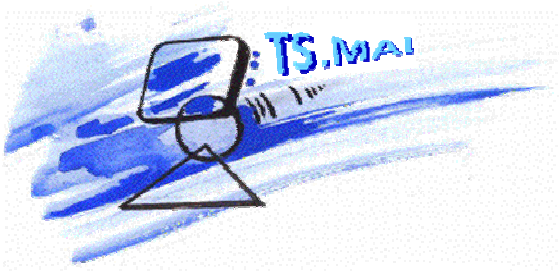
LES AUTOMATISMES

LA

PROGRAMMATION

Lycée L.RASCOL 10, Rue de la République
BP 218. 81012 ALBI CEDEX



SOMMAIRE

LA NORME CEI 1131

I.LA NORME CEI 1131

La normalisation des langages en informatique est un fait acquis (Pascal, C, C++, etc…). C'est un critère d'achat, aujourd'hui il n'existe plus sur le marché des progiciels de ce type qui ne respecterais pas les normes. En automatismes, bien que la part des automates programmables dans les systèmes automatisés soit très importante, la diversité est de règle. La part du logiciel devenant de plus en plus importante dans les applications, la nécessité de passer d'une programmation quasi artisanale à une certaine harmonisation s'est petit à petit imposer aux utilisateurs ainsi qu'aux constructeurs.
Les besoins d'une normalisation des langages pour API s'exprime, pour les industriels, en termes de:

- faciliter la formation des réalisateurs de configurations d'automates programmables
- obtenir un bon niveau de portabilité des programmes
- favoriser la création de bibliothèques de blocs fonctionnels fiables
- faciliter les configurations en réseau d'automates
- améliorer la qualité des applications (sûreté de fonctionnement, maintenabilité, extensibilité)
- réutiliser les outils de configuration et de programmation
- produire des dossiers d'applications homogènes
- faciliter la maintenance du logiciel d'application

Le CENELEC (Comité Européen de Normalisation Electrotechnique) a adopté en 1993 le texte (65B) de la CEI (Commission Electrotechnique Internationale) comme norme EN 61131 pour la commande des processus industriels. Cette norme traite des automates programmables en 5 parties :

- CEI 1131-1 définitions[1] , informations générales.
- CEI 1131-2 spécifications et essais matériels
- CEI 1131-3 langages de programmations
- CEI 1131-4 documentations
- CEI 1131-5 communications

L'originalité des langages de programmation pour automates est qu'ils sont généralement imagés par rapport à l'expression de la commande des machines automatisées. Ils sont souvent graphiques et depuis 20 ans des formes de langages se sont dégagés et sont mise à disposition par les constructeurs d'API (liste d'instruction mnémonique, réseau à contacts, GRAFCET).
La norme commence à être utiliser, des associations d'utilisateurs se sont crées en France et en Europe (EXERA, PLCopen) pour certifier des produits à un niveau de conformité "de base" et un niveau de "probabilité". Chez les constructeurs des langages de programmation "proches" sont annoncés (XTel et PL7Micro pour télémécanique, SFCd'Allen Bradley, Step5 et 7 de Siemens, etc…) des test de conformité sont fait par le CETIM relatifs aux prescriptions et à la syntaxe. Les nouveaux ateliers logiciels pour le Génie automatique sont développés pour fonctionner sur des PC avec des interfaces standard et des langages normalisés.

[1] Automate Programmable (AP) Système électronique fonctionnant de manière numérique, destiné à être utilisé dans un environnement industriel, qui utilise une mémoire programmable pour le stockage interne des instructions orientées utilisateur aux fins de mise en œuvre de fonctions spécifiques, telles que des fonctions de logique, de mise en séquence de temporisation, de comptage et de calcul numérique, pour commander au moyen d'entrées et de sorties tout ou rien ou analogiques divers types de machines ou processus. L'AP et ses périphériques associés sont conçus pour pouvoir facilement s'intégrer à un système d'automatisme industriel et être facilement utilisés dans toutes leurs fonctions prévues.

L'étude de la norme nous permettra de constater des principes largement éprouvés par le Génie logiciel tels que la structuration et la modularité. De distinguer les modules logiciels des langages dans lesquels ils peuvent être écrits. De faire la différence entre le modèle de spécification et le langage de programmation (code automate).
La norme CEI 1131 s'applique aux automates programmables industriels et à leurs périphériques. Les objectifs sont:

- **donner les définitions et identifier** les principales caractéristiques permettant de sélectionner et utiliser les automates programmables et leurs périphériques associés.
- **déterminer les prescriptions minimales** relatives aux caractéristiques fonctionnelles, aux conditions de service, aux caractéristiques constructives, à la sécurité générale ainsi qu'aux essais applicables aux automates programmables et à leurs périphériques.
- **définir pour les langages de programmation** les principaux champs d'applications, les règles syntaxiques et sémantiques ainsi que des ensembles de base simples mais exhaustifs d'éléments de programmation.
- fournir à l'utilisateur des informations générales didactiques et des recommandations quant à son application.

La partie 3 de cette norme définit :

Les langages de programmation
Les modules logiciels ou unités d'organisation de programmes

I.1. Ateliers logiciels

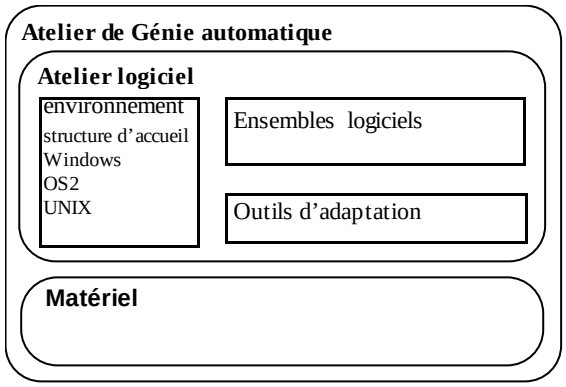
I.1.1 L'atelier de Génie Automatique

La complexité des traitements est induite par l'augmentation de la taille et par la multiplicité des fonctions demandées à l'API. Les fonctions classiques (combinatoire, séquentiel) représentent environ un tiers du logiciel à développer pour une application, le reste est souvent destiné:

- à l'aide à la maintenance et au diagnostic de panne
- à la communication
- aux fonctions spécifiques (régulation)
- aux fonction de commande d'axes, …

L'atelier logiciel vise à répondre au problème et est composé d'une gamme de produits dont certains peuvent être vendus séparément.

- structuration d'une application (architecture de commande)
- conception, codage, test
- gestion du processus de développement (# versions, projet, …)
- production des documents
- maintenance
- communication avec d'autres outils de CAO, …



L'atelier de Génie automatique ci-dessous est composé d'une partie matérielle (PC ou station de travail) et d'un atelier logiciel conçu dans un souci d'ergonomie. Des environnement graphiques différents peuvent être utilisés (OS2, Windows, Unix …), leur contribution est importante par rapport à des logiciels sous DOS.

I.1.2. L'atelier logiciel et les configurations d'API

Les potentialités des API sont déterminées par des fonctions programmables que l'on peut subdiviser en groupes orientés « application ». Dans le cadre d'une application, la partie

matérielle qui assure ces différentes fonctionnalités et les périphériques associés constituent une configuration d'automate programmable[2] .
Les ateliers logiciels disposent de mécanismes et de fonctionnalités communes , et s'appuient sur la norme CEI 1131.

I.1.3 Modèle logiciel
Le modèle logiciel écrit en langage évolué se compose d'éléments qui sont :

Les programmes et les blocs fonctionnels
Les éléments de configuration
- configuration
- ressources
- tâches
- variables globales et chemin d'accès

Reprise à froid
Reprise de la configuration d'AP et de son programme d'application après que toutes les données dynamiques {variables telles qu'image *E/S*, mémoires internes, temporisateurs, compteurs, etc. et contextes du programme) aient été ramenées à un état prédéterminé. Une reprise à froid peut être automatique (par exemple après une coupure de courant, une perte d'information dans la ou les parties dynamiques de la ou des mémoires, etc.) ou manuelle (par exemple bouton de réinitialisation, etc.).

Reprise immédiate
Reprise après une coupure d'alimentation intervenant pendant le laps de temps maximal fonction du processus alloué à la configuration d'AP pour rétablir son fonctionnement comme s'il n'y avait pas eu de coupure d'alimentation. Toutes les informations d'E/S et toutes les autres données dynamiques ainsi que le contexte du programme d'application sont restaurée ou demeurent inchangés.

Reprise à chaud
Reprise après une coupure d'alimentation avec un ensemble de données dynamiques prédéterminé et programmé par l'utilisateur et un contexte programme d'application prédéterminé par le système. Une reprise à chaud se caractérise par une signalisation d'état ou tout autre moyen apparenté mis à la disposition du programme d'application indiquant que l'interruption d'alimentation de la configuration d'AP à été détectée en mode run

II.LES LANGAGES DE PROGRAMMATION

II.1. Les éléments communs

La norme 61131.3 spécifie la syntaxe et la sémantique d'une série unifiée de langages pour automate programmable. Elle se compose de cinq langages classés en trois familles:

- Langages LITTERAUX
  - Langage IL : liste d'instructions
  - Langage ST : littéral structuré
- Langages GRAPHIQUES
  - Langage LD : langage à contacts (Ladder)
  - Langage FBD : diagramme fonctionnel — logigramme
- Langage SFC : diagramme fonctionnel en séquence — GRAFCET

[2] Configuration d'AP : Configuration réalisée par l'utilisateur et comportant un automate programmable et des périphériques associés, nécessaires au système automatisé prévu. Elle consiste en des unités interconnectées au moyen de câbles ou de raccords enfichables pour l'installation permanente, et au moyen de câbles ou d'autres dispositifs pour des périphériques portables et transportables.

Les représentations des données, dans les divers langages de programmation d'automates programmables, doivent se composer de libellés numériques, de cordons de caractères et de libelles de datation.

Libellés numériques;

Libellés entiers -----> -12 ; 0 ; +986 ; 123

Libellés réels ------> -12,0 ; 0,0 ; 0,45623 ; 3,1416 ; 1,34E-1,2

Libellés de cordons de caractères;

Séquence de zéro à plusieurs caractères précédée et terminée par une apostrophe (‘),’A’

Libellés de datation ; nécessité de distinguer deux types de données temporelles:

Relative à la durée pour la mesure et le contrôle du temps écoulé, TIME #10s

Relative à l'heure du jour pour l'expression d'une date, DATE # 2000-10-28

Identificateurs

Un *identificateur* est un cordon de lettres de chiffres et de caractères de soulignement qui doit commencer par une lettre ou par un caractère de soulignement.

Les identificateurs ne doivent pas comporter de caractères d'espaces intercalaires (SP).

Exemple : Cs_tpav, dcy,

Commentaires

Les *commentaires* de l'utilisateur doivent être respectivement délimités, à leur début et à leur fin, par la combinaison de caractères spéciaux « (* » et « *) ».

Exemple : dcy : (* bouton poussoir départ cycle *)

II.2. Les données

II.2.1 données élémentaires

Défini le mot clé relatif à chaque type de donnée, le nombre de bits, la plage des valeurs relative à chaque type de données (tableau 10 annexe A)

II.2.2 données génériques

Rassemblement de données élémentaires d'une même famille, la hiérarchie des types de données génériques est donnée dans le tableau ci-dessous. Elles sont identifiées par le préfixe “ANY” (tableau 11 annexe A)

II.2.3 données dérivées

Les données dérivées sont celles spécifiées par l'utilisateur ou le fabricant et sont déclarées à l'aide de la construction littérale:

TYPE

END_TYPE

II.3. Les variables

Les variables fournissent un moyen pour identifier des objets de données dont le contenu peut varier, comme par exemple les données associées aux entrées/sorties.

II.3.1 variable à un élément

Une variable à un seul élément est définie comme une variable qui représente une valeur mono élément. La représentation d'une variable à un seul élément est assurée par le symbole “**%**” combiné avec un préfixe d'emplacement et un préfixe de taille.

II.3.2 variable à plusieurs éléments

Les variables à plusieurs éléments sont des tableaux et des structures. Les indices sont mis entre crochets [ ] et séparés par des virgules.

ARRAY [1…10 , 1… 8] → tableau 2 dimensions à 80 éléments

II.3.3 déclaration de variable

Chaque unité d'organisation de programme doit comporter une partie déclaration qui spécifie le type (et l'emplacement si possible) des variables utilisées dans le module logiciel. Cette déclaration se fait sous forme littérale en utlisant des mots clés (VAR … END_VAR) ou sousn forme graphique. Les valeurs initiales peuvent être données dans la déclaration.

Mot clé	Désignation de la variable
VAR	interne au module
VAR_INPUT	variable d'entrée du module
VAR_OUTPUT	variable de sortie du module
VAR_IN_OUT	fournie par des entitées et modifiable dans le module (A=A+1
VAR_EXTERNAL	fournie par la configuration
VAR_GLOBAL	déclaration de variable globale
VAR_ACCES	déclaration de chemin d'accès
RETAIN	variable non volatile
CONSTANT	constante
AT	énoncé d'emplacement
END_VAR	fin de déclaration

II.3.4 initialisation des variables

Chacune des variables associées à l'élément de configuration et à son programme peut prende une valeur initiale.

II.4. Langage IL (Liste d'Instructions)

Langage à liste d'instructions (IL) est composé d'une suite d'instruction, chaque instruction doit débuter une nouvelle ligne de programme et doit contenir un opérateur suivi d'une ou plusieurs opérandes. L'instruction peut être précédée d'une étiquette d'identification avec (:)
S'il y a présence d'un commentaire il doit constituer le dernier élément de la ligne.
Le format d'une ligne d'instruction et les opérateurs standards sont définis (tableaux 51&52 annexe A)

Tableau 51 – Exemples de champs d'instruction

Etiquette	Opérateur	Opérande	Commentaire
START:	LD	%IX1	(* BOUTON POUSSOIR *)
	ANDN	%MX5	(* NON INHIBÉE *)
	ST	%QX2	(* MARCHE VENTILATEUR *)

II.5. Langage structuré ST

Le langage litteral structuré est constitué d'expression litterale composée d'opérateur et d'opérande libellé conformément à la norme. Les opérateurs booléens du langage ST sont ressemblant aux précédents, on retrouve : AND, OR, XOR, et NOT pour la complémentation (voir tableau 55 annexe A).

L'écriture du programme se fait sous forme d'énoncés littérale . Une ligne de programmation peut avoir la structure générale suivante

Commentaires

Etiquettes : énoncé …..

AFFECTATION

L'énoncé d'affectation remplace la valeur actuelle d'une variable par le résultat de l'évaluation d'une expression. Un énoncé d'affectation doit se composer d'une référence de variable à gauche, suivie de *l'opérateur d'affectation* **(:=)**, suivi de l'expression à évaluer.

SELECTION

Les énoncés de sélection comprennent les énoncés **IF** et **CASE**

L'énoncé IF indique qu'un groupe d'énoncés ne doit être exécuté que si la valeur prise par l'expression booléenne associée est 1 (vraie) ;THEN. Si la condition n'est pas vérifiée, soit aucun énoncé n'est exécuté, soit le groupe d'énoncés suivi du mot clé ELSE doit être exécuté.

L'énoncé CASE se compose d'une expression qui doit être évaluée sur une variable de type INT (que J'on appelle le "sélecteur"), et d'une liste de groupes d'énoncés, chacun d'entre eux étant étiqueté par une ou plusieurs plages de valeurs entières. Cet énoncé précise que le premier groupe d'énoncés, dont l'une des plages contient la valeur calculée du sélecteur, doit être exécuté. Si la valeur du sélecteur n'apparaît dans aucune plage d'aucun cas, la suite d'énoncés qui suit le mot clé ELSE. Dans le cas contraire aucune des suites d'énoncés ne doit être exécuté

ITERATION

Les énoncés d'itération comprennent les énoncés **FOR**, **WHILE** et **REPEAT**

L'énoncé FOR indique qu'une suite d'énoncés doit être exécutée de manière répétitive jusqu'à ce que le mot clé END-FOR soit atteint, une suite de valeurs progressives étant affectée à la variable de commande de la boucle FOR. La variable de commande, la valeur initiale et la valeur finale doivent être des expressions du même type entier (SINT, INT ou DINT). L'énoncé FOR augmente ou diminue la variable de commande d'une valeur initiale jusqu'à une valeur finale par incréments déterminés par la valeur d'une expression; par défaut, cette valeur est 1.

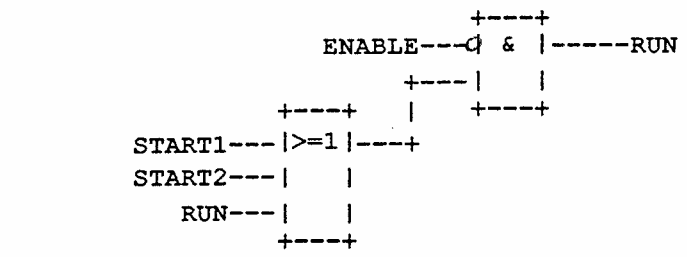
L'énoncé WHILE déclenche l'exécution répétée de la suite d'énoncés s'étendant jusqu'au mot clé END-WHILE, jusqu'à ce que l'expression booléenne associée soit fausse. Si, à l'origine, l'expression est fausse, le groupe d'énoncés n'est pas exécuté du tout.

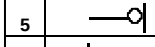
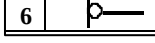
L'énoncé REPEAT entraîne l'exécution répétée (et au moins une fois) de la suite d'énoncés s'étendant jusqu'au mot clé UNTIL, jusqu'à ce que la condition booléenne associée soit vraie.

l'énoncé EXIT doit permettre de sortie de toutes les boucles d'itérations

II.6. Langage FBD (langage en Blocs Fonctionnels)

Le langage FBD est un langage graphique de programmation (type logigramme) compatible avec la norme CEI 617-12.

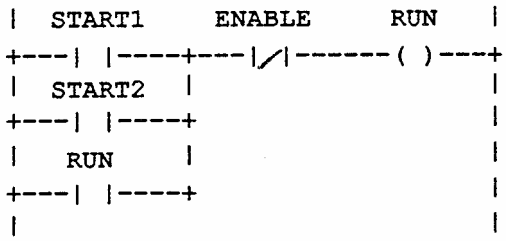


N°	symbole	désignation
1	&	ET
2	>=1	OU
3	=1	OU exclusif
4	1	égalité
5		complément d'entrée
6		complément de sortie

II.7. Langage à contacts LD (Ladder)

Le langage à contacts permet la programmation à l'aide de symboles graphiques. Ces symboles sont organisés en réseau reliés à gauche et à droite à des barres d'alimentation.

N°	Symbole	Désignation
1	⊣ ⊢	contact Normalement Overt
2	⊣/⊢	Contact Normalement Fermé
3	⊣P⊢	Contact détection front montant
4	⊣N⊢	Contact détection front descendant
5	—( )—	bobinage
6	—(/)—	bobinage complémenté
7	—(S)—	bobinage SET (vérouillage)
8	—(R)—	bobinage RESET (dévérouillage)
9	—(P)—	bobinage Front Montant
10	—(N)—	bobinage Front descendant



III. UNITES D'ORGANISATION DE PROGRAMMES.

III.1. Définition

Les unités d'organisation de programmes, définies par la norme, sont aux nombre de trois:

- la FONCTION
- le BLOC FONCTIONNEL
- le PROGRAMME

Ces unités d'organisation de programmes (ou modules logiciels) peuvent être fournies par le constructeur d'automate programmable (modules standards), ou programmées par l'utilisateur à l'aide des modules standards

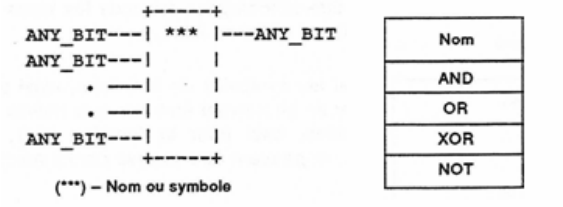
III.2. La fonction

III.2.1 définition

La fonction est définie comme un module logiciel pouvant avoir plusieurs variables d'entrées, mais une et une seule variable de sortie. Le lancement d'une fonction dotée des mêmes valeurs d'entrées doit toujours donner la même valeur de sortie. Une fonction n'a donc pas de mémoire.

III.2.2 représentation

Les fonctions peuvent être représentées soit graphiquement, soit littéralement(Voir annexe A)
Une entrée EN (Enable, validation) et une sortie ENO (pas d'erreur) booléenne supplémentaire peuvent être utilisées.



III.2.3 déclaration

Une fonction doit être déclarée graphiquement ou littéralement. La déclaration d'une fonction doit comporter les éléments suivant.

- le mot clé FUNCTION … END_FUNCTION
- la déclaration des variables uitlisées par la fonction
- le corps de fonction dans un des langages normalisé.

Exemple:

Etude de la fonction pesage : poids NET = poids BRUT – tare

FONCTION_PESEE

III.3. Le bloc fonctionnel

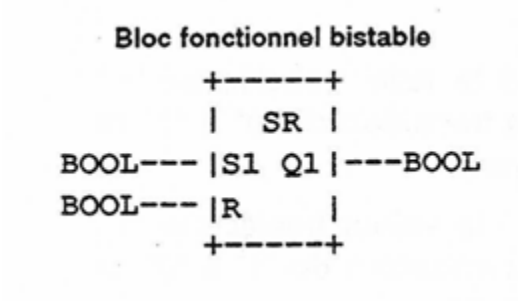
III.3.1 définition

Le bloc fonctionnel est défini comme un module logiciel pouvant avoir plusieurs sorties. Le bloc fonctionnel possède une mémoire interne, les variables internes sont transparentes pour l'utilisateur. Il est possible de créer plusieurs instances d'un bloc fonctionnel, chaque instances étant identifiée par un nom.

III.3.2 représentation

La représentation d'un bloc fonctionnel ou d'une instance de bloc fonctionnel peut être soit graphique soit littérale.

La norme défini un certains nombre de blocs, les blocs fonctionnels standards (mémoire, tempo, ect …), mais des blocs fonctionnels utilisateur peuvent être construit à l'aide de Fonctions et de Blocs fonctionnels standards.



III.3.3 déclaration

Un bloc fonctionnel peut être déclaré graphiquement ou littéralement. Cette déclaration doit comporter les éléments suivant:

- les mots clés FUNCTION_BLOCK …… END_FUNCTION_BLOCK
- la déclaration des variables utilisées par le bloc fonctionnel
- la définition du corps du bloc fonctionnel

Exemple: Extrait de la norme EN 61131

Le bloc fonctionnel CMD_MONITOR correspond à la commande d'un ensemble d'exploitation qui réagit a un ordre de marche (la sortie CMD) et renvoie un signal de bonne exécution (entrée FDBK). Le bloc fonctionnel permet une commande manuelle (MAN_CMD) ou auto (AUTO_CMD) suivant l'entrée AUTO_MODE. La vérification de MAN_CMD se fait par MAN_CMD_CHK.

Si aucune confirmation de la commande CMD dans un delais (T_CMD_MAX) la commande est annulée et une alarme est signalée.

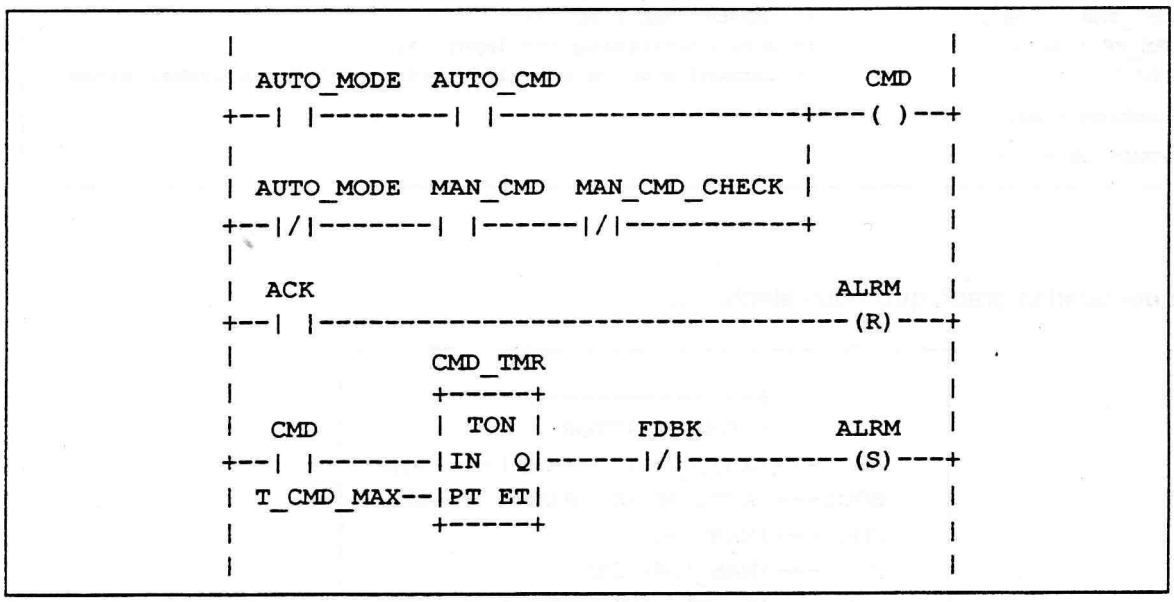
Déclaration sous forme littérale

```
FUNCTION_BLOCK CMD_MONITOR
  VAR_INPUT AUTO_CMD : BOOL ;   (* Automated command *)
  AUTO_MODE : BOOL ;            (* AUTO_CMD enable *)
  MAN_CMD : BOOL ;              (* Manual command *)
  MAN_CMD_CHK : BOOL ;          (* Negated MAN_CMD to debounce *)
  T_CMD_MAX : TIME ;            (* Max time from CMD to FDBK *)
  FDBK : BOOL ;                 (* Confirmation of CMD completion by operative unit *)
  ACK : BOOL ;                  (* Acknowledge/cancel ALRM *)
END_VAR
VAR_OUTPUT CMD : BOOL ;         (* Command to operative unit *)
  ALRM : BOOL ;                 (* T_CMD_MAX expired without FDBK *)
END_VAR
VAR CMD_TMR : TON ;             (* CMD-to-FDBK timer *)
  ALRM_FF : SR ;                (* Note over-riding "S" input: *)
END_VAR                         (* Command must be cancelled before "ACK" can cancel alarm *)
(* Function Block Body *)
END_FUNCTION_BLOCK
```

Corps du bloc fonctionnel en langage ST.

```
CMD := AUTO_CMD & AUTO_MODE
       OR MAN_CMD & NOT MAN_CMD_CHK & NOT AUTO_MODE ;
CMD_TMR (IN := CMD, PT := T_CMD_MAX) ;
ALRM_FF (S1 := CMD_TMR.Q & NOT FDBK, R := ACK) ;
ALRM := ALRM_FF.Q1 ;
```

Corps du bloc fonctionnel en langage LD



III.4. Le programme

Le programme est défini comme un *" ensemble logique de tous les éléments et constructions des langages de programmation nécessaires pour le traitement des données requis pour contrôler une machine ou un processus au moyen d'une configuration d'automate programmable"* .Un programme est construit à l'aide de Fonctions et de Blocs Fonctionnels.
Un programme ne peut être instancié que dans des ressources, alors que les blocs fonctionnels ne peuvent être instanciés que dans des programmes ou dans d'autres blocs fonctionnels.

III.4.1 déclaration.

La déclaration et l'utilisation de programmes sont identiques aux blocs fonctionnels :

- mots clés PROGRAM …… END_PROGRAM
- les déclarations de variables, un programme pouvant utiliser des variables globales ou/et des chemins d'accès.

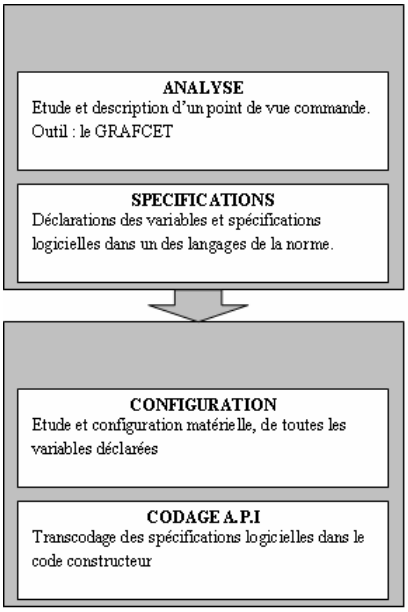
Exemple :
```
PROGRAM  ESSAI_1
        VAR_INPUT  x : BOOL ; END_VAR
        VAR_OUTPUT  y : BOOL ; END_VAR
        VAR_EXTERNAL  z : WORD ; END_VAR
    FB1 (…) ;
    FB2 (…) ;
END_PROGRAM
```

Conclusion.

La conformité à une norme est pour les utilisateurs un gage de qualité, de maintenabilité, et d'évolutivité. A la demande de certains constructeurs, des tets sont réalisés par le CETIM[3] et de l'EXERA[4]qui ont pour objectifs de vérifier la conformité. La norme représente une synthèse d'un état de l'art du génie logiciel sur des principes éprouvés tels que la structuration et la modularité. Dans l'application de cette norme , il est important:

- de distinguer les modules logiciels des langages dans lesquels ils peuvent être écrits
- de différencier le modèle de spécifications et le langage de programmation.

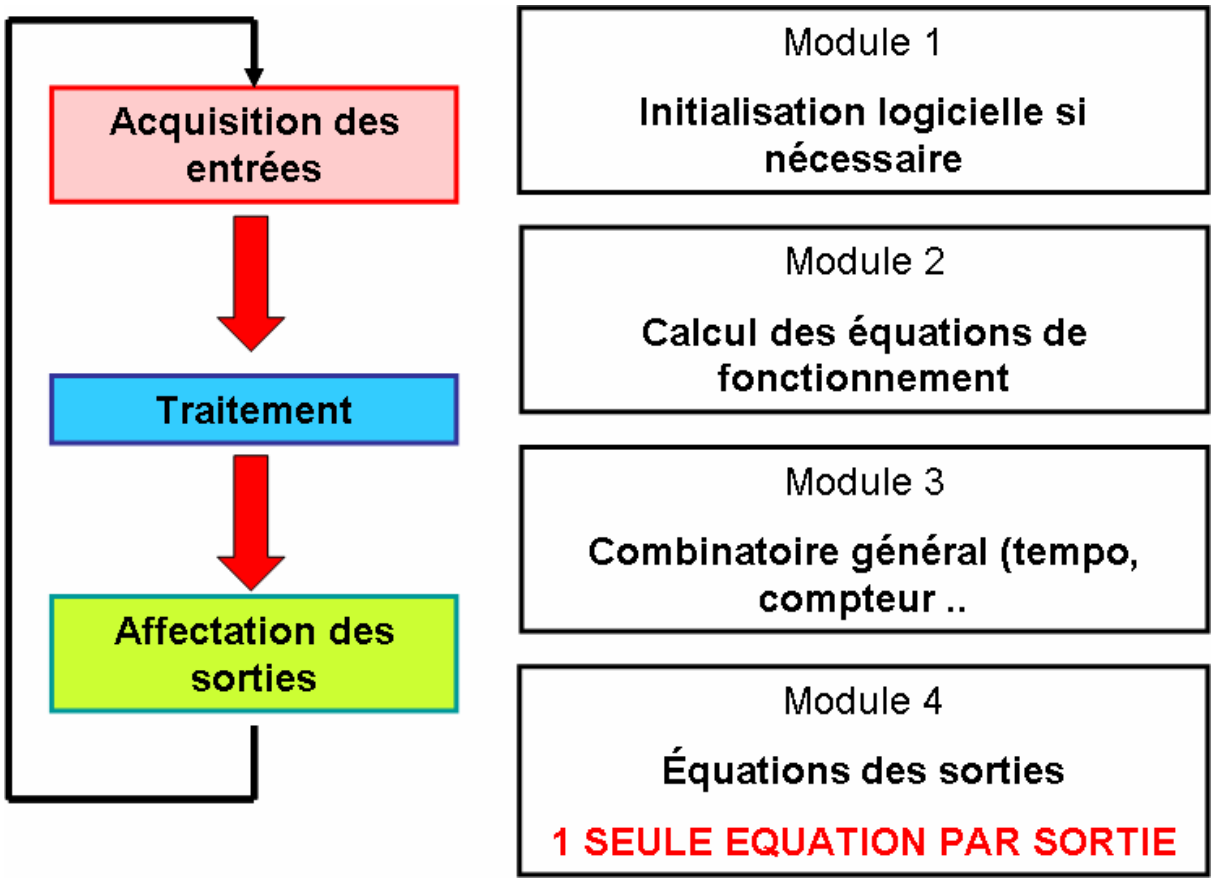
La démarche, ci-contre peut être mise en œuvre pour mener une étude de comception logicielle. Le bloc conception et le bloc réalisation n'ont pas obligatoirement les mêmes auteurs.



[3] Centre Technique des Industries Mécaniques
[4] Association des Exploitants d'Equipements de Mesure, de Régulation et d'Automatismes

IV. STRUCTURE D'UN PROGRAMME

Le moniteur système de l'automate gérant l'acquisition et l'affectation des entrées/sorties la partie programme de l'application de l'utilisateur respectera l'organisation générale ci-dessous. L'implémentation des différents modules est fonction de la structure de la mémoire programme de l'automate.
Avant tout travail de codage API, la configuration matérielle est obligatoire. Cette configuration prévoit **l'adressage des entrées sorties, document important du dossier d'automatisme,** en fonction des caractéristiques du matériel mis en service. Ce document peut se présenter sous la forme de tableaux directement accéssibles dans l'atelier logiciel.



Exemple de spécification programme :

Etude du comparateur 4 bits
Deux valeurs A{a0;a1;a2;a3} et B{b0;b1;b2 ;b3} répondant aux équations suivantes :

F(A>B)=a3./b3+(a3Θb3)a2/b2+(a3Θb3)(a2Θb2)a1./b1+(a3Θb3)(a2Θb2)(a1Θb1)a0./b0
F(A<B)= /a3.b3+(a3Θb3)/a2.b2+(a3Θb3)(a2Θb2)/a1.b1+(a3Θb3)(a2Θb2)(a1Θb1)./a0.b0
F(A=B)= (a3Θb3)(a2Θb2)(a1Θb1)(a0Θb0)

```
FONCTION_COMPA
(* déclaration des variables*)
    VAR_INPUT
        a : ARRAY[0..3] OF BOOL     ;   (*valeur des poids binaire de A*)
        b : ARRAY[0..3] OF BOOL     ;   (*valeur des poids binaire de B*)
    END_VAR
    VAR_OUPUT
        Sup : BOOL ; (*valeur A>B *)
        Inf :  BOOL ; (* valeur A<B *)
        Egal : BOOL ; (* valeur A=B *)
    END_VAR
    VAR
        Vi : ARRAY [0..15] OF BOOL ;   (* variables internes utilisées *)
    END_VAR

(* corps de fonction *)

    Vi[0] := a[0] XOR b[0]
    Vi[1] := a[1] XOR b[1]
    Vi[2] := a[2] XOR b[2]
    Vi[3] := a[3] XOR b[3]
    Vi[4] := (NOT Vi[3]) AND (NOT Vi[2])
    Vi[5] := (NOT Vi[4]) AND (NOT Vi[1])
    Vi[6] := (NOT Vi[5]) AND (NOT Vi[0])
    Vi[7] := (NOT Vi[3] AND NOTb[2] ANDa[2]
    Vi[8] := Vi[4] AND NOTb[1] ANDa[1]
    Vi[9] := Vi[5] AND NOTb[0] ANDa[0]
    Vi[10] := (NOT Vi[3] AND NOTa[2] ANDb[2]
    Vi[11] := Vi[4] AND NOTa[1] ANDb[1]
    Vi[12] := Vi[5] AND NOTa[0] ANDb[0]

    Sup := a[3] AND NOTb[3] OR Vi[7] OR Vi[8] OR Vi[9]
    Inf := b[3] AND NOTa[3] OR Vi[10] OR Vi[11] OR Vi[12]
    Egal := NOT ( Sup OR Inf )

END_FONCTION
```

V.TRAVAIL SUR BIT

L'objet de ce paragraphe est la programmation d'équations booléennes, combinatoires et/ou séquentielles. La norme défini les fonctions et les blocs fonctionnels utilisables dans les spécifications pour la programmation d'équations binaires.

V.1. Fonctions booléennes. (Voir annexe A)

V.1.1 fonctions booléennes standard.

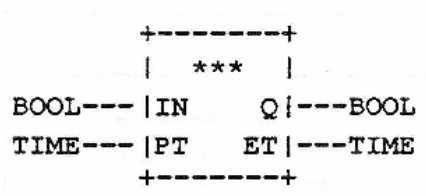
V.1.2 fonctions spéciales de traitement de cordons de bits.
Ces fonctions sont liées au décalage droite ou gauche

V.2. Blocs fonctionnels. (voir annexe A)

V.2.1 blocs fonctionnels standards: Bistables et détection de front
La sortie “Q “d’un bloc fonctionnel front passe à la valeur “1” au passage de “ 0 —>1” de l’entrée “CLK” d’un bloc R_TRIG ou, de “ 1—>0” pour un bloc F_TRIG , et reste à “1” de l'exécution du bloc à la suivante.

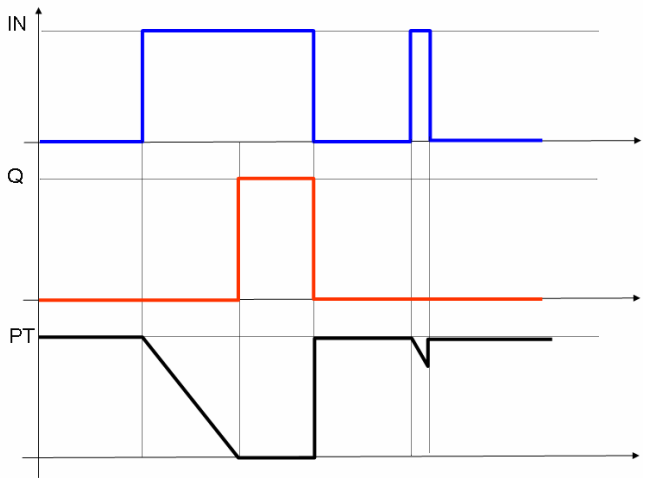
V.2.2 blocs fonctionnels de comptage
Les blocs fonctionnels standards de comptage sont au nombre de trois :
- CTU bloc compteur
- CTD bloc décompteur
- CTUD bloc fonctionnel mixte de comptage/décomptage

V.2.3 blocs foctionnels de temporisation

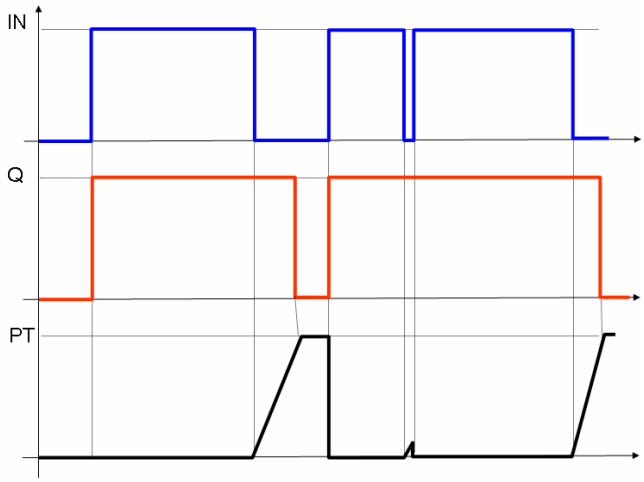


TP (impulsion)
TON (Enclenchement)
T---0 (Enclenchement)
TOF (Déclenchement)
0---T (Déclenchement)

Les blocs fonctionnels standards de temporisateurs (TON)



Les blocs fonctionnels standards de temporisateurs (TOF)



Annexe A Norme 61131 Eléments communs

identificateurs

Un *identificateur* est un cordon de lettres. de chiffres et de caractères de soulignement qui doit commencer par une lettre ou par un caractère de soulignement.

Les identificateurs ne doivent pas comporter de caractères d'espaces intercalaires (SP).

Exemple : Cs_tpav, dcy,

commentaires

Les *commentaires* de l'utilisateur doivent être respectivement délimités, à leur début et à leur fin, par la combinaison de caractères spéciaux « (* » et « *) ».

Exemple : dcy : (* bouton poussoir départ cycle *)

Tableau 10 – Types de données élémentaires

N°	Mot clé	Type de donnée	Bits	Étendue
1	BOOL	Booléen	1	Note 8
2	SINT	Entier court	8	Note 2
3	INT	Entier	16	Note 2
4	DINT	Entier double	32	Note 2
5	LINT	Entier long	64	Note 2
6	USINT	Entier court non signé	8	Note 3
7	UINT	Entier non signé	16	Note 3
8	UDINT	Entier double non signé	32	Note 3
9	ULINT	Entier long non signé	64	Note 3
10	REAL	Nombre réels	32	Note 4
11	LREAL	Réels longs	64	Note 5
12	TIME	Durée	Note 1	Note 6
13	DATE	Date (uniquement)	Note 1	Note 6
14	TIME_OF_DAY ou TOD	Heure du jour (uniquement)	Note 1	Note 6
15	DATE_AND_TIME ou DT	Date et heure du jour	Note 1	Note 6
16	STRING	Cordon de caractères de longueur variable	Note 1	Note 7
17	BYTE	Cordon de bits de longueur 8	8	Note 7
18	WORD	Cordon de caractères de longueur 16	16	Note 7
19	DWORD	Cordon de caractères de longueur 32	32	Note 7
20	LWORD	Cordon de caractères de longueur 64	64	Note 7

NOTES

1 La longueur de ces données dépend de l'application concernée.

2 L'étendue des valeurs, relatives à des variables de ce type de donnée, est comprise entre -(2**(Bits-1)) et (2**(Bits-1))-1.

3 L'étendue des valeurs, relatives à des variables de ce type de donnée, est comprise entre 0 et (2**Bits)-1.

4 L'étendue des valeurs, relatives à des variables de ce type de donnée doit être telle que définie dans la CEI 559 pour le format de virgule flottante à largeur binaire unique.

5 L'étendue des valeurs, relatives à des variables de ce type de donnée, doit être telle que définie dans la CEI 559 pour le format de virgule flottante à double largeur binaire.

6 L'étendue des valeurs relatives à des variables de ce type de donnée dépend de l'application concernée.

7 Une étendue numérique de valeurs ne s'applique pas à ce type de donnée.

8 Les valeurs que peut prendre variables de ce type de donnée doivent être 0 et 1, correspondant respectivement aux mots clés FAUX et VRAI.

Tableau 11 – Hiérarchie des types de données génériques

ANY ANY_NUM ANY_REAL LREAL REAL ANY_INT LINT, DINT, INT, SINT ULINT, UDINT, UINT, USINT ANY_BIT LWORD, DWORD, WORD, BYTE, BOOL STRING ANY_DATE DATE_AND_TIME DATE TIME_OF_DAY TIME Dérivés (voir notes)
NOTES 1 Les types de données génériques ne doivent pas être utilisés dans des unités d'organisation de programmes déclarées par l'utilisateur, telles que spécifiées en 2.5. 2 Le type générique d'un type dérivé à partir d'une *sous-étendue* (caractéristique 3 du tableau 12) doit être ANY_INT. 3 Le type générique d'un type *directement dérivé* (caractéristique 1 du tableau 12) doit être identique au type générique du type élémentaire dont il est dérivé. 4 Le type générique de tous les autres types dérivés définis au tableau 12 doit être ANY.

Le Langage IL (Liste d'Instructions)

Tableau 52 – Opérateurs de liste d'instructions

N°	Opérateur	Modificateurs	Opérande	Sémantique
1	LD	N	Note 2	Rendre le résultat courant égal à l'opérande
2	ST	N	Note 2	Mémoriser le résultat à l'emplacement de l'opérande
3	S	Note 3	BOOL	Positionner l'opérande booléen à 1
	R	Note 3	BOOL	Remettre l'opérande booléen à 0
4	AND	N, (	BOOL	AND booléen
5	&	N, (	BOOL	AND booléen
6	OR	N, (	BOOL	OR booléen
7	XOR	N, (	BOOL	OR exclusif booléen
8	ADD	(	Note 2	Addition
9	SUB	(	Note 2	Soustraction
10	MUL	(	Note 2	Multiplication
11	DIV	(	Note 2	Division
12	GT	(	Note 2	Comparaison: >
13	GE	(	Note 2	Comparaison: >=
14	EQ	(	Note 2	Comparaison: =
15	NE	(	Note 2	Comparaison: <>
16	LE	(	Note 2	Comparaison: <=
17	LT	(	Note 2	Comparaison: <
18	JMP	C, N	LABEL	Saut vers l'étiquette
19	CAL	C, N	NAME	Appel d'un bloc fonctionnel (note 4)
20	RET	C, N		Retour d'une fonction appelée ou d'un bloc fonctionnel
21	)			Evaluation d'une opération différée

NOTES

1 Se reporter au texte précédent pour toute explication relative aux modificateurs et à l'évaluation des expressions.

2 Ces opérateurs doivent être soit surchargés soit saisis comme défini en 2.5.1.4. Le résultat courant et l'opérande doivent être du même type.

3 Ces opérations sont effectuées si et seulement si le résultat courant a la valeur booléenne 1.

4 Le nom du bloc fonctionnel est suivi par une liste d'arguments entre parenthèses, telle que définie en 3.2.3.

5 Lorsqu'une instruction JMP est contenue dans une construction ACTION...END_ACTION, l'opérande doit être une étiquette à l'intérieur de la même construction.

Le Langage ST (littéral structuré)

Le langage littéral structuré est constitué d'expression. Une expression est une construction syntaxique qui, lorsqu'elle est évaluée, fournit une valeur correspondante à l'un des types de données définies. Les expressions sont composées d'opérateur et d'opérandes..

Tableau 55 – Opérateurs du langage ST

N°	Opération	Symbole	Priorité
1	Mise entre parenthèses	(Expression)	MAXIMALE
2	Evaluation de Fonction Exemples:	Identificateur (liste d'arguments) LN(A), MAX(X,Y), etc.	
3	Exponentiation	**	
4 5	Négation Complément	– NOT	
6 7 8	Multiplication Division Modulo	* / MOD	
9 10	Addition Soustraction	+ –	
11	Comparaison	< , > , <= , >=	
12 13	Egalité Inégalité	= <>	
14 15	AND booléen AND booléen	& AND	
16	OR exclusif booléen	XOR	
17	OR booléen	OR	MINIMALE

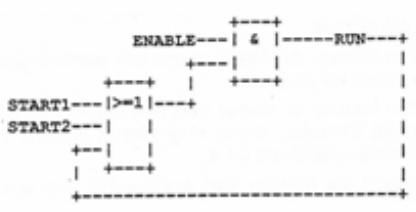
NOTES

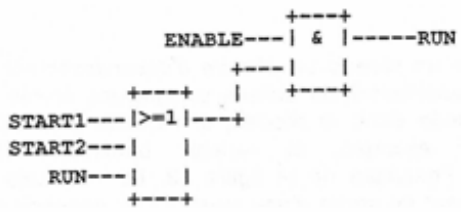
1 Les mêmes restrictions s'appliquent aux opérandes de ces opérateurs et aux entrées des fonctions correspondantes définies en 2.5.1.5.

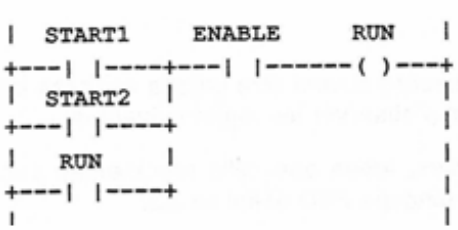
2 Le résultat de l'évaluation de A**B doit être le même que le résultat de l'évaluation de la fonction EXPT (A, B) telle que définie au tableau 24.

Les Langages Graphiques

Les langages graphiques sont le langage à contacts (LD) et le langage en blocs fonctionnels (FBD).







UNITES D'ORGANISATION DE PROGRAMMES

LES FONCTIONS

Tableau 26 – Fonctions booléennes standards au niveau du bit

Forme graphique	Exemple d'application
	A := AND(B,C,D) ; ou A := B & C & D ;

N°	Nom	Symbole	Description
5	AND	& (note 1)	OUT := IN1 & IN2 & ... & INn
6	OR	>=1 (note 2)	OUT := IN1 OR IN2 OR ... OR INn
7	XOR	=2k+1 (note 2)	OUT := IN1 XOR IN2 XOR ... XOR INn
8	NOT		OUT := NOT IN1 (note 4)

NOTES

1 Ce symbole convient à l'utilisation en tant qu'opérateur dans des langages littéraux, comme l'illustrent les tableaux 52 et 55.

2 Ce symbole ne convient pas à l'utilisation en tant qu'opérateur dans des langages littéraux.

3 Les notations IN1, IN2, ..., INn se rapportent aux entrées dans un ordre descendant (haut vers le bas); OUT se rapporte à la sortie.

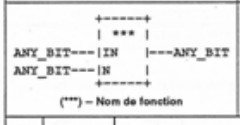
4 L'inversion graphique de signes de type BOOL peut être également effectuée conformément aux indications du tableau 19.

5 Lorsque la représentation nommée d'une fonction est acceptée, cela doit être indiqué par le suffixe "n" dans l'énoncé de conformité. Par exemple: "5n" représente la notation "AND".

6 Lorsque la représentation symbolique d'une fonction est acceptée, cela doit être indiqué par le suffixe "s" dans l'énoncé de conformité. Par exemple: "5s" représente la notation "&".

7 Des exemples d'application et des descriptions sont donnés en langage ST, tel qu'il est défini en 3.3.

Tableau 25 – Fonctions standards de décalage binaire

Forme graphique	Exemple d'application
	A := SHL(IN := B, N := 5) ; (Langage ST – voir 3.3)

N°	Nom	Description
1	SHL	OUT := IN Décalage à gauche de N bits, remplissage de zéros à droite
2	SHR	OUT := IN Décalage à droite de N bits, remplissage de zéros à gauche
3	ROR	OUT := IN Rotation à droite de N bits, circulaire
4	ROL	OUT := IN Rotation à gauche de N bits, circulaire

NOTE – La notation "OUT" se rapporte à la sortie de la fonction.

LES BLOCS FONCTIONNELS

Tableau 34 – Blocs fonctionnels bistables standards

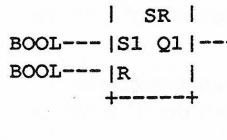
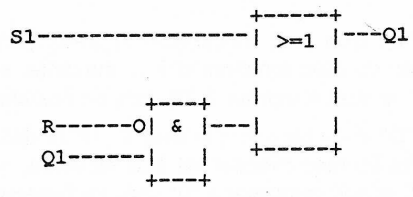
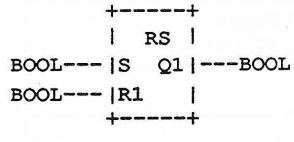
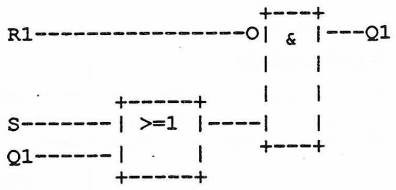
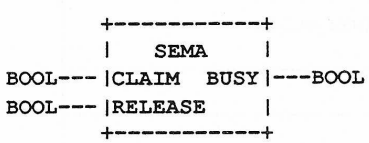
N°	Forme graphique	Corps de bloc fonctionnel
1	Bloc fonctionnel bistable (forcé dominant) (notes 1 et 2)	
		
2	Bloc fonctionnel bistable (réinitialisé dominant) (notes 1 et 2)	
		
3	Sémaphore avec "Essai et Positionnement" en continu (notes 3, 4, 5 et 6)	
		VAR X : BOOL := 0 ; END_VAR BUSY := X ; IF CLAIM THEN X := 1 ; ELSIF RELEASE THEN BUSY := 0 ; X := 0 ; END_IF

Tableau 35 – Blocs fonctionnels standards de détection de fronts

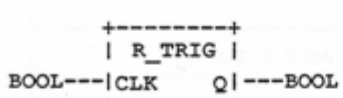
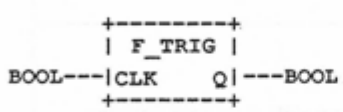
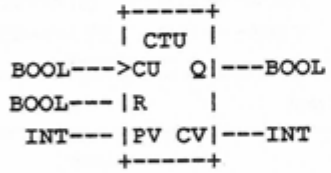
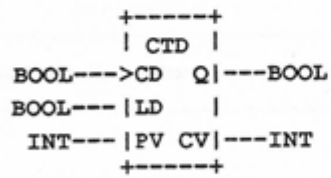
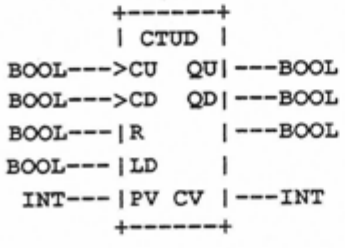
N°	Forme graphique	Définition (langage ST – voir 3.3)
1	Détecteur de front montant	
		FUNCTION_BLOCK R_TRIG VAR_INPUT CLK : BOOL ; END_VAR VAR_OUTPUT Q : BOOL ; END_VAR VAR M : BOOL := 0 ; END_VAR Q := CLK AND NOT M ; M := CLK ; END_FUNCTION_BLOCK
2	Détecteur de front descendant	
		FUNCTION_BLOCK F_TRIG VAR_INPUT CLK : BOOL ; END_VAR VAR_OUTPUT Q : BOOL ; END_VAR VAR M : BOOL := 1 ; END_VAR Q := NOT CLK AND NOT M ; M := NOT CLK ; END_FUNCTION_BLOCK

Tableau 36 – Blocs fonctionnels standards de pteurs

N°	Forme graphique	Corps de bloc fonctionnel (langage ST – voir 3.3)
1	Compteur (comptage)	
		IF R THEN CV := 0 ; ELSIF CU AND (CV < PVmax) THEN CV := CV+1 ; END_IF ; Q := (CV >= PV) ;
2	Compteur (décomptage)	
		IF LD THEN CV := PV ; ELSIF CD AND (CV > PVmin) THEN CV := CV-1 ; END_IF ; Q := (CV <= 0) ;
3	Compteur (comptage-décomptage)	
		IF R THEN CV := 0 ; ELSIF LD THEN CV := PV ; ELSIF CU AND (CV < PVmax) THEN CV := CV+1 ; ELSIF CD AND (CV > PVmin) THEN CV := CV-1 ; END_IF ; QU := (CV >= PV) ; QD := (CV <= 0) ;

NOTE - Les valeurs numériques des variables limites PVmin et PVmax sont propres à l'application concernée.

Tableau 37 – Blocs fonctionnels standards de temporisateurs

<table>
<tr><th>N°</th><th>Description</th><th>Forme graphique</th></tr>
<tr><td>1</td><td>***est: TP (impulsion)</td><td rowspan="4"><pre>
+-------+
| *** |
BOOL---|IN   Q|---BOOL
TIME---|PT   ET|---TIME
+-------+
</pre></td></tr>
<tr><td>2a</td><td>TON (Enclenchement)</td></tr>
<tr><td>2b</td><td>T---0 (Enclenchement)</td></tr>
<tr><td>3a</td><td>TOF (Déclenchement)</td></tr>
<tr><td>3b</td><td>0---T (Déclenchement)</td><td></td></tr>
<tr><td>4</td><td colspan="2">Horloge temps réel</td></tr>
<tr><td></td><td>PDT = Date et heure prédéfinies, chargées sur le front montant de EN
CDT = Date et heure du jour, valables lorsque EN=1
Q = copie de EN</td><td><pre>
+-------+
| RTC |
BOOL---|EN   Q|---BOOL
DT-----|PDT CDT|-----DT
+-------+
</pre></td></tr>
<tr><td colspan="3">NOTE - Dans les langages littéraux, les caractéristiques 2b et 3b ne doivent pas être utilisées.</td></tr>
</table>

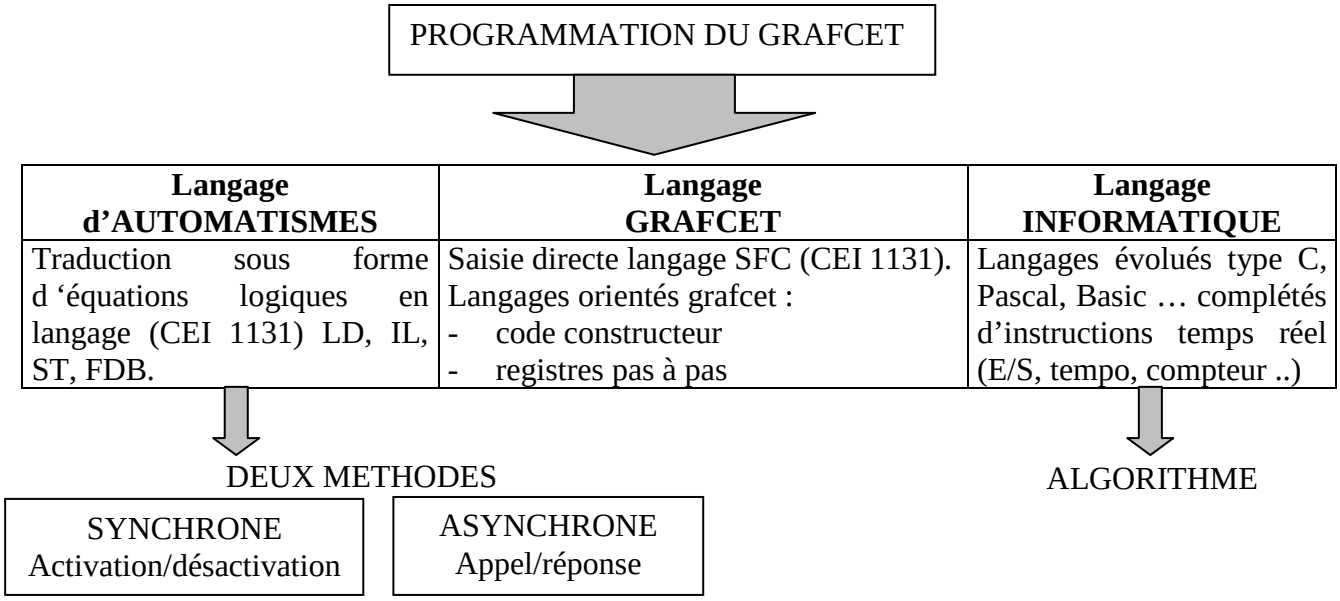
LA PROGRAMMATION DU GRAFCET

VI. INTRODUCTION

La programmation ou l'implantation ou **l'implémentation** du GRACET dans un automate programmable comporte , d'un point de vue du programmeur, plusieurs volets :

VI.1. Volet de programmation.

La saisie du GRAFCET, soit au moyen d'une console dédiée du constructeur, soit par des moyens informatiques (Atelier de programmation) est fonction du type de langage disponible au niveau de l'automate.



VI.2. volet machine (API)

Un automate programmable est une machine synchrone par conception. Ce niveau de synchronisme n'a pas de réelle influence sur la programmation. Le principe de scrutation séquentielle et cyclique de l'API est important et doit être pris en compte au moment de l'écriture programme (voir chapitre B).
Un niveau de synchronisme est mis en place à la programmation en fonction du type de GRAFCET à implémenter en mémoire. De ce point de vue, on distingue trois classes de GRAFCET pour la programmation.
Selon le point de vue réalisation (point de vue PC) le programmeur classe le grafcet selon trois types ou familles en fonction des règles utilisées dans la description.

- *grafcet asynchrone : famille ou type 1*

Grafcet ne mettant en œuvre que les trois premières règles d'évolutions. Il peut utiliser les variables d'activités (Xi) dans les réceptivités mais sans recours à la règle 4.

- *grafcet synchrone : famille ou type 2*

Grafcet non hiérarchisé, mettant en œuvre toutes les règles d'évolutions, notamment les règles 4 et 5, sur un ou plusieurs graphes partiels. Il peut utiliser les variables d'activités (Xi) pour de la synchronisation, mais ne comporte pas d'ordres de forçage.

- *grafcet hiérarchisé : famille ou type 3*

Exploite le concept de forçage et peut comporter des grafcets partiels synchrones et/ou asynchrones.

VI.3. Volet stabilité du grafcet

Cet aspect concerne le comportement attendu du Grafcet vis à vis de ces entrées / sorties lorsque celui-ci comporte des étapes ou des situations instables.

VI.4. Volet rapidité de traitement

Il s'agit d'un besoin de plus en plus important en raison de l'accroissement de la complexité des applications, complexité qui s'exprime en taille programme et donc en temps de scrutation et de réponse.
La majorité des API fonctionne actuellement en mode asynchrone vis à vis du traitement, l'amélioration de la rapidité ne peut donc être obtenue qu'au détriment soit:

- de la simplicité du programme
- de la taille du programme.

Par trois techniques ou procédés de programmation:

- standard, exploration globale du programme
- amélioré: par exploration minimisée au moyen de sauts conditionnels
- scrutation situationnelle limité à la seule situation actuelle.

VII.LE LANGAGE SFC

Voir annexe B

VII.1. L'étape

Le drapeau d'étape est représenté par la valeur logique d'une variable binaire « ***,X » où « *** » est le nom de l'étape.
***,X = 1 si l'étape correspondante est active
***,X = 0 si l'étape correspondante est inactive.
Le temps écoulé, « ***,T » depuis l'activation de l'étape est une donnée de type time

VII.2. La transition

A chaque transition est associée une réceptivité résultat s'une expression booléenne.

VII.3. action

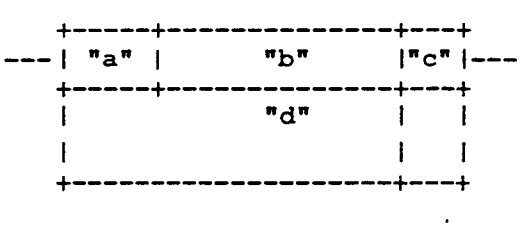
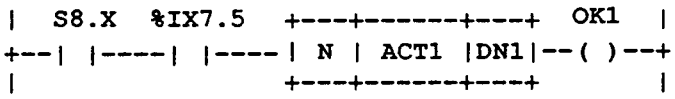
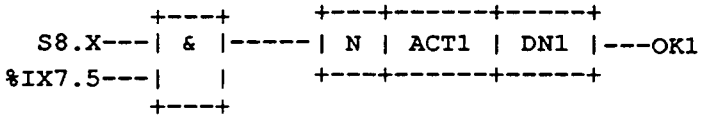
Une action, ou plusieurs actions doivent être associées a une étape. Une action peut être une variable booléenne ou un ensemble d'instructions dans un langage défini par la norme.

II.3.1 qualificatif d'action

Un qualificatif d'action doit être associé à chaque action.

qualificatif	explication	qualificatif	explication
N ou rien	Non mémorisé	D	Temporisé
R	Remise à zéro prioritaire	P	Impulsion
S	Positionné (mémorisé)		Combinaison possible
L	Limité dans le temps		

II.3.2 caractéristiques du bloc d'action

N°	Caractéristique	Forme graphique
1 2 3 4 5 6 7	"a" : Qualificatif conforme à 2.6.4.4 "b" : Nom d'action "c" : Variables booléennes d'asservissement "d" : Action utilisant: le langage IL (3.2) le langage ST (3.3) le langage LD (4.2) le langage FBD (4.3)	
	Caractéristique/Exemple	
8	Utilisation de blocs d'actions en schémas à contacts (voir 4.2):	
9	Utilisation de blocs d'actions en schémas de blocs fonctionnels (voir 4.3):	

NOTES

1 Le champ "a" peut être omis lorsque le qualificatif est "N".

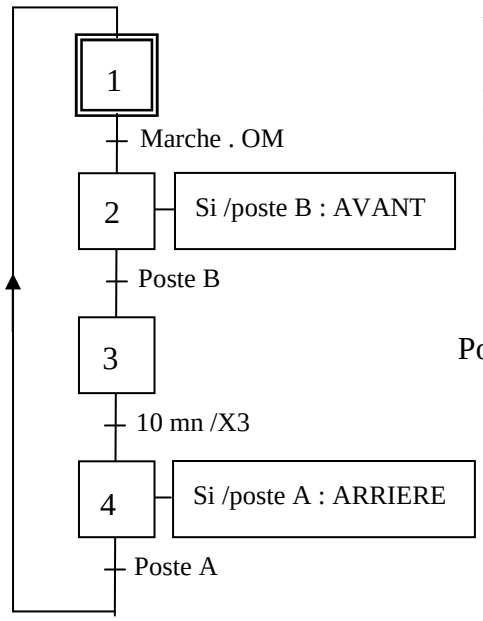
2 Le champ "c" peut être omis lorsqu'aucune variable d'asservissement n'est utilisée.

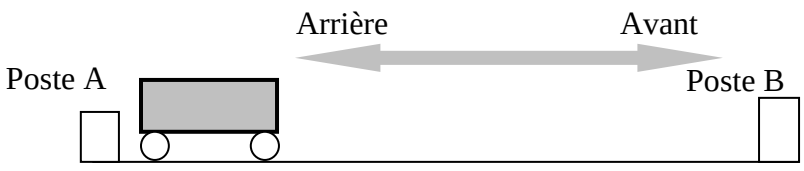
VII.4. exemple

Commande d'un chariot de transfert.

Un chariot motorisé transfère des produit du poste A au poste B sur demande de l'opérateur par un bouton poussoir "marche". Le retour du poste B vers A se fait après une temporisation d'attente en B de 10mn (temps de déchargement).

Grafcet de fonctionnement.



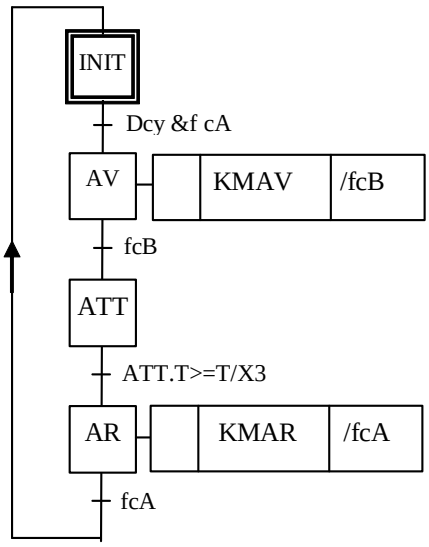


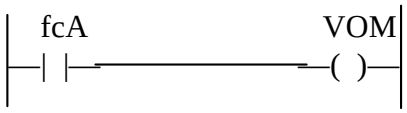
Déclaration littérale pour le programme CHARIOT

```
PROGRAMME_CHARIOT
    VAR_INPUT
            Dcy : BOOL ;            (*BP de marche*)
            FcA : BOOL ;            (*fin de course poste A*)
            FcB : BOOL ;            (*fin de course poste B*)
            T/X3 :TIME :=t#10mn     (*variable d'attente en B, initialisée à 10mn*)
    END_VAR
    VAR_OUPUT
            KMAV :BOOL ;            (*contacteur marche avant*)
            KMAR :BOOL ;            (*contacteur marche arrière*)
            VOM :BOOL ;             (*voyant origine machine*)
    END_VAR
```

Corps du programme CHARIOT en langage SFC sous forme graphique.

(* commande du chariot*) (*commande voyant origine*)





corps du programme chariot en langage SFC sous forme littérale.

(*commande du chariot*) (*commande du voyant OM*)

INITIAL_STEP INIT : **END STEP** VOM := fcA
TRANSITION FROM INIT **TO** AV
:= dcy & fcA ;
END_TRANSITION
STEP AV **:**
KMAV(fcB) ;
END_STEP
TRANSITION FROM AV **TO** ATT
:= fcB ;
END_TRANSITION
STEP ATT **: END_STEP**
TRANSITION FROM ATD **TO** AR
:= ATT.T>= t/X12 ;
END_TRANSITION
STEP AR **:**
KMAR(fcA) ;
END_STEP
TRANSITION FROM AR **TO** INIT
:= fcA ;
END_TRANSITION

VIII.PRINCIPES GENERAUX

L'acquisition est effectuée de manière systématique en début de traitement sur la quasi totalité des API, mais doit être programmée sur les autres constituants. Certains API permettent des acquisitions en cours de traitement et d'autres possèdent la possibilité d'associer des entrées à une tâche de traitement rapide sous interruption. L'affectation des sorties est également implicite sur la majorité des automates actuels.

Toutes les méthodes d'implémentation du GRAFCET devront aborder les 5 points suivants:

1 _ Initialisation :

Peut être assurée par la déclaration des étapes initiales (langage SFC) ou par programmation dans le respect de la règle 1 du grafcet. Généralement exécutée à la mise sous tension.

2 _ Calcul des conditions d'évolutions :

Doit permettre de respecter la règle 2 et éventuellement la règle 4. Associé au franchissement d'une transition, ce calcul peut concerner l'ensemble des transitions ou seulement celles validées.

3 _ Calcul des conditions d'activation des étapes :

Correspond à la règle 3 du grafcet et si nécessaire à la règle 5

4 _ Combinatoire général :

Instruction de blocs fonctionnels standards (tempo, compteur, …)

5 _ Equations des sorties :

Calcul des équations logiques des actions associées aux étapes correspondant aux sorties de la PC

Le développement des langages grafcet graphique ne supprime pas la nécessité de bien maîtriser les techniques de programmation du grafcet à partir de langages booléens ou de langages évolués, car:

- de nombreux API ne possèdent pas encore cette forme de langage.
- le constituant utilisé peut être universel (micro ou mini ordinateur) ou au contraire spécifique pour lequel le développement d'un langage grafcet n'est pas justifié.
- le besoin, lié au niveau du grafcet et/ou à la rapidité attendue, dépasse les caractéristiques de la solution constructeur(Hiérarchie).
- la compréhension fine de l'interprétation du grafcet et notamment des aléas pouvant survenir en raison de cette interprétation.

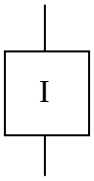
Le choix d'une méthode de programmation doit être adaptée au niveau de complexité de l'application actuelle, mais aussi future, en cas de modifications du programme.

Le choix de la méthode de programmation doit prendre en compte le critère de rapidité, qui peut orienter vers une méthode plus générale que celle à priori suffisante pour le type de grafcet à programmer.

VIII.1. Matérialisation des composantes du Grafcet

VIII.1.1. concept d'étape

Une étape peut être active ou inactive. L'opérateur logique, image de l'étape, doit donc avoir deux états stables afin de mémoriser l'activation et la désactivation de l'étape. Le profil de l'opérateur ainsi défini correspond à la fonction **MEMOIRE**. La variable Xi représentant l'étape sera travaillée selon le principe suivant.



III.1.2 concept de transition et réceptivité

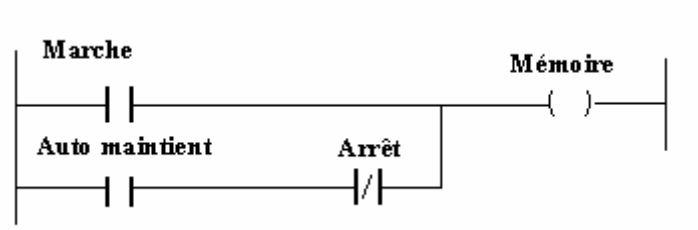
Ce concept est l'élément lié aux possibilités d'évolution du GRAFCET. Les règles 2 et 3 précisent que l'évolution ne peut avoir lieu que si : la transition est validé ET la réceptivité est vraie

III.1.3 généralisation des concepts

La généralisation de ces deux concepts emmène à la mise en œuvre, pour chaque étape de grafcet, de la fonction mémoire générale de l'automatisme.

La mémoire utilisée est monostable et à marche prioritaire, ce qui est la traduction exacte du concept.

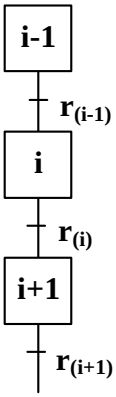
Equation générique :



IX. METHODE SYNCHRONE

La méthode synchrone ou méthode ACTIVATION / DESACTIVATION est la traduction directe des règles d'évolutions du grafcet. La méthode activation/désactivation s'applique à tous les grafcets (3 types). Elle permet de traiter la simultanéitée de franchissement, au sens du grafcet[5], par le calcul préalable de toutes les conditions d'évolutions avant le calcul des activations d'étapes.
Le principe consiste à l'association d'une mémoire à chaque étape et au calcul des conditions d'activation et de désactivation en appliquant les règles 2 et 3 du modèle grafcet.

ACTIVATION : équation logique des conditions d'évolutions d'entrée de l'étape Xi ; mise à 1 de la mémoire d'étape.
DESACTIVATION : équation logique des conditions d'évolutions de sortie de l'étape Xi ; mise à 0 de la mémoire d'étape.

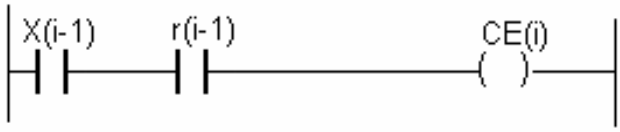


La méthode consiste à l'évaluation préalable de TOUTES les conditions d'évolutions AVANT d'effectuer une seule évolution, de sorte que la situation reste invariante pendant la phase de calcul des activations d'étapes.
Cette méthode de programmation comprend deux blocs distincts, obligatoirement programmés en suivant :

1 °) le calcul de toutes les conditions d'évolution du grafcet
2 °) le calcul des activations d'étapes du même grafcet

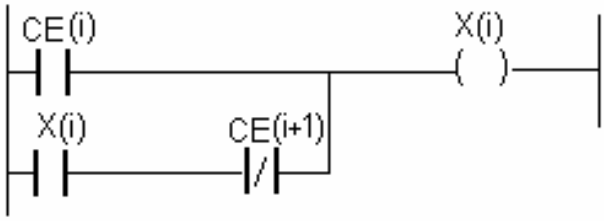
IX.1. Equations d'étape

Calcul des CE(i)



[5] Ne pas confondre une évolution simultanée au sens du grafcet et au niveau du traitement. En effet, avec la majorité des API, il n'est pas possible d'assurer une évolution simultanée de l'ensemble des transitions franchissables, le traitement étant par essence séquentiel.

Calcul des X(i)

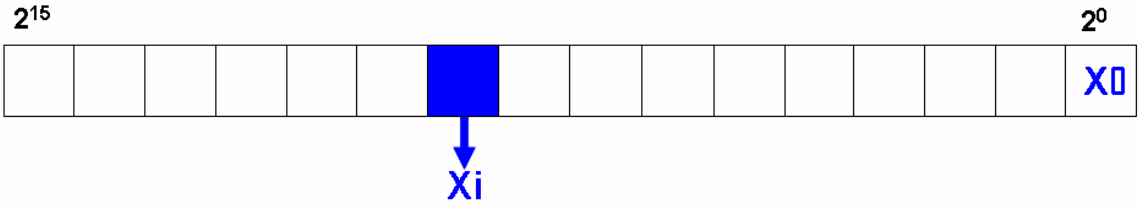


IX.2. Programmation des équations

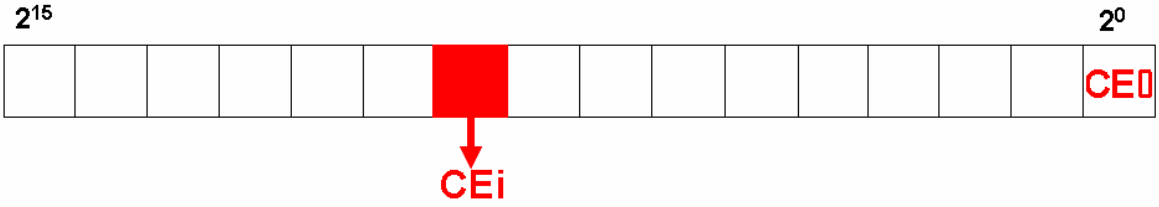
Toutes les étapes du grafcet à programmer seront traduites à partir des équations définies ci-dessus. Ces équations n'utilisent que des données de type booléen, il faut donc utiliser des bits internes pour matérialiser chaque variable d 'étape Xi.

1 étape = 1 bit

Dans la pratique on utilisera les données de type WORD (cordon de 16 bits) pour implémenter un grafcet, ceci nous autorise 16 étapes, mais ne représente pas un nombre limite d'étape pour un grafcet.



Nous appliquerons le même principe à calcul des conditions d'évolutions en utilisant un deuxième mot.



Adressage

Adressage type : **variable d'étape Xi = donnée de type bit ; W mm, xx**

avec : mm : adresse du mot utilisé

xx : bit du mot utilisé pour l'étape Xi

Adressage type : **condition d'évolution CEi = donnée de type bit ; W nn, xx**

avec : nn : adresse du mot utilisé

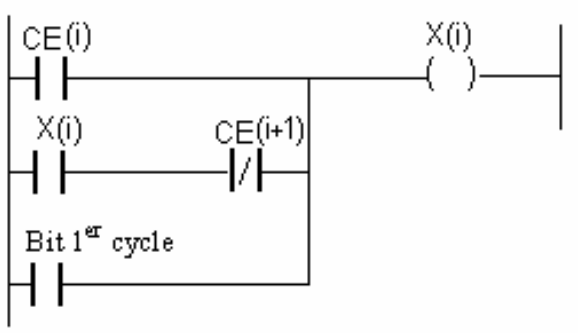
xx : bit du mot utilisé pour la condition CEi

Les adresses des deux mots seront choisies avec un certain rapport de façon à faciliter la lecture.

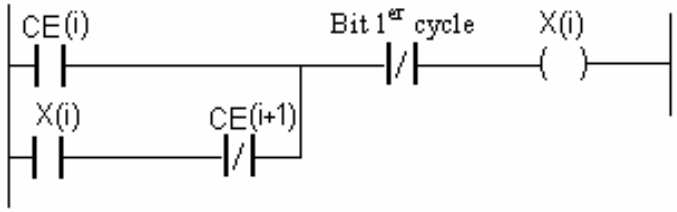
IX.3. initialisation

La règle 1 stipule que tout grafcet doit être initialisé pour pouvoir évoluer de façon correcte. Elle défini la situation initiale comme la ou les étapes actives au début du fonctionnement de la partie commande. La matérialisation du grafcet dans l'automate programmable passe par des mots internes (mot registre grafcet) utilisés de manière particulière.
Les API de dernière génération possèdent des bits systèmes utilisables par le programmeur. Ces bits définis par le constructeur sont gérés par le système, dans cette panoplie existe un bit **«premier tour de cycle »** qui sera utilisé pour l'initialisation du grafcet à la mise sous tension.

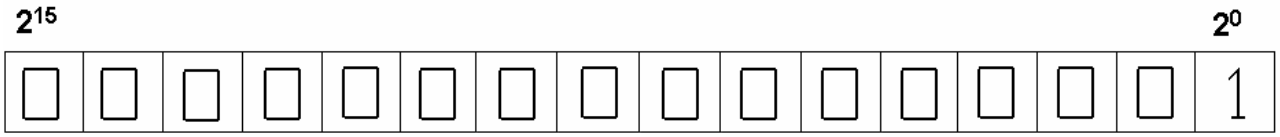
IV.3.1 modification des équations ***Etape initiale***



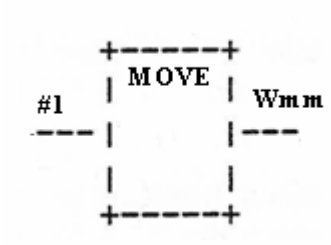
Etapes non initiales



IV.3.2 travail sur mot (X_0 étape initiale)

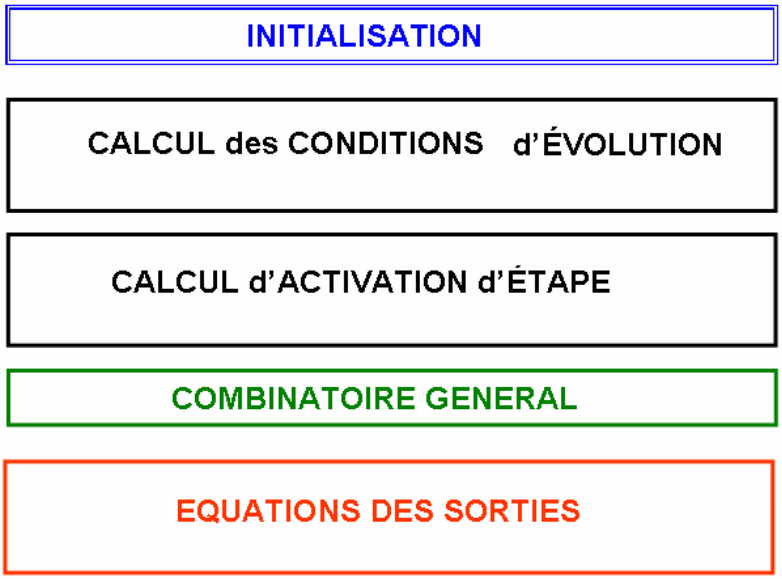


Xi



IX.4. Structure du programme de la méthode

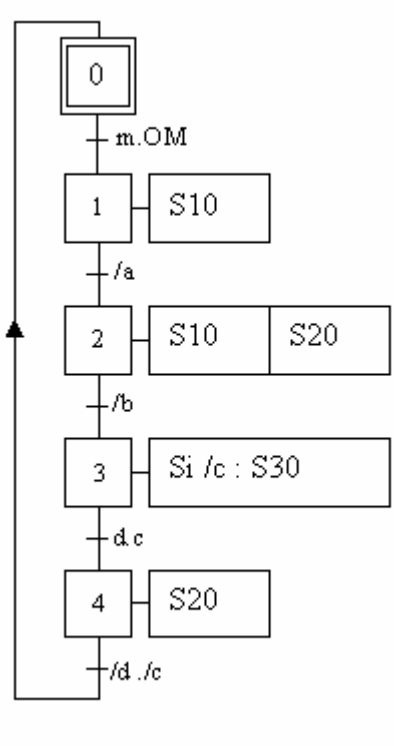
Cette méthode nécessite cinq phases. La structure du programme aura la forme ci_dessous



Exemple de programmation : grafcet GRAF_1

OM = a ./d

INITIALISATION
Wmm := 1



CALCUL des CONDITIONS d'EVOLUTION
CE1 = X0 . m . OM
CE2 = X1 . /a
CE3 = X2 . /b
CE4 = X3 . d . c
CE0 = X4 . /d . /c

CALCUL D'ACTIVATION D'ETAPE
X0 = CE0 + (X0 . /CE1)
X1 = CE1 + (X1 . /CE2)
X2 = CE2 + (X2 . /CE3)
X3 = CE3 + (X3 . /CE4)
X4 = CE4 + (X4 . /CE0)

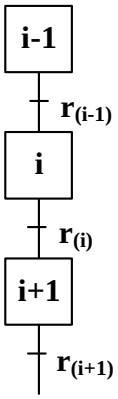
COMBINATOIRE GENERAL
OM = a . /d

EQUATIONS DES SORTIES
S10 = X1 + X2
S20 = X2 + X4
S30 = X3 . /c

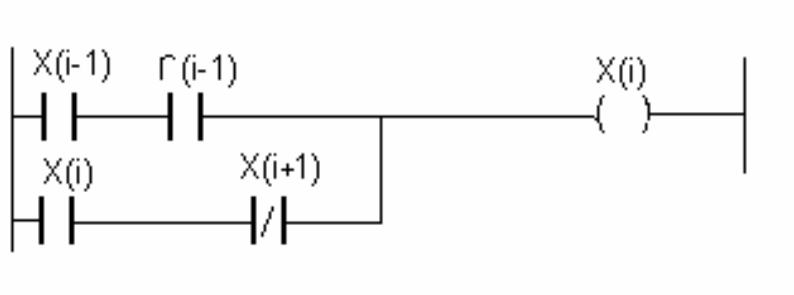
X.METHODE ASYNCHRONE

La méthode de programmation asynchrone ou APPEL/REPONSE du grafcet est une méthode simple à mettre en œuvre et qui ne peut s'utiliser que pour des grafcets de type 1 (famille 1). La condition d'enclenchement de la mémoire d'étape correspond à l'APPEL et le déclenchement à la REPONSE.

APPEL : mise à 1 de la mémoire d'étape, correspond à la condition d'évolution de la transitions d'entrée de l'étape.
REPONSE : mise à 0 de la mémoire étape, correspond à l'activation de l'étape suivante.



Equation :



X.1. programmation de l'équation

Toutes les étapes du grafcet à programmer seront traduites à partir de l'équation définie ci-dessus. Cette équation n'utilise que des données de type booléen, il faut donc utiliser des bits internes pour matérialiser chaque variable d 'étape Xi.

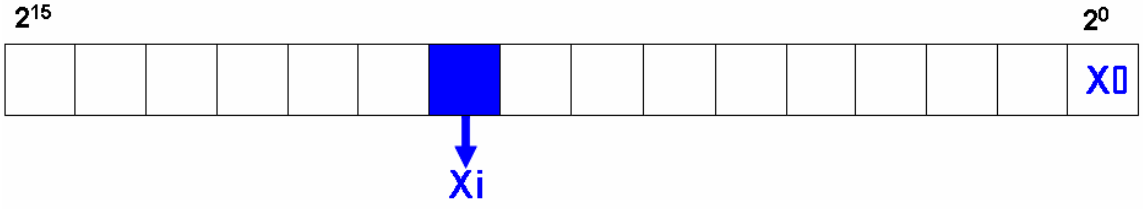
1 étape = 1 bit

Comme dans la méthode précédente on utilisera les données de type WORD (cordon de 16 bits) pour implémenter un grafcet, ceci nous autorise 16 étapes, mais ne représente pas un nombre limite d'étape pour un grafcet.

Adressage type : **variable d'étape Xi = donnée de type bit ; W mm, xx**

avec : mm : adresse du mot utilisé

xx : bit du mot utilisé pour l'étape Xi



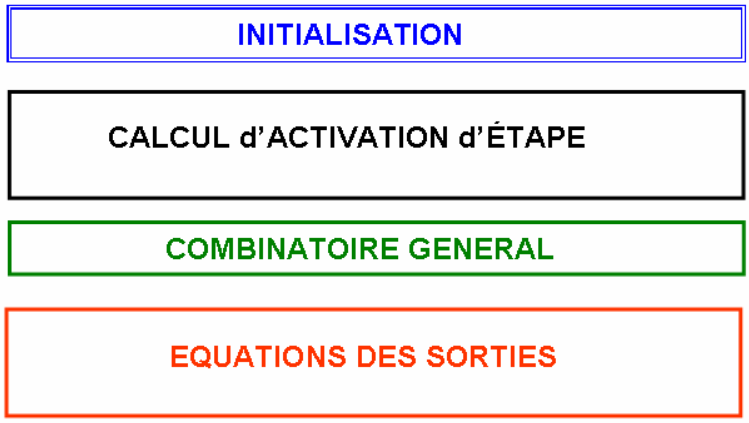
X.2. initialisation

L'initialisation se fait de la même façon que dans la méthode précédente
V.2.1 modification de l'équation

V.2.2 travail sur mot

X.3. Structure du programme

Cette méthode est l'équivalent programmé des séquenceurs cablés. Elle utilise un minimum de données mémoire et ce structure de la façon suivante :



Exemple de programmation : grafcet GRAF_1

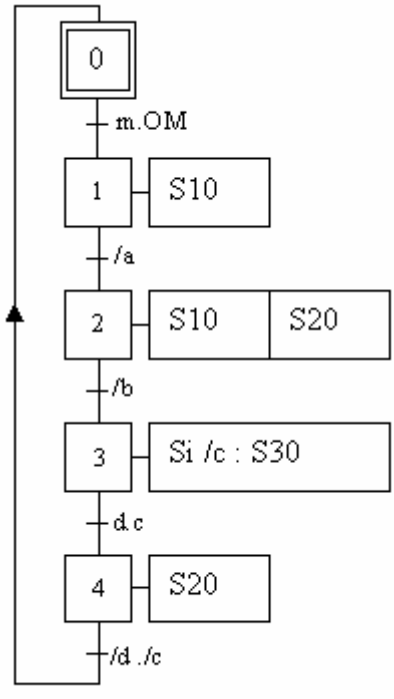
OM = a ./d

INITIALISATION
Wmm := 1

CALCUL D'ACTIVATION D'ETAPE
X0 = (X4 . /d . /c) + (X0 . /X1)
X1 = (X0 . m . OM) + (X1 . /X2)
X2 = (X1 ./a) + (X2 . /X3)
X3 = (X2 ./b) + (X3 . /X4)
X4 = (X3 ./d ./c) + (X4 . /X0)

COMBINATOIRE GENERAL
OM = a . /d

EQUATIONS DES SORTIES
S10 = X1 + X2
S20 = X2 + X4
S30 = X3 . /c



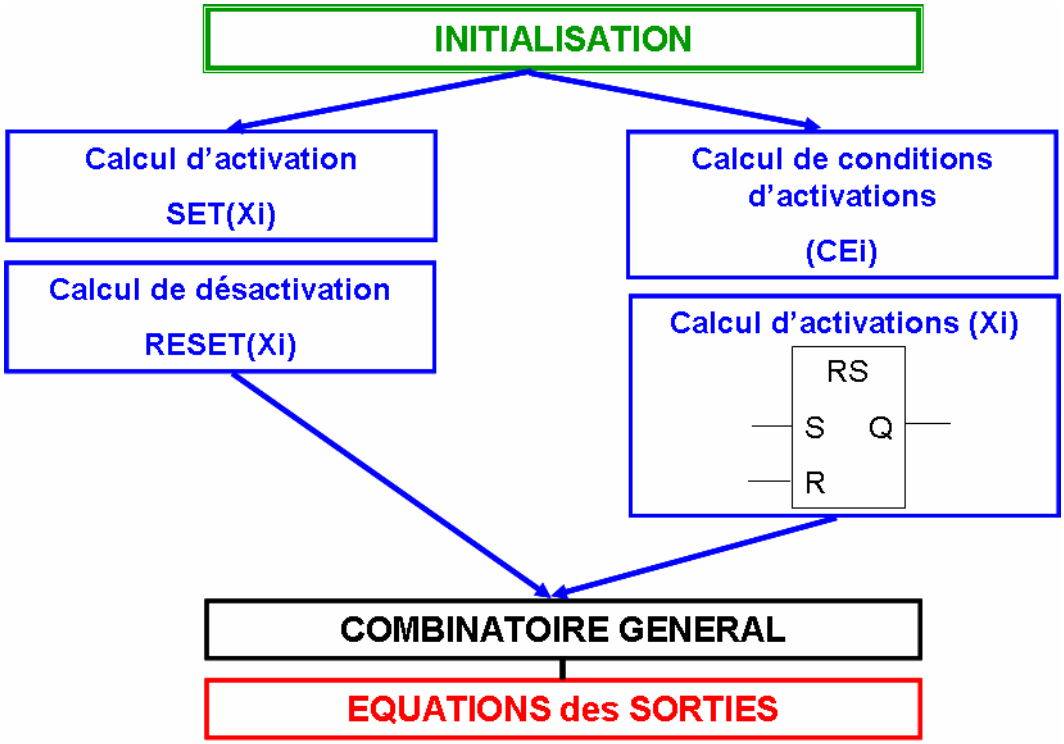
X.4. Comparaison des deux méthodes

	synchrone	asynchrone
Mémoire de données	2 mots	1 mot
Mémoire programme	15 lignes	10 lignes

La méthode appel/réponse est une méthode simple à implémenter, peu consommatrice de données automate mais qui trouve rapidement ses limites.

XI. PROGRAMMATION BISTABLE

La majorité des automates programmables actuels possèdent des adresses de bits bistables dans la mémoire de données. L'accés à ces bits s'accompagne généralement par les instructions "SET" et "RESET", ou par l'utilisation de blocs fonctionnels (norme 1131-3) du type "RS".



L'utilisation en programmation des bits bistables permet, dans la majorité des automates et pour une grande partie des grafcets, de ne pas avoir à calculer les conditions d'évolutions si l'on prend la précaution de regrouper les activations et les désactivations dans le programme. La structure du programme peut alors avoir deux organisations suivant la technique utilisée.

XII. IMPLEMENTATION DU GRAFCET GLOBAL

XII.1. Grafcet structuré

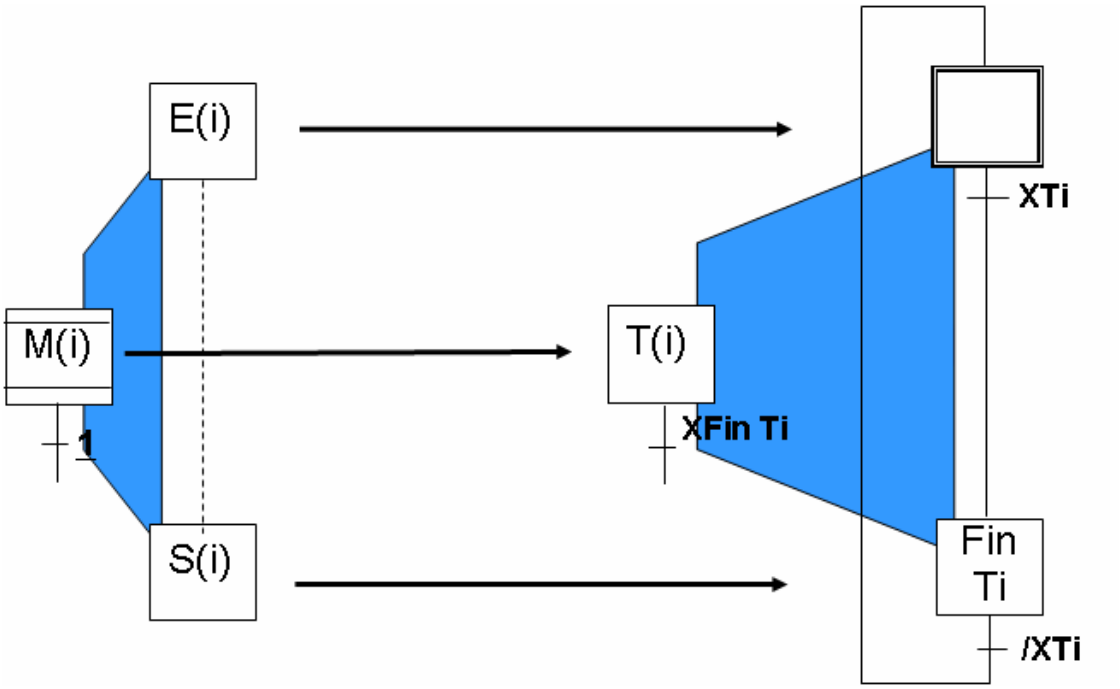
Le concept de macro représentation, utilisé dans le cas de la description du Grafcet de Production Normale (GPN), ne présente pas de problèmes particuliers. La programmation est fonction de l'outil de développement utilisé.

1° cas :

Le jeu d'instruction de l'automate inclut la programmation des macros-étapes, il suffit dans ce cas de respecter la syntaxe donnée par le constructeur.

2° cas :

Le passage au point de vue PC du grafcet oblige à interpréter les macros-étapes par des grafcets connexes bouclés.



La programmation suivant la méthode Activation-Désactivation est applicable dans son intégralité

Rappel :

La programmation synchrone utilise la notion de calcul des conditions d'évolution et de calcul d'activation d'étape. Ceux-ci se programmes dans deux modules différents utilisant des mots différents:

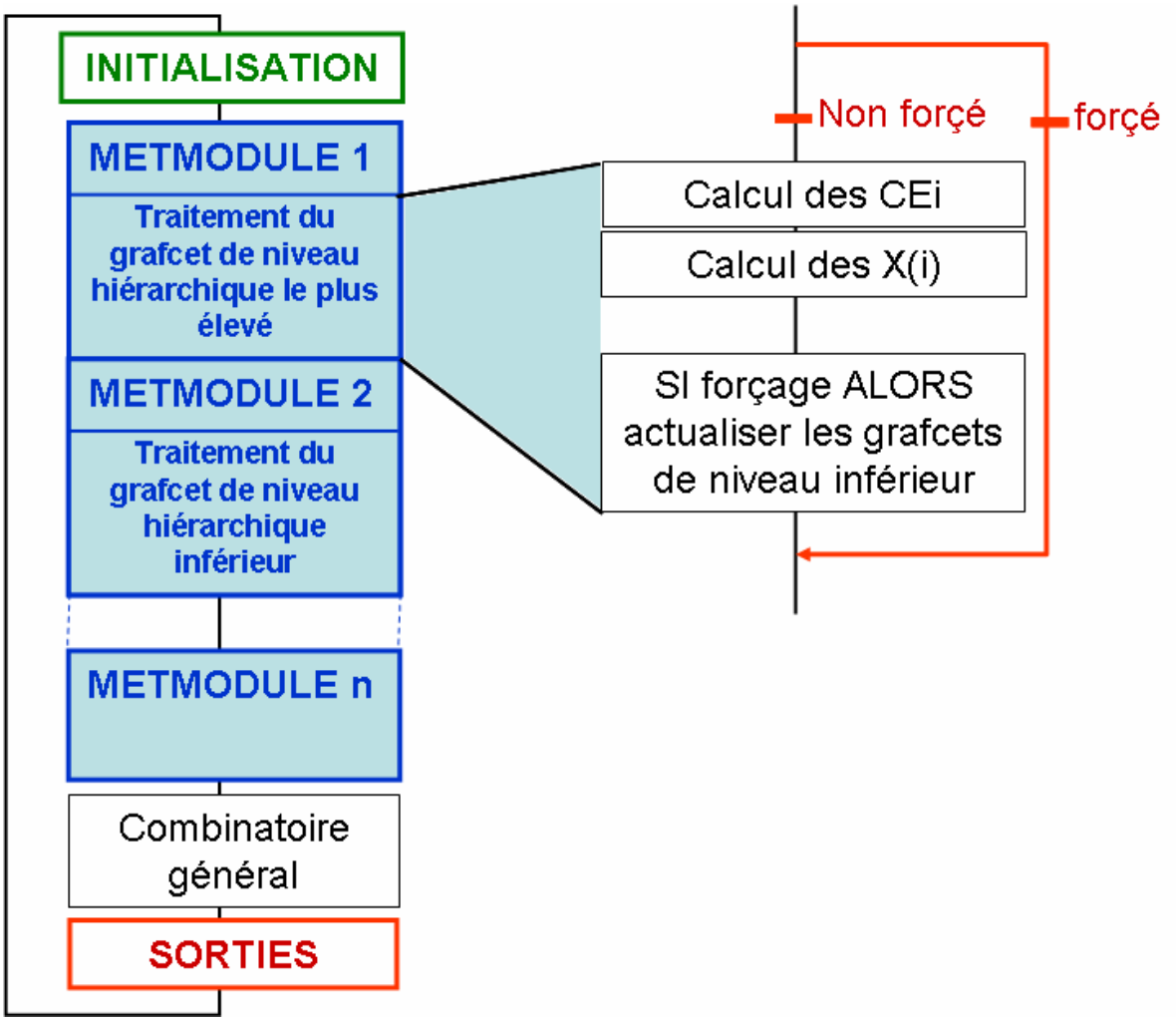
- module 1 : calcul des conditions d'évolutions utilisant un mot
- module 2: calcul des activations d'étapes utilisant un mot différent souvent appelé: mot REGISTRE grafcet.

XII.2. Grafcet hiérarchisé

La programmation d'une structure hiérarchisée nécessite l'application d'une métarègle de stucturation du programme. *Le traitement est structuré en "métamodules", chaque métamodule est associé à un seul niveau de grafcet.*

Règle 1 : Traiter les métamodules dans l'ordre de la hiérarchie décroissante
Règle 2 : Pour chaque métamodule, SI et seulement SI ce niveau n'est pas forcé :
- **effectuer la traitement correspondant au grafcet**
- **exécuter les actions de forçage de niveau inférieur**

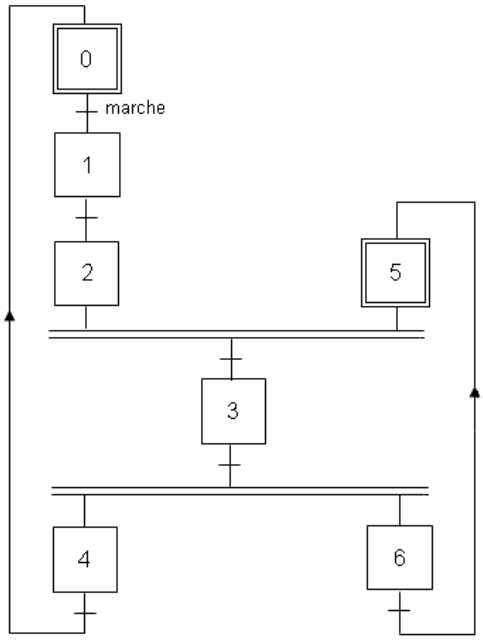
Règle 3 : Sauter le traitement des grafcets partiels forcés
Règle 4 : Effectuer les actions opératives de tous les métamodules en fin de programme



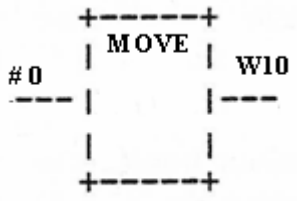
XII.3. Programmation des forçages

Le forçage d'un grafcet , programmé en méthode synchrone en utilisant des mots registres, se fait par modification directe de la valeur du mot en utilisant le bloc fonctionnel de transfert de valeur MOVE.

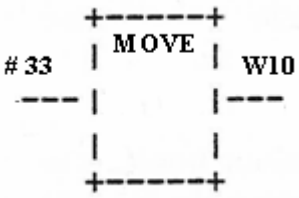
Mot Registre grafcet W10 # 8
Etape X0 bit 0
Etape X1 bit 1
Etape X2 bit 2
Etape X3 bit 3
Etape X4 bit 4
Etape X5 bit 5
Etape X6 bit 6



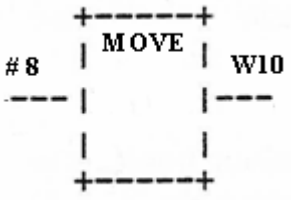
- **Situation vide :** Le forçage en situation vide [F/G# :{}] correspond au forçage à zéro du mot registre utilisé pour le grafcet forcé. Exemple



- **Situation initiale :** Le forçage en situation initiale [F/G# : (init)] correspond à la mise à 1 dans le mot registre utilisé par le Grafcet des bits correspondant aux étapes initiales Exemple:



- **Situation donnée** : **:** Le forçage dans une situation donnée [F/G# : (Xj)] correspond à la mise à 1 dans le mot registre utilisé par le Grafcet du bit correspondant a l' étape Xj
  Exemple: [F/G# : (X3)]



- **Situation courante** : Le figeage du grafcet [F/G# (*)] peut s'obtenir de plusieurs façons:

1) Par blocage de son évolution en travaillant au niveau des équations des conditions d'évolutions.

2**)** Par forçage direct et permanent (durée du figeage) du mot registre utilisé par le grafcet figé en deux étapes :

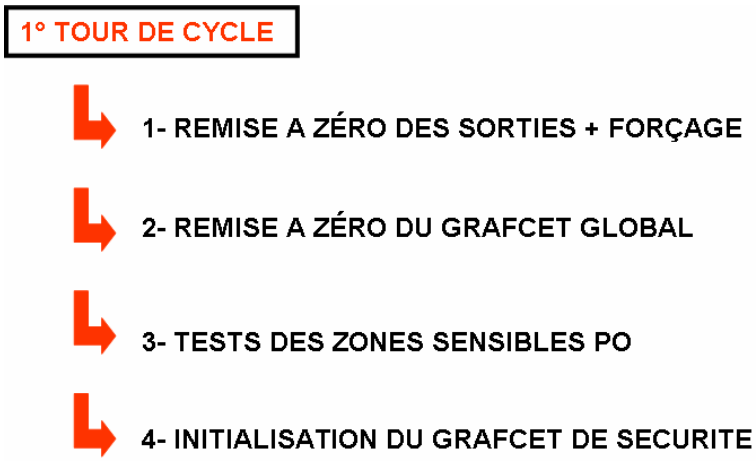
- enregistrement permanent du registre du grafcet forcé dans un registre tampon
- écriture tout le temps du forçage du registre forcé par la valeur du tampon.

3) Par la programmation du saut de traitement du grafcet forcé tout le temps du forçage.

XII.4. Module logiciel de démarrage

A la mise sous tension, l'automate programmable fait un certain nombre de tests internes avant de commencer son cycle de scrutation. Pour une initialisation contrôlée de l'application, il est nécessaire d'effectuer des commandes particulières en début de programme. Nous appellerons cette phase: **« le module d'initialisation ».**

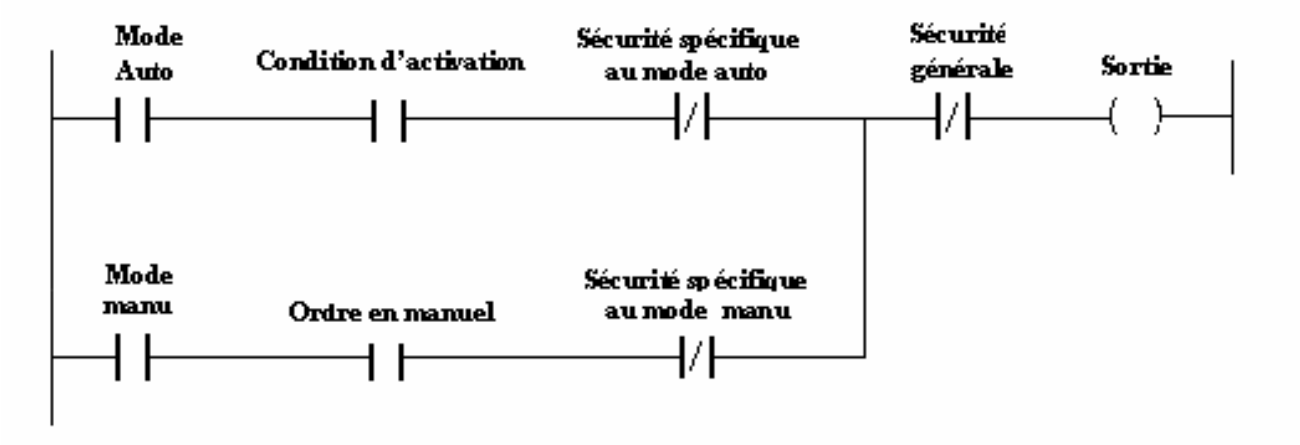
1° tour de cycle : Le premier tour de cycle est signalé par un bit système.



Le déverrouillage des sorties se fait par le Grafcet de sécurité au moment de l'alimentation des sorties.

XII.5. Montage des équations de sorties

Pour le respect de la règle de programmation des sorties (une seule équation par sortie), la forme finale de l'équation d'une sortie aura la structure suivante.

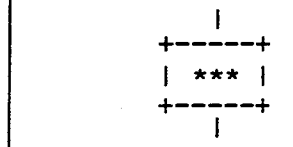
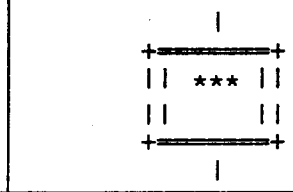


ANNEXE B Norme 61131 LANGAGE GRAPHIQUE SFC

L'extrait de norme suivant représente les éléments du diagramme fonctionnel en séquence (SFC) destinés à être utilisés pour la description d'une unité d'organisation de programme en langage Grafcet.

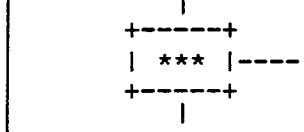
(a) ETAPES

Le tableau ci-dessous illustre la représentation d'une étape.

N°	Représentation	Description
1		Etape - Forme graphique avec liaisons dirigées "***" = Nom d'étape
		Etape initiale - Forme graphique avec liaisons dirigées "***" = Nom d'étape initiale (note 2)

2	`STEP *** :`
	`(* Corps d'étape *)`
	`END_STEP`
		Etape - Forme littérale sans liaisons dirigées (voir 2.6.3)
		"***" = Nom d'étape

	`INITIAL_STEP *** :`
	`(* Corps d'étape *)`
	`END_STEP`
		Etape initiale - Forme littérale sans liaisons dirigées (voir 2.6.3)
		"***" = Nom d'étape

3a	`***.X`	Drapeau d'étape - Forme générale "***" = Nom d'étape ***.X = 1 booléen quand *** est actif = sinon 0 booléen
3b		Drapeau d'étape - Connexion directe de variable booléenne ***.X à droite de l'étape "***"
4	`***.T`	Temps écoulé pour une étape - Forme générale "***" = Nom d'étape ***.T = une variable de type TIME (voir définition 2.6.2)

NOTES

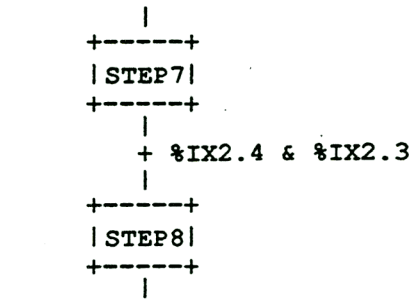
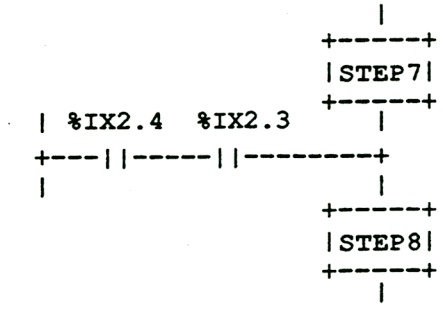
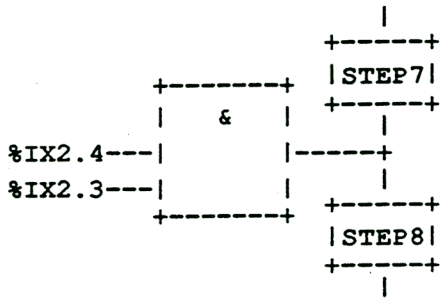
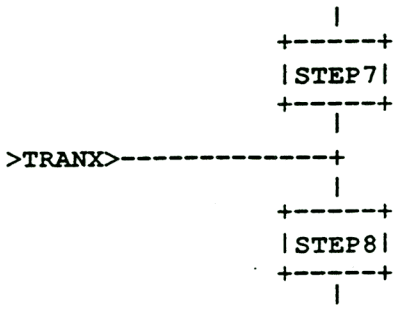
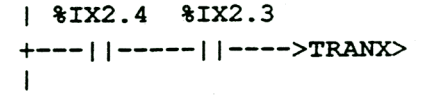
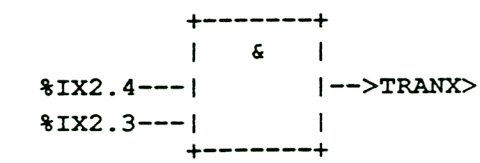
1 Lorsque la caractéristique 3a, 3b, ou 4 est acceptée, elle doit constituer une erreur si le programme utilisateur tente de modifier la variable associée. Par exemple: si S4 est un nom d'étape, alors les énoncés suivants devraient être des erreurs dans le langage ST défini en 3.3:

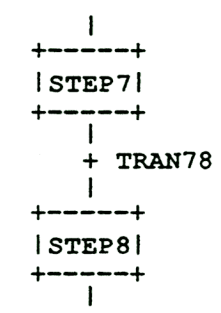
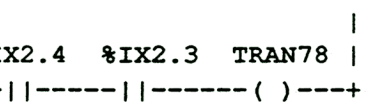
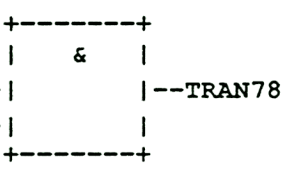
S4.X := 1 ; (* ERROR *)

S4.T := t#100ms ; (* ERROR *)

2 La liaison dirigée supérieure n'est pas requise si l'étape initiale n'a pas de prédécesseur.

TRANSITION

N°	Exemple	Description
1		Etape précédente Condition de transition utilisant le langage ST (voir 3.3) Etape suivante
2		Etape précédente Condition de transition utilisant le langage LD (voir 4.2) Etape suivante
3		Etape précédente Condition de transition utilisant le langage FBD (voir 4.2) Etape suivante
4		Utilisation de connecteur: Etape précédente Connecteur de transition Etape suivante
4a		Condition de transition utilisant le langage LD (voir 4.2)
4b		Utilisant le langage FBD (voir 4.3)

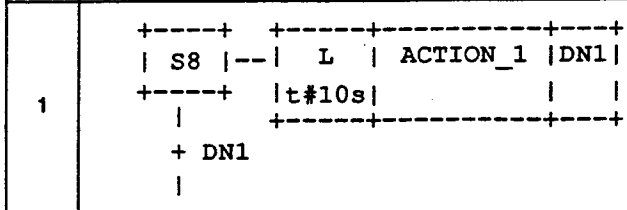
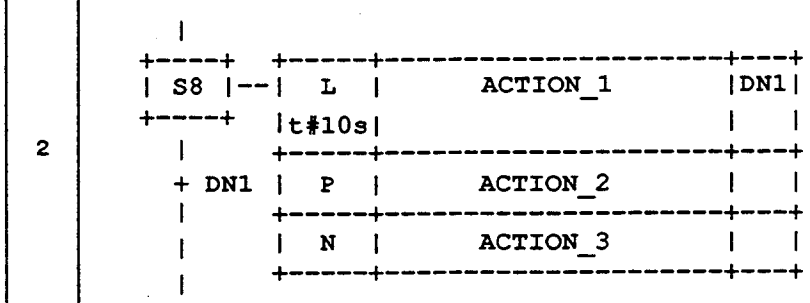
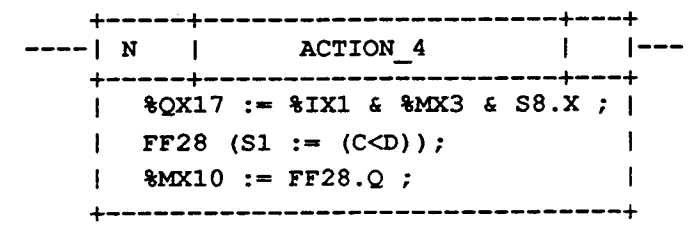
N°	Exemple	Description
5	STEP STEP7 : END_STEP TRANSITION FROM STEP7 TO STEP 8 := %IX2.4 & %IX2.3 ; END_TRANSITION STEP STEP8 : END_STEP	Equivalent littéral de la caractéristique n° 1 utilisant le langage ST (voir 4.3)
6	STEP STEP7 : END_STEP TRANSITION FROM STEP7 TO STEP 8: LD %IX2.4 AND %IX2.3 END_TRANSITION STEP STEP8 : END_STEP	Equivalent littéral de la caractéristique n° 1 utilisant le langage IL (voir 3.2)
7		Utilisation de nom de transition: Etape précédente Nom de transition Etape suivante
7a	TRANSITION TRAN78 :  END_TRANSITION	Condition de transition utilisant le langage LD (voir 4.2)
7b	TRANSITION TRAN78 :  END_TRANSITION	Condition de transition utilisant le langage FBD (voir 4.3)
7c	TRANSITION TRAN78 : LD %IX2.4 AND %IX2.3 END_TRANSITION	Condition de transition utilisant le langage IL (voir 3.2)
7d	TRANSITION TRAN78 : := %IX2.4 & %IX2.3 ; END_TRANSITION	Condition de transition utilisant le langage ST (voir 3.3)

NOTES

1 Si la caractéristique 1 du tableau 40 est acceptée, alors une ou plusieurs des caractéristiques 1, 2, 3, 4 ou 7 du présent tableau doivent être acceptées.

2 Si la caractéristique 2 du tableau 40 est acceptée, alors la caractéristique 5 ou 6 du présent tableau, ou les deux, doivent être acceptées.

ACTIONS

N°	Exemple	Caractéristique
1		Bloc d'action (Voir 2.6.4.3)
2		Bloc d'action enchaînés
3	STEP S8 : ACTION_1(L,t#10s,DN1) ; ACTION_2(P) ; ACTION_3(N) ; END_STEP	Corps d'étape littéral
4		Bloc d'action champ "d" (Voir 2.6.4.3)

NOTE - Lorsque la caractéristique 4 est utilisée, le nom d'action correspondant ne peut être utilisé dans aucun autre bloc d'action.