AUTOMATE PROGRAMMABLE INDUSTRIEL

- A P I -

I- Introduction

L'objectif du développement de la recherche scientifique dans tous les domaines est d'aboutir à un progrès technologique voué principalement au bien être de l'humanité. Par sa nature de chercher à avoir la quantité, la qualité, le confort avec le minimum d'effort et du coût, l'homme a commencé à réfléchir, à concevoir et à réaliser des systèmes autonomes qui peuvent lui assurer tout dont il a besoin. Par conséquent, les outils et les appareils à énergies musculaires actionnés par des opérateurs sont remplacés par leurs équivalents à énergies électriques, mécaniques ou hydrauliques. La substitution d'opérateurs par ces systèmes définit l'automatisation.

La figure ci-dessous montre un schéma synoptique simplifié d'un système automatisé.

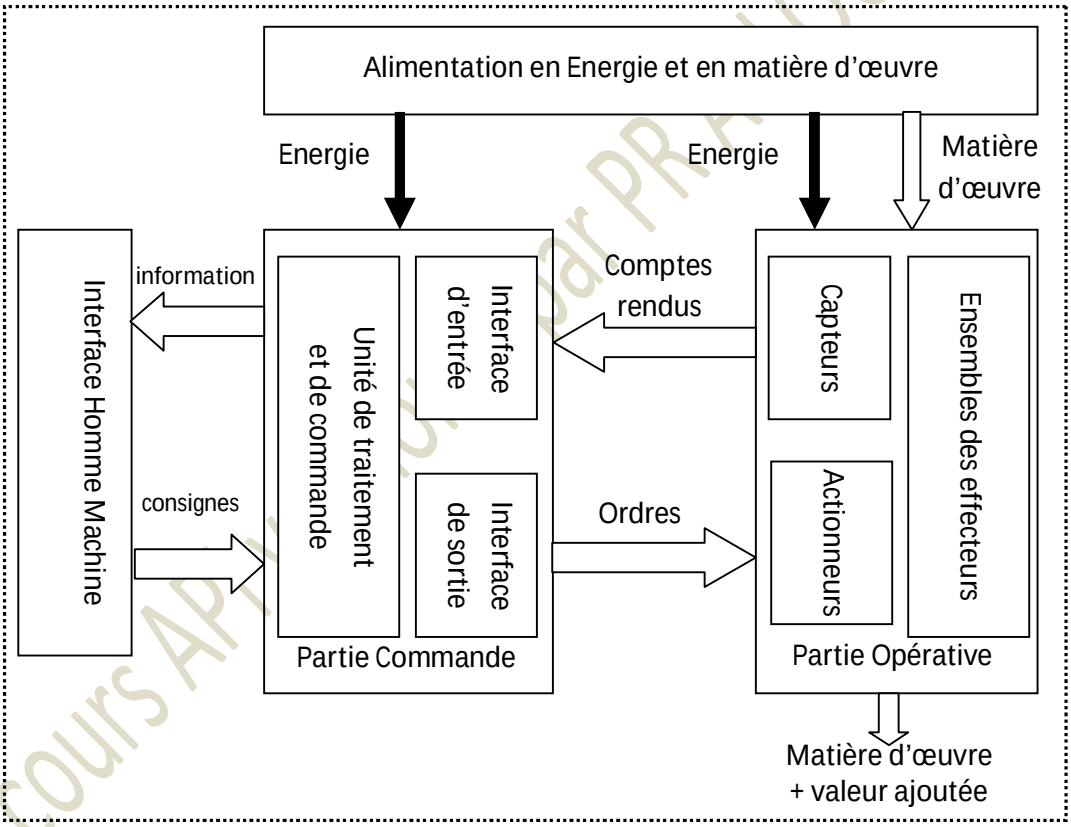


Figure-1 : Schéma de principe d'un système automatisé

L'industrie automobile est parmi les industries les plus développées au monde. C'est au niveau de cette industrie que les Automates Programmables Industriels (API) ont apparus. Le critère de conception des premiers API était spécifié pour la première fois en 1968 par General Motors. Leur objectif principal est le remplacement de la logique à relais qui était la technique dominante dans le contrôle des processus industriels et qui présente une structure rigide et encombrante par un système flexible capable d'être exploité dans des environnements industriels. Les premiers API ont fait leur apparition en 1969 et ils ont ouvert la porte vers le développement

d'une nouvelle technologie de contrôle. Par conséquent, ils remplacent la logique câblée basée sur l'emploi de relais et ils présentent une structure modulaire pour d'éventuelles extensions. Parmi les objectifs de l'industrie automobile est l'augmentation de la production par le développement des chaines de production et de leur automatisation.

Les Automates Programmables Industriels ou API (PLCs Programmable Logic Controllers) sont des appareils électroniques conçus autour de microprocesseurs. Ils sont destinés spécialement pour piloter des processus industriels. Leur fonctionnement diffère de celui des systèmes à microprocesseurs usuels. Leurs emplois permettent de rendre les machines employées autonomes et totalement indépendantes de l'intervention d'opérateurs humains. L'API peut alors être défini comme étant un dispositif électronique programmable adapté à l'automatisation des systèmes de production. L'architecture, la programmation et l'exploitation des API répondent aux exigences de son milieu d'utilisation et au personnel qui va l'exploiter.

D'une façon simple et facile à retenir, l'API sera défini comme un système à microprocesseur dédié à l'industrie; il doit répondre aux attentes de l'industrie :

- Concernant l'API
  - Moins encombrant
  - Facile et simple à mettre en œuvre
  - Protégé contre les influences du milieu de travail
- Concernant le système de production
  - Flexibilité
  - Augmenter la productivité
  - Améliorer la qualité du produit
  - Réduction des couts
  - Travail continu

II- Structure d'un API

Pour une raison de simplicité et pour mieux comprendre les API on donne sur la figure 2 le schéma synoptique d'une structure simplifiée d'un API regroupant les principaux modules qui le constituent.

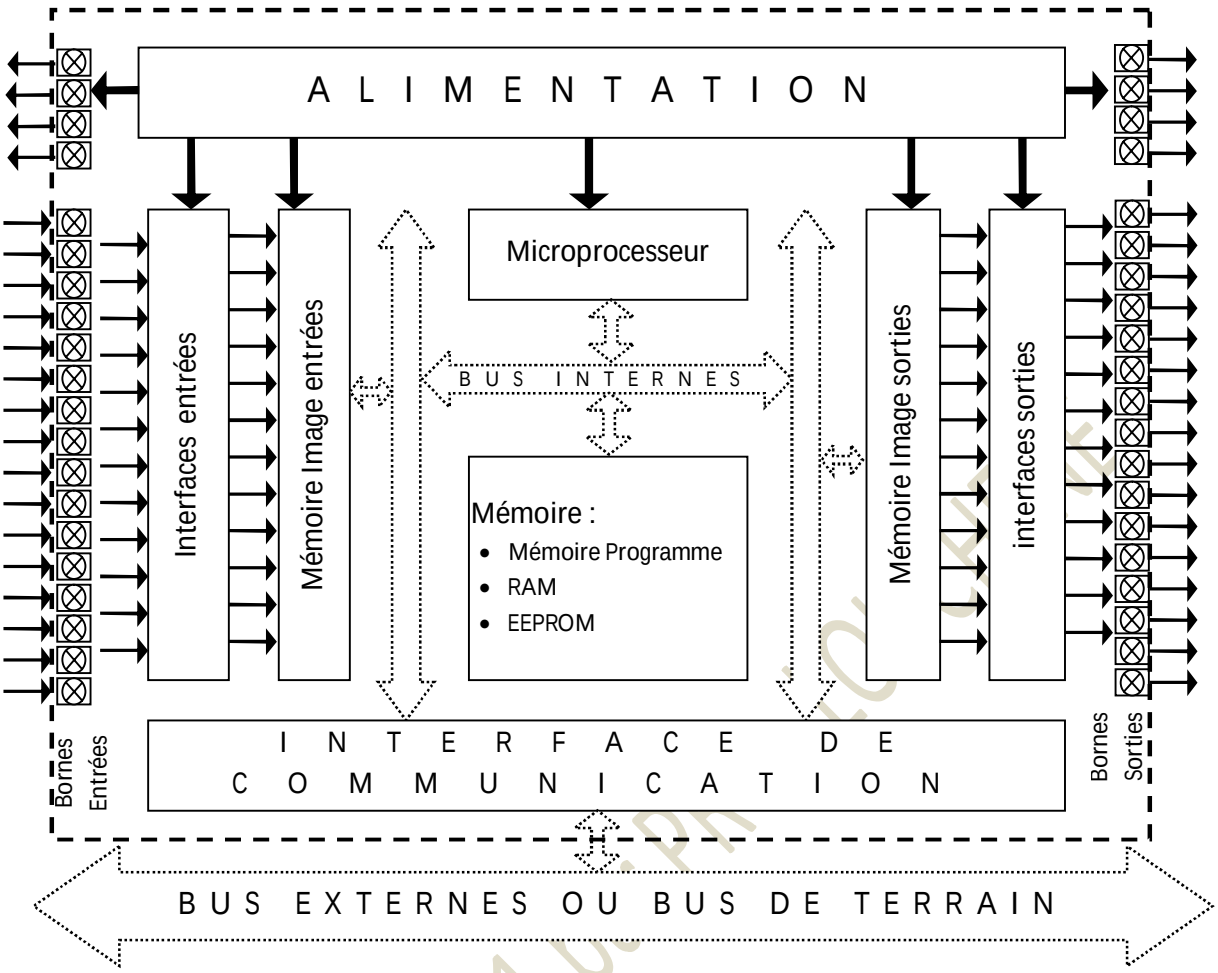


Figure 2 : Structure d'un Automate Programmable Industriel (API)

II-1 Microprocesseur

L'unité de traitement ou Unité centrale constitue le noyau central de l'automate programmable. Le microprocesseur définit l'élément principal d'une unité centrale. C'est à ce niveau que s'effectuent les traitements et les prises de décisions suivant l'interprétation et l'exécution d'un programme écrit par un utilisateur. Dans le cas général, l'exécution d'un programme se fait instruction par instruction d'une façon séquentielle ; de la première instruction jusqu'à la dernière. Si on veut qu'un programme soit exécuté d'une façon cyclique continue, la dernière instruction devra être du type saut vers le début du programme principal. Le temps réservé à l'exécution de chacune des instructions est fixé par un nombre de cycles de l'horloge du système. Les opérations effectuées par cette unité peuvent être réparties sur deux catégories:

- Logiques: AND, OR, XOR, NOT, SHIFT et ROTATE (ET, OU, OU EXCLUSIF, NON, DECALAGE et ROTATION)
- Arithmétiques l'ADDITION, la SOUSTRACTION, la MULTIPLICATION, la DIVISION.

Du moment que l'API est destiné à travailler dans le domaine industriel, il est conçu à travailler d'une façon continue. Par conséquent, l'exécution d'un programme est réalisée d'une façon cyclique. Il serait inutile de prévoir une instruction de reprise d'exécution à la fin d'un programme.

II-2 Mémoire

La partie mémoire est destinée à la sauvegarde de programmes ou de données. Dans le cas des automates programmables la mémoire des données ou RAM est généralement divisée en espaces mémoires suivant la nature de leur fonction. On trouve la mémoire spéciale qui est destinée à la sauvegarde des états du système lors de l'exécution d'un programme, elle affiche aussi les drapeaux indiquant l'état de la solution d'une opération tels que résultat NUL, résultat NEGATIF,….

Une mémoire répartie en registres joue le rôle d'image des entrées et des sorties numériques ; elle permet de verrouiller l'état des entrées ou des sorties afin de permettre au système d'y accéder aux entrées et d'envoyer les sorties avec un temps très réduit. Une troisième partie de la mémoire est réservée aux variables, on l'appelle dans la plus part des cas mémoire de l'utilisateur ou mémoire des données.

L'accès à cette mémoire peut se faire par Bit, par Byte, par Word ou par Long Word. Le tableau 1 montre la capacité d'une mémoire exprimée en Bit, Byte , Word et Long Word. Il est important de savoir que le Kilo correspond toujours à 1000, cependant en notation binaire la valeur la plus proche à 1000 et 1024 qui correspond à 2^{10}.

Tableau 1- mémoires exprimées en Bits, Bytes , Words et Long Words

Bit	Byte (8 bits)	Word (16 bits)	Long Word (32 bits)
1024	128	64	32
256K	32K	16K	8K
1M	128K	64K	32K

La mémoire d'un API peut être répartie sur différents types suivant la nature de son utilisation :

ROM ou PROM : elle est réservée au système d'exploitation du système. C'est ce qui est exécuté en premier dès la mise sous tension de l'API.

EEPROM (FLASH) : c'est la mémoire programme. Le programme écrit par un utilisateur est téléchargé dans cette mémoire pour être exécuté lors de l'activation du bouton RUN.

RAM : C'est une mémoire volatile; elle ne conserve pas son contenu après coupure de son alimentation.

Contrairement aux systèmes à microprocesseur à usage général, des parties de la mémoire sont réservées à des fonctions bien déterminées :

Mémentos : c'est une mémoire dont les bits seront utilisés comme des drapeaux de sauvegarde des résultats intermédiaires. Ils peuvent être aussi comme des états de conditions.

Mémoire spéciale : Chacun des bits de cette mémoire est destiné à représenter un état lors de l'exécution d'un programme. Prenons comme exemple quelques bits de la mémoire spéciale dans le cas d'un API siemens de la famille S7-200 tel qu'il est illustré par les tableaux 2 et 3.

Tableau -2 Octet de mémento spécial SMB0 (SM0.0 à SM0.7)
Ref. Manuel de référence S7-200

Bits SM	Descriptio
SM0.0	Ce bit est toujours à 1.
SM0.1	Ce bit est à 1 au premier cycle. Il sert, entre autres, à l'appel d'un sous-programme d'initialisation.
SM0.2	Ce bit est mis à 1 pour la durée d'un cycle si des données rémanentes ont été perdues. Vous pouvez l'utiliser comme mémento d'erreur ou pour appeler une séquence de mise en route particulière.
SM0.3	Ce bit est mis à 1 pour la durée d'un cycle si une mise sous tension entraîne le passage à l'état de marche (RUN). Il permet, par exemple, de fournir un temps de chauffe de l'installation avant de commencer l'exploitation.
SM0.4	Ce bit fournit une impulsion d'horloge en fonction pendant 30 secondes et hors fonction pendant 30 secondes, pour une période d'une minute. Vous disposez ainsi d'un retard d'emploi simple ou d'une impulsion d'horloge d'une minute.
SM0.5	Ce bit fournit une impulsion d'horloge en fonction pendant 0,5 seconde et hors fonction pendant 0,5 seconde, pour une période d'une seconde. Vous disposez ainsi d'un retard d'emploi simple ou d'une impulsion d'horloge d'une seconde.
SM0.6	Ce bit est une horloge de cycle en fonction pendant un cycle et hors fonction pendant le cycle suivant. Vous pouvez l'utiliser comme entrée de comptage d'un cycle.
SM0.7	Ce bit indique la position du commutateur de mode (0 correspondant à la position TERM et 1 à la position RUN). Si vous validez la communication programmable à l'aide de ce bit lorsque le commutateur de mode est sur RUN, vous pouvez valider la communication normale avec la console de programmation en mettant le commutateur en position TERM.

Tableau -3 Octet de mémento spécial SMB1 (SM1.0 à SM1.7)
Ref. Manuel de référence S7-200

Bits SM	Descriptio
SM1.0	Ce bit est mis à 1 lors de l'exécution de certaines opérations si leur résultat est égal à zéro.
SM1.1	Ce bit est mis à 1 lors de l'exécution de certaines opérations en cas de débordement ou de valeur numérique illicite.
SM1.2	Ce bit est mis à 1 lorsqu'une opération arithmétique fournit un résultat négatif.
SM1.3	Ce bit est mis à 1 lors d'une tentative de division par zéro.
SM1.4	Ce bit est mis à 1 lorsque l'opération « Inscrire dans table » provoque un débordement de la table.
SM1.5	Ce bit est mis à 1 lorsque des opérations LIFO ou FIFO tentent de lire dans une table vide.
SM1.6	Ce bit est mis à 1 lors de la tentative de conversion d'une valeur non DCB en valeur binaire.
SM1.7	Ce bit est mis à 1 lorsqu'une valeur ASCII ne peut pas être convertie en valeur hexadécimale correcte.

Mémoire image des entrées : Durant l'exécution d'un programme, l'API commence toujours par recopier les états des signaux issus des capteurs connectés aux différentes entrées dans lis bits correspondant de la mémoire images des entrées. Le programme de l'utilisateur manipule l'image d'une entrée au lieu de l'entrée elle-même.

Mémoire image des sorties : Les sorties générées lors de l'exécution d'un programme sont sauvegardées dans les bits correspondants de la mémoire de sortie. A la fin de chaque cycle d'exécution du programme en cours les images sauvegardées dans la mémoire de sortie passent par l'interface de sortie à l'extérieur à travers les bornes correspondantes.

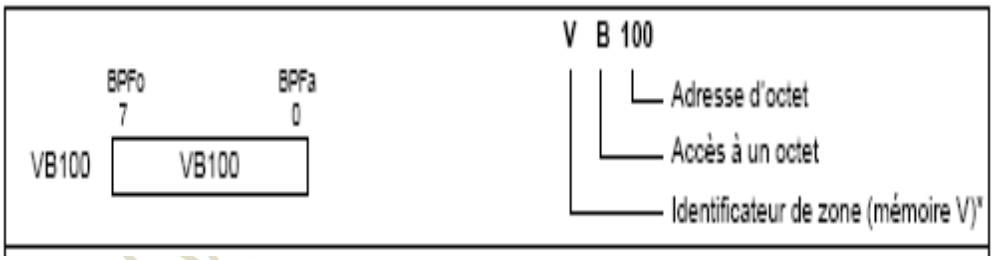
II-3 Accès à la mémoire

La manipulation de la mémoire dans le cas des automates programmables est très souple. Contrairement aux systèmes à microprocesseurs à usage générale, l'accès à la mémoire par un API peut être par Bit, par Byte (octet), par Word (Mot) ou par Long Word (Long Mot). Certes, ceci peut être rencontré sur une machine quelconque. Cependant, dans le cas des API les mots et les longs Mots peuvent chevaucher. Pour mieux comprendre le principe prenons le cas d'automate programmables de SIEMENS de la famille S7 qui sont très utilisés dans le milieu industriel. Les figures illustrant l'accès à la mémoire sont prises du manuel de Siemens.

- Byte (octet) = 8 Bits
- Word (mot) = 2 Bytes
- Long Word (Long Mot) ou Double Word (Double Mot)= 2 Word

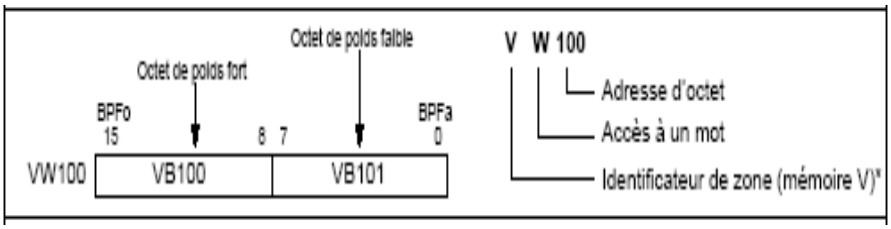
Accès par		CPU 221	CPU 222	CPU 224, CPU 226	CPU 226XM
bit (octet.bit)	I	0.0 à 15.7	0.0 à 15.7	0.0 à 15.7	0.0 à 15.7
	Q	0.0 à 15.7	0.0 à 15.7	0.0 à 15.7	0.0 à 15.7
	V	0.0 à 2047.7	0.0 à 2047.7	0.0 à 5119.7	0.0 à 10239.7
	M	0.0 à 31.7	0.0 à 31.7	0.0 à 31.7	0.0 à 31.7
	SM	0.0 à 179.7	0.0 à 299.7	0.0 à 549.7	0.0 à 549.7
	S	0.0 à 31.7	0.0 à 31.7	0.0 à 31.7	0.0 à 31.7
	T	0 à 255	0 à 255	0 à 255	0 à 255
	C	0 à 255	0 à 255	0 à 255	0 à 255
	L	0.0 à 59.7	0.0 à 59.7	0.0 à 59.7	0.0 à 59.7

Figure 3 : accès à la mémoire par Bit. Ref. Manuel de référence S7-200



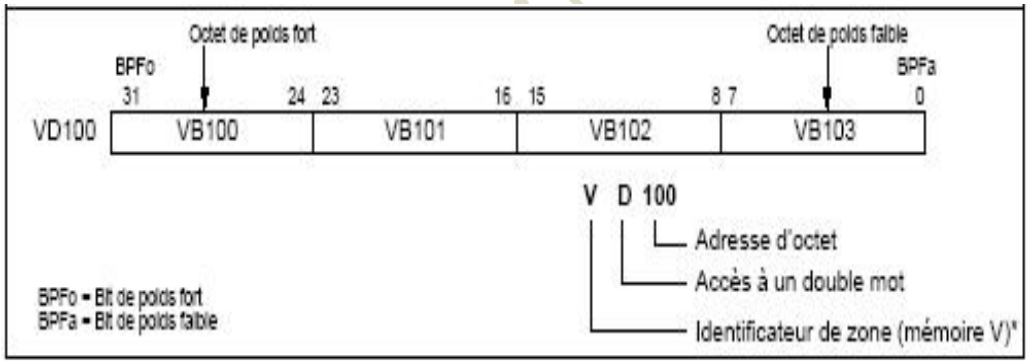
Description		CPU 221	CPU 222	CPU 224	CPU 226	CPU 226XM
octet	IB	0 à 15	0 à 15	0 à 15		0 à 15
	QB	0 à 15	0 à 15	0 à 15		0 à 15
	VB	0 à 2047	0 à 2047	0 à 5119		0 à 10239
	MB	0 à 31	0 à 31	0 à 31		0 à 31
	SMB	0 à 179	0 à 299	0 à 549		0 à 549
	SB	0 à 31	0 à 31	0 à 31		0 à 31
	LB	0 à 63	0 à 63	0 à 63		0 à 255
	AC	0 à 3	0 à 3	0 à 3		0 à 255

Figure 4 : accès à la mémoire par Byte. Ref. Manuel de référence S7-200



Description		CPU 221	CPU 222	CPU 224	CPU 226	CPU 226XM
mot	IW	0 à 14	0 à 14	0 à 14	0 à 14	
	QW	0 à 14	0 à 14	0 à 14	0 à 14	
	VW	0 à 2048	0 à 2048	0 à 5118	0 à 10238	
	MW	0 à 30	0 à 30	0 à 30	0 à 30	
	SMW	0 à 178	0 à 298	0 à 548	0 à 548	
	SW	0 à 30	0 à 30	0 à 30	0 à 30	
	T	0 à 255	0 à 255	0 à 255	0 à 255	
	C	0 à 255	0 à 255	0 à 255	0 à 255	
	LW	0 à 58	0 à 58	0 à 58	0 à 58	
	AC	0 à 3	0 à 3	0 à 3	0 à 3	
	AIW	*Néant*	0 à 30	0 à 62	0 à 62	
	AQW	*Néant*	0 à 30	0 à 62	0 à 62	

Figure 5 : accès à la mémoire par Word. Ref. Manuel de référence S7-200



Description		CPU 221	CPU 222	CPU 224	CPU 226	CPU 226XM
double mot	ID	0 à 12	0 à 12	0 à 12	0 à 12	
	QD	0 à 12	0 à 12	0 à 12	0 à 12	
	VD	0 à 2044	0 à 2044	0 à 5116	0 à 10236	
	MD	0 à 28	0 à 28	0 à 28	0 à 28	
	SMD	0 à 176	0 à 296	0 à 546	0 à 546	
	SD	0 à 28	0 à 28	0 à 28	0 à 28	
	LD	0 à 56	0 à 56	0 à 56	0 à 56	
	AC	0 à 3	0 à 3	0 à 3	0 à 3	
	HC	0, 3, 4, 5	0, 3, 4, 5	0 à 5	0 à 5	

Figure 6 : accès à la mémoire par Long Word. Ref. Manuel de référence S7-200

Pour comprendre le chevauchement de mots, prenons l'exemple du mot double VD 100 de la figure 5 :

Le mot VW100 est composé de deux octets VB100 et VB101,

Le mot VW101 est composé de deux octets VB101 et VB102,

On peut bien remarquer que l'octet VB101 représente l'octet poids faible du mot VW100 et l'octet du poids fort du mot VW101 à la fois. En effet, les deux mots VW100 et VW101 chevauchent par le partage de l'octet VB101.

II-4 Description fonctionnelle

Du moment que les automates programmables sont des systèmes à microprocesseurs dédiés à des applications industrielles où un contrôle continu est indispensable, ils sont conçus à exécuter leurs programmes d'une façon séquentielle et continue.

La figure 7 montre un schéma qui trace l'ordre des cycles lors de l'exécution d'un programme. L'API commence par scanner les états de ses entrées dont une image sera recopiée dans des registres d'entrée. Une deuxième étape sera réservée au traitement des états des entrées au niveau de l'unité de traitement de l'API suivant le programme de l'utilisateur. Dans une troisième étape, l'API recopie les résultats obtenus après exécution du programme dans des registres images de sorties pour être finalement envoyés aux éléments concernés de la partie opérative. Et le cycle recommence et continu sans s'arrêter jusqu'à l'obtention d'un ordre d'arrêt hardware ou software.

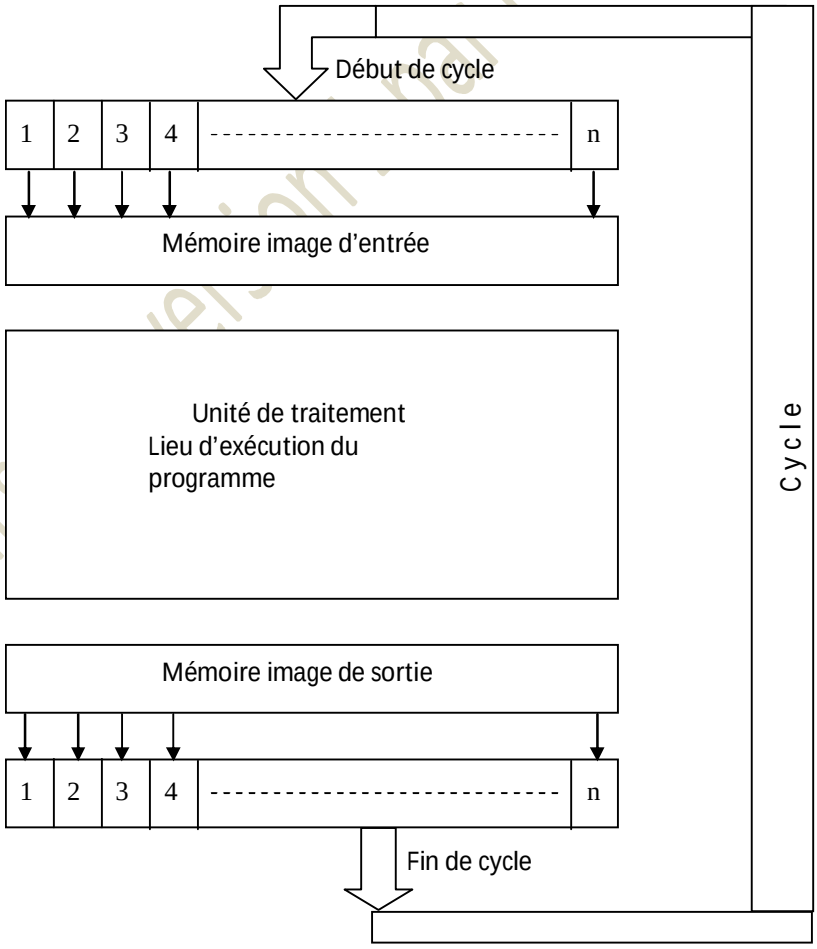


Figure 7 : Représentation des séquences d'opération pour les API

II-4 Interface d'entrée et de sortie

II-4-1 Interface d'entrée et de sortie numérique

Les circuits d'interface d'entrée et de sortie ont deux fonctions principales ; la première concerne l'isolation de l'automate de son mode extérieur alors que la deuxième permet l'adaptation des niveaux des tensions de l'API avec celle qui vont ou viennent de l'extérieur. Les schémas des figures 8 et 9 montrent les circuits d'interface d'une entrée et d'une sortie respectivement où on constate une isolation optique et le passage du niveau de tension 5V à un niveau 24V et inversement. Sur le circuit de la figure 9 on trouve aussi une isolation électromagnétique entre 24V et 220V alternatif. Les figurent 10 et 11 illustrent des exemples de connexion externes à un API.

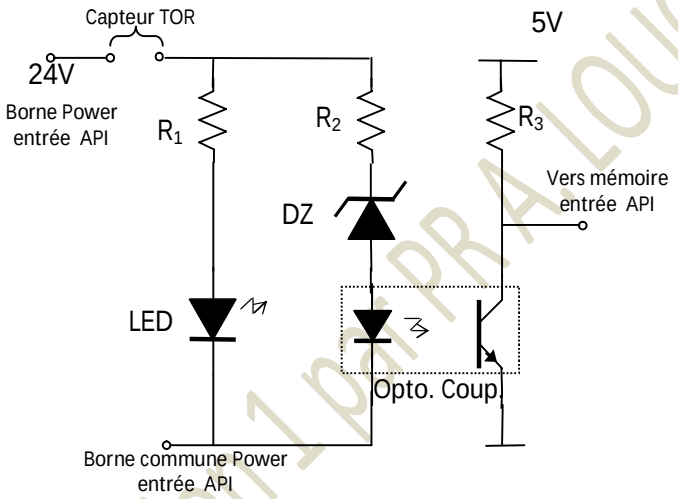


Figure 8: exemple d'une interface d'entrée

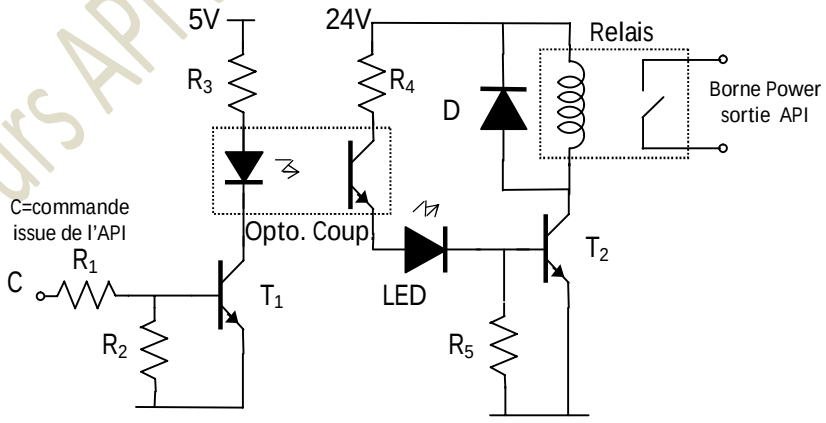


Figure 9 : exemple d'une interface de sortie

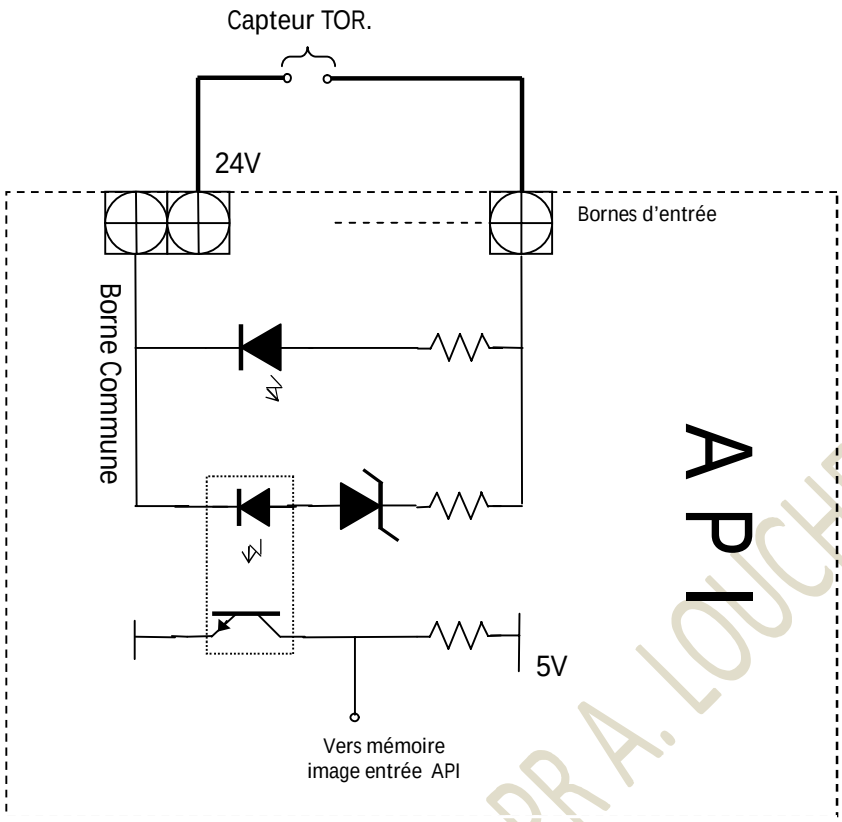


Figure 10 : Exemple de connexion d'un capteur TOR à une entrée de l'API

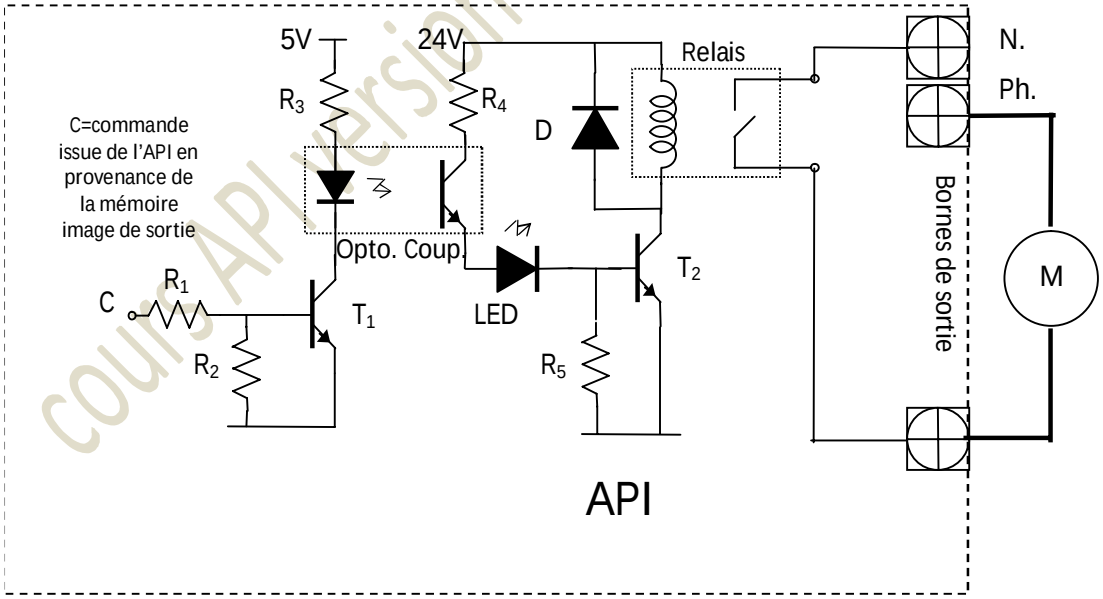


Figure 11 : Exemple de connexion d'un moteur à une sortie de l'API

II-4-2 Interface d'entrée et de sortie analogique

Actuellement, les systèmes informatiques occupent une place très importante dans le

domaine du contrôle. L'opérateur humain est substitué par un système informatique, qui est capable de lire, de traiter et surtout de prendre une décision suivant l'état présent du processus contrôlé. Cependant, généralement les grandeurs contrôlées sont des grandeurs physiques traduites par des capteurs en grandeurs électriques analogiques, alors que les systèmes informatiques ne sont que des systèmes numériques pouvant manipuler des 1 et des 0.

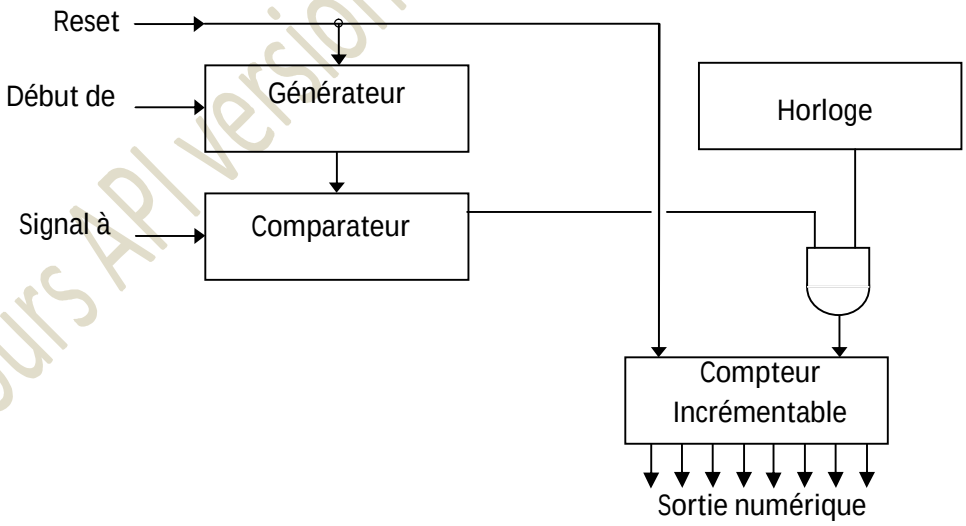
Par conséquent, une fois la grandeur est conditionnée et prête à être traitée, elle doit être convertie en une suite de combinaisons numériques pour enfin être présentées au système numérique. Les circuits permettant une telle transformation sont appelés convertisseurs analogiques numériques CAN ou plus exactement ADC (Analog-Digital Converters).

Sur le marché nous trouvons une large gamme de CAN, des plus simples à ceux compatibles de travailler avec des microprocesseurs, et dont la liaison aux systèmes informatiques est directe.

II-4-2-1 convertisseurs analogiques numériques

a- convertisseurs " simple rampe "

Le schéma de base d'un tel type de convertisseur est illustré par la figure-12. Son principe de fonctionnement est basé sur la comparaison d'une dent de scie générée localement avec le signal utile à convertir. A chaque début de la dent de scie, l'état de la sortie du comparateur permet le passage de l'horloge au compteur et le compte commence; juste au moment t_0 où la dent de scie ait une valeur égale au signal à convertir la sortie du comparateur bascule et le comptage sera arrêté. La sortie ainsi affichée par le compteur représente le code numérique du signal à convertir. La figure-13 présente le chronogramme de fonctionnement des convertisseurs "simple rampe".



Figure—12: CAN '' Simple Rampe ''

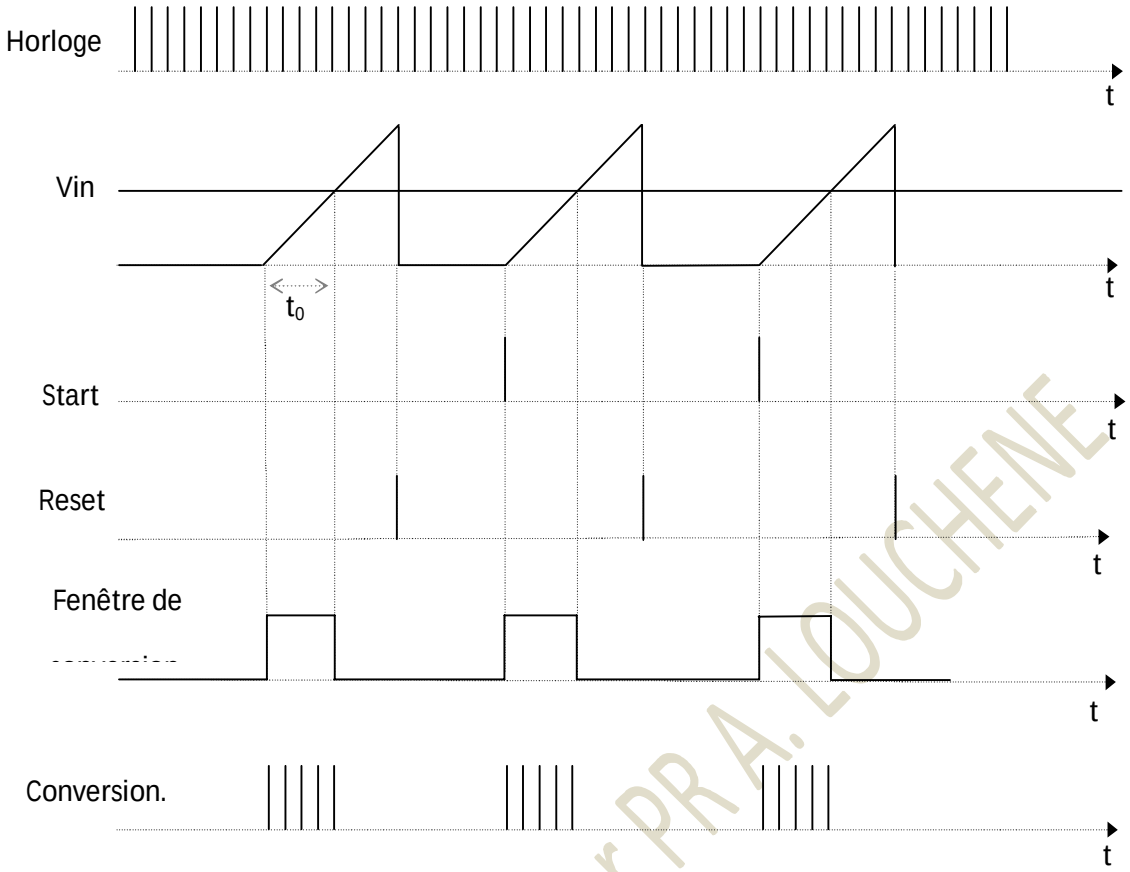


Figure-13 : Chronogramme de fonctionnement d'un CAN "simple rampe"

L'inconvénient majeur d'un tel convertisseur analogique numérique réside dans son temps de conversion qui est relativement élevé et qui est étroitement lié à la valeur de la grandeur à convertir.

b- Convertisseurs "double rampes"

L'élément de base au niveau de ce type de convertisseur est un intégrateur. On les appelle dans certains ouvrages '' Convertisseurs Intégrateurs '' comme illustré sur le schéma de la figure-14. Le principe de fonctionnement consiste à intégrer la grandeur à convertir pendant un temps fixe T_0, figure-15. Puis à la place de la grandeur à convertir, on applique au même intégrateur une référence V_{ref} de polarité opposée à celle du signal à convertir. L'intervalle de temps T_1, qui définit le temps mis entre la commutation au niveau de l'entrée de l'intégrateur et la valeur nulle affichée à sa sortie, correspond au temps donné au compteur pour afficher à sa sortie la combinaison numérique correspondante à la valeur analogique à convertir.

Ces convertisseurs sont immunisés contre le bruit surtout celui lié à la fréquence du secteur. Il suffit de prendre une période d'intégration multiple de celle correspondante à la fréquence du réseau et toutes les perturbations alternatives correspondantes à la fréquence du réseau et de ses multiples seront éliminées.

Les CAN '' Double Rampes '' sont surtout utilisés dans les appareils de mesures numériques qui sont sensés être utilisés dans des environnements perturbés. Comme il peut être constaté, l'immunité au bruit est choisie au dépend du temps de conversion.

La génération du signal de fin de conversion EOC « End Of Conversion » peut facilement être exploité pour l'interfaçage du convertisseur avec des systèmes numériques tels que système à microprocesseur, micro-controlleur, PC….

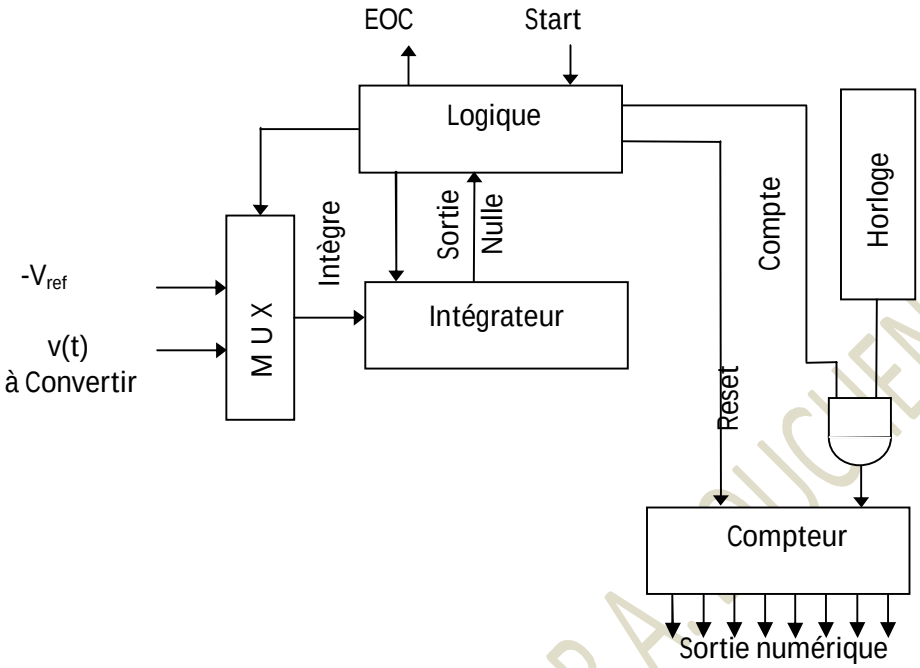


Figure-14 : CAN " Double Rampes "

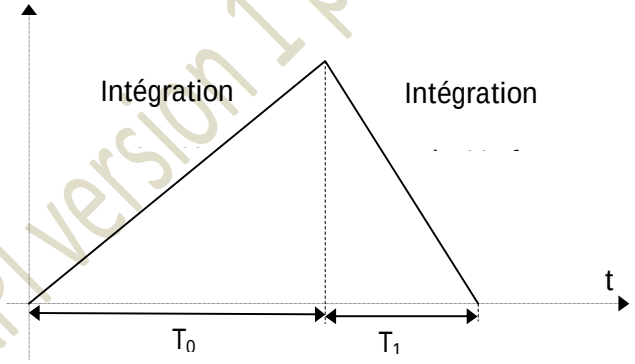


Figure-15 : Réponse de l'intégrateur

Par intégration de la valeur V_x de v(t) sur l'intervalle $[0,T_0]$ on obtient:

$$K.V_x.T_0 \qquad (1)$$

Par intégration de $-V_{ref}$ sur l'intervalle $[T_0,T_1]$ on aura:

$$K.(-V_{ref}).(T_1-T_0) \qquad (2)$$

Avec K comme constante d'intégration du circuit intégrateur.

Pour un cycle d'horloge de T_H on a:

$$T_0 = N_0.T_H \quad \text{et} \quad T_1-T_0 = N.T_H \qquad (3)$$

Où N_0 est fixe et N représente le résultat de la conversion.

Si la valeur crête de l'intégration est V_0 de l'équation (3) nous tirons:

$$V_0 = K.V_x.N_0.T_H = K.V_{ref}.N.T_H \quad (4)$$

$$N = \frac{V_X}{V_{ref}} N_0 \quad (5)$$

c- Convertisseurs " suiveur "

Le convertisseur analogique numérique suiveur appartient au groupe de convertisseurs dits convertisseurs à contre réaction dont la boucle de retour comprend généralement un deuxième convertisseur numérique analogique.

La valeur du signal à convertir est continuellement comparée avec la sortie du CNA de la boucle de retour. Suivant l'état de sortie du comparateur, l'opération d'incrémentation ou de décrémentation du compteur sera sélectionnée par la logique de contrôle.

Si la sortie du CNA est supérieure à la valeur du signal à convertir le compteur sera décrémentée; dans le cas contraire, il sera incrémenté. C'est ainsi que la grandeur à convertir sera suivie d'où le nom de ce convertisseur. Relativement aux deux convertisseurs précédents, ce CAN est plus rapide. Le schéma de principe de la figure-16 représente le bloc fonctionnel du CAN suiveur.

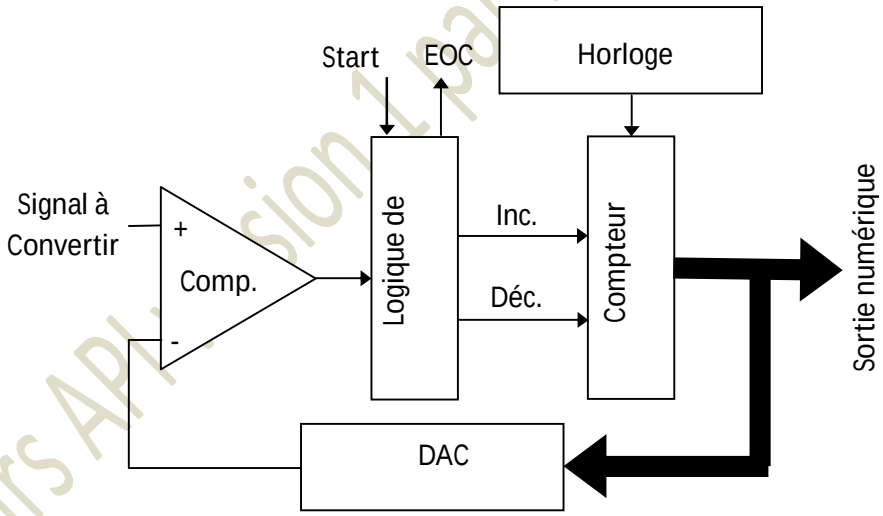


Figure-16 : CAN ''suiveur ''

D- CONVERTISSEURS ''A APPROXIMATION SUCCESSIVE ''

Le convertisseur à approximation successive est le plus rapide des convertisseurs cités auparavant. Son temps de conversion est fixe, il ne dépend pas de la valeur du signal à convertir, seule la précision du convertisseur le fixe.

L'élément essentiel dans ce convertisseur est le registre à approximation successive utilisé. Il peut être considéré comme un registre à décalage spécial. La présence de ce dernier introduit une stratégie efficace dans la génération du code numérique de sortie. La figure-17 donne le schéma de principe d'un convertisseur à approximation successive.

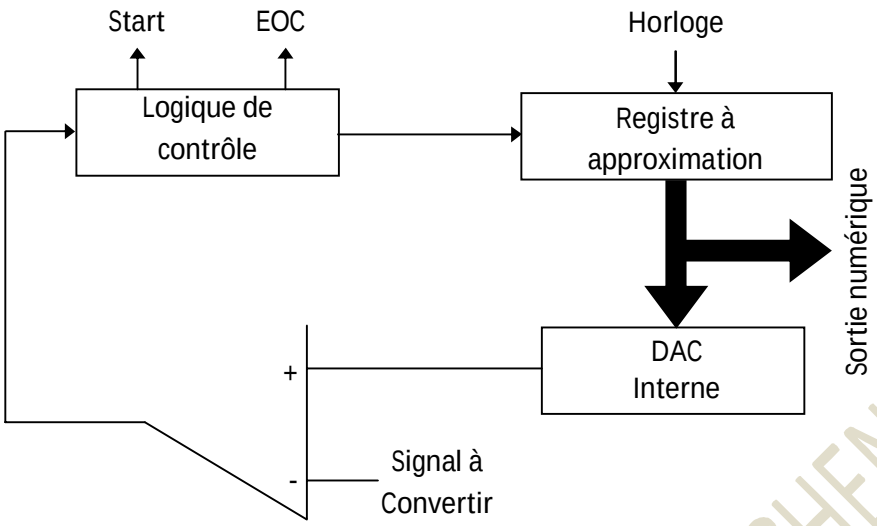


Figure-17 : CAN ''à approximation successive ''

Pour bien comprendre le principe de fonctionnement de ce convertisseur, prenons l'exemple suivant :

Etant donné un convertisseur à approximation successive unipolaire à 4 bits de précision et une tension de 16V en pleine échelle. Supposant que la valeur de la tension à convertir est de 5,5V. Alors la génération du code, correspondant à 5,5V, passe par les étapes résumées dans le tableau-4 ci-dessous :

Tableau-4 : Principe du CAN ''à approximation successive ''

Rang du bit	Code correspondant	Tension V_x correspondante	V_x par rapport à V_{in}	Code maintenu.
4ième	1000	8V	8V>5.5V	0000
3ième	0100	4V	4V<5,5V	0100
2ième	0110	6V	6V>5.5V	0100
1er	0101	5V	5V<5,5V	0101

Le résultat final de la conversion sera donné par le code 0101. Nous pouvons constater que le nombre d'opérations effectuées lors de la recherche du code est égal à la précision du convertisseur. Par conséquent un convertisseur à N bits de sortie effectue N opérations lors de la conversion, ce qui correspond à un temps de conversion de NT_H avec T_H cycle d'horloge.

e- CAN parallèle

Bien que leur principe de fonctionnement soit le plus facile à comprendre, les convertisseurs parallèles sont les plus rapides de tous les convertisseurs. Ils sont les plus difficiles à réaliser avec un coût plus élevé.

La tension à convertir est appliquée simultanément aux entrées non inverseuses des 2^N comparateurs pour être comparées à des fractions de la tension de référence. Ces fractions de la tension de référence sont obtenues grâce à un pont diviseur constitué de 2^N +1 résistances, où N représente la précision du convertisseur. Delà on peut imaginer la complexité de ce type de convertisseur qui est liée au nombre de composants demandés. A titre d'exemple, un convertisseur parallèle d'une précision de 8 bits utilise 256 comparateurs et 257 résistances. A

cause de leur rapidité on les appelle convertisseurs FLASH. La figure-18 illustre le schéma de principe d'un convertisseur parallèle à 3 bits de précision.

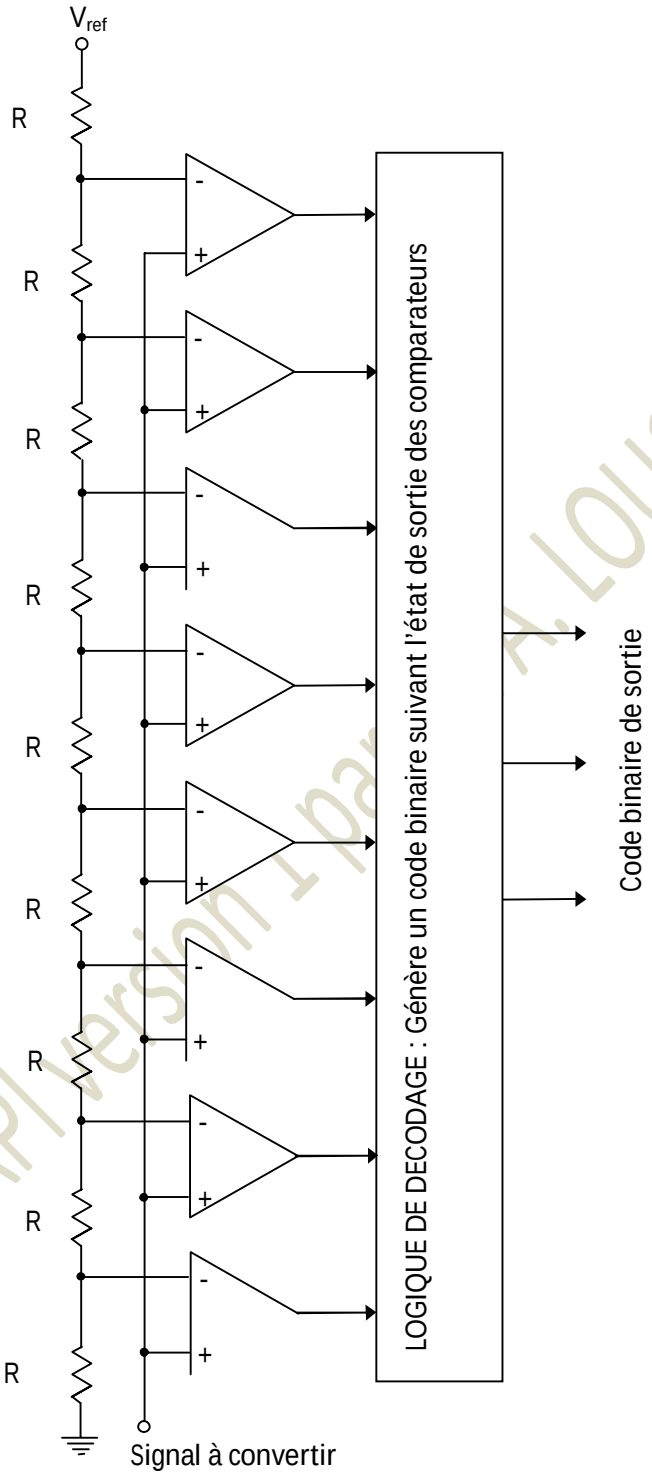


Figure-18: CAN " Parallèle " ou " flash ADC "

II-4-2-2 INTERFACE CAN-SYSTEME PROGRAMMABLE

a- premier cas

Dans ce premier cas nous considérons que le code généré par le CAN correspond à la largeur du bus de données du système à microprocesseur. Pour ce type de convertisseur l'interface est directe, seul un latch (verrou) à trois états est indispensable pour éviter un conflit au niveau des données issues des différents périphériques connectés à ce même bus. Pour éviter d'éventuelles pertes d'information, les deux signaux START et EOC sont utilisés comme un moyen de « Handshake » entre le CAN et le microprocesseur.
La figure-19 donne un exemple d'interface d'un convertisseur à 8 bits et un système à microprocesseur avec bus de données à 8 bits. Une fois la conversion est terminée, le CAN génère un signal EOC qui verrouille l'information dans un latch à trois états et en même temps il informe le microprocesseur de la présence d'un code validé. Par l'association d'une adresse d'identification du latch et du signal READ, un signal ENABLE active le latch pour placer son contenu sur le bus de données pour être véhiculer vers le microprocesseur.

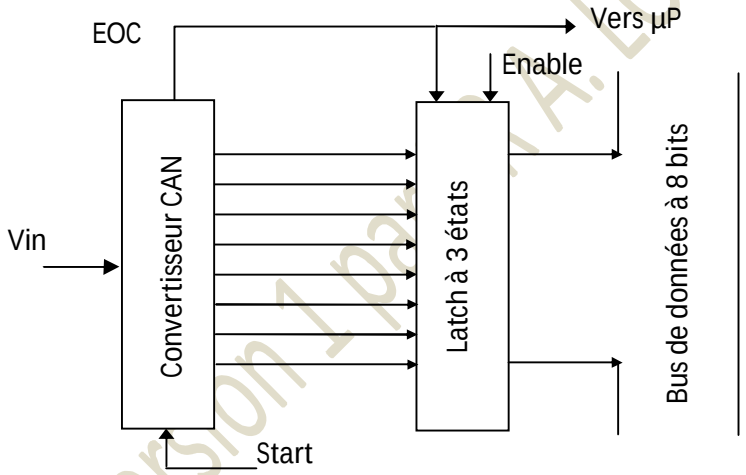


Figure-19 : CAN à 8 bits connecté à un système à µP 8 bits

b- deuxieme cas

Dans ce deuxième cas nous considérons l'interface d'un CAN dont la précision dépasse la largeur du bus du système à microprocesseur. La lecture du code de sortie du convertisseur nécessite plus d'un cycle machine. L'information obtenue suite à une conversion doit être mémorisée dans des latch pour être ensuite lues par le microprocesseur.

Prenons le cas d'un convertisseur analogique numérique à 12 bits travaillant dans un environnement de microprocesseur à 8bits. Le schéma d'interface est illustré par la figure-20 où une adresse est attribuée pour chacun des deux latch.

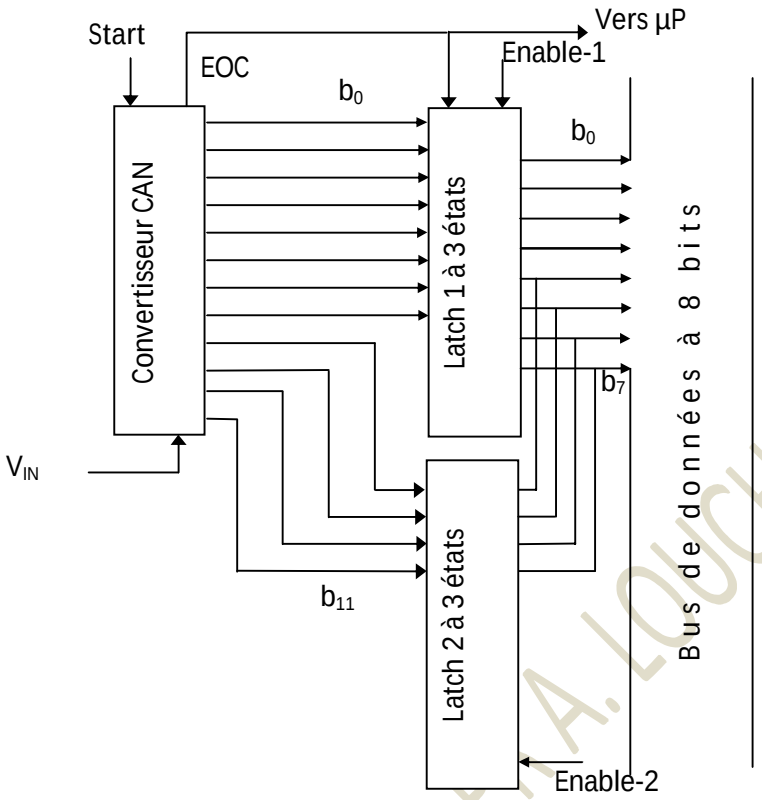


Figure-20 : CAN à 12 bits connecté à un système à µP 8 bits

c- cas de contrôle de plusieurs grandeurs

Au niveau d'un processus automatisé, le contrôle ne se limite pas à un seul paramètre. Par conséquent, dans une chaîne de mesure on trouve plusieurs grandeurs physiques à contrôler. Dans ce cas précis, on doit concevoir soit pour chacune des grandeurs sa propre chaîne de mesure, soit on conçoit une chaîne commune pour toutes les grandeurs. Généralement, c'est le deuxième cas de conception qui est retenu. Dans le cas de chaîne de mesure pour chacune des grandeurs à contrôler, beaucoup de composants sont nécessaires, ce qui augmente considérablement le prix et l'encombrement du circuit réalisé.

On donne sur la figure-21, un schéma de principe d'une chaîne de mesure à plusieurs voies.

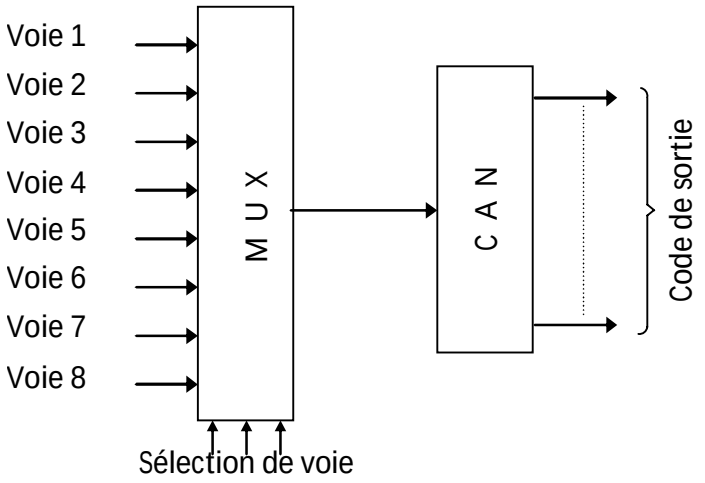


Figure-21 : Interface pour 8 voies

II-4-2-3 Choix d'un convertisseur analogique numérique

Dans une chaîne de mesure numérique, le convertisseur analogique numérique constitue le maillon le plus important. Son choix est fonction de la qualité de mesure que nous voulons effectuer.

Parmi les caractéristiques de choix d'un convertisseur analogique numérique, nous citons :

Tension analogique d'entrée : Elle définit la plage de variation de la tension qu'on peut appliquer à l'entrée. Pour un CAN unipolaire, elle est exprimée par $[0, V_{MAX}]$, alors que pour un CAN bipolaire elle est donnée par $[0, \pm V_{MAX}]$.

Longueur du code de sortie N : N est égal au nombre de bit de la combinaison numérique délivrée à la sortie du CAN.

Temps de conversion : C'est le temps que met un CAN pour présenter un code à sa sortie correspondant à l'échantillon appliqué à son entrée.

Résolution : La variation du signal d'entrée, qui provoque un changement au niveau du code de sortie. Elle est exprimée par le rapport de la tension maximale applicable à l'entrée sur 2^N.

II-5 Programmation des API

Dans la plus part des cas, les automates programmables industriels n'ont pas seulement une architecture modulaire pour plus de souplesse dans l'ajout de modules d'extension, mais ils ont un ensemble de langages de programmation standardisé sous la norme IEC 61131-3. Cette norme regroupe 5 langages :

Ladder (LD) : c'est un langage de programmation graphique. Il ressemble au schéma d'un schéma électrique où on trouve des interrupteurs dans la commande et des relais comme éléments d'action. Suivant la condition sur les états des interrupteurs le relais sera activé ou non. Ce langage permet d'interpréter des équations booléennes.

Instruction Liste (IL) : C'est une liste d'instructions où chaque instruction est sous forme d'une abréviation. C'est un langage qui ressemble à l'assembleur utilisé dans les systèmes à microprocesseurs en général. Ce langage est généralement dans le cas où on utilise de simples consoles pour éditer un programme se trouvant la mémoire de l'API pour d'éventuelles modifications ou introduire un nouveau programme.

Function Bloc Diagram (FBD) : Ce langage, qui lui aussi est un langage de programmation graphique, est composé par des blocs qui chacun d'eux correspond à une fonction bien déterminée. La syntaxe de ces blocs est généralement des rectangles avec des variables d'entrées et des sorties, avec à l'intérieur un symbole ou un texte qui définit la fonction.

GRAFCET : Ce langage permet de d'écrire l'évolution du procédé par un graphe qui montre les différentes étapes et les conditions qui permettent le passage d'une étape à une autre. Les étapes montrent les états stables, et la transition d'une étape à une autre est conditionnée par le résultat d'une expression booléenne. A chaque étape correspond une action.

Structured text (ST) : C'est un langage évolué qui permet de traduire des algorithmes complexes tels que C ou autres.

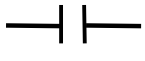
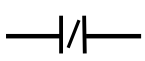
II-5-1 Le LADDER (LD) et Liste d'Instructions (IL°

On commence l'étude des langages de programmation le LADDER et le IL parce qu'ils représentent les langages les plus rencontrés et les utilisés dans les milieux industriel. Ils sont faciles à manipuler. Le principe du Ladder repose sur les schémas électriques que les électriciens utilisent pour schématiser une installation électrique.

Nous donnons sur les tableaux qui suivent quelques instructions dont la syntaxe est celle rencontrée dans les environnements de développement des API de Siemens.

On donne dans le tableau un quelques instructions mais pour plus d'information sur le reste des instructions voir le manuel la famille des API S7 200 de siemens (peut être facilement téléchargé de l'internet version anglaise ou française).

Tableau 5 : Exemples d'instructions de contact

LADDER	Description de la fonction de l'instruction	IL
	Interrupteur ouvert au repos	LD
	Interrupteur fermé au repos	LDN

	Actif au front montant	EU remarque : la conséquence ou le résultat activé par le front montant dure un seul cycle
	Actif au front Descendant	ED remarque : la conséquence ou le résultat activé par le front descendant dure un seul cycle

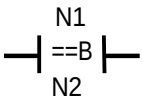
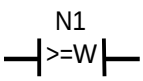
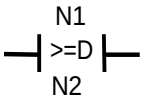
Tableau 6: Exemples d'instructions de comparaison		
LADDER	Description de la fonction de l'instruction	IL
	Comparaison entre les octets N1 et N2 N1=N2 ?	LDB= N1 , N2
	Comparaison entre les mots N1 et N2 N1≥N2 ?	LDW>= N1 , N2
	Comparaison entre les longs mots N1 et N2 N1≥N2 ?	LDD>= N1 , N2

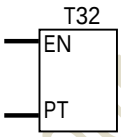
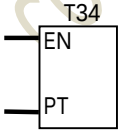
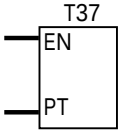
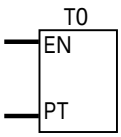
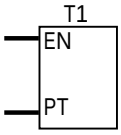
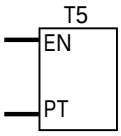
Tableau 7 : Exemples d'instructions de timers TON		
LADDER	Description de la fonction de l'instruction	IL
	Timer T32 est du type TON : EN=0 il est remis à zéro EN=1 s'incrémente suivant sa base de temps 1ms	TON T32 , PT
	Timer T34 est du type TON : EN=0 il est remis à zéro EN=1 s'incrémente suivant sa base de temps de 10ms	TON T34 , PT
	Timer T37 est du type TON : EN=0 il est remis à zéro EN=1 s'incrémente suivant sa base de temps de 100ms	TON T37 , PT

Tableau 8 : Exemples d'instructions de timers TONR		
LADDER	Description de la fonction de l'instruction	IL
	Timer T0 est du type TONR EN=0 arrête l'incrémentation et conserve la valeur comptée EN=1 s'incrémente suivant sa base de temps de 1ms	TONR T0 , PT
	Timer T1 est du type TONR EN=0 arrête l'incrémentation et conserve la valeur comptée EN=1 s'incrémente suivant sa base de temps de 10ms	TONR T1 , PT
	Timer T5 est du type TONR EN=0 arrête l'incrémentation et conserve la valeur comptée EN=1 s'incrémente suivant sa base de temps de 10ms	TONR T5 , PT

II-5-2 Exemple de programme en Ladder et IL :

Soit un circuit électrique d'allumage d'une lampe de 220V. La lampe s'allume que si les deux interrupteurs K1 et K2 sont fermés. Si l'un ou les deux s'ouvrent la lampe s'éteint.

Comme on peut le constater sur la figure 22, le fait que les deux interrupteurs K1 et K2 sont en série, le courant arrivera à la lampe (le circuit se fermera) que si les deux interrupteurs sont fermés. Dans ce cas de figure, la lampe affiche le résultat d'un ET logique (AND) de K1 et K2.

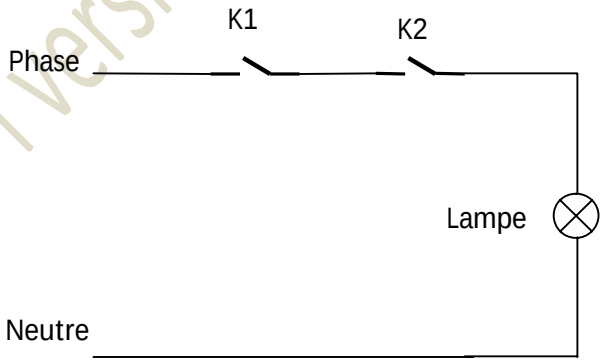


Figure 22 : Cas d'une lampe actionnée par deux interrupteurs en série

Pour automatiser cet exemple en utilisant un API, deux points seront considérés :

- la connexion : - en entrées des deux capteurs traduisant les états de K1 et K2.

  - en sortie de la lampe.

- Le programme à écrire pour l'API pour réaliser cette tâche.

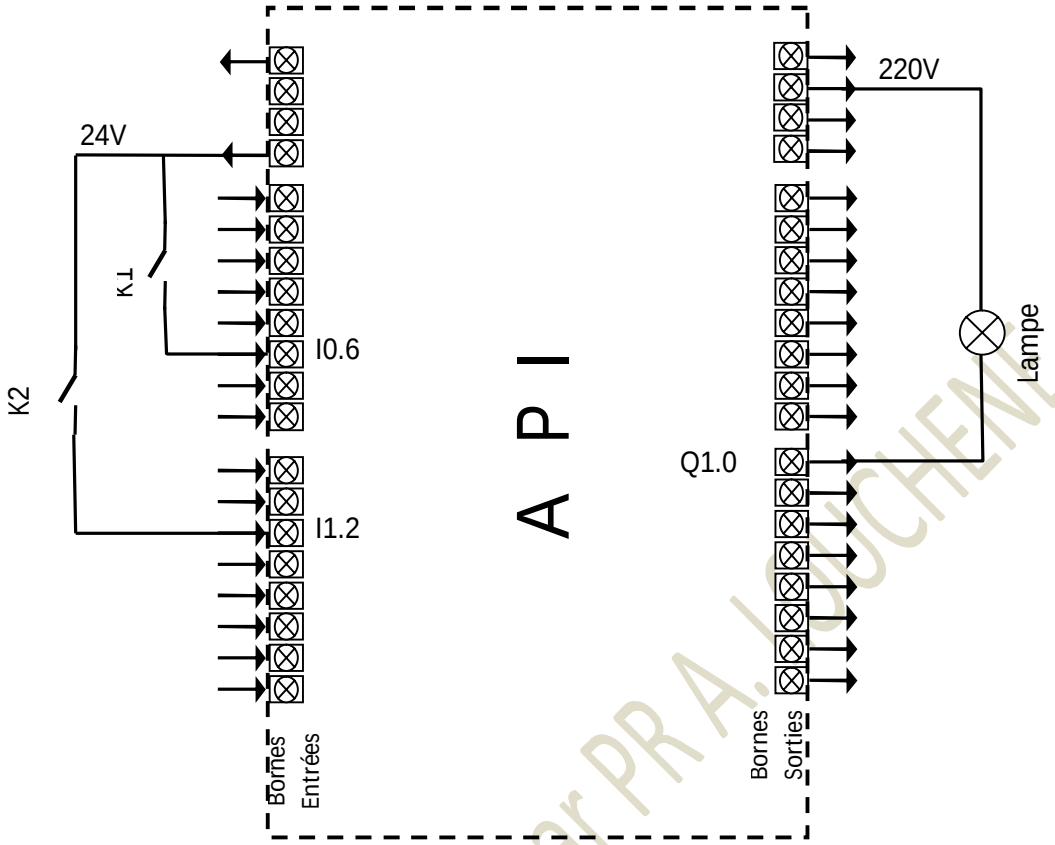


Figure 23 : Exemple de connexion d'une lampe actionnée par deux interrupteurs

Programme de l'utilisateur répondant à l'exemple ci-dessus.

Programme en LADDER	Programme en IL
Network1 I0.6 I1.2 Q1.0 Network2 end	Network 1 LD I0.6 A I1.2 = Q1.0 Network 2 End

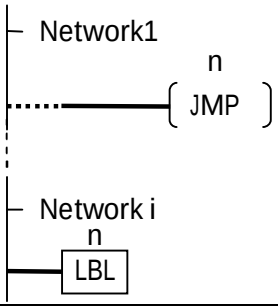
II-6 SAUT, APPEL et INTERRUPTION

Instructions de saut

Tout processeur exécute d'une façon continue un programme en passant d'une instruction à l'instruction suivante. Les instructions de SAUT permettent au processeur d'effectuer à un saut vers une instruction au sein d'un même programme :

- *Instruction de saut dans le programme principal : le saut s'effectue exclusivement vers une instruction du programme principal.*
- *Instruction de saut dans une subroutine : le saut s'effectue exclusivement vers une instruction de la subroutine*
- *Instruction de saut dans une routine d'interruption : le saut s'effectue exclusivement à vers une instruction de la routine d'interruption.*

Tableau 9 : Exemples d'instructions de SAUT

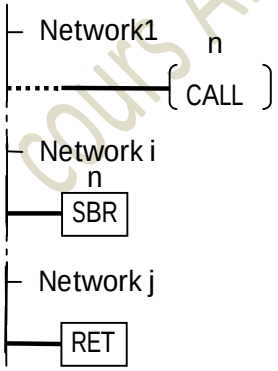
LADDER	Description de la fonction de l'instruction	IL
	Le network 1 est une instruction de saut vers le network i identifié par l'étiquette (Label LBL numéro n)	Network 1 JMP n . . . Network i LBL n

Généralement, Le numéro n de l'étiquette vers laquelle on effectue le saut peut prendre des valeurs allant de 0 à 255.

Instructions d'appel de subroutine

Comme on peut le remarquer sur le titre de ce paragraphe, les appels de subroutine se font par l'exécution d'instruction d'appel qui fait partie du programme principal. Cette instruction arrête temporairement l'exécution du programme principal pour passer à l'exécution d'un autre petit programme (subroutine) pour une tâche bien déterminée. Une subroutine doit être terminée par une instruction de retour au programme principal.

Tableau 10 : Exemples d'instructions d'APPEL de Subroutine

LADDER	Description de la fonction de l'instruction	IL
	Le network 1 est une instruction d'appel de la subroutine n commençant au network i et s'arrête au network j par une instruction de RET de retour au programme principal.	Network 1 CALL n . . Network i SBR n . . Network i RET

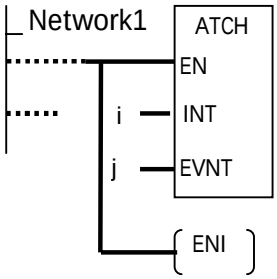
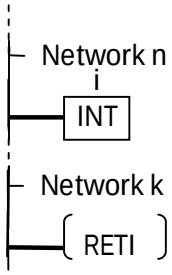
Du fait que dans le cas d'un appel de subroutine, le processeur doit retourner au programme principal pour continuer son exécution à partir du point où il s'est arrêté, ce dernier doit sauvegarder d'une façon automatique certains de ses registres (compteur ordinal,

accumulateurs…) avant de passer à la subroutine pour les récupérer à la suite de l'exécution de l'instruction RET.

Interruption

L'interruption est due essentiellement à un évènement, de nature interne ou externe au système, qu'on ne peut savoir à quel moment ça peut arriver. Il est généré intérieurement au système par le débordement d'un compteur, par le watchdog ou autre. Il est aussi le plus souvent généré extérieurement par le changement d'état d'une ligne du système réservée à cette fonction généralement notée par IRQ (Interrupt request).

Tableau 11 : Exemples d'interruption

LADDER	Description de la fonction de l'instruction	IL
	Au début du programme principal dans la partie d'initialisation on doit affecter un évènement à une interruption et valider les interruptions telles qu'il est montré par network1. Les pointillés montrent l'endroit d'une éventuelle condition.	Network 1 LD …condition. ATCH i, j ENI
	Le network n est le début de la routine d'interruption qui se trouve hors programme principal. Cette routine est appelée et exécutée à chaque occurrence de l'évènement j. L'exécution du network k permet le retour au programme principal.	Networkn INT n . . Network k RETI

II-6 Exercices

Exercice-1(ratt. du 04/2012 dpt ELN UB par Pr. A. LOUCHENE)

1. Donner le cycle d'exécution d'un programme par un API en expliquant chacune des étapes.
2. Que comprend-on dans le cas d'un S7 200 par SM2.5, VW100, M3.6, IB6 ?
3. En se limitant au LADDER, donner un exemple d'instructions pour les cas suivants :
   - Instruction de contact.
   - Instruction de transfert d'un octet
   - Instruction de transfert d'un mot
   - Instruction de comparaison avec un octet
   - Instruction de comparaison avec un mot.
4. Donner l'instruction et le type de timer pour :
   - une temporisation de 15,5s
   - une temporisation de 38 s
   - une temporisation de 330s

Exercice-2(exo 1 du ctl 2016 dpt ELN UB par Pr. A. LOUCHENE)

Convertir IL la partie de programme écrite en Ladder donné sur la figure24 .

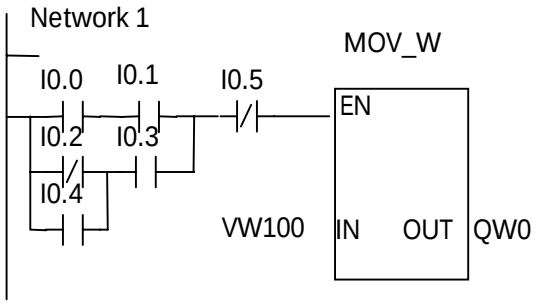


Figure 24

Exercice-3(exo 2 du ctl 2016 dpt ELN UB par Pr. A. LOUCHENE)

On veut automatiser le remplissage d'un réservoir d'eau alimentant un petit village. Le réservoir est alimenté par deux puits grâce à deux pompe P1 et P2. Quand le réservoir est au dessous du niveau minimal indiqué par NV_B les deux pompes démarrent. Quand le niveau moyen VN_M est atteint la pompe P2 s'arrête et P1 continue à fonctionner jusqu'à ce que le niveau haut NV_H soit atteint.

Exercice-4(exo 3 du ctl 2016 dpt ELN UB par Pr. A. LOUCHENE)

Avant d'écrire un programme de gestion pour le wagon de la figure 25, on demande de proposer un algorithme ou un organigramme qui permet de décrire le cycle complet du wagon. Au repos le wagon est en poste A qui est indiqué par la fermeture du fin de course FC_A. Dans cette position initiale et uniquement cette position, l'activation du bouton poussoir BP_A permet au wagon de se déplacer à droite vers les postes B puis vers C puis vers D. L'arrivée du wagon à un poste est indiquée par l'activation du Fin de Course correspondant (FC_A, FC_B, FC_C et FC_D pour poste A, B, C et D respectivement). Les séquences de déplacement du wagon sont : Arrivant à B il marque un temps d'arrêt de 10s puis continue vers C où il marque le même d'arrêt de 10s pour ensuite passer à D où il marque aussi un temps d'arrêt de 10s avant de retourner en marche arrière au poste de repos A..

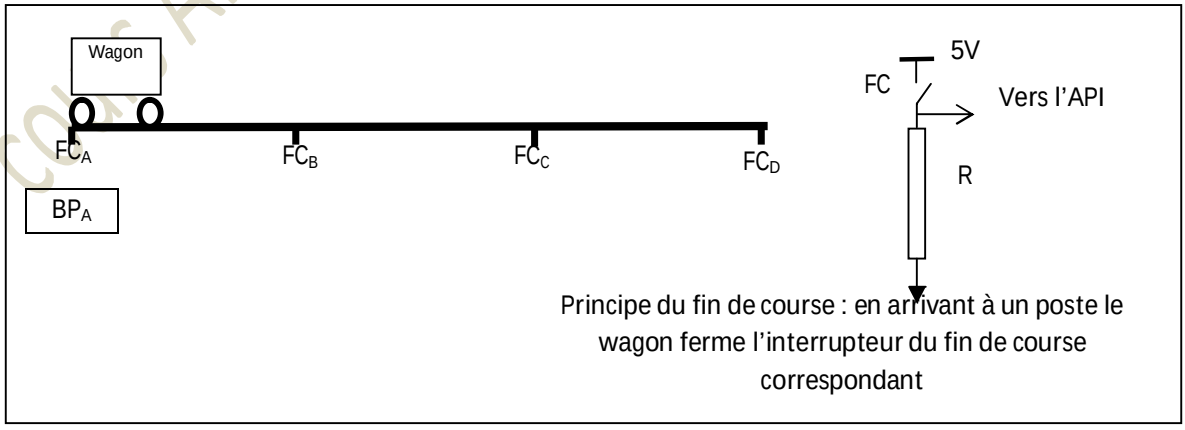


Figure 25

Exercice 5(int.1 du 12/2015 dpt ELN UB par Pr. A. LOUCHENE)

Convertir le programme ci dessous en Ladder.

```
NETWORK1
LD  I0.0
A   I0.1
TON T33,300

NETWORK2
LD  I1.0
LD  I1.1
LDW>= T33, 150
OLD
ALD
=   Q0.0
NOT
=   Q0.1

NETWORK3
end
```

Exercice 6(int.2 du 1/2016 dpt ELN UB par Pr. A. LOUCHENE)

- Convertir le programme ci-dessous en LADDER.
- Donner le chronogramme des signaux reçus à travers les sorties en spécifiant la fréquence pour chacune des sorties.

```
NETWORK1
LDW>=T32,200
LDW< T32,100
OLD
= Q0.1
NETWORK2
LDW>= T32,100
= Q0.0
```

```
NETWORK3
LD M0.5
TON T32, 300
NETWORK4
LD T32
NOT
=M0.5
NETWORK5
end
```

Exercice-7 (exo 1 du ctl 2011 dpt ELN UB par Pr. A. LOUCHENE)

Le schéma donné sur la figure 26 présente l'intersection de trois routes secondaires B, C et D avec une route principale A. Les séquences du pilotage des feux de signalisation au niveau des trois intersections sont montrées sur la figure 27.

On demande de donner la structure du programme de gestion de ces feux de signalisation.

- La première lettre R, V ou O indique la couleur du feu de signalisation respectivement Rouge , Vert ou Orange
- *VA_B : la lampe correspondante au feu vert pour la route A au niveau de l'intersection AB*
- *RB : : la lampe correspondante au feu rouge pour la route B*

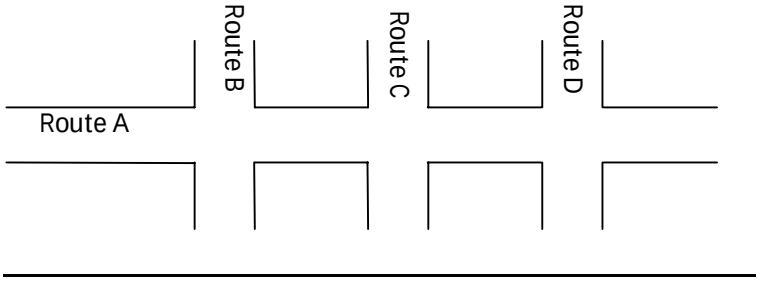


Figure 26

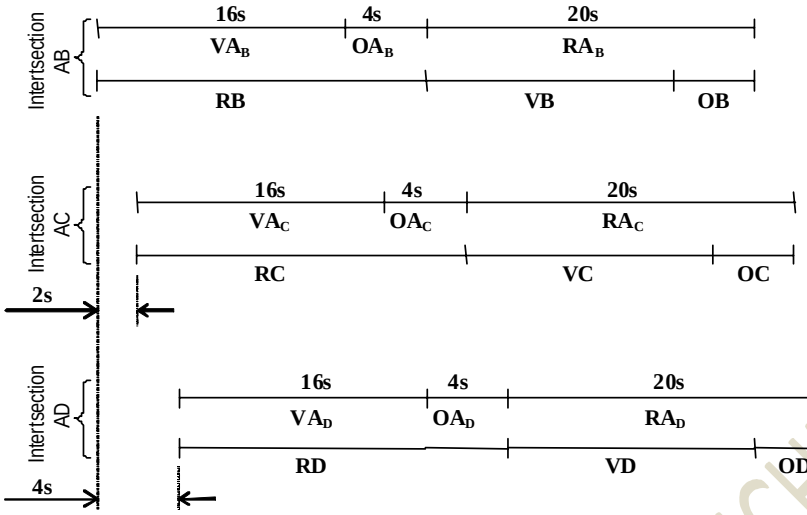


Figure 27

Exercice-8 (exo 2 du ctl 2011 dpt ELN UB par Pr. A. LOUCHENE)

On se propose de générer deux signaux carrés à travers les sorties Q0 et Q1 de l'API S7 200 de siemens, le chronogramme des signaux à générer est donné sur la figure 28. On demande de :

1. Justifier le type de timer que vous utilisez.
2. d'écrire le programme en LADDER correspondant à la génération des deux signaux.

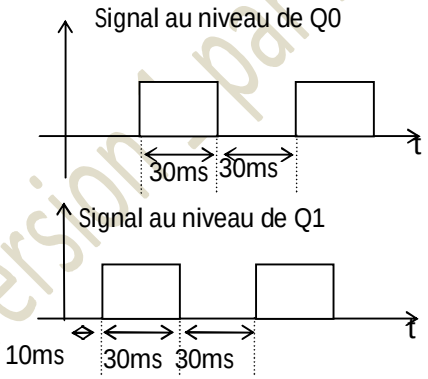


Figure 28

II-6 GRAFCET

II-6-1. Introduction

Un système automatisé peut généralement être décrit par deux parties interconnectées comme montré sur la figure 29. Une première partie représente la commande qui est aussi appelée l'automatisme alors qu'une deuxième partie décrit le processus ou la partie opérative.

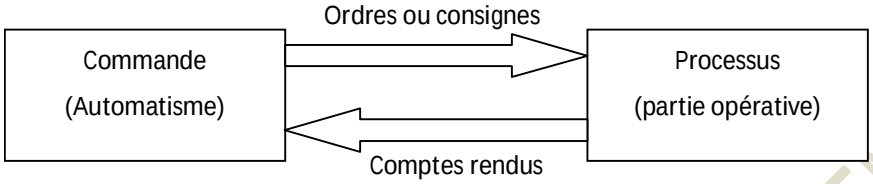


Figure29 : Structure d'un processus automatisé

Deux types d'automatismes peuvent être rencontrés : l'automatisme de régulation dont l'action de réglage est essentiellement basée sur l'asservissement, et l'automatisme séquentiel où un ensemble d'actions sont balayées dans un ordre bien déterminé.

Pour l'automatisation d'un processus, le concepteur doit disposer de tous les détails sur la nature des données spécifiant les échanges « Consignes – Comptes rendus » entre la partie commande et la partie opérative, sous forme d'un cahier des charges traçant les différents modes de fonctionnement et les sécurités que devra posséder cet automatismes. Les langages courants ou naturels se révèlent mal adaptés à la description de l'évolution d'un système séquentiel à cause du risque d'incompréhension ou de malentendu entre ce qui décrit le système et ce qui l'interprète. Ce qui a poussé les spécialistes à développer un langage intermédiaire universel qui permet, à un système automatisée, une description fonctionnelle claire, précisé et sans ambiguïté. C'est un langage graphique connu sous le nom de GRAFCET (GRAphe Fonctionnel de Commande Etape et Transition).

La description d'un automatisme peut être généralement étalée sur deux niveaux :

Niveau 1 : Description fonctionnelle

A ce niveau ce sont les fonctions, les informations et les commandes impliquées dans l'automatisme du processus, qui doivent être définies d'une façon la plus précise possible. C'est ainsi que le comportement de l'automatisme envers la partie opérative sera décrit sans préjuger en aucune façon les technologies employées dans les deux parties.

Niveau 2 : Description technologique et opérationnelle

a- Spécification technologique

Pour les spécifications technologiques, des détails viennent s'ajouter pour compléter la description du niveau 1 afin de préciser comment cette dernière sera réalisée pratiquement. Par conséquent, le circuit d'interface entre la commande et le processus à automatiser sera défini et les éléments qui le constituent seront donnés tels que les capteurs, les actionneurs, les compteurs…..Sans pour autant oublier la partie d'interface qui permet la communication avec l'opérateur.

b- Spécification opérationnelle

Cette étape concerne l'après réalisation de l'automatisme. A ce niveau, seule l'exploitation est concernée en donnant le maximum d'information sur les caractéristiques du système afin de rendre son exploitation plus facile. Ces informations concernent en général la fiabilité, la maintenance, la possibilité de modification, le mode d'utilisation…..

II-6-2. Définitions

Un grafcet est la description graphique des séquences de fonctionnement d'un automatisme spécifié par un cahier des charges. Ce graphe, appelé aussi diagramme, est constitué par différentes étapes interconnectées par l'intermédiaire d'arcs ou de liaisons dans un ordre bien déterminé à travers des transitions qui valident ou non leur exécution. On peut donc définir un grafcet comme un organigramme de fonctionnement d'un automatisme. Il est généralement décrit par : Etape-Transition-Arc auxquels sont associés Action et Réceptivité.

Etape : Elle définit l'état ou la situation où le comportement d'une partie d'un système ou sa totalité reste invariant par rapport à ses entrées et ses sorties.

Action : A chaque étape sont associées des actions qui par leur exécution la situation du système sera caractérisée. Cependant, une action ne peut être exécutée que si l'étape à laquelle elle est associée est active.

Transition : Le passage entre deux étapes successives est conditionné par le franchissement d'une transition. Une transition caractérise la possibilité de passer d'une étape à une autre par désactiver la première et activer la seconde.

Réceptivité : A chaque transition est associée une fonction combinatoire logique appelée réceptivité dont la validation est conditionnée par l'activation de toutes les étapes qui juste la précèdent. C'est à cette condition que la transition à laquelle est associée cette réceptivité est dite franchissable..

Liaison : Dans un grafcet les étapes sont interconnectées par des liaisons orientées qui indiquent l'ordre d'exécution des séquences de l'automatisme.

II-6-3. Représentation d'un grafcet

C'est l'Association française pour la Cybernétique et Technique (AFCET) qui a dégagé la représentation Grafcet comme une synthèse de tous les outils existants. Au début une notation originale lui a été attribuée par l'AFCET qui a été juste après remplacée par une représentation normalisée qui est introduite par l'Agence nationale pour le Développement de la Production Automatisée (ADEPA).

Représentation originale :

- L'étape est représentée par un cercle avec un nombre placé en son centre indiquant le numéro de l'étape. L'action associée à l'étape est écrite sur le côté droit du cercle

comme montré par la figure 30. Des fois un point est placé à l'intérieur du cercle pour indiquer que l'étape correspondante est active.

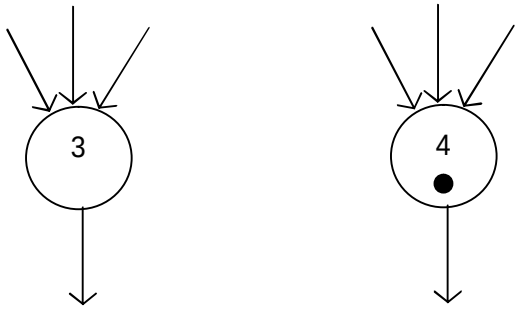


Figure 30 : Représentation d'une étape dans la notation originale

- La transition est représentée par un trait horizontale sur lequel arrive une ou plusieurs liaisons et à partir duquel part une ou plusieurs d'autres liaisons. Un nombre sera placé à droite de la transition indiquant son numéro comme montré sur la figure 31. On trouve aussi sur la droite de chaque transition une expression logique qui définit la réceptivité ; elle correspond à la condition à satisfaire pour que le cycle puisse évoluer.

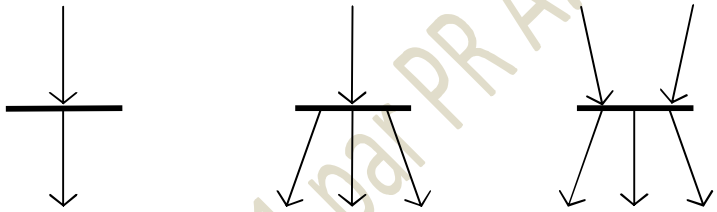


Figure 31 Représentation d'une transition dans la notation originale

- Les liaisons orientées servent à connecter à travers les transitions les étapes qui se suivent. Les flèches indiquent le sens d'évolution du grafcet comme on peut le constater sur les figures 30 et 31.

Représentation normalisée :

Seul le graphisme opté dans cette représentation prétend à une normalisation officielle ; il est donc indispensable de le connaître quand on veut toucher au domaine de l'automatisme. Par conséquent, seule cette représentation sera utilisée dans le reste de ce document.

- L'étape dans la notation normalisée est représentée par un carré dont un nombre se trouvant à l'intérieur indique le numéro de l'étape. L'action à entreprendre associée à une étape sera écrite à l'intérieur d'un rectangle se trouvant à droite de l'étape et connecté avec cette dernière par un arc. La présence d'un point à l'intérieur du carré d'une étape indique l'état actif de cette étape un exemple d'étapes est illustré par la figure 32.

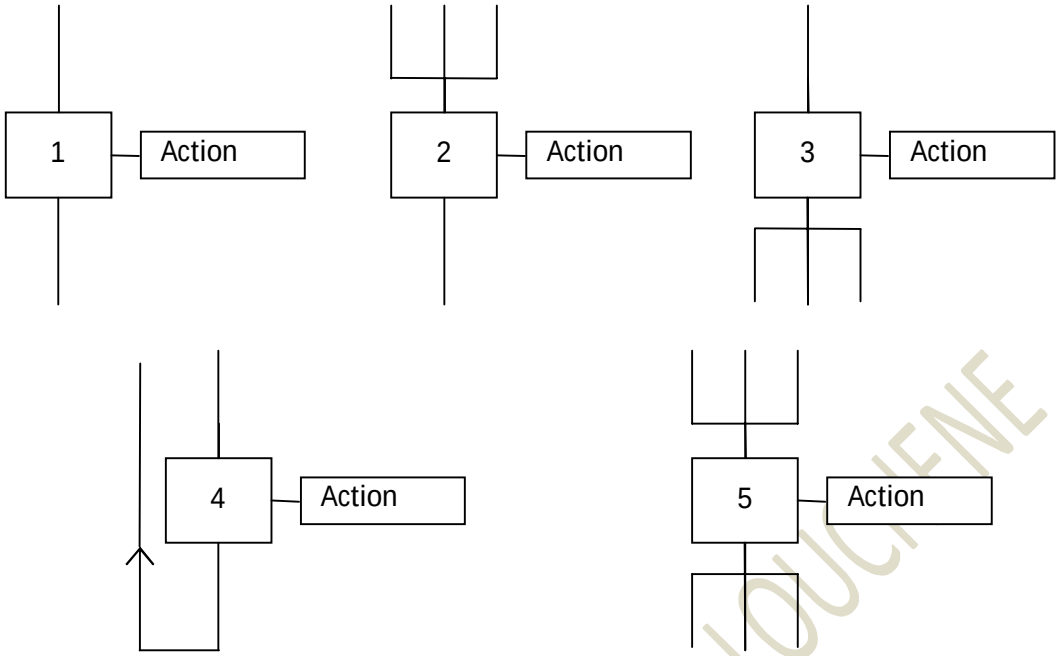


Figure 32 : Exemple d'étapes dans la notation normalisée

- La transition est maintenue identique à celle donnée dans la représentation originale, la seule différence entre les deux représentations est que dans le cas où plusieurs liaisons arrivent ou partent on peut distinguer la convergence et la divergence pour le ET ou le OU logique, comme il est illustré par la figure 33.

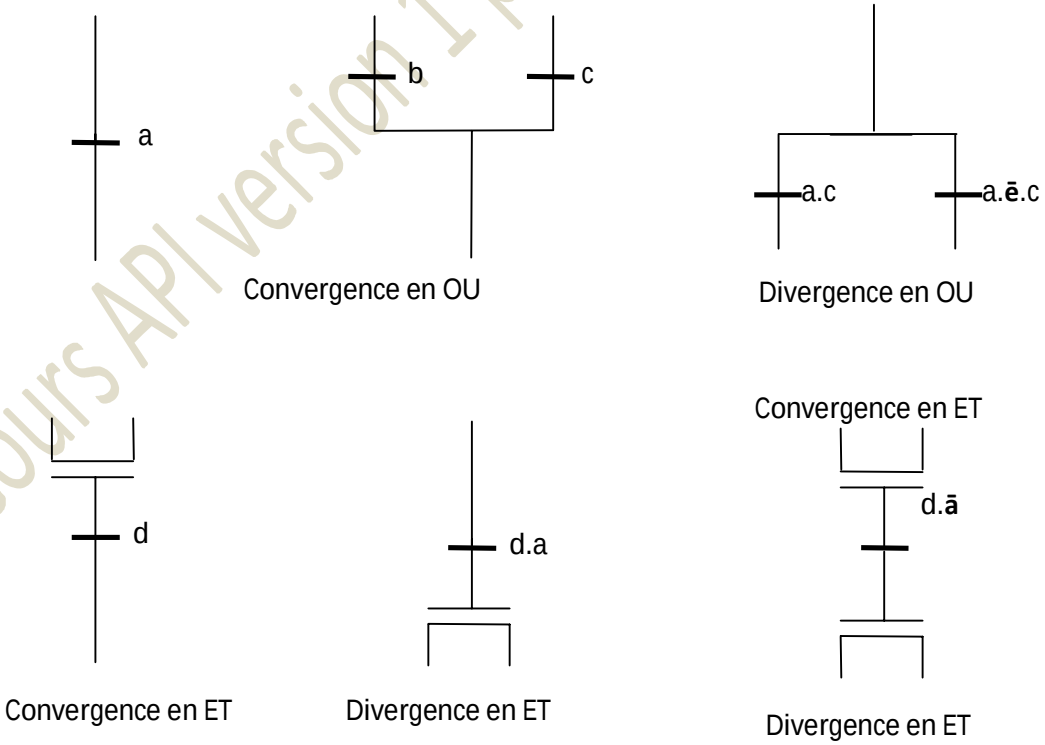


Figure 33 : Exemple de transition dans la notation normalisée.

II-6-4. Evolution d'un grafcet

L'évolution d'un grafcet se fait toujours du haut vers le bas, c'est pour cette raison que les liaisons ne sont pas encombrées de flèches. Dans le cas de saut ou de retour vers le haut des flèches devront être ajoutées.

L'évolution d'un grafcet obéit à cinq importantes règles :

Règle 1 : Etape d'initialisation. L'initialisation précise les étapes qui sont actives au début de chaque fonctionnement. Ce sont des étapes qui marquent le commencement de l'évolution d'un grafcet. Une étape d'initialisation est inconditionnellement activée et elle est représentée dans un grafcet par un carré dont les cotés sont en double trait comme il est montré par la figure 34.

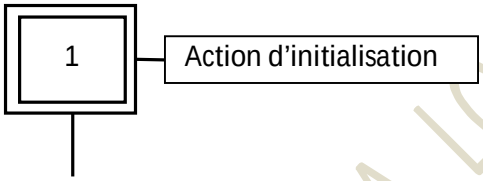


Figure 34 : Etape d'initialisation

Règle 2 : Validation d'une transition. Une transition peut être dans deux situations validée ou non validée. Une transition est dite validée si et seulement si toutes les étapes qui juste la précèdent sont actives, dans le cas contraire elle est dite non validée tel qu'il est montré dans la figure 35. Pour activer l'étape ou les étapes qui suivent immédiatement une transition, cette transition doit être franchissable. Le franchissement d'une transition est conditionnée par :

- Sa validation.
- ET que la réceptivité qui lui est associée soit vraie

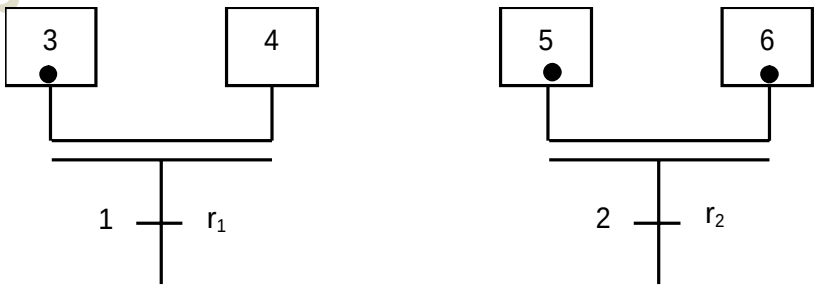


Figure 35 : La transition 1 non franchissable. La transition 2 franchissable

Règle 3 : Franchissement d'une transition. Une transition validée est franchissable, elle sera franchie si la réceptivité qui lui est associée est vraie. Le franchissement d'une transition active ses étapes de sortie et désactive

ses étapes d'entrée comme montré par la figure 36 où après franchissement de la transition 1 les étapes 3, 4 et 5 sont activées alors que les étapes 1 et 2 sont inactives.

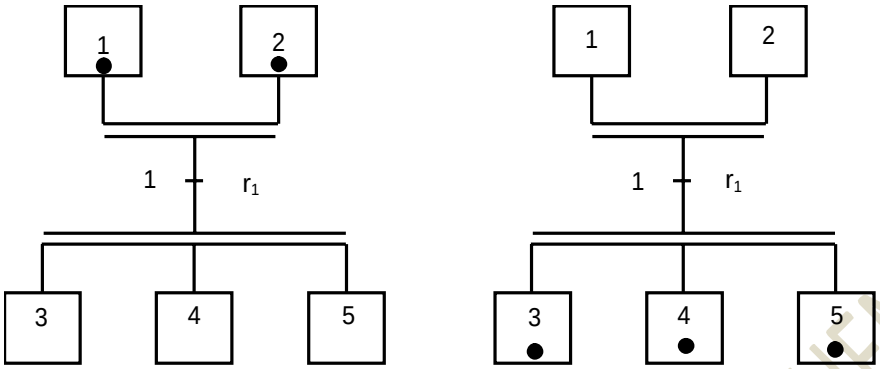
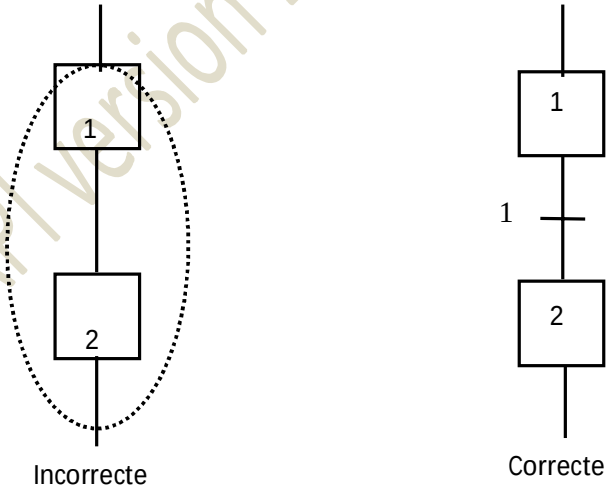


Figure 36 : Exemple illustrant le franchissement d'une transition

Règle 4 : Plusieurs transitions qui sont simultanément franchissables seront simultanément franchies.

Règle 5 : La désactivation et l'activation simultanée d'une étape entraîne son activation continue.

Règle 6 : Deux symboles de même nature ne doivent jamais se suivre. L'alternance étape-transition doit être respectée. La figure 37 montre des cas de notation non permis et leurs correspondants permis.



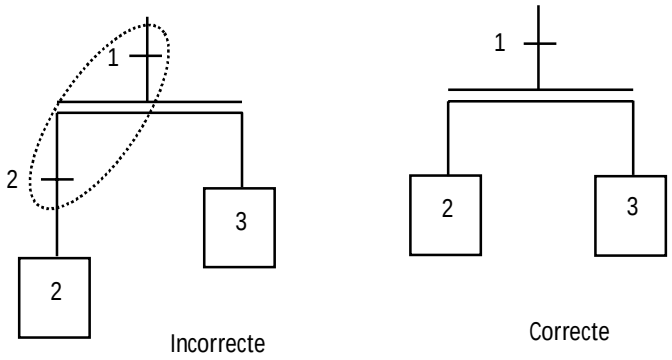


Figure 37 : Exemples de notations non permises

Règle 7 : Les regroupements de liaisons, simple trait et double traits, ne peuvent ni être isolés, ni être commun à plusieurs symboles.

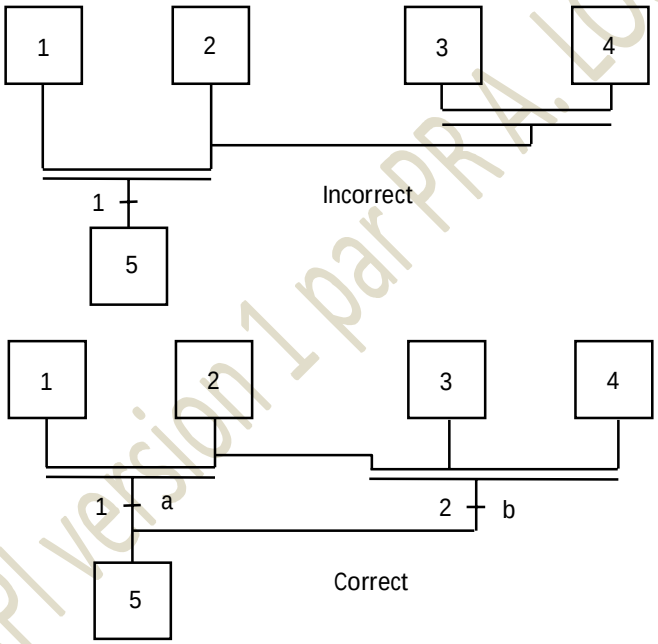


Figure 38-a : Exemple de regroupement isolé

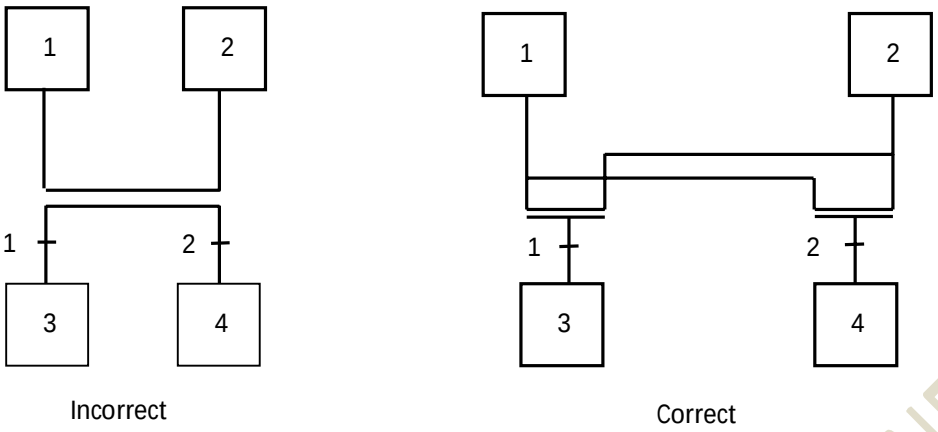


Figure 38 – b : Exemple de regroupement commun à plusieurs symboles

II-6-5. Synchronisation entre grafcet

Dans le cas de systèmes complexes, leur fonctionnement peut être décrit par plus d'un grafcet qui doivent être synchronisés dans certains cas. La synchronisation entre grafcet est obtenue en faisant intervenir l'état actif ou inactif d'étapes de l'un des grafcet comme réceptivités de transition dans l'autre grafcet. Prenant l'exemple de la figure 39 où le concepteur veut que les deux grafcet commencent à évoluer en même temps, dans ce cas l'état x1 de l'étape 1 du premier grafcet intervient dans la réceptivité 4 du deuxième grafcet au moment où l'état x4 de l'étape 4 du deuxième grafcet intervient dans la réceptivité 4 du premier grafcet.

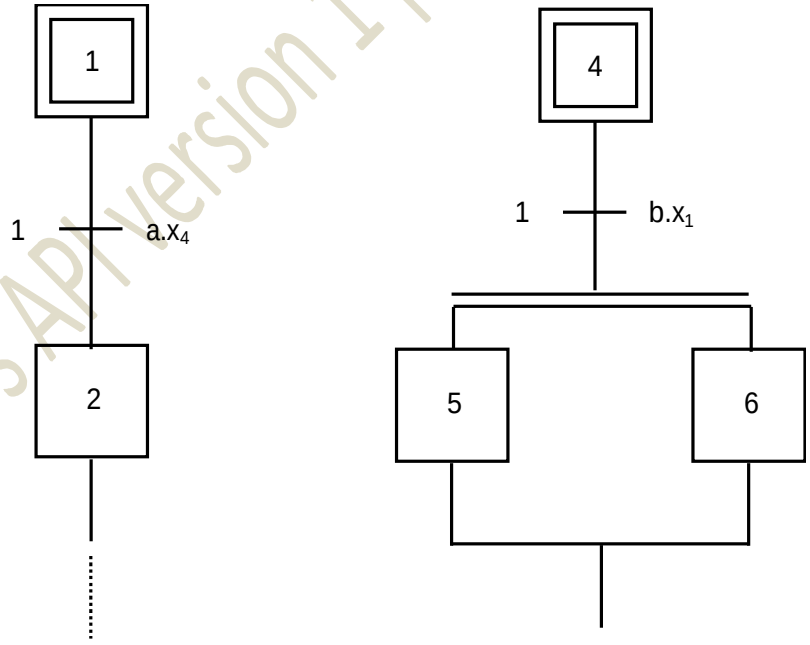


Figure-39 : Exemple de synchronisation entre grafcet

Si sur un même grafcet on trouve l'état de certaines étapes intervient comme condition de franchissement au niveau des réceptivités de quelques transitions du même grafcet bien sur, on dit que ce grafcet est auto-synchronisé.

II-6-6. Mise en équation d'un grafcet

En matérialisant une étape i par une variable logique E_i et la réceptivité de la transition qui la suit par la variable R_i , suivant les règles d'évolution d'un grafcet nous obtenons le tableau qui résume la partie d'un grafcet montré par la figure 40.

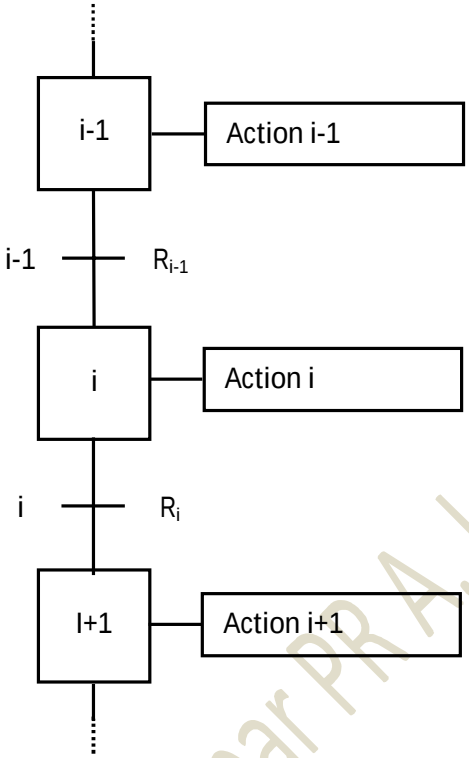


Figure-40 : partie d'un grafcet pour mise en équation

La variable logique E_i associée à l'étape i affiche son état :

E_i=1 → l'étape i est active

E_i=0 → l'étape i est inactive

Par analogie on obtient pour le reste du grafcet de la figure 40 :

E_{i-1}=1 → l'étape i-1 est active

E_{i-1}=0 → l'étape i-1 est inactive

E_{i+1}=1 → l'étape i+1 est active

E_{i+1}=0 → l'étape i+1 est inactive

Etape	Condition d'activation	Condition de désactivation
i	E_{i-1}active ET R_{i-1} vraie Donc $E_i = E_{i-1}\, R_{i-1}$	E_{i+1} active ce qui correspond à

Etape i active E_i=1

Etape i désactive E_i=0

- Condition d'activation d'une étape i

$$AE_i = E_{i-1}.\, R_{i-1}$$

- Condition de désactivation d'une étape i

$DE_i = AE_{i+1} = E_{i+1}$

A l'instant (t-dt)			A l'insant (t)
E_i	AE_i	DE_i	Ei
0	0	0	0
0	1	0	1
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1
1	0	1	0
1	1	1	1

E_i \ AE_i DE_i	00	01	11	10
0	0	0	1	1
1	1	0	1	1

Ce qui permet d'écrire l'évolution de l'étape i par :

$$E_i = AE_i + \overline{DE_i} \bullet E_i$$

ou

$$E_i = E_{i-1} \bullet R_{i-1} + \overline{E_{i+1}} \bullet E_i$$