République Algérienne Démocratique et Populaire
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Ecole Nationale Polytechnique
Laboratoire de Commande des Processus
Département du Génie Electrique
Spécialité Automatique

PROJET DE FIN D'ETUDES EN VUE DE L'OBTENTION

DU DIPLOME D'INGENIEUR D'ETAT

EN AUTOMATIQUE

Thème

SIMULATION ET COMMANDE EN TEMPS REEL DES SYSTEMES CONTINUS ET DISCONTINUS PAR AUTOMATES PROGRAMMABLES APPLICATIONS SUR AUTOMATE SIEMENS

Présenté par :

Adlane ACHAB et Fateh MEFTOUT

Proposé et dirigé par :

Pr. El Madjid BERKOUK

Ecole Nationale Polytechnique
10, Avenue Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie

Je dédie ce modeste travail tout d'abord à mon père qui nous a tant donné sans jamais rien demander

A ma mère sans laquelle je ne serais pas ce que je suis.

A mes frères et à ma sœur qui ont su m'épauler dans les moments difficiles

A mon oncle El Hadj (tu resteras dans la mémoire de tous)

A Mista et Oussama, merci pour tout cousins

Ainsi qu'a toute ma famille

Dédicace spéciale

A Karim, vive les allemands

A Ouahab qui devrait suivre un régime

A Massi, arrête de faire du sport !! Tu ne me rattraperas jamais !!

A Brahim, Farid et les autres, y a pas que le foot !!

A Latifa, Ouahiba et à toute sa famille

A Salimo, toto et massi, vive ouled les verts

A Fateh (binomi) Moh, Oumar, Kellou, Saleh, Mehdi…. Toute ma promo

A Miroine , Didan et tous mes amis de Rouiba , vivent les ENPEIST !!

A Idir, El Hadi, Naim et Toufik, ça fait déjà 6 ans !!

Merci à toutes les personnes qui m'ont aidé de près ou de loin

Adlane ACHAB

Je dedie ce modeste travail :

A ma tres chere Mere, pour ses sacrifices depuis qu'elle m'a mis au monde, et qui n'a pas cesse de m'encourager, de me soutenir dans les moments difficiles et qui a su m'entourer de toute son affection et son amour pour que je puisse reussir.

A mon Pere, qui m'a toujours soutenu et aide a affronter les difficultes.

A mes frères Khelaf et sa femme, Brahim et sa fiancée, Samir et Mohamed et ma très chère sœur.

A toute ma famille.

A mon binome et frere Adlane ainsi que toute sa famille.

A tous mes amis en particulier :

MASSI, MOUH, DIDAN, KARIM, MEROUANE, OMAR et ALAEDINE.

A tous mes amis de l'ENP.

A tous ceux qui me sont chers.

Fateh MEFTOUT

Remerciements

Nous remercions en premier lieu Mr E.M. BERKOUK, notre encadreur, pour son aide et disponibilité tout au long de ce projet.

Nous remercions Mr FAKHAR, de nous avoir accueillis, ainsi que pour son aide et son assistance.

Nous remercions les membres du jury, qui ont eu l'amabilité d'examiner ce document et d'évaluer son contenu.

Nos sincères remerciements à tous nos professeurs de l'Ecole Nationale Polytechnique, ainsi que ceux de l'Ecole Nationale Préparatoire aux Etudes d'Ingénieur pour l'enseignement et les connaissances qu'ils nous ont apportés

Merci à toutes et à tous

Adlane ACHAB

Fateh MEFTOUT

Sommaire

Chapitre III: Applications

Section A: Simulation numérique

Section B: Commande des systèmes continus

Sommaire des Tableau

Sommaire des figures

INTRODUCTION

INTRODUCTION

Le développement massif des techniques de l'automatisme a permis le passage de la machine automatisée à celui des systèmes automatisés de production, qui gèrent l'alimentation en énergie et qui permettent d'avoir une meilleur qualité des produits en plus de la sécurité et de la flexibilité des processus, mais cela entraîne un accroissement des besoins, en particulier la manipulation d'un grand nombre de variables et la gestion de véritables flux de communication.

Cela explique que les systèmes câblés deviennent trop volumineux et trop rigides pour de telles applications, et que l'on se tourne donc vers des solutions utilisant les techniques de traitement de l'information par processeurs programmables ou les automates programmables industriels occupent une place de choix.

Un automate programmable est un système électronique, destiné à être utilisé dans un environnement industriel, il 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 arithmétique, pour commander au moyen d'entrées et de sorties divers types de machines ou de processus. [B02]

Les API (ou Programmable Logic Controller PLC) sont aujourd'hui les constituants les plus répandus des automatismes. On les trouve non seulement dans tous les secteurs de l'industrie, mais aussi dans les services (gestion de parkings, d'accès à des bâtiments, contrôle du chauffage, de l'éclairage, de la sécurité ou des alarmes) et dans l'agriculture (composition et délivrance de rations alimentaires dans les élevages) et cela est due surtout à leurs performances de sécurité.

Dans tous les secteurs ou ils sont utilisés les API doivent remplir des taches de commande en élaborant des actions suivant une algorithmique appropriée, à partir des informations données par des détecteurs (Tout ou Rien) ou des capteurs (analogiques ou numériques), et des taches de communication avec des opérateurs humains ou avec d'autres processus (automates, calculateurs de gestion de production,…).

Bien que les automates programmables inclus des modules et des options permettant le contrôle ou la commande d'un système continu, beaucoup d'utilisateurs pensent que les APIs ne sont destinées qu'à la commande séquentielle.

Le but de ce travail est de faire la mise en œuvre d'automates programmables industriels de la famille SIMATIC S7-300 pour la commande, la simulation et la régulation de systèmes continues.

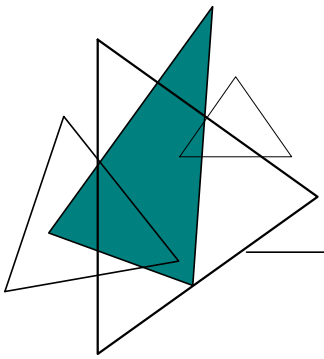
Trois applications principales seront présentées :

- Simulation numérique et résolutions des équations différentielles.
- Régulation de la température d'un four électrique (régulateur PID & régulation à hystérésis).
- Commande d'un moteur à courant continue avec boucle de mesure et de traitement (commande en boucle ouverte).

Ce document englobe l'étude du logiciel de programmation STEP7, des automates utilisés ainsi que l'implémentation de quelques applications.

CHAPITRE I

LES AUTOMATES PROGRAMMABLES INDUSTRIELS



Section A

GENERALITES SUR LES AUTOMATES PROGRAMMABLES

I.A.1 Architecture générale

En général un automate programmable se constitue essentiellement d'une unité centrale, un module d'entrées/sorties, un module d'alimentation, un module de communication et des auxiliaires.

Figure I.A.1 Architecture d'un Automate Programmable

I. A.1.1 Le module d'alimentation "PS"

Il est composé de blocs qui permettent de fournir à l'automate l'énergie nécessaire à son fonctionnement, il convertit la tension du réseau (AC 220 V) en tension de service (DC 24V, 12V ou 5V) et assure l'alimentation de l'automate ainsi que circuits de charge.

Un voyant est positionné en générale sur la façade pour indiquer la mise sous tension de l'automate.

I. A.1.2 L'unité centrale "CPU"

La CPU est une carte électronique bâtie autour d'un ou plusieurs processeurs, elle comprend aussi des moyens de stockage, qui sert à sauvegarder les programmes et les données.

I. A.1.2.1 Le processeur

Le processeur est chargé d'exécuter le programme utilisateur, il doit assurer des opérations logiques et arithmétiques ainsi que des fonctions de temporisation et du comptage. Il peut être issu de la technologie câblée ou de la technologie à microprocesseur.

En général un microprocesseur est composé d'une Unité Arithmétique et Logique (UAL), de registres, un Décodeur d'Instructions, un Compteur Programme et une horloge [B01].

I. A.1.2.2 La mémoire

La mémoire est l'élément fonctionnel qui peut recevoir, conserver et restituer l'information. Elle est découpée en zones :

- une zone mémoire programme,
- une zone mémoire donnée.
- une autre pour les variables internes.

Pour un automate, il faut connaître la capacité mémoire minimale utile et la capacité maximale que l'on peut obtenir par diverses extensions.

I. A.1.3 Le module d'entrées/sorties "SM"

Le module E/S assure le rôle d'interface pour la partie commande, qui distingue une partie opérative (les sorties), où les actionneurs agissent physiquement sur le processus, et une partie d'acquisitions (les entrées) récupérant les informations sur l'état de ce processus et coordonnant en conséquence les actions pour atteindre les objectifs prescrits (matérialisés par des consignes).

En plus d'assurer la communication entre la CPU et les organes externes, le module d'E/S doit garantir une protection contre les parasites électriques, c'est pour quoi la plus part des modules E/S font appel au découplage optoélectronique.

Il existe deux types d'interface E/S:

- **Le module E/S Tout Ou Rien (TOR) :** Permet de raccordé l'automate à des capteurs TOR (boutons poussoirs, fins de course, capteurs de proximité, capteurs photoélectriques …) ou à des pré-actionneurs (vannes, contacteurs, voyant pneumatique, électrovannes, relais de puissance, LED….). L'état de chaque entrée ou sortie est visualisé par une diode électroluminescente. Le nombre d'entrées sur une carte est de : 4, 8, 16, 32.
- **Le module E/S analogique :** Permet de traiter les signaux analogiques. Il est muni d'un convertisseur analogique/numérique pour les entrées et un autre numérique/analogique pour les sorties. Il existe des modules à 2, 4, 8 voies.

I. A.1.4 Le module de fonction "FM" (Les cartes spécialisés)

Le module de fonction ou «Function Module » est un module additionnel ou des cartes spécialisées peuvent être connectés. Ces cartes comportent un processeur spécifique ou une carte

électronique spécialisée, elles assurent non seulement la liaison avec le monde extérieur mais aussi une partie du traitement pour soulager le processeur. On peut citer : les cartes d'axe, les concentrateurs de communication, les cartes E/S déportées, les cartes de comptage rapide, les cartes de pesage, les cartes de régulations PID…

I. A.1.5 Le module de communication "CM"

Le module de communication comprend les consoles et les boîtiers de tests.

I. A.1.5.1 Les consoles

Les consoles permettent la programmation, le paramétrage et les relevés d'informations, ils peuvent également afficher le résultat de l'autotest comprenant l'état des modules d'entrées et de sorties, l'état de la mémoire, de la batterie, etc. Ils sont équipés (pour la plupart) d'un écran à cristaux liquides.

Pendant la phase de programmation les consoles permettent : l'écriture, la modification, l'effacement et le transfert d'un programme dans la mémoire de l'automate ou dans une mémoire EPROM.

Pendant la phase de réglage et d'exploitation elles permettent : de visualiser ou d'exécuter le programme pas à pas, de forcer ou de modifier les données (les entrées, les sorties, les bits internes, les registres de temporisation, les compteurs…).

Certaines consoles ne peuvent être utilisées que connectées à un automate car c'est ce dernier qui leurs fournit l'alimentation et la mémoire de travail, c'est les consoles de programmation On-line, avec ces consoles le programme introduit par l'utilisateur est directement mémorisé dans l'automate.

D'autres consoles peuvent fonctionner de manière autonome grâce à leurs mémoires interne et à leurs alimentations, c'est les consoles de programmation Offline, elles offrent un plus grand confort, le programme écrit de cette façon est appelé source, il est compilé par la console puis transféré dans la mémoire de l'automate. [B03]

I. A.1.5.2 Les boîtiers de tests

Les boîtiers de testes quand a eux sont destinées aux personnels d'entretien, ils permettent de visualiser le programme ou les valeurs des paramètres (affichage de la ligne de programme à contrôler, visualisation de l'état des entrées et des sorties…)

I. A.1.6 Les auxiliaires

Il s'agit principalement de :

- Un support mécanique (un rack) : l'automate se présentant alors sous forme d'un ensemble de cartes, d'une armoire, d'une grille et des fixations correspondantes.
- Un ventilateur : il est indispensable dans les châssis comportant de nombreux modules ou dans le cas où la température ambiante est susceptible de devenir assez élevée (plus de 40 °C).
- Un indicateurs d'état : il indique la présence de tension, l'exécution du programme (mode RUN), la charge de la batterie, le bon fonctionnement des coupleurs.

Conclusion

L'automate programmable est un composant électronique adapté aux conditions industrielles. Il intègre des fonctions et des modules qui facilitent son interaction avec l'environnement extérieur. Vu le nombre important de marques disponibles, Le choix d'un automate programmable est en premier lieu le choix d'une société ou d'un groupe, les grandes sociétés privilégieront deux fabricants pour faire jouer la concurrence et pouvoir "se retourner" en cas de "perte de vitesse" de l'une d'entre elles.

Pour des applications de commande de systèmes séquentiels ou discrets un automate utilisant des langages de programmation de type GRAFCET est préférable pour assurer les mises au point et dépannages dans les meilleures conditions, des outils permettant une simulation des programmes sont également souhaitables.

Il faut ensuite quantifier les besoins :

- Nombre et types d'entrées / sorties : tension de sorties des capteurs et tension d'entrées des actionneurs ainsi que le nombre de voies nécessaire.
- Type de CPU : la taille mémoire, la vitesse de traitement et les fonctions spéciales intégrés.
- Fonctions ou modules spéciaux : certaines cartes (commande d'axe, pesage ...) permettront de soulager le processeur et devront offrir les caractéristiques souhaitées (résolution, ...).

- Fonctions de communication : l'automate doit pouvoir communiquer avec les autres systèmes de commande (API, supervision ...) et offrir des possibilités de communication avec des standards normalisés (Profibus ...).
- Alimentation : Elle doit couvrir les besoins énergétiques de l'unité centrale et de toutes les extensions. Quand elle existe sur l'automate de base, elle ne couvre pas les besoins d'un nombre important d'extension et il faudra rajouter une deuxième alimentation.

I. A.2 Les langages de programmation

I. A.2.1 Introduction

La norme IEC 1131-3 (la Commission Électrotechnique Internationale) définit cinq langages qui peuvent être utilisés pour la programmation des automates programmables industriels. Ces langages peuvent être divisés en deux grandes catégories :

- **Langages graphiques:**

- SFC « Sequential Funiculite Chart » ou GRAFCET
- LD « Ladder Diagram »ou schéma à relais
- FBD « Function Block Diagram »ou schéma par bloc

- **Langages textuels:**

- ST « structured text » ou texte structuré
- IL « Instruction List » ou Liste d'instructions

I. A.2.2 Les langages graphiques

I. A.2.2.1 Le GRAFCET

L'acronyme GRAFCET signifie: GRAphe Fonctionnel de Commande Etape Transition (SFC Sequential Function Chart). C'est une méthode de représentation graphique permettant de décrire le cahier de charge d'un automatisme. Il est adapté aux systèmes à évolution séquentielle, il est défini par un ensemble d'éléments graphiques de base traduisant le comportement de la partie commande vis-à-vis de ses entrées et de ses sorties.

Un programme GRAFCET décrit un procédé comme une suite d'étapes, reliées entre elles par des transitions. À chaque transition est associée une réceptivité, celle-ci est une condition logique

qui doit être vraie pour franchir la transition et passer à l'étape suivante. Des actions sont associées aux étapes du programme.

Le format graphique d'un programme GRAFCET est le suivant :

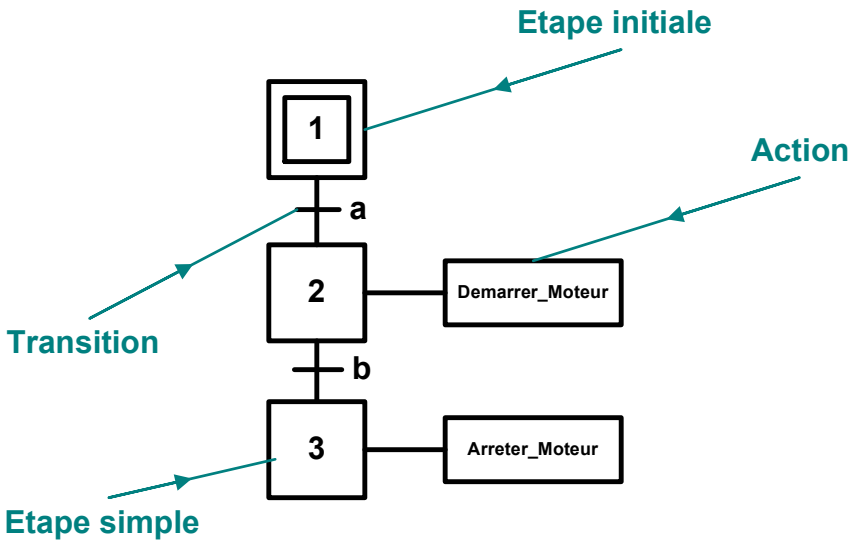


Figure I.A.2.1 Le GRAFCET

Une étape représentée par un carré qui a un numéro identificateur et les actions associées sont indiquées dans un rectangle relié à la partie droite du carré; (l'étape initiale est représentée par un carré double).

Une liaison orientée représentée par une ligne, parcourue par défaut de haut en bas ou de gauche à droite.

Une transition entre deux étapes à qui est associé une réceptivité inscrite a ça droite, est représentée par une barre perpendiculaire aux liaisons orientées qui relient ces étapes.

I. A.2.2.2 Ladder Diagram :

Le LD est une représentation graphique qui traduit directement des équations booléennes en un circuit électrique en combinant des contacts et des relais a l'aide des connexions horizontales et verticales, les contacts représentent les entrées (contact normalement ouvert, contact normalement fermé,…) et les relais représentent les sorties (relais directs, relais inversés,…), les diagrammes LD sont limités sur la gauche par une barre d'alimentation et par la masse sur la droite.

Par exemple pour réaliser la fonction logique $x = (a + b)(\bar{c} + \bar{d}e)$, voici le programme LD :

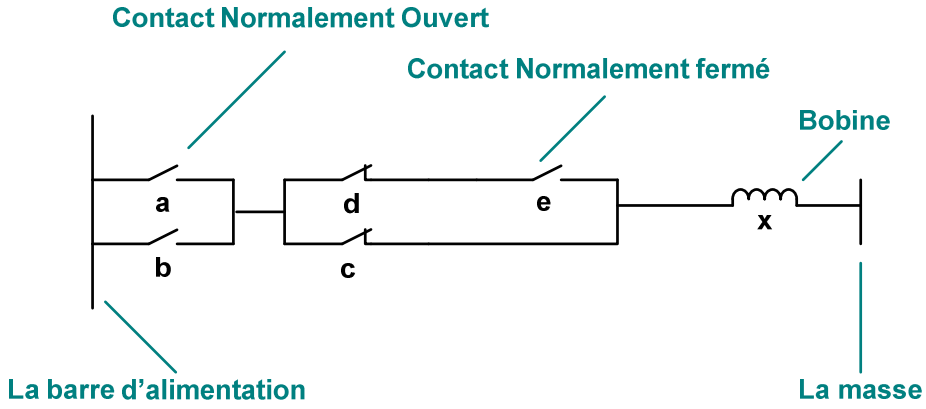


Figure I.A.2.2 Exemple 1 d'un programme LADDER

Le langage LD propose d'autre type de fonction tel que les fonctions de comptages et de temporisations, les fonctions arithmétiques et logiques, les fonctions de comparaison et de transfert. Par exemple pour réaliser la fonction $Z = (X \geq Y)$, on utilise directement la fonction déjà disponible.

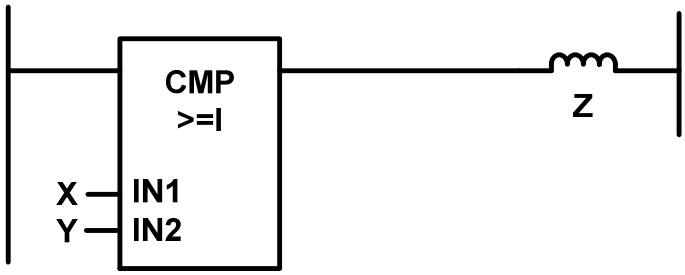


Figure I.A.2.3 Exemple 2 d'un programme LADDER

I. A.2.2.3 Bloc de Fonction

C'est un langage graphique qui permet la construction d'équations complexes à partir des opérateurs standard, ou de blocs fonctionnels, il se compose de réseaux de fonctions préprogrammées ou non, représentées par des rectangles qui sont connectés entre eux par des lignes.

La programmation avec le FBD est très souple et facile à apprendre, la plupart des fonctions nécessaires (les fonctions arithmétique et logique, les fonctions de temporisation, des blocs

fonctionnels PID…) sont déjà disponible dans la bibliothèque, il suffit juste de les connecter et bien paramétrer les entrées et les sorties, c'est-à-dire respecter le type des variables lors de la connexion.

Par exemple pour réaliser l'opération arithmétique $W = \frac{20(X+Y)}{Z}$ on aura besoin de trois blocs : un pour addition, un pour la multiplication et un autre pour la division.

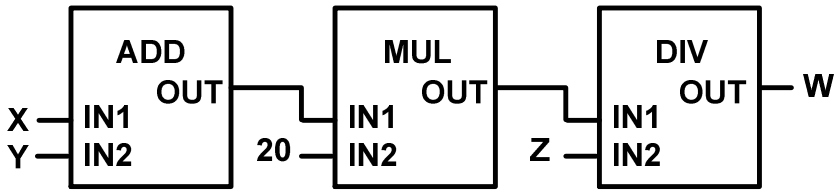


Figure I.A.2.4 Exemple d'un programme en Fonction Bloc

I. A.2.3 Les langages textuels

I. A.2.3.1 Texte Structuré

Le langage ST (Structured Text) est un langage de programmation textuel de haut niveau dédié aux applications d'automatisation, il est utilisé principalement pour décrire les procédures complexes et difficilement modélisables avec les langages graphiques, il peut aussi être utilisé en tant que sous programme avec d'autre langage de programmation.

Il utilise les même énoncés que les langages de programmation de haut niveau (Pascal, C, C++…) comme: les assignations, les appels de fonction, les énoncés de contrôle (IF, THEN, ELSE, CASE) ou d'itération (FOR, WHILE, REPEAT) en plus des opérations arithmétiques et logiques.

Exemple : calcul de l'écart entre deux points dans un plan cartésien.

```
FUNCTION  Ecart: REAL
VAR_INPUT
X1, X2, Y1, Y2 : REAL;
END_VAR
BEGIN
RESULT := SQRT((X1-X2)^2 + (Y1-Y2)^2);
END_FUNCTION
```

Figure I.A.2.5 Exemple d'un programmation en Langage Structuré

I. A.2.3.2 Liste d'instructions

Le langage IL est un langage textuel de bas niveau (proche du langage machine), qui utilise un jeu d'instruction simple, il trouve sa puissance dans les applications de petites tailles, et dans la création de sous programme ou procédure, car il permet un contrôle totale et une optimisation parfaite du code, par contre dans les grandes applications il est très difficile de programmer avec le IL, les programmes dans ce langage peuvent être traduit ou déduit des autres langages.

Le IL a la même structure que l'assembleur, il utilise un ou plusieurs registres de travail. Les valeurs intermédiaires nécessaires pour l'exécution d'une instruction donnée seront mémorisées dans ces registres le temps de leurs utilisations et il possède un jeu d'instruction d'assez riche pour décrire toutes les opérations arithmétiques et logiques, les opérations de comptage et temporisation, la comparaison et le transfert…

Par exemple pour réaliser l'opération $W = \frac{20(X+Y)}{Z}$ on utilise le code suivant :

```
LD   X
LD   Y
+R
LD   +20
*R
LD   Z
/R
ST   W
```

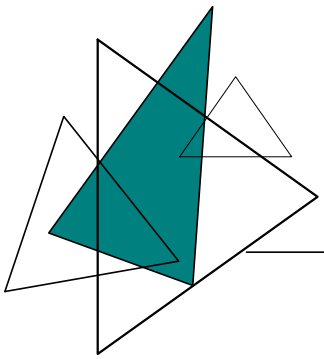
Figure I.A.2.6 Exemple d'un programme en IL

Conclusion

La plus part des grands constructeurs d'automates programmables, fournissent des logiciels de configuration et de programmation munis des langages SFC, LD, FBD, ST et IL.

Le choix d'un langage s'appuyé sur la complexité de l'application et de la tache de commande. Il est préférable d'utilisé les langages graphiques (SFC, LD et FBD) pour la réalisation des programmes de commande séquentielles. Le SFC est la réalisation direct d'un GRAFCET de commande, les langages LD et FBD sont plus utiles pour les opérations combinatoires sur bits ou mots.

Les langages textuels sont beaucoup plus performent pour le traitement de variables continues ou analogiques ainsi que pour la commande des systèmes continus. Les programmes en IL sont un peu fastidieux à mettre en œuvre, mais connaisse une optimisation optimale pour le temps de traitement et l'occupation de la mémoire. Le ST est le langage par excellence, très utiles pour des utilisateurs ayant des connaissances en langages évolués tel que PASCAL.



Section B

DESCRIPTION DE L'AUTOMATE S7-313

I.B. Description de l'automate S7-313

L'automate programmable utilisé est le S7-313, c'est un automate modulaire constitué d'un module d'alimentation, une CPU, un module d'entrées TOR, un module de sorties TOR et un module d'E/S analogiques.

Figure I.B.1 S7-313

I.B.1 La CPU

Sur la plus part des CPU 300, on peut remarquer les éléments suivants: des LEDs, un commutateur, une pile, une carte mémoire et une interface MPI.

1. Commutateur du mode de fonctionnement

Position	Signification	Explications
RUN-P	RUN-PROGRAM	La CPU traite le programme utilisateur. La clé ne peut être retirée dans cette position.
RUN	Mode de marche	La CPU traite le programme utilisateur. Ce dernier ne peut être modifie.
STOP	Mode d'arrêt	La CPU ne traite aucun programme utilisateur
MRES	Effacement général	Position instable du commutateur pour effacement général de la CPU.

Tableau I.1 Positions du commutateur du mode de fonctionnement

2. **Les LEDs de visualisations**

SF (Rouge) : Défaut de matériel ou de logiciel
BATF (Rouge) : Défaillance de la pile
5V cc (Verte) : L'alimentation 5V CC et bus S7-300 est correcte
FRCE (Jaune) : Le forçage permanent est actif
RUN (Verte) : CPU en RUN ; la LED clignote à 1 Hz au démarrage ; à 0.5 Hz en mode d'attente.
STOP (Jaune) : CPU en STOP ou ATTENTE ou en démarrage ; la LED clignote en cas de demande d'effacement gén

Figure I.B.2 Les LEDs de visualisation

3. **Pile de sauvegarde :** Elle permet de sauvegarde le programme utilisateur (s'il n'est pas sauvegardé sur une carte mémoire) et de plus grandes zones de données que celles qui son rémanentes dans les blocs de données sans pile, la durée de sauvegarde des données peut attendre une année. **Nota :** La CPU peut sauvegarder une partie des données, même sans pile. Le recours à une pile n'est nécessaire que si on désire étendre la rémanence à une plus grande quantité de données.
4. **Carte mémoire** : Elle étend la mémoire de chargement de la CPU et permet de sauvegarder le programme utilisateur et les paramètres qui déterminent le comportement de la CPU et des modules. On peut également sauvegarder le système d'exploitation de la CPU sur une carte mémoire. Si le programme utilisateur a été sauvegardé sur une carte mémoire, il est conservé après une mise hors tension de la CPU même si celle-ci ne contient pas de pile de sauvegarde. Les cartes mémoires disponibles pour la 313 sont du type FEPROM 5 V (16Ko, 32Ko…4Mo).
5. **Interface MPI** : elle est utilisée pour connecté l'automate à la console de programmation PG, un pupitre opérateur OP ou pour la communication au sein d'un sous-réseau MPI. La vitesse de transmission typique (par défaut) est de 187,5 kbauds.

Caractéristique technique de la CPU

Les tableaux suivant résument les principales caractéristiques de la CPU 313

Mémoire	
Mémoire de travail intégré Mémoire de chargement intégrée RAM extensible	12 Ko 20 Ko de RAM Jusqu'à 4 Mo
Sauvegarde avec pile	Toutes les données

Sauvegarde sans pile	72 octets.
Temps de traitement (min)	
Opérations sur bit Opérations sur mot Opérations arithmétiques sur nombre entiers Opérations arithmétiques sur nombre réels	0.6 µs 2 µs 2 µs 60 µs
Compteurs	
Nombre de compteurs Plage de comptage	64 1 à 999
Temporisations	
Nombre de temporisations Plage de temps	128 10 ms à 9990 s
Mémentos	
Taille - Rémanence réglable - Par défaut Mémentos de cadence	2048 bits (256 octets) de MB 0 à MB 71 de MB 0 à MB 15 8 bits (1 octet de mémento)
Blocs de données	
Nombre Taille	127, de DB1 à DB127 (DB 0 réservé) Max 8 Ko
Blocs	
FB FC	128, de FB0 à FB127, taille max 8 Ko 128, de FC0 à FC127, taille max 8 Ko

Zones d'adresses (E/S)	
- Numérique - Analogique Mémoire image Voies numériques Voies analogiques	0.0 à 31.7/0.0 à 31.7 256 à 383/256 à 383 32 octets/32 octets Max 256/256 Max 64/32
Configuration	
Profilé-support Module par profilé-support	1 Max 8
Dimension	
L x H x P (mm^3) Poids	80 x 125 x 130 0.53 Kg
Tensions, courants	
Tension d'alimentation - Plage admissible Consommation Courant d'appel à l'enclenchement Puissance dissipée Pile - Durée de sauvegarde à 25°C	24 V cc 20,4 à 28,8 V 0.7 A 8 A 8 W min. 1 an

Programmation	
Langage de programmation	STEP7
Niveau de parenthèses	8
Blocs d'organisations	14
- Cycle libre	OB1
- Alarmes horaires	OB10
- Alarmes cycliques	OB35
- Alarmes processus	OB40
- Alarme de diagnostic	OB82
- Alarmes d'erreur asynchrones	OB80, OB81, OB85, OB86 et OB87
- Arrière-plan	OB90
- Démarrage	OB100
- Réactions aux erreurs	OB121 et OB122

Tableau I.2 Description de la CPU 313

I.B.2. Module d'alimentation

Le module d'alimentation PS 307 2 A présente les propriétés suivantes :

- courant de sortie 2 A.
- tension nominale de sortie 24 V cc, stabilisée, tenue aux courts-circuits et à la marche à vide.
- raccordement à un réseau alternatif monophasé (tension nominale d'entrée 120/230 V ca, 50/60 Hz).
- séparation de sécurité des circuits selon EN 60 950.
- peut servir de tension d'alimentation des capteurs et actionneurs.
- Signalisation de la présence d'une tension de sortie par une LED verte.
- Dimensions L x H x P (mm^3) : 50 x 125 x 120, Poids enivrent 420 g.

I.B.3. Modules d'entrées TOR

Il s'agit du SM321 DI 16x 24V cc, il contient 16 entrées Tout Ou Rien de 24 V cc munis par une séparation galvanique par groupes de 8.

Le tableau suivant résume ses caractéristiques techniques

Dimensions & Poids	
L x H x P (mm^3) Poids	40 x 125 x 117 200 g
Caractéristiques spécifiques	
Nombres d'entrées Signalisation de l'état Longueur max du câble	16 Une LED verte par voie 600 m
Caractéristiques pour la sélection d'un capteur	
Tension d'entrée - Valeur nominale - Pour signal '1' - Pour signal '0' Courant d'entrée - Pour signal '1' Temporisation d'entrée - De '0' à '1' - De '1' à '0'	 24 V cc 13 à 30 V De -30 jusqu'a +5 V 7 mA 1.2 à 4.8 ms 1.2 à 4.8 ms

Tableau I.3 Description du module d'entrée TOR SM321 DI 16x 24V cc

I.B.4. Modules de sorties TOR

Le SM322 DO 16x24 V cc comprend 16 sorties Tout Ou Rien de 24 V cc munis d'une séparation galvanique par groupes de 8, il délivre en sortie un courant de 0.5 A.

Le tableau suivant résume ses caractéristiques techniques

Dimensions & Poids	
L x H x P (mm^3) Poids	40 x 125 x 117 190 g
Caractéristiques spécifiques	
Nombres de sorties Signalisation de l'état Longueur max du câble	16 Une LED verte par voie 600 m
Caractéristiques pour la sélection d'un actionneur	
Tension de sortie - Pour signal '1' Courant de sortie - Pour signal '1' o Valeur nominale o Plage admissible - Pour signal '0' Temporisation de sortie - De '0' à '1' - De '1' à '0' Plage de résistance de charge Protection des sorties contre les courts-circuits	 24 V cc 0.5 A 5 mA à 0.6 A Max. 0.5 mA Max. 100µs Max. 500 µs 48Ω à $4K\Omega$ Oui, par hachage électronique.

Tableau I.4 Description du module de sortie TOR SM322 DO 16x 24V cc

I.B.5. Module d'entrées sorties analogiques

Le module SM 334 AI 4/AO 2 x 8/8 bits présente les propriétés suivantes :

- 4 entrées, 2 sorties
- résolution 8 bits
- étendue de mesure au choix 0 à 10 V ou 0 à 20 mA
- étendue de sortie au choix 0 à 10 V ou 0 à 20 mA
- choix libre entre tension et courant s
- ans séparation galvanique par rapport au couplage de bus interne
- séparation galvanique par rapport à la tension d'alimentation

Voici dans le tableau ci-dessous quelques spécifications techniques

Dimensions & Poids	
L x H x P (mm^3) Poids	40 x 125 x 117 285 g
Caractéristiques spécifiques	
Nombres d'entrées Nombres de sorties Signalisation de l'état Longueur max du câble	4 2 Une LED verte par voie max. 200 m
Formation des valeurs analogiques	
Résolution Temps de conversion par voie	8 bits 500 µs
Caractéristiques pour la sélection d'un capteur	
Etendue d'entrée (valeurs nominales/résistance d'entrée) - Tension - courant	 0 à 10 V/100kΩ 0 à 20 mA/50Ω

Tension d'entrée admissible	Max. 20V permanents
Limite de destruction	75 V durant max. 1s
Courant d'entrée admissible	40 mA
Caractéristiques pour la sélection d'un actionneur	
Etendue d'entrée (valeurs nominales/résistance d'entrée)	
- Tension	0 à 10 V
- courant	0 à 20 mA
Résistance de charge (dans l'étendue de sortie nominale)	
- sortie de tension	Min. 5 KΩ
o charge capacitive	Max. 1 µF
- sorties de courant	Max. 300 Ω
o charge inductive	Max. 1 mH
Tension à vide	Max. 15 V
Protection contre les courts-circuits	Oui
Limite de destruction	
- Tension	Max 15 V permanents
- Courant	max. DC 50 mA

Tableau I.5 Description du module d'E/S analogique SM 334 AI 4/AO 2 x 8/8 bits

Remarque :

1. Les entrées et sorties du module sont adressées à partir de l'adresse initiale du module. L'adresse d'une voie correspond à l'adresse de début du module plus un déport d'adresse (2 octets).
2. Les voies d'entrées non utilisées doivent être court-circuitées. Par cette mesure, on obtient une immunité optimale aux perturbations pour le module analogique.
3. Les voies de sortie libres doivent être laissées en l'air.
4. Le SM 334 AI4/AO2 x 8/8bits n'a pas d'étendues de mesure négatives.

La correspondance valeurs analogiques tension est représentée dans le tableau suivant

Système			Etendue de tension	
Pourcentage %	Décimale	Hexadécimale	Tension	Domaine
117.56 %	32511	7EFF	11.76 V	Domaine de dépassement
	27649	6C01		
100%	27648	6C00	10 V	Etendue nominale
75%	20736	5100	7.5 V	
0.0036%	1	1	361.7 µV	
0%	0	0	0 V	
	-1	FFFF	0 V	Impossible la tension min. d'entrées ou de sortie est limitée à 0 V
	-32768	8000	0 V	

Tableau I.6 Correspondance valeurs analogiques tension

La correspondance valeurs analogiques courant est représentée dans le tableau suivant

Système			Etendue de tension	
Pourcentage %	Décimale	Hexadécimale	Courant	Domaine
117.56 %	32511	7EFF	32.52 mA	Domaine de dépassement
	27649	6C01		
100%	27648	6C00	20 mA	Etendue nominale
75%	20736	5100	15 mA	
0.0036%	1	1	723.4 nA	
0%	0	0	0 mA	

	-1	FFFF	0 mA	Impossible le courant min. d'entrées ou de sortie est limitée à 0 mA.
	-32768	8000	0 mA	

Tableau I.7 Correspondance valeurs analogiques courant

CHAPITRE II

LE STEP7

II.1 Description du STEP7

STEP7 est le logiciel de base pour la programmation et la configuration de systèmes d'automatisation SIMATIC. Il permet : la création et la gestion de projets, la configuration et le paramétrage du matériel et de la communication, la gestion des mnémoniques, la création de programmes.

Il inclut 6 applications :

1. **Gestionnaire de projets SIMATIC** : il gère toutes les données relatives à un projet d'automatisation. Il démarre automatiquement les applications requises pour le traitement des données sélectionnées.
2. **Editeur de mnémoniques** : il permet de gérer toutes les variables globales. C'est à dire la définition de désignations symboliques et de commentaires pour les signaux du processus (entrées/sorties), mémentos et blocs, l'importation et l'exportation avec d'autres programmes Windows.
3. **Diagnostic du matériel** : il fournit un aperçu de l'état du système d'automatisation. Dans une représentation d'ensemble, un symbole permet de préciser pour chaque module, s'il est défaillant ou pas. De plus permet l'affichage d'informations générales sur le module et son état, l'affichage d'erreurs sur les modules de la périphérie centrale et des esclaves DP et l'affichage des messages de la mémoire tampon de diagnostic.
4. **Langages de programmation** : trois langages de programmation sont inclus dans le logiciel de base : CONT (LD Ladder Diagram), LIST (IL Instruction List) et LOG (FBD Function Bloc Diagram), d'autre langage de programmation peuvent être procurés sous forme de logiciel additionnel : le SCL (ST Structured Text) et le GRAPH (GRAFCET).
5. **Configuration matérielle** : il permet de configurer et paramétrer le matériel d'un projet d'automatisation. Il suffit juste de sélectionner le châssis (Rack) dans un catalogue électronique et leurs affecter les modules sélectionnés aux emplacements souhaités dans les racks (CPU, SM, FM…). De plus il permet le paramétrage de la CPU (comportement à la mise en route, surveillance du temps de cycle), des modules fonctionnels (FM) et de processeurs de communication (CP).
6. **NetPro** : il permet le transfert de données via MPI tout en offrant les possibilités de choisir les participants à la communication et de définir les liaisons de communication.

II.2 Création d'un projet STEP7 le pas à pas

Un projet comprend deux données essentielles, les programmes et la configuration du matériel, on peut commencer par définir l'une ou l'autre, mais tout d'abord il faut démarrer le programme SIMATIC Manager. Ce programme est l'interface graphique qui permet la manipulation du projet et l'accès aux autres programmes de STEP7.

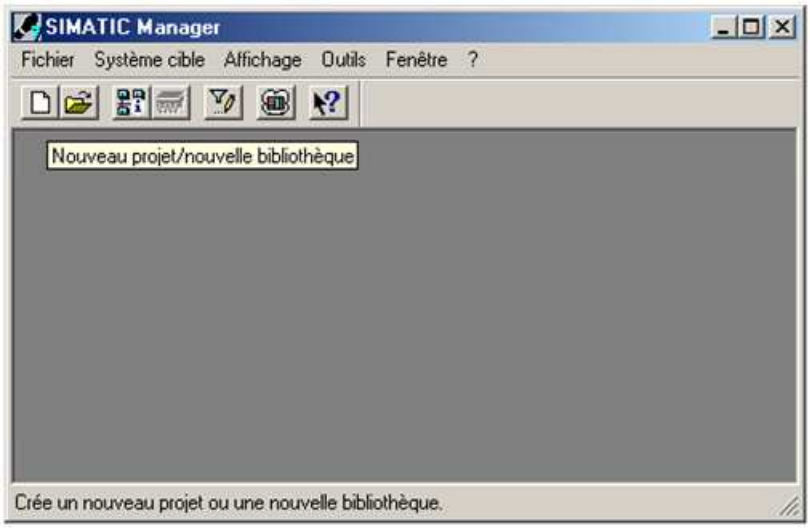


Figure II.1 Création d'un nouveau projet

Pour en créer un nouveau, il suffit de cliquer sur le bouton 'Nouveau projet', attribuer un nom et valider. Ensuite il faut choisir une station de travail.

Une station SIMATIC représente une configuration matérielle S7 comportant un ou plusieurs modules programmables. Il existe différents types:

- SIMATIC 400 : Automate à performances extrêmes, adapté à l'exécution de programme de lourds calculs.
- SIMATIC 300 : Automate à extensibilité modulaire.
- SIMATIC H : Automate insensible aux défaillances, il se compose de 2 CPU du même type, en cas de problème elle commute de l'une vers l'autre sans perte de données.
- SIMATIC PC : ou Station PC, représente un PC ou une station OS contenant des composants SIMATIC : des applications (WinCC, par ex.), un automate logiciel ou une carte CPU enfichée dans le PC.
- Autres stations : se sont soit des appareils d'autres fabricants ou bien des stations de SIMATIC S7 contenus dans un autre projet.
- SIMATIC S5 : liaison vers un projet S5.

- PG/PC : Outils de programmation pour contrôleurs SIMATIC, c'est une console de programmation compatible avec le milieu industriel.

Figure II.2 Station PC et PG/PC

Par exemple on va choisir une station SIMATIC 300

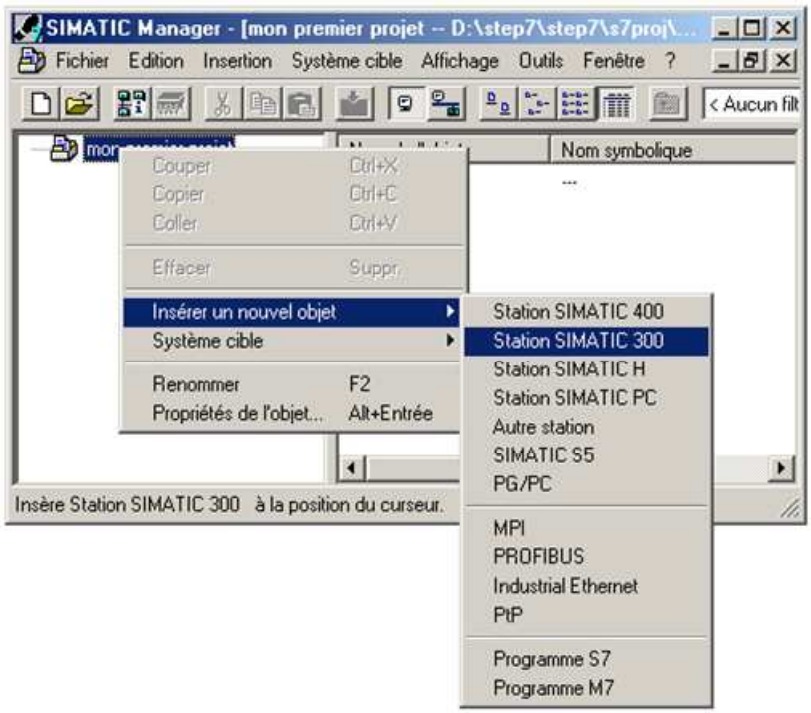


Figure II.3 Choix de la station de travail

Pour commencer, le plus simple est de configurer le matériel, d'éditer les programmes puis les charger dans la CPU. Double clique sur 'Matériel' démarre l'application HW Config.

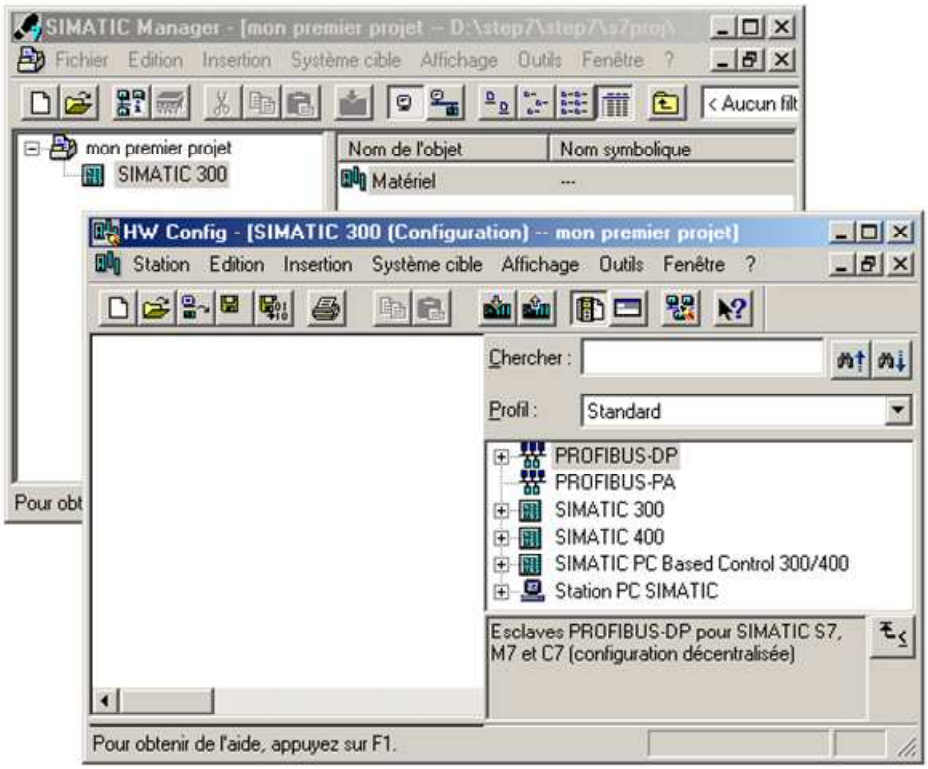


Figure II.4 Configuration du matériel

II.2.1 Configuration du matériel

Pour configurer le matériel, il suffit de faire glisser des éléments du catalogue dans l'emplacement approprié, on choisit le 'Rack', l'alimentation, la CPU et les E/S...

Dans le catalogue on trouve les modules qu'on peut affecter à chaque type de station, on distingue:

- C7 : Système intégré compact qui regroupe automate programmable et interface homme-machine (pupitre opérateur) pour la réalisation de commandes de machines sous encombrement réduit.
- CP : Communication Processor, module de communication (PROFIBUS, Industriel Ethernet, Peer to Peer…).
- FM : Function Module, il regroupe les modules de fonctions (regulation, comptage…).
- IM : Coupleurs d'extension, il permet l'ajout d'autres modules.
- M7 : Modules d'extension et cartouches interface pour SIMATIC M7.
- PS : Module d'alimentation.
- Rack : Support mécanique.
- Routeur : Relie Industriel Ethernet à PROFIBUS.

- SM : Signal Module, c'est le module d'E/S, il contient le AI module d'entrées analogiques, le AO module de sorties analogiques, le DI module d'entrées TOR et le DO module de sorties TOR.
- CPU : L'Unité Centrale, noté CPU xxx a b.
    - xxx est la famille de la CPU
    - a, b sont les propriété de la CPU (éléments additionnels, port de communication…). Par exemple :
        - C : compact, la CPU intègre des modules E/S ainsi que des fonctions spécialisées.
        - PtP : Peer to Peer, la CPU intègre un port de communication Point to Point.
        - H : Fault-tolerant, des unités de traitements insensibles aux défaillances
        - DP : Decentralized Periphery, la CPU intègre un port de communication PROFIBUS

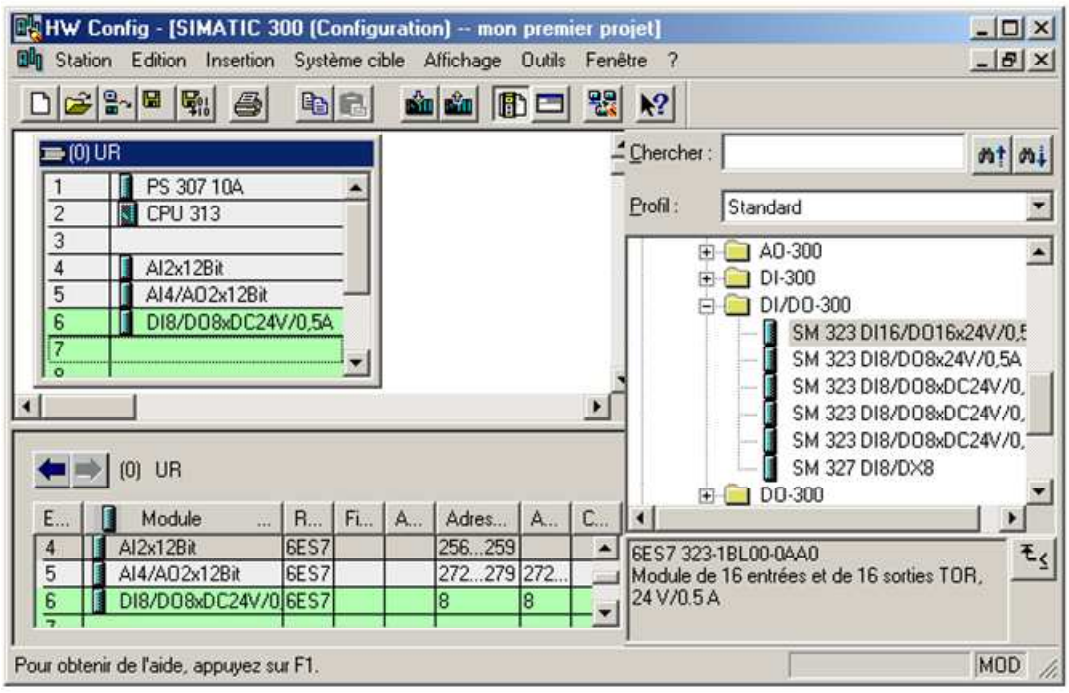


Figure II.5 Selection des modules

Si l'insertion de l'élément choisie est possible dans le Rack, la case appropriée devient verte. Une fois le matériel choisi on sauvegarde, on compile et on charge dans la CPU.

II.2.2 Définition des mnémoniques

De retour dans le SIMATIC Manager, on trouve de nouveaux éléments. On commence par créer les mnémoniques dans la section programmes.

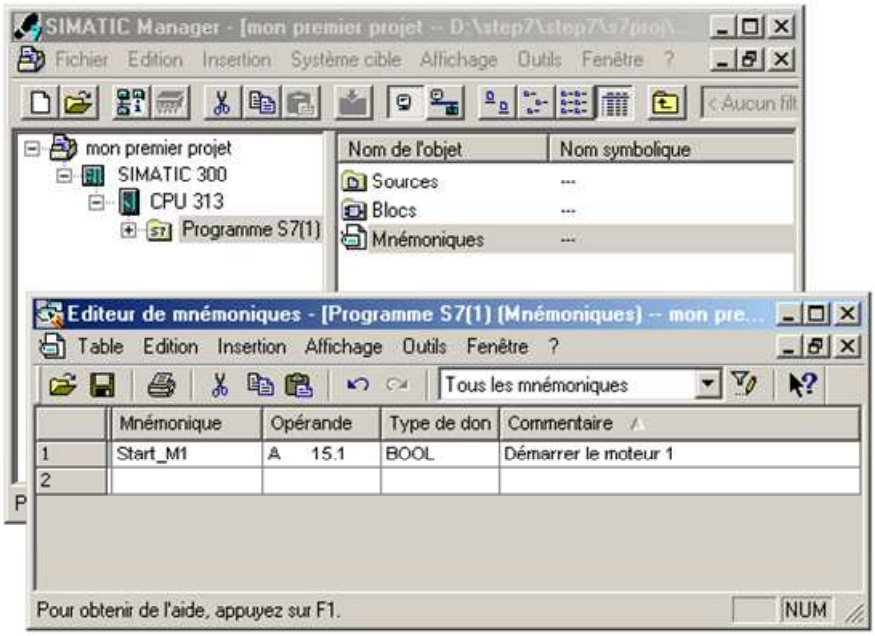


Figure II.6 Edition des Mnémoniques

En affectant des noms symboliques aux adresses absolues, les programmes deviennent plus lisible, faciles à corrigé et à mettre à jour.

Il y quatre différents types d'opérande : le bit, l'octet, le mot et le double mot. Ces types définissent l'accès à une zone mémoire. Pour chaque opérande un certain types de données est permet :

- Pour le bit : BOOL : variable booléenne (True ou False, 1 ou 0).
- Pour l'octet : deux types de données sont possibles :
    1. BYTE: nombre hexadécimal de B#16#0 à B#16#FF.
    2. CHAR : Caractère ASCII, 'A', 'B'…
- Pour le mot : quatre types de données sont possible :
    1. WORD : nombre hexadécimal de W#16#0 à W#16#FFFF.
    2. INT : nombre entier de -32768 à 32767.
    3. S5TIME : Durée S7 en pas de 10 ms (valeur par défaut), de S5T#0H_0M_0S_10MS à S5T#2H_46M_30S_0MS.
    4. DATE : Date en incréments de 1 jour, de D#1990-1-1 à D#2168-12-31.

- Pour le double mot : cinq types de données :
    1. DWORD : nombre hexadécimal de DW#16#0000_0000 à DW#16#FFFF_FFFF.
    2. DINT : nombre entier de L#-2147483648 à L#2147483647.
    3. REAL : nombre à virgule flottante, Limite supérieure : ±3.402823e+38
       Limite inférieure : ±1.175 495e-38.
    4. TIME : Durée en incréments de 1 ms, de −T#24D_20H_31M_23S_648MS à T#24D_20H_31M_23S_647MS.
    5. TIME_OF_DAY : Heure en pas de 1 ms, de TOD#0:0:0.0 à TOD#23:59:59.999.

Le tableau suivant présente les différentes zones mémoire :

Nom de la zone	Description	Accès à la zone par
Mémoire Image des Entrées (MIE)	Au début du cycle le système d'exploitation lit les entrées provenant du processus et enregistre ces valeurs dans la MIE. Le programme utilise ces valeurs pendant son traitement normal.	Bit : E Octet : EB Mot : EW Double mot : ED
Mémoire Image des Sorties (MIS)	Pendant le cycle le programme calcule les valeurs de sortie et les déposes dans la MIS. A la fin du cycle le système d'exploitation lit les valeurs de soties figurées dans la MIS et les transmet aux sorties du processus.	Bit : A Octet : AB Mot : AW Double mot : AD
Mémentos	Se sont des zones mémoires qui permettent de sauvegarder les résultats intermédiaires calculés dans le programme.	Bit : M Octet : MB Mot : MW Double mot : MD
Périphérie d'entrée et de	Cette zone permet d'accéder directement aux modules d'entrées et de sorties.	Octet : PEB, PAB

sortie		Mot : PEW, PAW Double mot : PED ou PAD
Temporisations	Cette zone sert d'espace mémoire pour les cellules de temporisation, l'horloge accède à ces cellules afin de les mettre à jour en décrémentant la valeur de temps.	T
Compteurs	Cette zone mémoire, sert d'espace mémoire pour les opérations de comptage.	Z
données locales	Cette zone contient les données temporaires, elle est utilisé dans les blocs de code (OB, FB ou FC). Ces données sont rangées dans la pile des données locales, elles seront perdues une fois le bloc de code achevé.	Bit : L Octet : LE Mot : LW Double mot : LD

Tableau II.1 Les zones mémoires

II.2.3 Edition des programmes

Dans la section 'bloc' du SIMATIC Manager, on trouve par défaut le bloc d'organisation 1 'OB1' qui représente le programme cyclique. On peut rajouter d'autres blocs à tout moment par un clique droit dans la section Bloc de SIMATIC Manager.

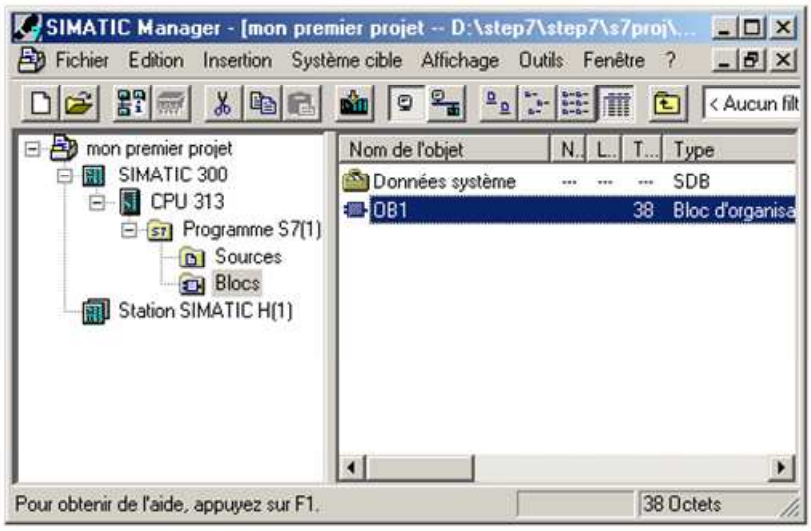


Figure II.7 Edition des programmes

Deux programmes différents s'exécutent dans la CPU: le système d'exploitation et le programme utilisateur.

Le système d'exploitation, organise toutes les fonctions et procédures dans la CPU qui ne sont pas liées à une tâche d'automatisation spécifique. Il gère le déroulement du démarrage à chaud et du redémarrage, l'actualisation de la mémoire image des entrées et l'émission de la mémoire image des sorties, l'appel du programme utilisateur, la gestion des zones de mémoire l'enregistrement des alarmes et l'appel des OB d'alarme…

Le programme utilisateur contient toutes les fonctions nécessaires au traitement des tâches d'automatisation spécifique. Ce programme doit être créé et chargé dans la CPU par l'utilisateur. Il détermine les conditions pour le démarrage à chaud et le redémarrage de la CPU (par exemple, initialiser des signaux), il traite les données du processus (par exemple, combiner des signaux binaires, lire et exploiter des valeurs analogiques), il doit réagir aux alarmes et traiter les perturbations dans le déroulement normal du programme.

Le STEP 7 permet de structurer le programme utilisateur en le subdivisant en différentes parties autonomes ou dépendantes. Ceci permet d'écrire des programmes importants mais clairs, simples à tester et à modifier.

II.2.3.1 Blocs d'organisation

Les blocs d'organisation (OB) constituent l'interface entre le système d'exploitation et le programme utilisateur. Ils sont appelés par le système d'exploitation et gèrent le traitement du programme cyclique et des programmes déclenchés par alarmes, ainsi que le comportement à la mise en route de l'automate programmable et le traitement des erreurs.

Les blocs d'organisation définissent l'ordre (événements de déclenchement) dans lequel les différentes parties du programme sont traitées.

L'exécution d'un OB peut être interrompue par l'appel d'un autre OB (les OB de priorité plus élevée interrompent les OB de priorité plus faible). Les blocs d'organisations les plus prioritaires sont ceux de la mise en route (OB100, OB101 et OB102) et le moins prioritaire est le cycle en arrière plan (OB90).

On appelle alarmes les événements qui déclenchent l'appel d'un OB donné. Le type d'alarme définit la classe de priorité de celle-ci. Il existe 13 types d'alarmes :

1. Cycle libre (OB1) : il s'exécute d'une façon continue. Son traitement constitue le traitement normal pour les automates programmables. Le système d'exploitation appelle l'OB1 cycliquement et déclenche ainsi le traitement cyclique du programme utilisateur.

2. Alarmes horaires (OB10 à OB17) : Elles peuvent être déclenchées une seule fois à un moment donné (indication de temps absolue avec date) ou périodiquement avec indication du commencement et de la fréquence de répétition (par exemple, toutes les minutes, toutes les heures, tous les jours.
3. Alarmes temporisées (OB20 à OB23) : Elles permettent l'exécution retardée de certaines parties du programme utilisateur. Ce retard doit être définit précédemment.
4. Alarmes cycliques (OB30 à OB38) : Elles sont déclenchées à des intervalles de temps précis. La période de déclenchement est toujours un multiple entier de la période de base de 1 ms.
5. Alarmes de processus (OB40 à OB47) : Elles réagissent à des signaux provenant des modules (SM, CP ou FM). Elles sont déclenchées lorsqu'un module de signaux pouvant générer des alarmes de processus, avec validation d'alarme de processus paramétrée, transmet un signal de processus reçu à la CPU ou lorsqu'un module de fonction de la CPU signale une alarme.
6. Alarme DPV1 (OB55 à OB57) : Les esclaves DPV1 (modules décentralisés) peuvent déclencher des alarmes de diagnostic, de processus et de débrochage.
7. Alarme multiprocesseur (OB60) : Le mode "multiprocesseur" correspond au fonctionnement simultané de plusieurs CPU (quatre au maximum) dans un châssis central. Lors du traitement de l'alarme multiprocesseur, le programme utilisateur émetteur ainsi que les programmes utilisateur s'exécutant dans les autres CPU vérifient s'ils connaissent la tâche et réagissent ensuite selon la programmation.
8. Alarmes de synchronisme d'horloge (OB61 à OB64) : Elle permet la synchronisation du programme utilisateur contenu dans une CPU avec d'autres programmes répartis dans un réseau.
9. Erreur de redondance (OB70 et OB72) : Elle est appelée lorsqu'une perte de redondance se produit sur le réseau PROFIBUS DP, perte de redondance des CPU, erreur de synchronisation ou erreur dans un module SYNC.
10. Erreurs asynchrones (OB80 à OB87) : Elles sont appelées lorsqu'apparaît
    a. une erreur de temps : le dépassement du temps de cycle maximal, le saut d'alarmes horaires parce que l'heure a été avancée, un retard excessif pour le traitement d'une classe de priorité.

b. une défaillance dans l'appareil de base ou dans un appareil d'extension : de la tension d'alimentation 24 V, d'une pile, du système de sauvegarde entier.
c. débrochage/enfichage, erreur sur l'interface au réseau MPI, erreur d'exécution du programme (l'OB correspondant ne démarre pas) ou erreur de communication.

11. Cycle en arrière plan (OB90) : Il permet d'exécuter des processus à durée non critique et ainsi d'éviter des temps d'attente.
12. Mise en route (OB100 à 102) : On distingue entre les modes de mise en route suivants:
    a. redémarrage (n'existe pas pour les S7-300 et S7-400H) : OB101
    b. démarrage à chaud (le seul possible pour S7-300) : OB100
    c. démarrage à froid : OB102
13. Erreurs synchrones (OB121 et OB122) : Elles sont appelées lorsqu'apparaît une erreur de programmation (des temporisations adressées manquent, un bloc appelé n'est pas chargé…) ou lorsqu'une opération STEP 7 accède à une entrée ou une sortie d'un module de signaux à laquelle aucun module n'était associé lors du dernier démarrage, par exemple : erreur en cas d'accès direct à la périphérie (module défaillant ou manquant), accès à une adresse de périphérie inconnue de la CPU.

II.2.3.2 Fonctions et blocs fonctionnels

On peut programmer chaque bloc d'organisation en tant que programme structuré en créant des fonctions (FC) et des blocs fonctionnels (FB)

- Les blocs fonctionnels (FB) sont des blocs de code associés a des blocs de données d'instance, dans les quels sont sauvegardés les paramètres effectifs et les données statique des blocs fonctionnels.
- Les fonctions (FC) sont des blocs de code sans rémanence, c'est-à-dire qu'ils ne sont pas associe à des blocs de données, les paramètres effectifs ne sont pas sauvegardés automatiquement.

De plus il existe les blocs fonctionnels système (SFB) et les fonctions système (SFC), qui sont des fonctions préprogrammés. Ils peuvent être appelés à partir du programme utilisateur. On trouve des fonctions système pour la copie de blocs de données, le contrôle du programme utilisateur, la gestion des alarmes horaires et temporisées…

II.2.3.3 Bloc de données

Les blocs de données (DB) servent à l'enregistrement de données utilisateur. Les blocs de données globaux servent à l'enregistrement de données qui peuvent être utilisées par tous les autres blocs. Les blocs de données d'instances sont affectés à des blocs fonctionnels.

Les différents blocs cités ci-dessus peuvent être édités avec l'application 'CONT LIST LOG'.

II.2.4 Programmation des blocs

La programmation des blocs de codes peut se faire à l'aide de trois applications :

1. **CONT LIST LOG** : elle permet de programmer des blocs d'organisations 'OB', des blocs fonctionnels 'FB' et des fonctions 'FC'.
1. **GRAPH** : elle permet de programmer des blocs fonctionnels 'FB'.
2. **SCL** : elle permet de créer des sources de code. Une source de code est un fichier texte, qui contient une suite d'instructions, une fois compilé il peut être transférer dans la CPU. On peut trouver dans un même fichier source tout le programme utilisateur, c'est-à-dire les blocs d'organisations, les blocs fonctionnels et les fonctions.

Nota : les deux dernières applications GRAPH et SCL n'existent pas dans la version de base du STEP7, il faut les installées séparément.

On va présenter comment éditer les Blocs de code avec ces trois applications.

II.2.4.1 CONT LIST LOG

Un système de contrôle de qualité d'un certain produit fonction comme suit : L'élément X se déplace sur un tapis roulant, un premier capteur 'Cap1' vérifier sa hauteur, s'il est supérieur à une certaine hauteur h0, l'élément est éjecté en dehors du tapis grâce à un piston Pi1, sinon il poursuit son parcours vers un autre capteur 'Cap2' qui vérifier sa largeur, si elle n'est pas conforme à une certaine consigne, un piston Pi2 le pousse vers un casier en dehors du tapis.

Un front montant dans le capteur 1 (capteur 2) déclenche le piston 1 (piston 2).

On va éditer le bloc d'organisation 1, pour contrôler la gestion de ce système.

Dans SIMATIC Manager, cliquer sur le dossier programmes, ensuite sur Blocs, double clique sur l'icône du l'OB1ouvre l'application 'CONT LIST LOG'.

Choisissons par exemple le langage CONT

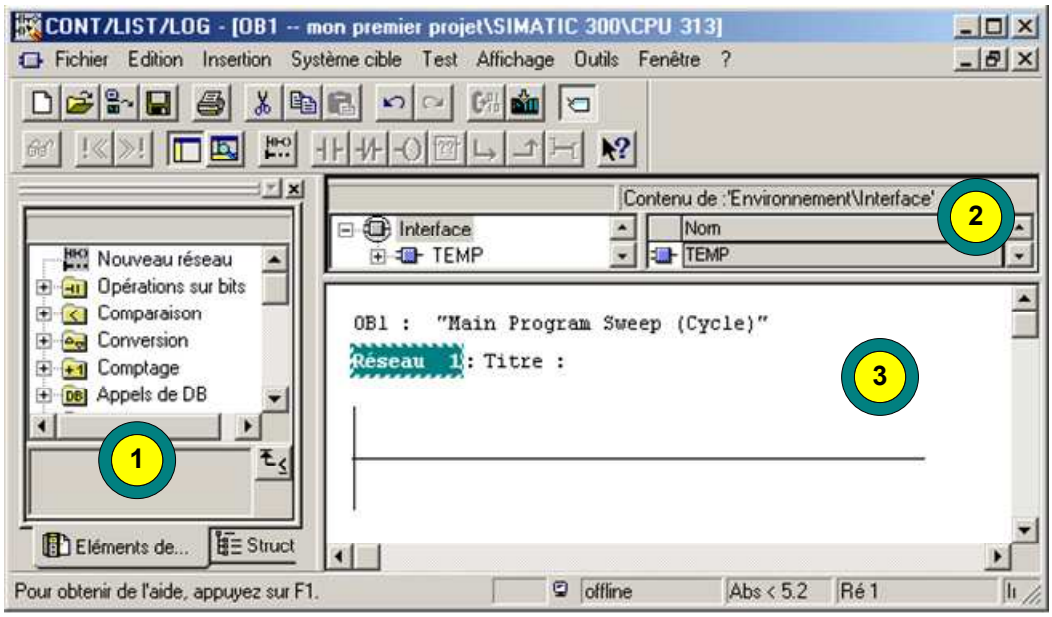


Figure II.8 CONT LIST LOG

On peut distinguer trois Zones principales :

1. Le catalogue : c'est la bibliothèque des opérations et instructions du langage choisit (ici CONT), ces différentes opérations sont regroupées par famille : opération sur les bits, sur les mots, sur les entiers… pour utilisé une opération il suffit de faire glisser l'élément vers zone souhaitée.
2. La zone de déclarations : dans cette zone on définit les variables locales, les variables d'entrées et sorties ou arguments pour les fonctions et les blocs fonctionnelles.
3. La zone d'édition : elle est structurée en réseaux, chaque réseau finit par une affectation ou un saut.

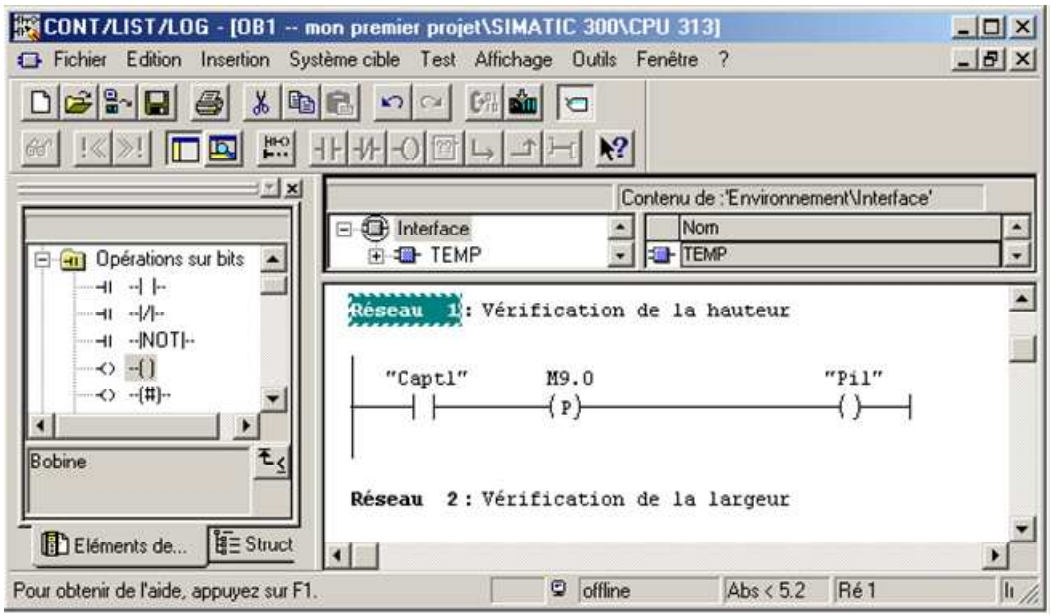


Figure II.9 Exemple d'un programme en CONT

Pour notre exemple, on a 2 réseaux seulement. La détection du front montant se fait à l'aide de l'opération –(P)–. Elle nécessite un bit pour sauvegarder l'état des opérations qui la précède, au prochain cycle, elle compare ce bit avec le nouvel état de ces opérations.

Il est très utile d'affecter des titres aux réseaux ainsi que des commentaires. Sa permet une détection plus rapide des erreurs et permet d'écrire des programmes plus clair (en programmant, il faut penser toujours que ce n'est pas nous qui va le lire).

II.2.4.2 SCL

Le SCL est un langage de haut niveau, il est très utile pour la programmation de blocs de code complexes. Contrairement aux autres langages, le SCL peut contenir dans un seul fichier source tout le programme utilisateur (blocs d'organisation, fonctions et bloc fonctionnelles), il suffit de déclarer chaque bloc en respectant l'ordre d'appel, c'est-à-dire que si un bloc X appel un autre bloc Y, ce dernier doit être déclaré en premier dans le fichier source.

Comme application de démonstration, on se propose de calculer le gain en décibels d'un circuit électrique donné : on envoi a l'entrée du système une tension via le module de sorties analogique, ensuite on récupère la tension de sortie du système via le module d'entrées de l'automate. Il suffit après de calculer $20 \cdot \log_{10} \frac{Tension\ de\ sortie\ du\ système}{Tensio\ d'entrée\ du\ système}$.

Dans SIMATIC Manager, cliquer su 'Programmes' ensuite sur source et créer un nouveau fichier source SCL. Double clique sur le nouveau objet inséré ouvre l'application d'édition.

Pour ajouter un bloc d'organisation, cliquer sur 'insérer', 'model de bloc' ensuite sur OB, l'application génère automatiquement quelque lignes de code.

Remplacer les xxx par le numéro du bloc, il faut faire attention de respecter la numérotation tel qu'elle est définit par la classe des blocs d'organisation.

Compléter le bloc, sauvegarder et compiler, ceci génère les blocs ajouter dans le dossier 'Bloc' de l'application SIMATIC Manager.

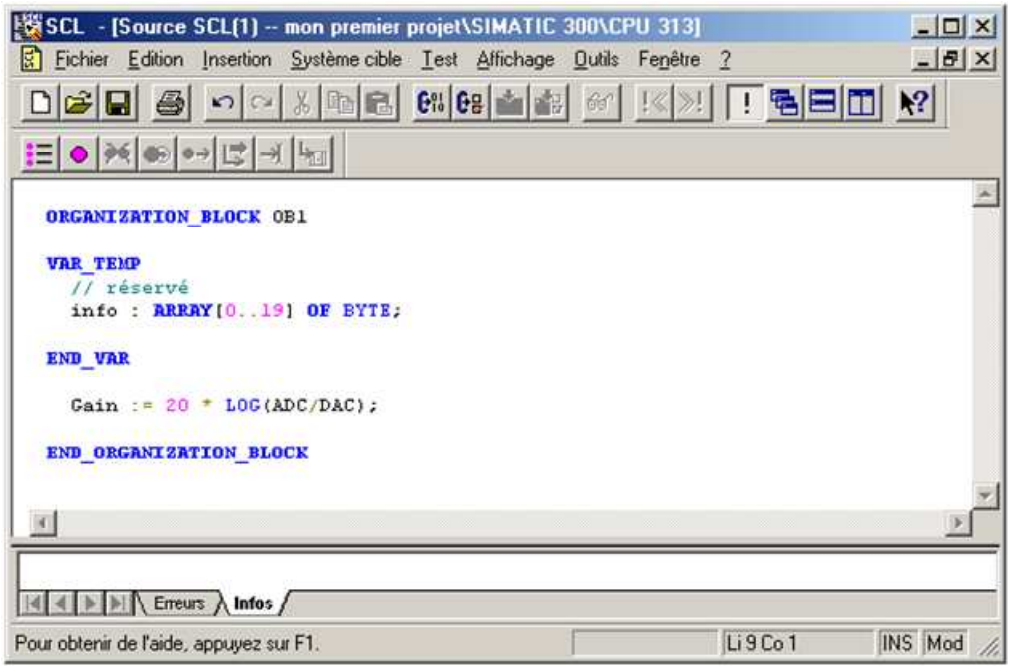


Figure II.10 Exemple d'un programme en SCL

Bien sûre, il faut déclarer les variables Gain, ADC et DAC dans l'éditeur des mnémoniques.

II.2.4.3 S7-GRAPH

Cette application permet l'édition des blocs fonctionnels, en effet elle ne permet pas d'éditer des fonctions car elles n'ont pas des blocs de données associes pour mémorisé l'état des différentes transitions et étapes.

On va illustrer l'édition avec S7-GRAPH, par cet exemple :

Un chariot de transport dans une mine se trouve à l'état initiale au point A, en appuyant sur le bouton poussoir m on démarre le moteur M1, arrivé au point B on démarre M2 jusqu'au point C, on arrête M1 et on continu avec M2 jusqu'au point D, on arrête le chariot.

Figure II.11 Commande d'un chariot

Pour satisfaire ce cahier de charge, on commence par créer un bloc fonctionnel et un autre de donnée d'instance dans le dossier Blocs du SIMTAIC Manager.

On choisit le GRAPH comme langage de programmation et on lance l'application.

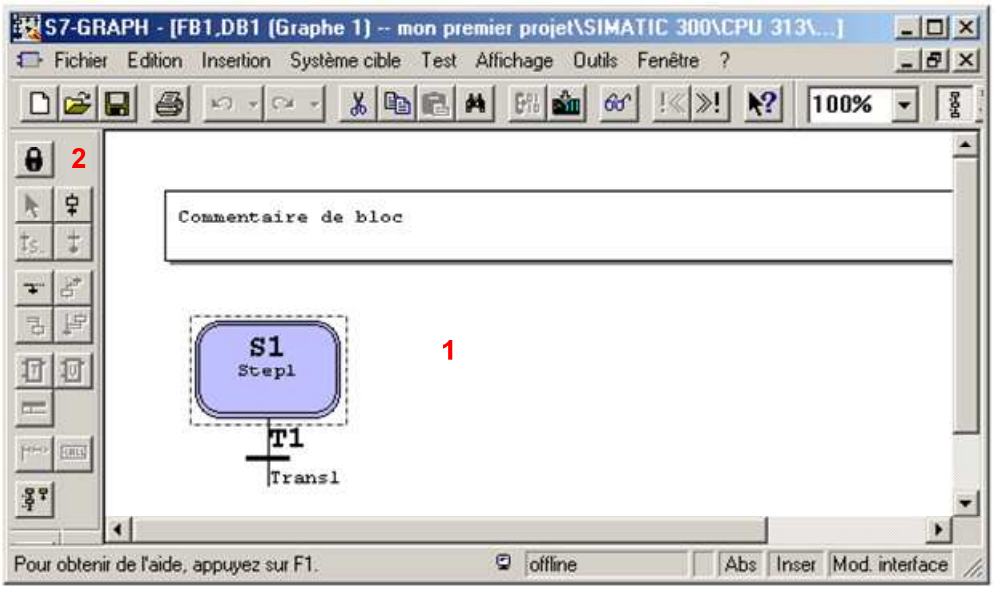


Figure II.12 Edition d'un Bloc fonctionnel avec S7-GRAPH

On remarque deux parties, la première est la zone d'édition, la deuxième inclus les éléments à ajouter (des transitions, des branchements, des étapes…).

Pour ajouter un élément à la zone d'édition, il suffit de cliquer sur l'emplacement souhaité puis sur l'élément.

Notre exemple de commande de chariot comprend cinq étapes et quatre transitions. On commence par les crées.

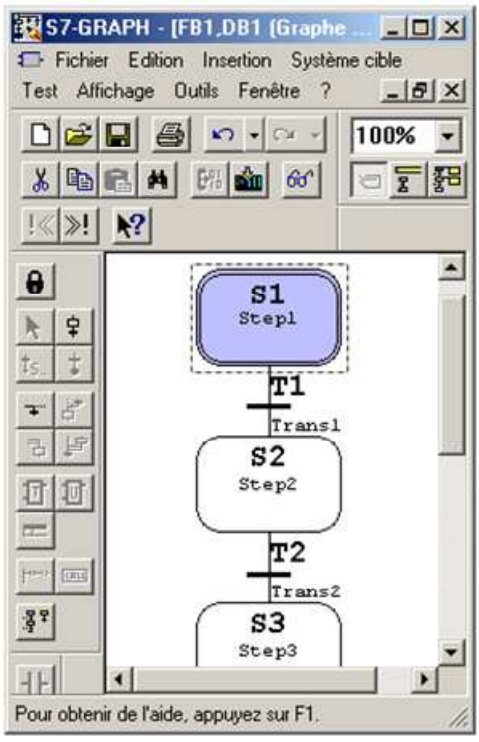


Figure II.13 Edition du Graph

Ensuite pour l'édition des actions et des transitions on change l'affichage de 'Graph' à 'Étape'. Pour ajouter une action, cliquer par le bouton droit de la souris sur la colonne à droite et choisissez 'insérer un nouvel élément' puis sur 'action'. Il existe différent type d'action, le tableau suivant présente les plus utilisées :

Opération	opérande	Signification
N	A, E, M ou D	Tant que l'étape est active l'opérande est à 1.
S	A, E, M ou D	SET : Dès que l'étape est active, l'opérande est mis à 1.
R	A, E, M ou D	RESET : Dès que l'étape est active, l'opérande est mis à 0.
D	A, E, M ou D	Delay : n secondes après l'activation de l'étape, l'opérande se mis à 1 pour la durée de l'activation.
L	A, E, M ou D	Impulsion : quand l'étape est active, l'opérande se mis à 1 pendant n secondes
CALL	FB, FC, SFB ou SFC	Appel de bloc : tant que l'étape est active, le bloc spécifié est appelé.

Tableau II.2 Les action dans le S7-GRAPH

Pour les transitions, elles peuvent être présentées en langage CONT ou LOG, cliquer sur la colonne à gauche et insérer des éléments (contacts…) depuis le catalogue.

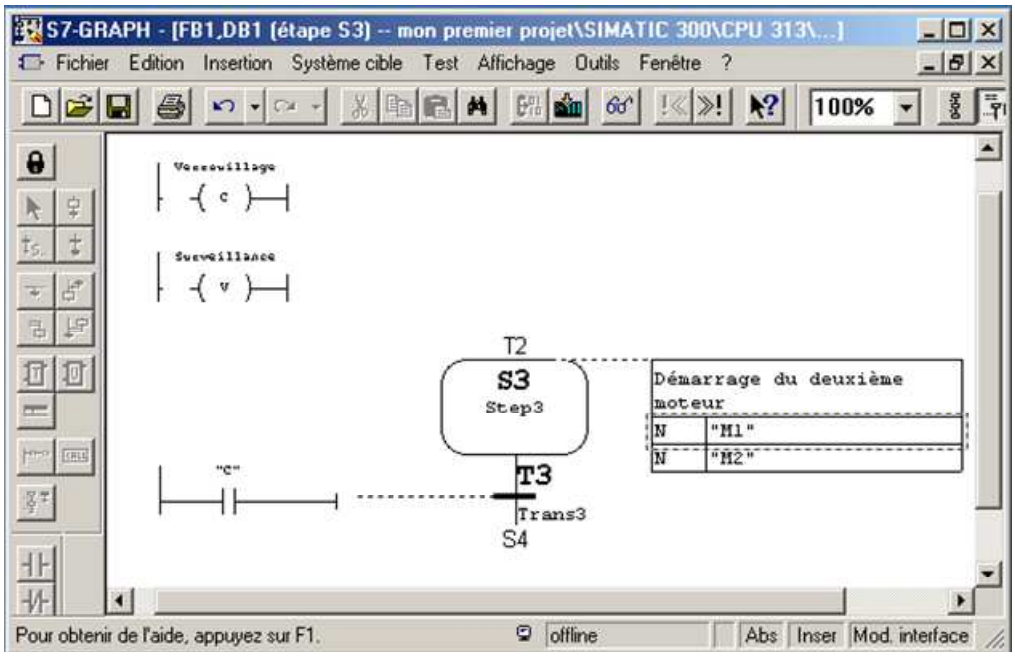


Figure II.14 Edition des actions & transitions

En haut à gauche, on remarque deux autres éléments :

1. Verrouillage : l'Interlock, c'est une condition programmable servant à verrouiller une étape et influençant l'exécution de certaines actions.
2. Surveillance : Supervision, c'est une condition programmable servant à surveiller une étape et influençant l'évolution du graphe entre cette étape et la suivante.

Pour plus de détail sur l'utilisation du S7-GRAPH, référez vous à l'annexe.

II.2.5 Simulation de modules

L'application de simulation de modules S7-PLCSIM permet l'exécution et le teste du programme utilisateur destinés aux CPU S7-300 et aux CPU S7-400, ainsi qu'à WinLC. La simulation étant complètement réalisée au sein du logiciel STEP 7, il n'est pas nécessaire qu'une liaison soit établie avec un matériel S7 quelconque. Lorsque S7-PLCSIM s'exécute, toute nouvelle liaison est automatiquement dirigée vers la CPU de simulation.

S7-PLCSIM dispose d'une interface simple permettant de visualiser et de forcer les différents paramètres utilisés par le programme (comme, par exemple, d'activer ou de désactiver des entrées).

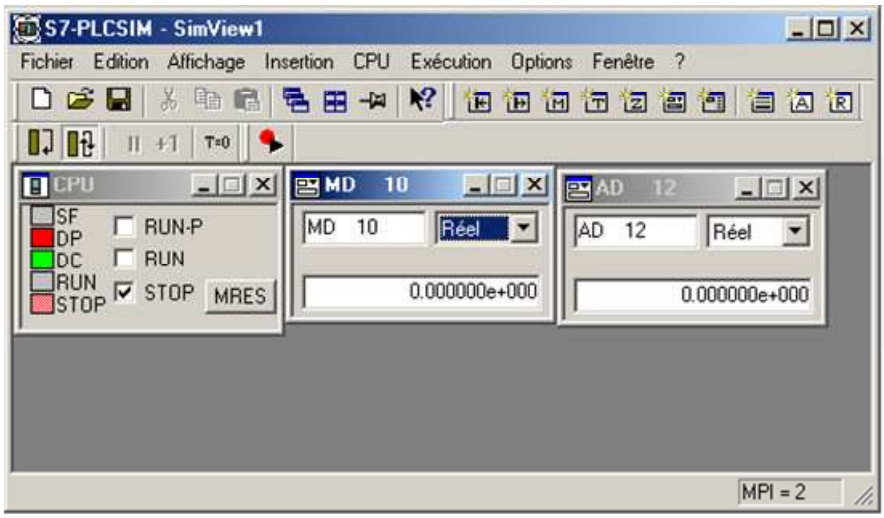


Figure II.15 Simulation de modules

En outre, S7-PLCSIM possède les fonctions suivantes :

- On peut créer des "fenêtres" dans lesquelles on a la possibilité d'accéder aux zones de mémoire d'entrée et de sortie, aux accumulateurs ainsi qu'aux registres de la CPU de simulation. On peut également accéder à la mémoire par adressage symbolique (il faut juste charger la table des mnémoniques dans ‘options’, puis sur ‘outils’ ‘insérer mnémoniques’).
- On peut sélectionner l'exécution automatique des temporisations ou encore les définir et les réinitialiser manuellement.
- On a la possibilité de changer l'état de fonctionnement de la CPU (STOP, RUN et RUN-P) comme pour une CPU réelle. De plus, on dispose d’une fonction de pause qui permet d'interrompre momentanément la CPU, sans affecter l'état du programme.

Bien que l'AP de simulation soit essentiellement logiciel, STEP 7 le considère comme une réelle composante matérielle, à quelques différences près :

- Contrairement à ce qui se passe avec une CPU réelle lors de la mise à l’arrêt de la CPU, l'état des sorties ne change pas.
- La CPU n'attend pas le début ou la fin du cycle pour actualiser une donnée qu’on a modifiée. Toute modification dans une fenêtre entraîne l'actualisation immédiate du contenu de l'adresse en mémoire.
- Les options d'exécution permettent de choisir le mode d'exécution du programme par la CPU :

- La commande Cycle unique exécute un cycle du programme, puis attend qu'on démarre l'exécution du cycle suivant.
- La commande Cycle continu exécute le programme de la même manière que dans un AP réel : elle démarre un nouveau cycle aussitôt que le cycle précédent est terminé.

- On peut déclencher manuellement les OB d'alarme (aller dans 'exécution' puis 'déclenchement OB Erreur').
- Les modules fonctionnels (FM) ne sont pas pris en charge.
- La communication d'égal à égal n'est pas possible.

On va essayer le fonctionnement du S7-PLCSIM avec un petit exemple. Dans le dossier Programmes du SIMATIC Manager, on ajoute un bloc, par exemple une fonction

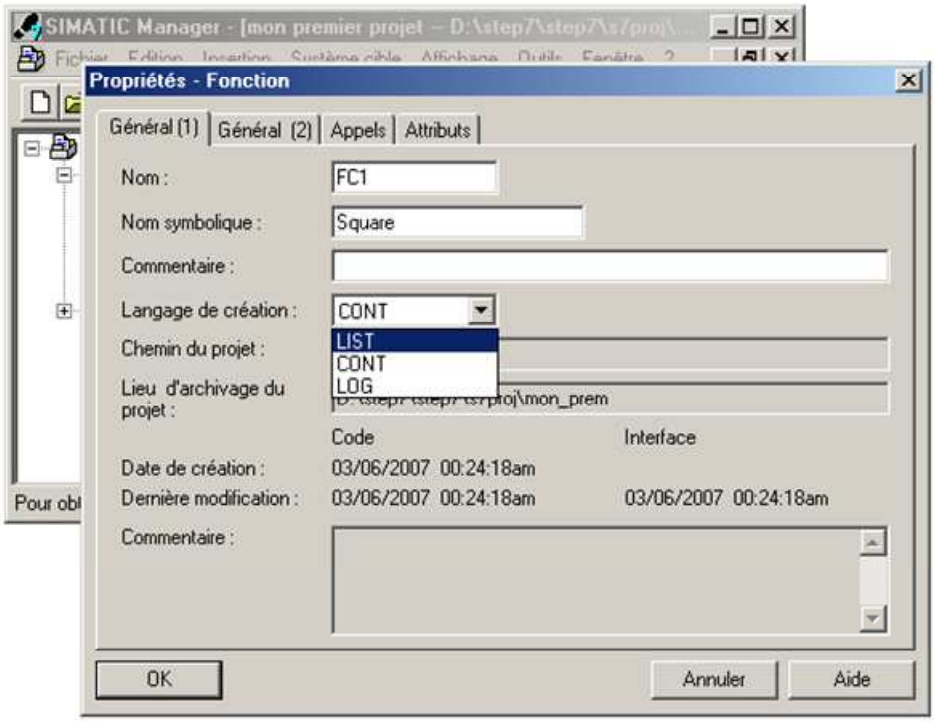


Figure II.16 Création d'une fonction

On donne à la fonction un nom symbolique, on choisit un langage d'édition parmi les trois proposés et on valide. Par exemple on va réaliser la fonction carrée $y = x^2$, celle-ci nécessite deux arguments, un en entrée (x) et un autre en sortie (y).

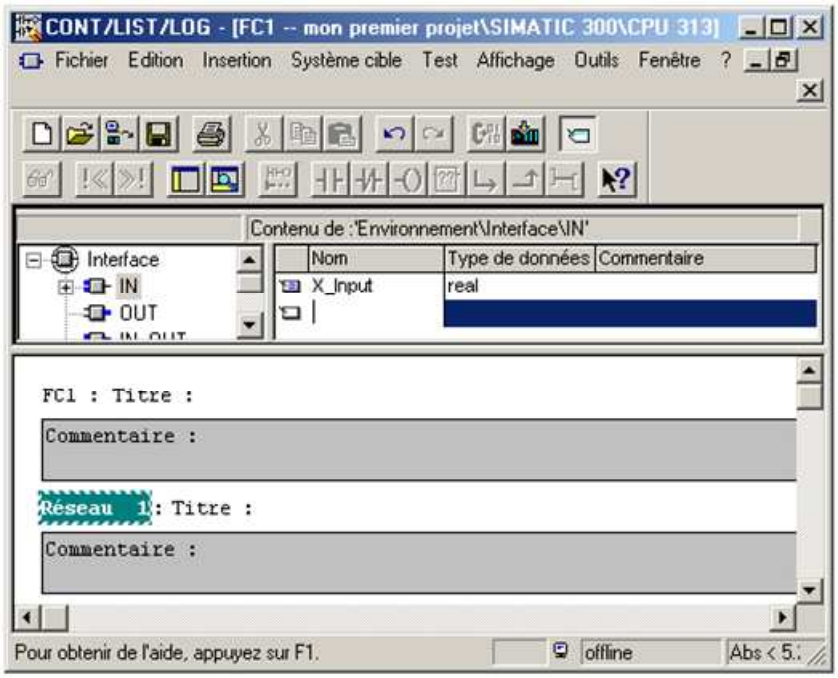


Figure II.17 Définition des arguments d'une fonction

On édit le code de cette fonction et on sauvegarde.

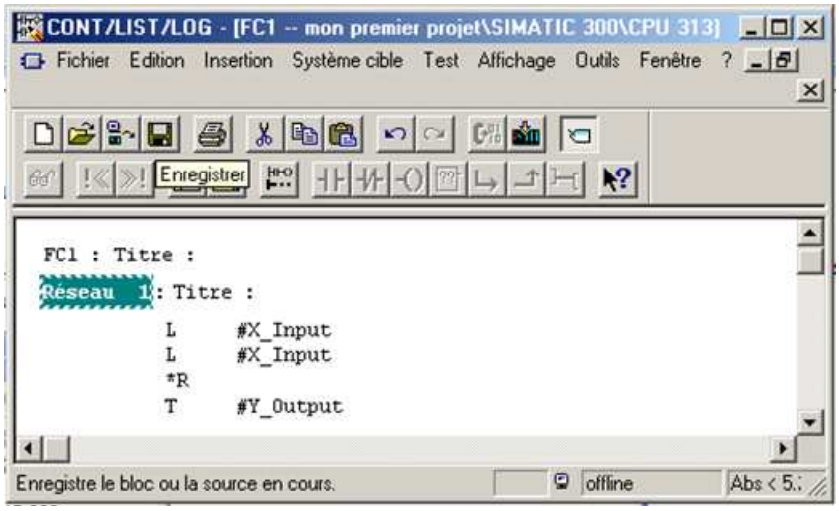


Figure II.18 Edition du code en LIST

On revient sur SIMATIC Manager, on ouvre l'OB1, on change le langage d'édition dans le menu supérieur 'Affichage' puis sur 'CONT'. Maintenant sur le catalogue à gauche on appel la fonction que nous venant de créer en lui affectant les variables d'E/S.

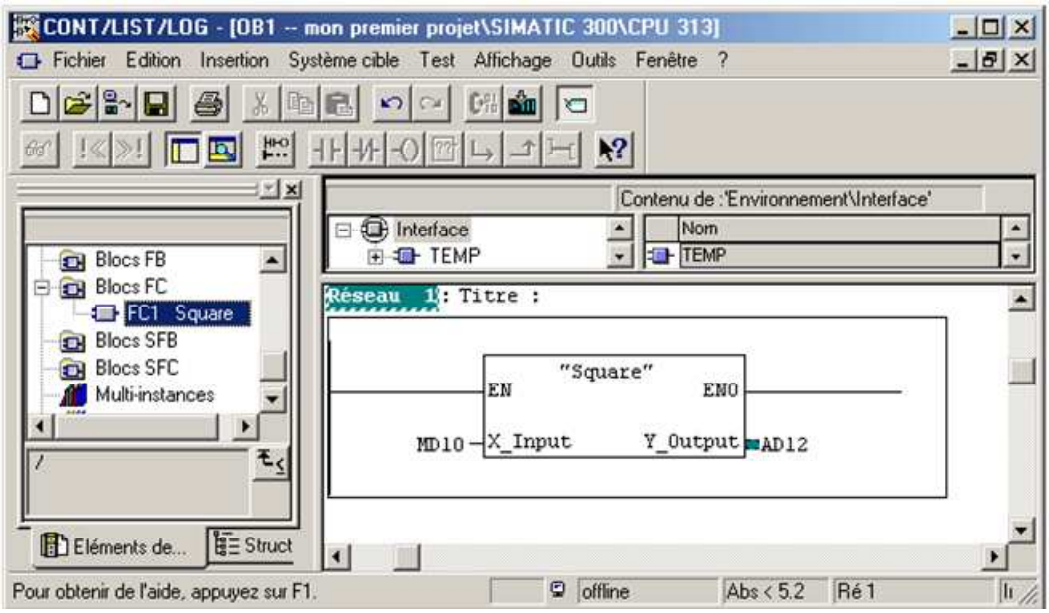


Figure II.19 Edition du code en CONT

Dans cette exemple, le bloc d'organisation Cyclique OB1 appel la fonction FC1 qui va affecter au double mot de la sortie analogique 'AD12' le carrée du double mot contenu dans le mémento 'MD10'. Le contenu de ces adresses mémoires va être interprété comme nombre réel.

On lance l'application S7-PLCSIM (aller dans 'Options' puis cliquer sur 'Simulation des modules') et on charge les blocs de code dans la CPU.

Dans le menu insertion, on ajoute le mémento 'MD10' et le double mot de la sortie analogique 'AD12', il faut sélectionner Réel comme type sinon on se retrouve avec des résultats erronés.

On démarre la simulation en cochant la case RUN, ensuite on introduit différents valeurs dans le mémento MD10 et on vérifié la valeur correspondante dans le double mot de la sortie analogique AD12.

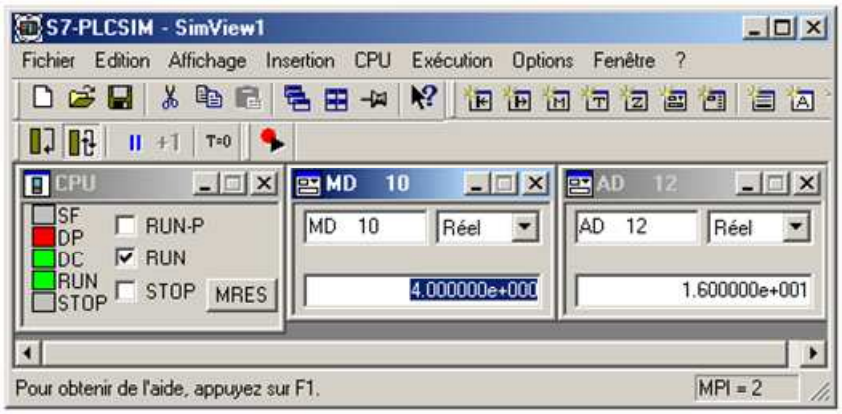
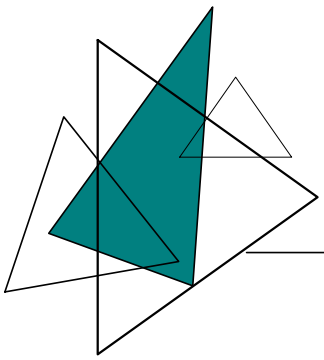


Figure II.20 Simulation du programme

Ça marche !!

CHAPITRE III

APPLICATIONS



Section A

SIMULATION NUMERIQUE

III.A.1 Introduction

Cette application a pour but de montrer comment utilisé le STEP7 pour faire de la simulation numérique. Celle-ci permet :

- de trouver des solutions numériques à des équations différentielles difficiles à résoudre analytiquement.
- de prédire le comportement et l'évolution d'un système dans le temps.

Il existe différentes méthodes de calcul numériques, les plus utilisés sont les celles de Runge-Kutta. Elles reposent sur le principe d'itération. C'est-à-dire que la sortie à l'instant i dépend de celle de l'instant i-1.

Dans ce qui suit on utilisera la méthode de Runge-Kutta d'ordre quatre, la RK4,

III.A.2 Runge-Kutta ordre 4

Considérons le problème suivant :

Pour $t \in [t_0, t_f]$

$$\begin{cases} \dot{y} = f(t, y, w) & y(t_0) = y_0 \\ \dot{w} = g(t, y, w) & w(t_0) = w_0 \end{cases}$$

Pour un pas de discrétisation h, l'algorithme du calcul numérique est donné par les équations :

$$\begin{cases} y_{n+1} = y_n + \frac{h}{6}(K1 + 2 \cdot K2 + 2 \cdot K3 + K4) \\ w_{n+1} = w_n + \frac{h}{6}(V1 + 2 \cdot V2 + 2 \cdot V3 + V4) \end{cases}$$

Où :

$$\begin{cases} K1 = f(t_n, y_n, w_n), & V1 = g(t_n, y_n, w_n) \\ K2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} \cdot K1, w_n + \frac{h}{2} \cdot V1\right), & V2 = g(t_n + \frac{h}{2}, y_n + \frac{h}{2} \cdot K1, w_n + \frac{h}{2} \cdot V1) \\ K3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} \cdot K2, w_n + \frac{h}{2} \cdot V2\right), & V3 = g(t_n + \frac{h}{2}, y_n + \frac{h}{2} \cdot K2, w_n + \frac{h}{2} \cdot V2) \\ K2 = f(t_n + h, y_n + h \cdot K3, w_n + h \cdot V3), & V4 = g(t_n + h, y_n + h \cdot K3, w_n + h \cdot V3) \end{cases}$$

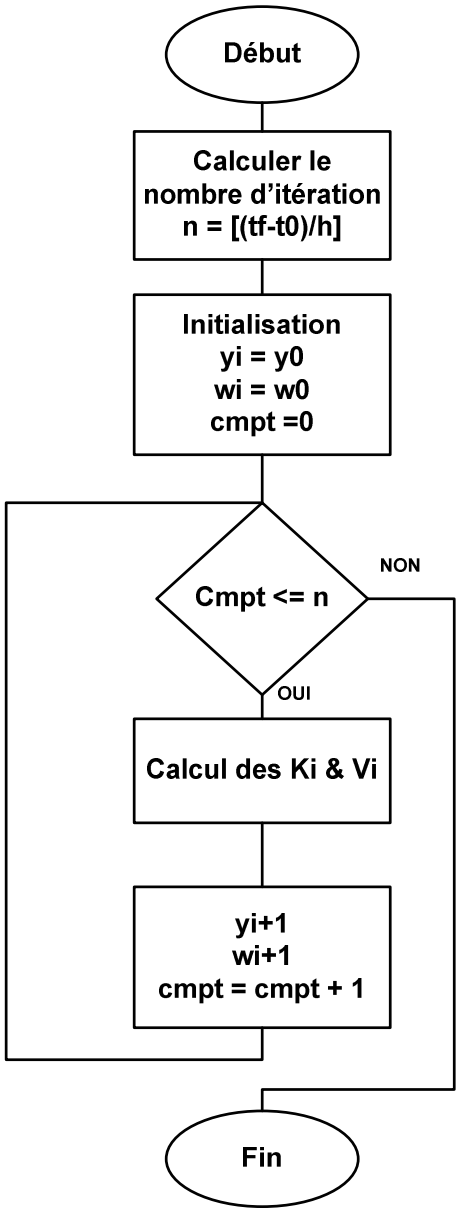


Figure III.A.1 Organigramme du calcul numérique

Pour visualiser la sortie il faut brancher un oscilloscope à la sortie analogique de l'automate.

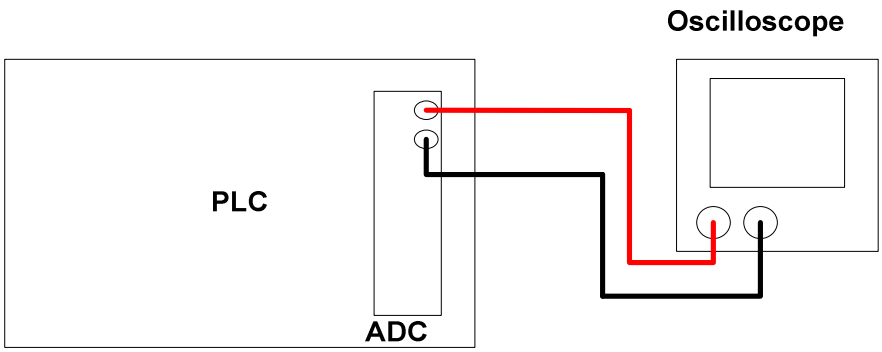


Figure III.A.2 Visualisation de la sortie analogique

La sortie maximale du module analogique de l'automate programmable utilisé étant égale à 10V, on doit choisir l'étalonnage le plus approprie pour avoir une réponse visualisable.

III.A.3. Simulation d'un système du premier ordre

Soit le circuit électrique suivant :

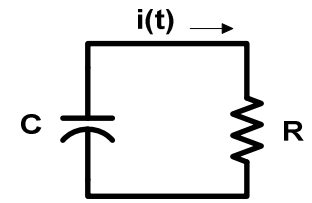


Figure III.A.3 Circuit RC

$$V_R + V_C = 0 \ avec \ V_C = \int i \cdot dt$$

$$d'ou \ RC \cdot \dot{V}_C = -V_C$$

Le condensateur étant initialement chargé, on pose

$$y = V_C, y_0 = V_{C0} \ et \ \tau = {}^{1}/_{RC}$$

On aura alors $\dot{y} = f(t,y) \ avec \ f(t,y) = -\tau \cdot y$

On va simuler l'évolution de ce système pour $t \in [0,2]s$ avec un pas de discrétisation h=0.01s pour les valeurs numériques suivantes : $R = 10K\Omega, \ C = 10\mu F \ \ et \ V_{C0} = 5V$

La valeur maximale pour système est égale à 5V, on prend un coefficient d'étalonnage égal à 2.

III.A.4. Résolution d'une équation différentielle du 2ème ordre

On veut résoudre l'équation différentielle du 2ème ordre

$$\ddot{x} + 2 \cdot \dot{x} + x = 0 \ avec \ x(0) = 0 \ et \ \dot{x}(0) = 2$$

On peut prendre $y = x \ \ et \ \ w = \dot{x}$, on aura le système d'équation suivant :

$$\begin{cases} \dot{y} = f(t,y,w) \\ \dot{w} = g(t,y,w) \end{cases} \quad avec \quad \begin{cases} f(t,y,w) = w \\ g(t,y,w) = -2 \cdot w - y \end{cases}$$

En prenant un pas de discrétisation h=0.01 et $t \in [0,10]s$, on simule le système via le programme présenté précédemment et on visualise la sortie sur un oscilloscope.

III.A.5. Résolution d'un système d'équations

Soit le système d'équation algébrique suivant :

$$\begin{cases} a_1 \cdot x_1 + b_1 \cdot x_2 = c_1 \\ a_2 \cdot x_1 + b_2 \cdot x_2 = c_2 \end{cases}$$

Si une solution existe, elle est forcément constate, on peut écrire alors que

$$\dot{x}_1 = \dot{x}_2 = 0$$

On exploitant cette relation, on peut écrie que :

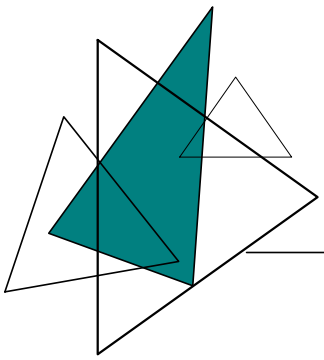
$$\begin{cases} \dot{x}_1 = a_1 \cdot x_1 + b_1 \cdot x_2 - c_1 \\ \dot{x}_2 = a_2 \cdot x_1 + b_2 \cdot x_2 - c_2 \end{cases}$$

Le système d'équation algébrique prend la forme d'une représentation dans l'espace d'état d'un système continu, la solution de ce système est la valeur du vecteur d'état dans le régime permanent.

Il suffit de prendre $y = x_1$ et $w = x_2$, on aura le système d'équation suivant :

$$\begin{cases} \dot{y} = f(t,y,w) \\ \dot{w} = g(t,y,w) \end{cases} \quad avec \quad \begin{cases} f(t,y,w) = a_1 \cdot y + b_1 \cdot w - c_1 \\ g(t,y,w) = a_2 \cdot y + b_2 \cdot w - c_2 \end{cases}$$

On simule le système via la méthode Runge-Kutta, néanmoins il faut assurer que le déterminent soit négatif.



Section B

Commande des systèmes continus

III.B. Introduction

Cette section a pour but de montrer les possibilités qu'offre un automate programmable pour commander un système continu.

Nous avons traité deux applications différentes : La commande en boucle fermé d'un four pour la régulation de la température (système long) et la commande en boucle ouverte de la vitesse d'un moteur à courant continu (système rapide).

Le schéma de principe d'une commande numérique via automate est le suivant :

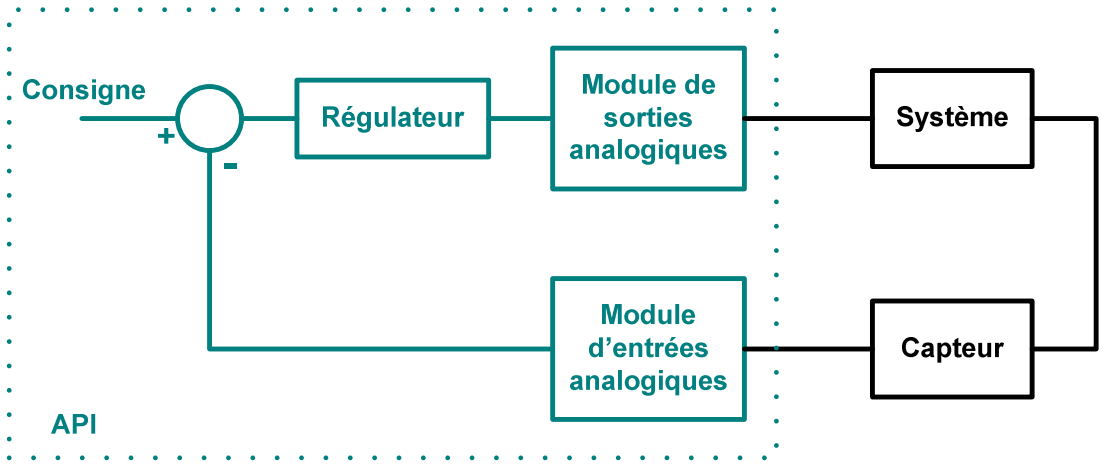


Figure III.B.1 Schéma de principe de la commande via automate

1ère Application

Régulation de la température d'un four électrique

III.B.1. Régulation de la température d'un four électrique

Il s'agit de réguler la température d'un four de laboratoire, le KL-600. C'est un Kit de formation constituer de

- un four électrique : KL-68001 Humidity & Temperature Load.
- un capteur de température : KL-63003 AD590
- une unité centrale : KL-61001 Main Unit

III.B.1.1. Description du four

1 - Afficheur digital + clavier

2 - Entrée du signal de commande

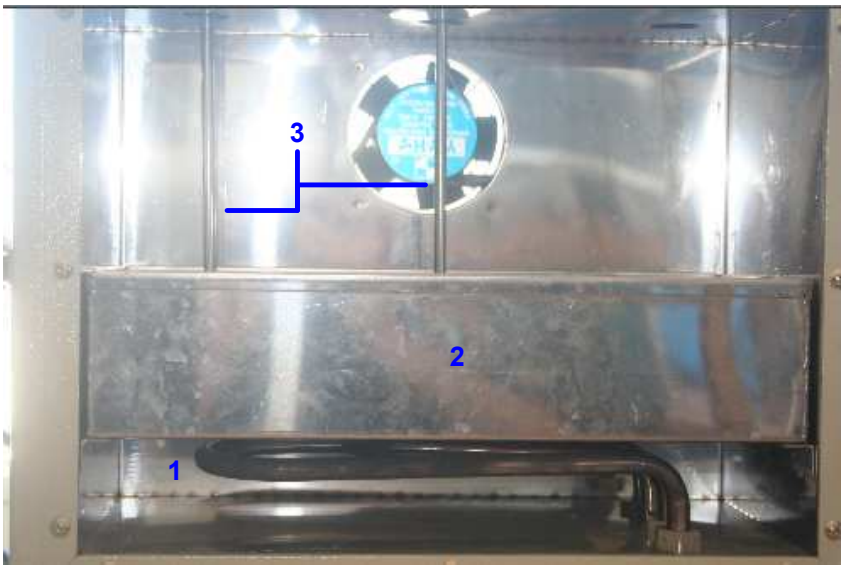
3 - Manuel / Auto

4 - Temperature/ Humidité

5 - On / Off

Figure III.B.2 Description du four

Ce four fonctionne avec une alimentation alternative de 220V~, il a deux modes de fonctionnement le mode manuel et le mode automatique. Dans le mode manuel la consigne température est introduite par le clavier alors que dans l'autre mode le signal de commande est une tension continue comprise entre 0 et 10 V.



1 - Résistance

2 - Banc

3 - Capteurs de Température

Figure III.B.3 Constitution du four

Il est constitué d'une résistance dont la tension d'alimentation est contrôlée par un triac, d'un banc ou on peut mettre le liquide et de deux capteurs de température. L'un d'eux est relié à l'afficheur digital, l'autre au circuit de transduction.

Figure III.B.4 Résistance du four

III.B.1.2 Description du capteur et de l'unité centrale

L'unité centrale fournit l'alimentation nécessaire au transducteur (+12 V), et permet l'interfaçage d'une éventuelle connexion avec le port parallèle d'un ordinateur.

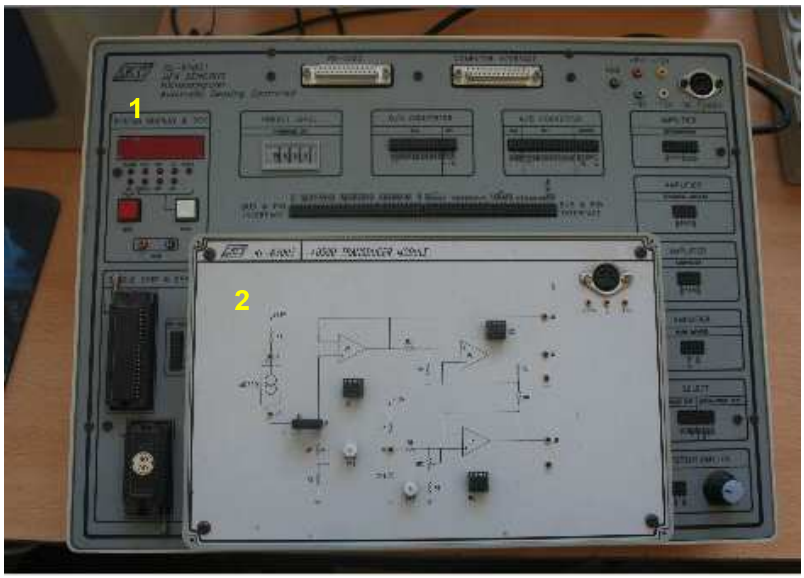


Figure III.B.5 L'unité centrale + transducteur

III.B.1.2.1. Le capteur

Le capteur de température utilisé est l'AD590. C'est un capteur à réponse rapide, il produit un courant proportionnel à la température absolue $1\mu A/°K$. Ce qui permet d'écrire : $T(°C) = I(\mu A) - 273.15$

Voici quelques de ces caractéristiques :

- Courant linéaire de sortie: $1\mu A/°K$
- Etendue de mesure : de −55°C à +150°C.
- Linéarité excellent: ±0.3°C au dessus de toutes les amplitudes (Aucun circuit de linéarisation n'est nécessaire).
- Etendue de tension d'alimentation: de 4 V à 30 V.

L'AD590 est un capteur à base de transistors, il exploite le changement de la tension V_{BE} avec la température. Si deux transistors opèrent à rapport constant de courant de collecteur alors la tension aux bornes de leur émetteur sera (kT/q).

Avec :

k : est la constante de Boltzmann.

q : la charge des électrons.

La tension résultante est directement proportionnelle à la température absolue. Grace à des résistances cette tension est convertit en courant. La figure suivante montre le schéma électrique de l'AD590.

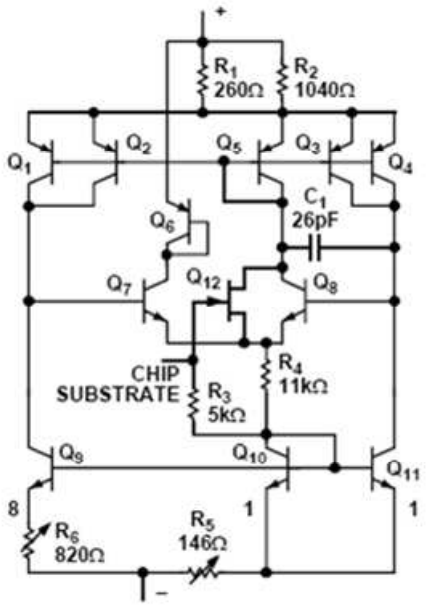
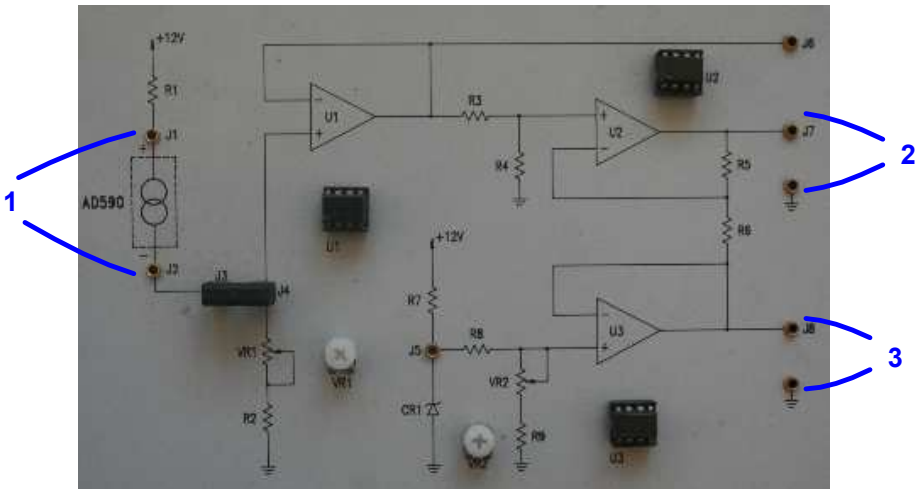


Figure III.B.6 Schéma électrique de l'AD590

Q8 et Q11 sont les transistors qui produisent la tension proportionnelle à la température, et las résistances R5 et R6 convertit cette tension en un courant.

III.B.1.2.2. Conversion du courant en tension

Vu que le capteur AD590 délivre un courant en sortie, un convertisseur courant-tension est utilisé pour le convertir en une tension comprise entre 0 et 10V, soit $0.1V/°C$



1 - Bornes d'entrées (courant) 2 - Bornes de sorties (tension)
3 - Réglage de l'origine (0 °C)

Figure III.B.7 Schéma du convertisseur (courant-tension)

En résumé on a :

- un four alimenté en $220V\sim$, qui délivre une température entre la température ambiante et 100°C pour un signal de commande compris entre 0 et 10 V,
- un capteur de température qui délivre 0.1V pour 1 °C.

Figure III.B.8 Photo de l'ensemble Four + Capteur + Automate

III.B.1.3. Identification du système

Le modèle mathématique du four étant inconnu, nous avons effectué des identifications de la réponse indicielle en boucle ouverte pour différents points de fonctionnement, car le four est un système fortement non linéaire (les modèles ne seront valables qu'autour des points de fonctionnement).

Le modèle choisi est celui de Broîda, qui est de la forme : $F(p) = \frac{K}{1+Tp} e^{-\tau p}$

Il s'agit d'identifier la réponse indicielle, on envoi au four une consigne de tension comprise entre 0 et 10V, et on relève la réponse du système. A partir du graphe obtenu on identifier les paramètres du modèle via les équations suivantes :

$$\begin{cases} K = \frac{\Delta Y_\infty}{\Delta E_\infty} \\ T = 5.5 \cdot (t_2 - t_1) \end{cases}$$

$$\tau = 2.8 \cdot t_1 - 1.8 \cdot t_1$$

t_1 *et* t_2 *s*ont les instants au bout des quelles la sortie atteint respectivement 0.28 et 0.4 de sa valeur finale.

Le tableau suivant résume les résultats trouvés :

Point de fonctionnement (°C)	Température initiale (°C)	Modèle identifié
30	18	$\frac{6.4}{1+324 \cdot p} e^{-263 \cdot p}$
35	20	$\frac{6.8571}{1+451 \cdot p} e^{-346 \cdot p}$
45	18	$\frac{5.33}{1+452 \cdot p} e^{-238 \cdot p}$
55	16	$\frac{4}{1+436 \cdot p} e^{-262 \cdot p}$

Tableau III.1 Fonctions de transfert autour du point de fonctionnement

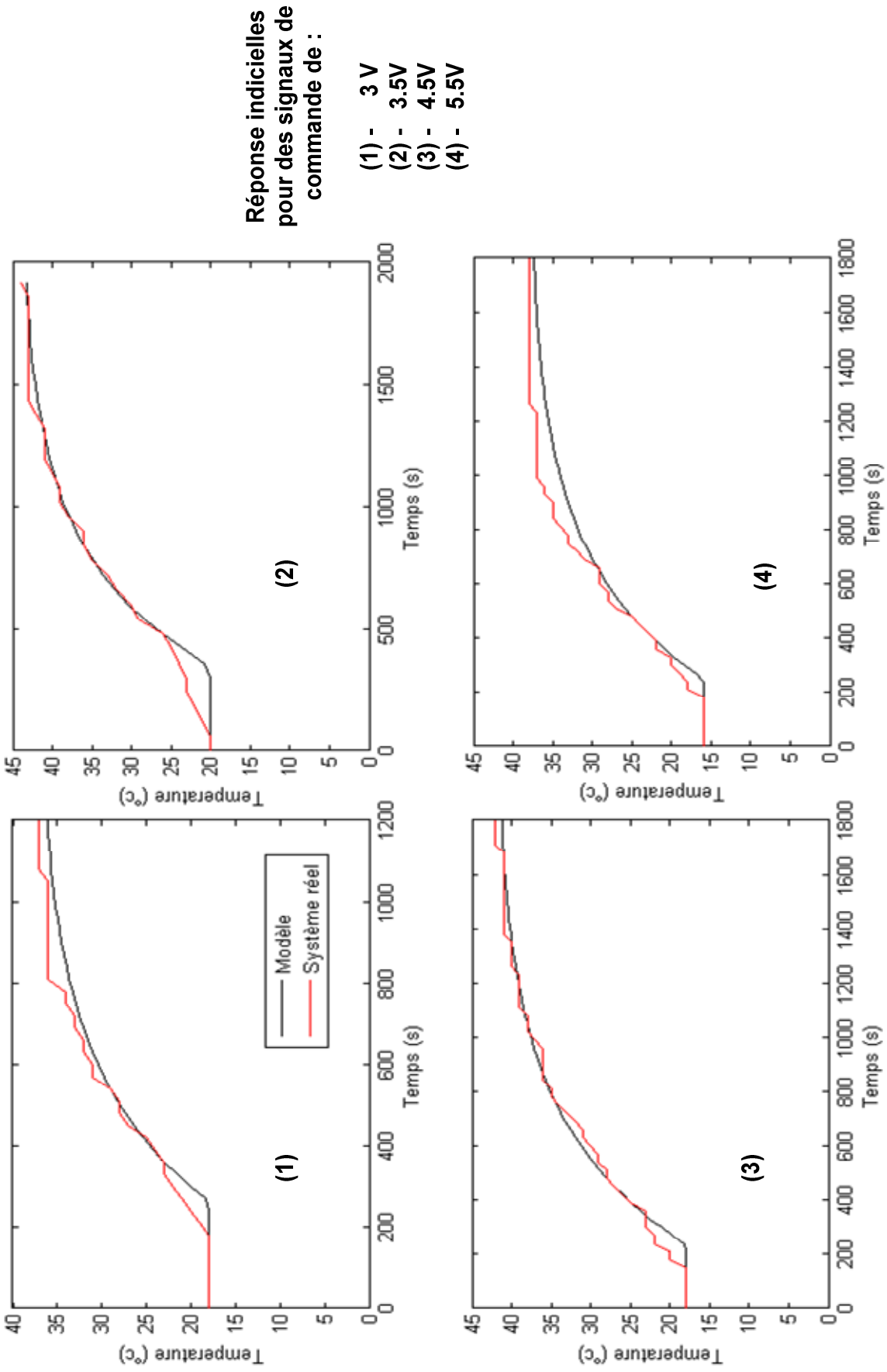


Figure III.B.9 Identification du four

III.B.1.4. Description du programme

On a utilisé deux type de régulation : régulation PID et régulation ON-OFF, l'organigramme général du calcul de la commande, est représenté sur la figure suivante :

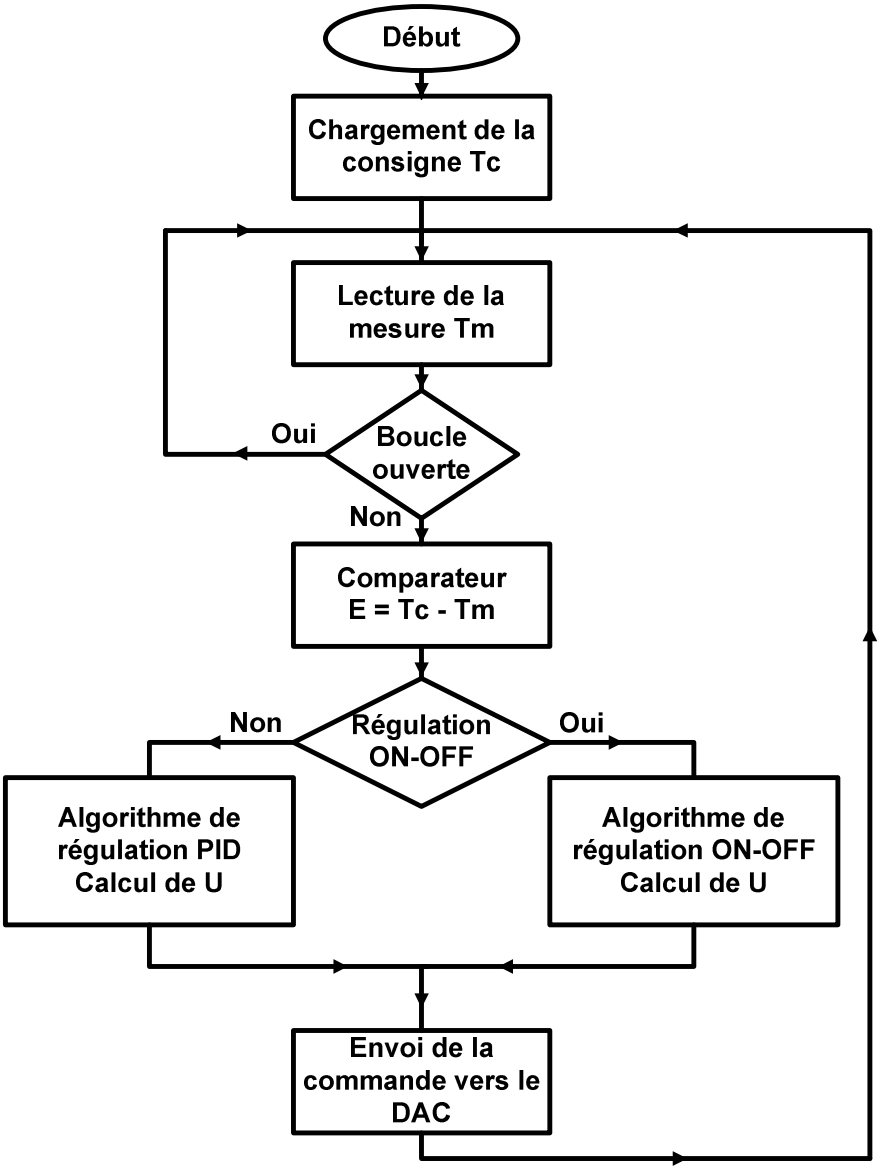


Figure III.B.10 Organigramme général du calcul de la commande

Chacun de ces blocs est représenté dans le programme par une fonction FC.

Le cheminement de cet organigramme est divisé en trois parties exécutées par trois blocs d'organisations différents.

1. Bloc d'organisation de démarrage OB100.
2. Bloc d'organisation cyclique (continu) OB1
3. Bloc d'organisation d'alarme cyclique OB35.

III.B.1.4.1. Bloc d'organisation de démarrage

L'OB100 s'exécute lors du démarrage de l'automate, on va l'utiliser pour l'initialisation des zones mémoires utilisées et pour exécuter certains blocs, comme le chargement de la consigne, le choix du type de régulation…

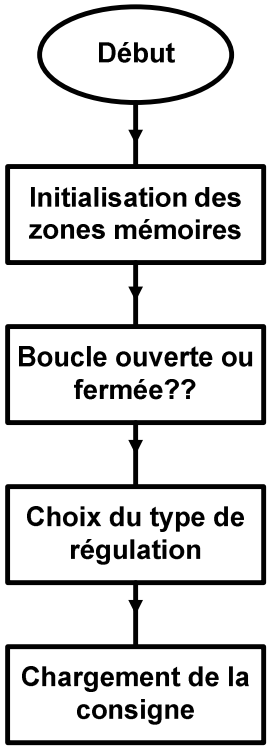


Figure III.B.11 Programme du démarrage OB100

III.B.1.4.2. Bloc d'organisation cyclique

L'OB1 s'exécute d'une façon continue, on va l'utiliser pour la lecture de la mesure, pour faire le filtrage et rechercher le max.

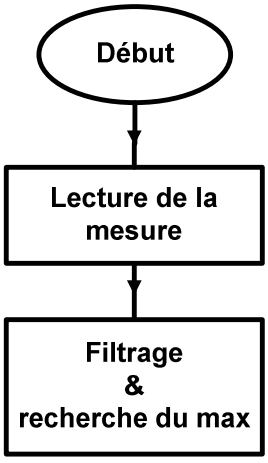


Figure III.B.12 Programme Cyclique OB1

III.B.1.4.3. Bloc d'organisation d'alarme cyclique

L'OB35 s'exécute à des périodes régulières, la période d'exécution peut être modifiée dans les propriétés de la CPU. Nous l'avons choisit de tel sorte quelle soit égale à la période d'échantillonnage du système (Te).

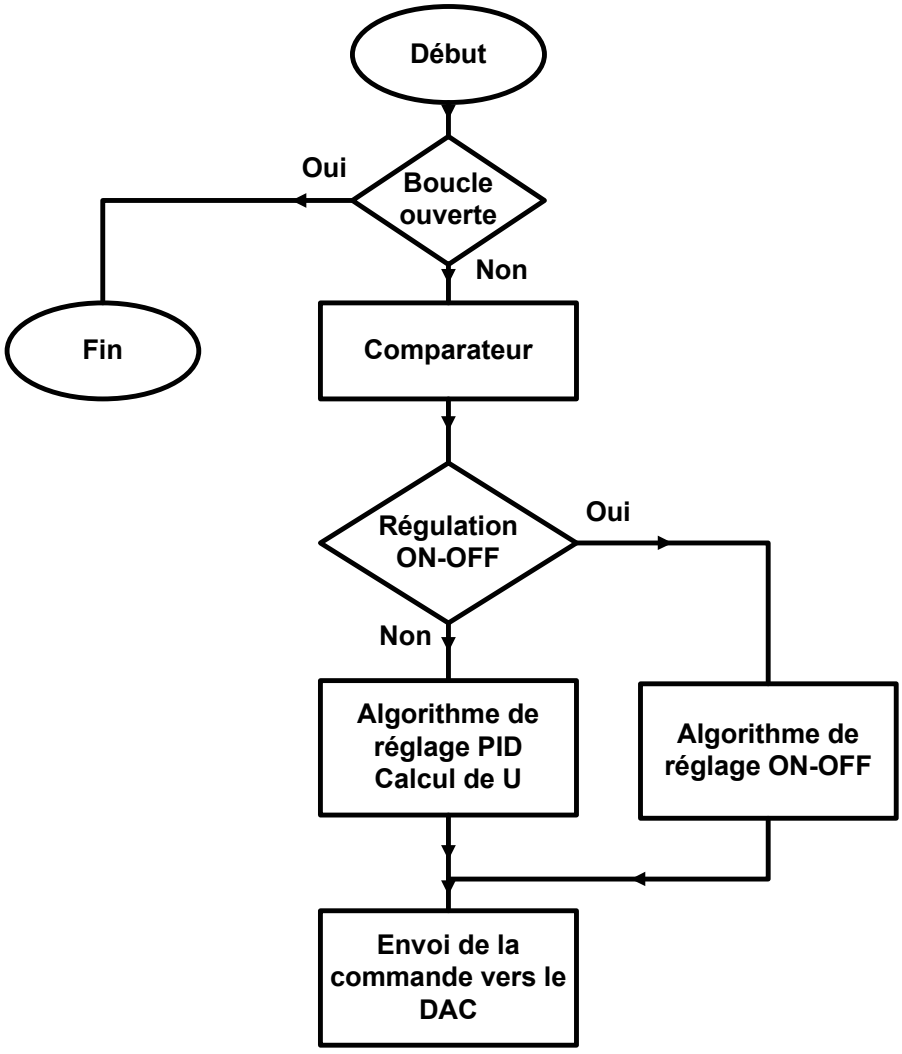


Figure III.B.13 Programme de l'alarme cyclique OB35

Dans ce qui suit On va essayer d'expliquer le fonctionnement de chacun de ces blocs.

III.B.1.5.1. Chargement de la consigne

La consigne peut être chargée de deux façons :

1. Manuel : la consigne est introduite dans le programme (mode hors-ligne)
2. Automatique : la consigne est introduite par en manipulant un potentiomètre qui délivre une tension entre 0 et 10V, ce signal est converti en donnée numérique après passage par l'entrée analogique channel 2

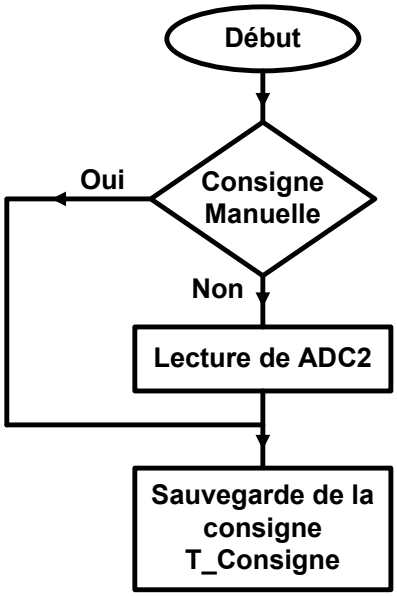


Figure III.B.14 Chargement de la consigne

Cette partie est représentée dans le programme par la fonction FC3, elle n'est exécutée qu'une seule fois lors du démarrage de l'automate (OB100).

III.B.1.5.2. Lecture de la mesure

La sortie du capteur de température après conversion en tension est connectée à l'entrée analogique ADC channel1. Pour chaque tension comprise entre 0 et 10V, on peut lire dans l'ADC une valeur entière entre 0 et 27600. Il suffit donc de diviser la valeur lue par 276 pour avoir la température correspondante.

Le seul problème est que le signal délivré par le capteur est fortement bruité (voir figure B.11), il faut donc le filtré. Plusieurs solutions sont possibles, la plus simple consiste à simuler un filtre passe bas à l'aide du simulateur numérique déjà réalisé.

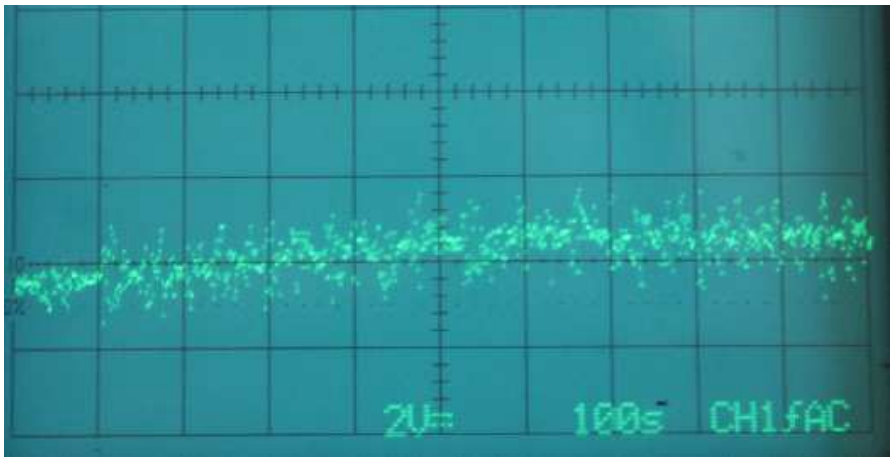


Figure III.B.15 Signal issu du capteur

On choisit la fréquence de coupure du filtre plus grande que celle du four (filtre passe bas) et moins que la fréquence d'échantillonnage pour ne pas perdre de donnée.

Une autre solution consiste à prendre le maximum des valeurs lues sur une période d'échantillonnage, car on a remarqué que les pics dans le signal bruité représentent au mieux les tensions correspondantes, lus sur un voltmètre connecté aux bornes du capteur.

L'organigramme suivant présente le cheminement des deux méthodes.

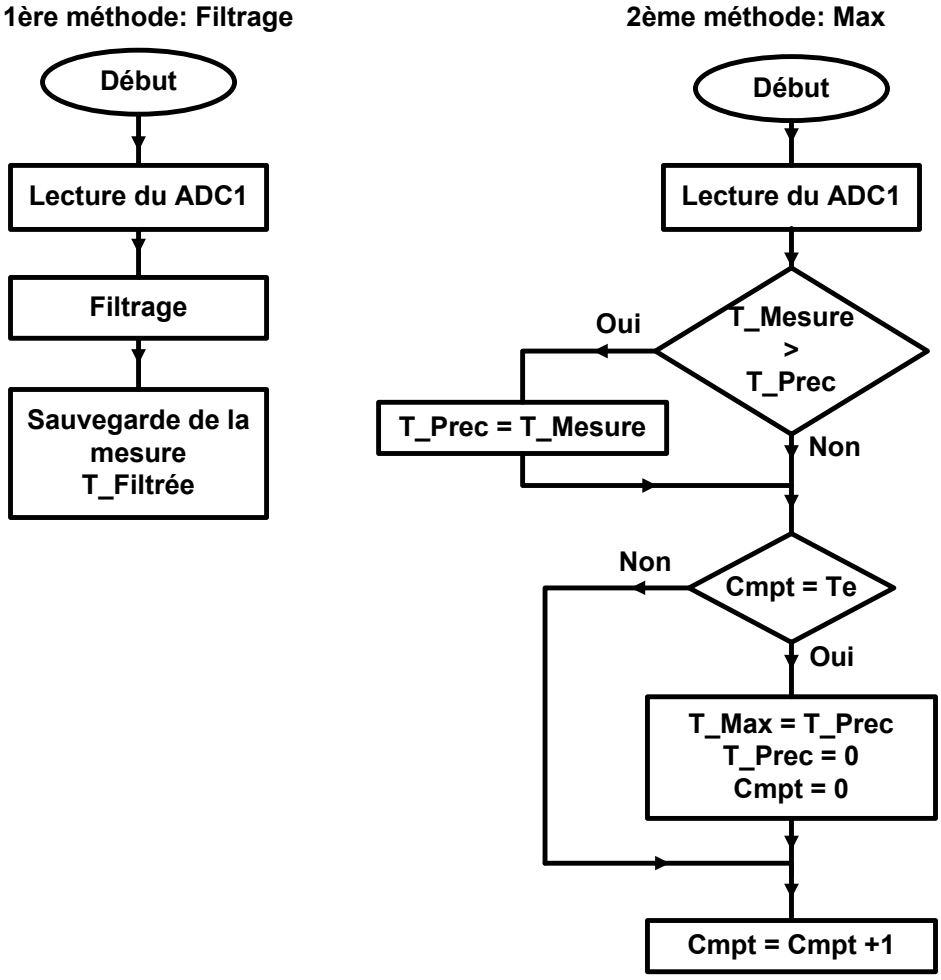


Figure III.B.16 Lecture de la mesure

La période d'échantillonnage Te, est donnée en millisecondes.

Les blocs Lecture Mesure, Filtrage et Max sont représentés dans le programme respectivement par les fonctions FC2, FC6 et FC11

III.B.1.5.3. Le comparateur

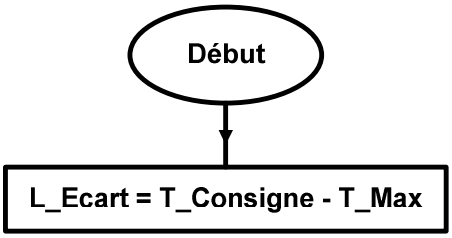


Figure III.B.17 Le comparateur

Ce bloc compare la température consigne à la mesure de température filtrée ou max (selon le choix validé).

Il est représenté dans le programme par la fonction FC, il est appelé et exécuté par l'alarme cyclique (OB35)

III.B.1.5.4. Régulation PID

On veut réguler la température du four autour d'une température donnée, en rendant le système plus rapide avec le moins de dépassement possible.

Un régulateur PID permet de respecter ce cahier de charge, le schéma de régulation général est le suivant :

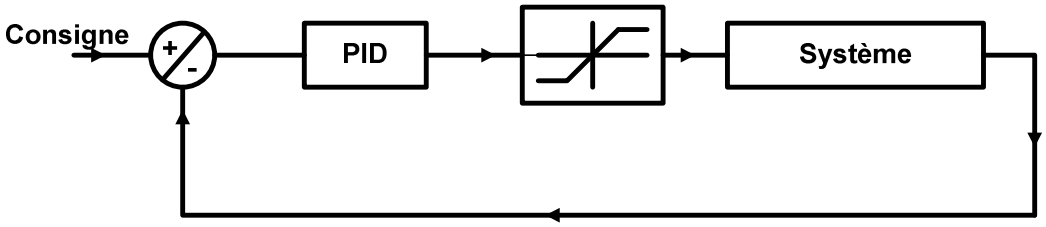


Figure III.B.18 Schéma de commande

Le signal de commande du four étant compris entre 0 et 10V, on doit ajouter à la sortie du régulateur un composant saturation pour limiter les variations de la commande à cet intervalle.

Pour le calcul des paramètres du régulateur, le plus simple est de trouver le contrôleur continu C(p) qui satisfait au mieux le cahier de charge, il suffit après de l'approximer par un autre numérique C(Z). Une des transformations les plus utilisées est celle qui consiste à remplacer la variable complexe 'p' par $\frac{1}{T_e} \cdot \frac{Z-1}{Z}$ avec T_e la période d'échantillonnage.

La structure du contrôleur numérique est représentée dans la figure B.15

C'est un contrôleur PID parallèle avec action dérivée filtrée, mené d'un anti-windup pour travailler le plus possible dans la zone linéaire.

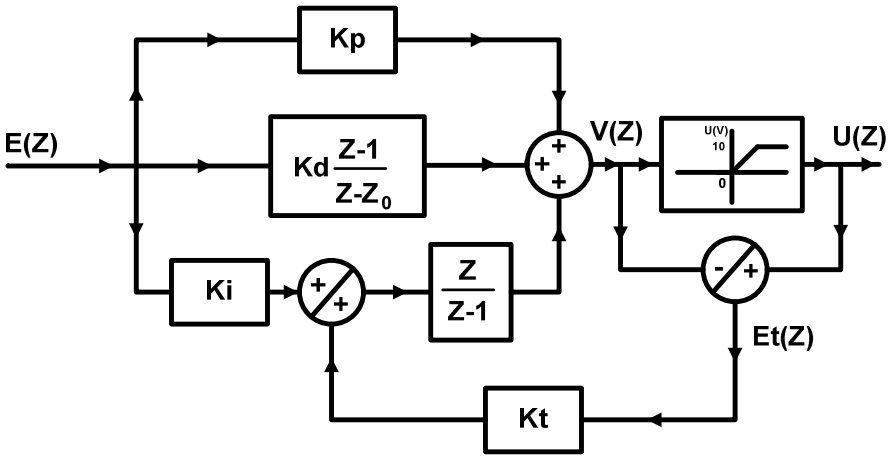


Figure III.B.19 PID parallèle avec antiwindup

E(Z) est la sortie du comparateur (consigne – mesure)

V(Z) est la commande calculé par l'algorithme de réglage PID (elle peut dépasser la valeur max ou min).

U(Z) est la commande après passage par le composant saturation.

Et(Z) est l'erreur entre ces deux commandes.

Le rôle de l'anti-windup est de réinitialiser la composante intégrale pour rendre le retour vers la zone linéaire plus rapide et donc travailler dans cette zone le plus longtemps possible

Explication :

$$U(Z) = \begin{cases} 10 & si\ V(Z) > 10 \\ V(Z) & si\ 0 \leq V(Z) \leq 10 \\ 0 & si\ V(Z) < 0 \end{cases}$$

1. Si la commande est dans la zone linéaire alors Et(Z) es t égale à 0, on aura simple PID parallèle, ce qui donne le schéma suivant :

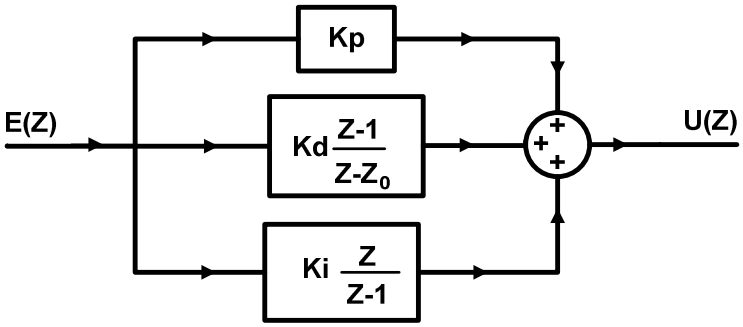


Figure III.B.20 PID sans l'antiwindup

2. Si la commande est supérieure à la commande max, alors Et(Z) est inférieur à 0. La composante intégrale va donc diminuer, pareil pour la commande V(Z), elle va diminuer jusqu'à ce qu'elle devient inférieure à la commande max. Elle retourne donc vers la zone linéaire.
3. Si la commande est inférieure à la commande min, alors Et(Z) est supérieur à 0. La composante intégrale va augmenter, pareil pour la commande V(Z), elle va augmenter jusqu'à ce qu'elle devient supérieure à la commande min. Elle retourne donc vers la zone linéaire.

Le gain K_t permet de contrôler la réinitialisation de la composante intégrale. La valeur recommandée est $K_t = \frac{1}{\sqrt{T_i \cdot T_d}}$

La commande V à l'instant k sera donnée par :

$$V_k = K_I \cdot x_{k-1} + K_{PID} \cdot e_k - K'_D \cdot e_{k-1} + K_t + E_{s\,k-1}$$

Avec : $x_{k-1} = x_{k-2} + e_{k-1}$, $x_{-1} = 0$ et $e_{-1} = 0$

$$K_{PID} = K_P + K_I + K_D$$

$$K'_D = K_D \cdot (1 - Z_0)$$

$$E_{s\,k-1} = E_{s\,k-2} + U_{k-1} - V_{k-1}, \quad E_{s-1} = 0$$

La commande U est donnée par :

$$U_k = \begin{cases} 10 & si\ V_k > 10 \\ V_k & si\ 0 \le V_k \le 10 \\ 0 & si\ V_k < 0 \end{cases}$$

L'algorithme du réglage PID est donné dans la figure ci-dessous

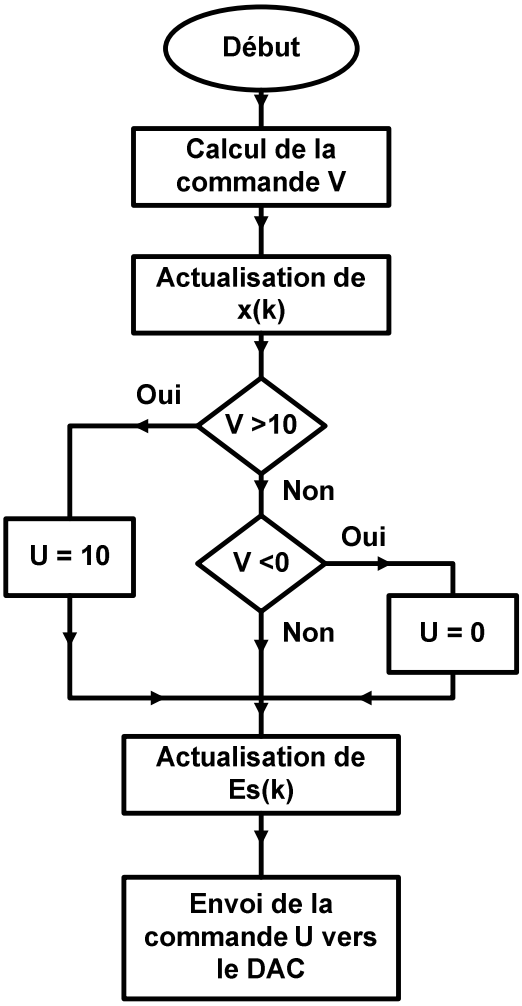


Figure Figure III.B.21 Algorithme de réglage PID

III.B.1.5.5. Régulation à Hystérésis ON-OFF

L'algorithme de réglage ON-OFF, consiste à délivrer une commande max ou min pour borner l'erreur entre la consigne et la mesure dans un intervalle $[-a,+a]$. Le schéma de commande est le suivant :

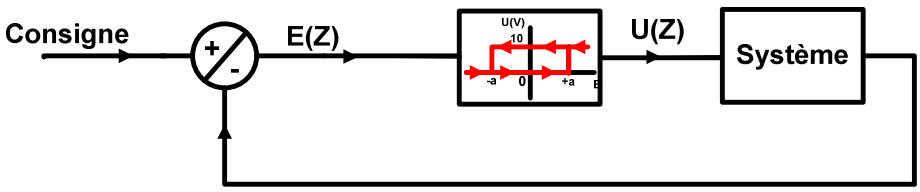


Figure III.B.22 Régulation ON-OFF

La commande U est donnée par :

$$U_k = \begin{cases} 10 & si\ e_k \geq 10 \\ U_{k-1} & si\ -a < e_k < +a \\ 0 & si\ e_k \leq -a \end{cases}$$

L'algorithme de calcul de la commande ON-OFF est donné dans la figure ci-dessous :

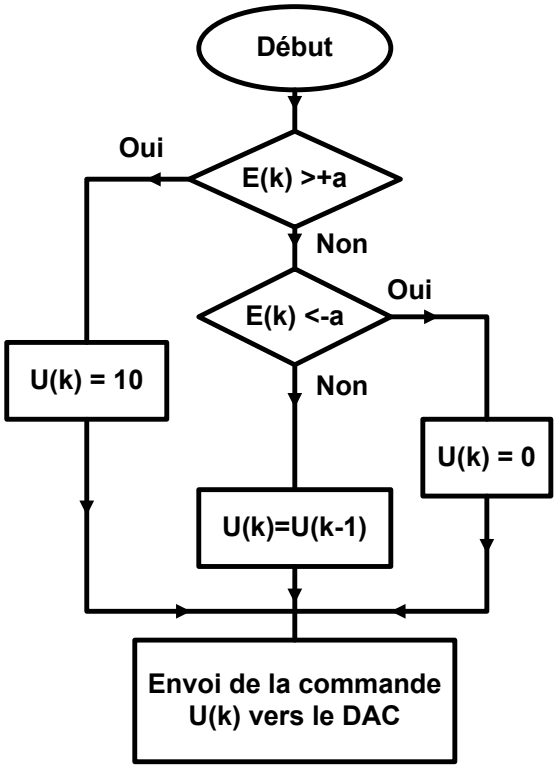


Figure III.B.23 Algorithme de réglage ON-OFF

III.B.1.6. Application de la commande

Nous avons appliqué la commande PID & ON-OFF pour réguler la température du four autour de 30°C.

III.B.1.6. 1. Commande PID

Pour le point de fonctionnement de 30°C, les paramètres du régulateur sont:

$K_p = 0.19$

$K_D = 10$

$K_I = 5 \cdot 10^{-4}$

$K_c = 8 \cdot 10^{-4}$ et $Z_0 = 0.2$

Nous avons effectué deux tests :

1. Température initiale 18°C, la réponse du système (température + signal de commande) est relevé à partir d'un oscilloscope à effet mémoire.
2. Température initiale 26°C, la réponse du système est relevé manuellement, puis visualiser par Matlab.

III.B.1.6.1.1. Premier test

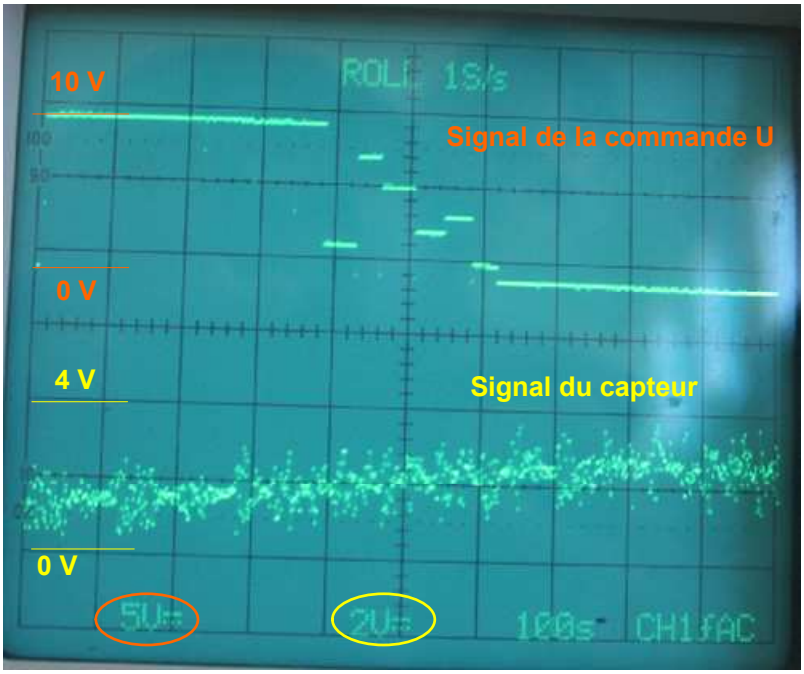


Figure III.B.24 Commande PID

On voit bien que la réponse est plus rapide qu'en boucle ouverte (pour la réponse en boucle ouverte voir figure B.10). A partir d'une température initiale de 18°C, le système a mis presque 600 s (soit 10 minutes) pour atteindre la consigne souhaitée. Malheureusement le signal du capteur est fortement bruité, pour mieux apercevoir l'évolution de la température il faut se référer au test 2.

III.B.1.6.1.2. Deuxième test

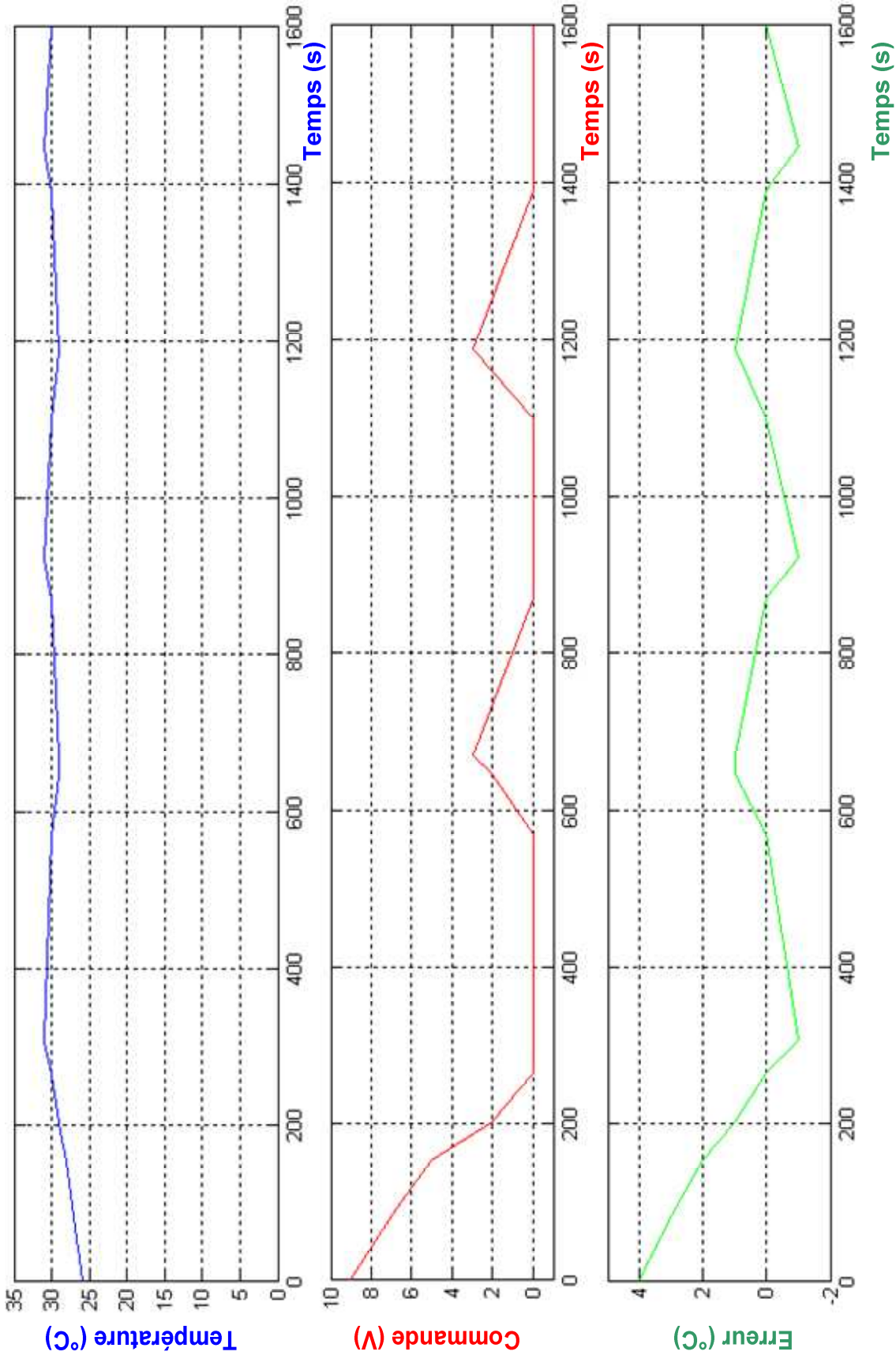


Figure III.B.25 Réponse du système Régulation PID

Dans la figure précédente, la réponse de température montrée est celle obtenu après filtrage du signale du capteur. On voit bien que le dépassement est très minime, dans le régime établi la température varie entre 31 et 29°C, soit un dépassement de plus ou moins 3.33%.

Le four n'a pas un système de refroidissement, ce qui fait que dans le cas de dépassement de la consigne, le signal de commande du four s'annule, mais la température continue a augmenté à cause de l'échange thermique existant entre l'air comprimé et les parois du four, ensuite après un certain temps à cause de l'inertie du système la température commence a diminué jusqu'à ce qu'elle devient plus petite que la consigne, la commande devient alors non nulle et la température augmente et ainsi de suite, ce qui donne les oscillations observés dans la figure B.17. L'erreur reste bornée entre -3.33% et +3.33% de la consigne.

III.B.1.6.2. Commande ON-OFF

L'intervalle de variation de l'erreur choisit est [-3, +3] °C

Nous avons effectué deux tests :

1. Température initiale 18°C, la réponse du système est relevée à partir d'un oscilloscope à effet mémoire.
2. Température initiale 26°C, la réponse du système est noté manuellement, puis visualiser par Matlab.

III.B.1.6.2.1. Premier test

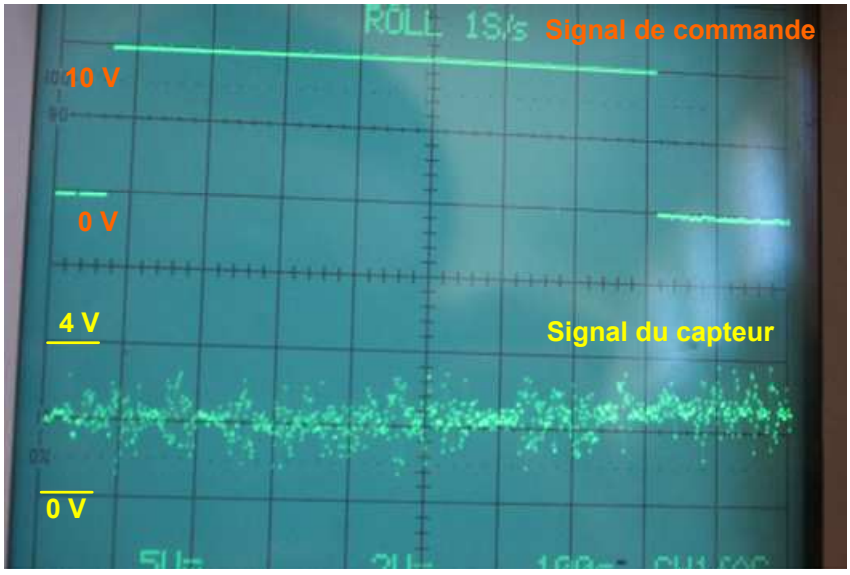


Figure III.B.26 Régulation à Hystérésis de la température

La figure ci-dessus montre bien la variation de la commande entre sa valeur min et sa valeur max. La mémoire de l'oscilloscope étant très limité (maximum 1000s) la figure ne montre qu'une seule oscillation. Le deuxième test montre beaucoup mieux l'évolution de la température.

III.B.1.6.2.2. Deuxième essai

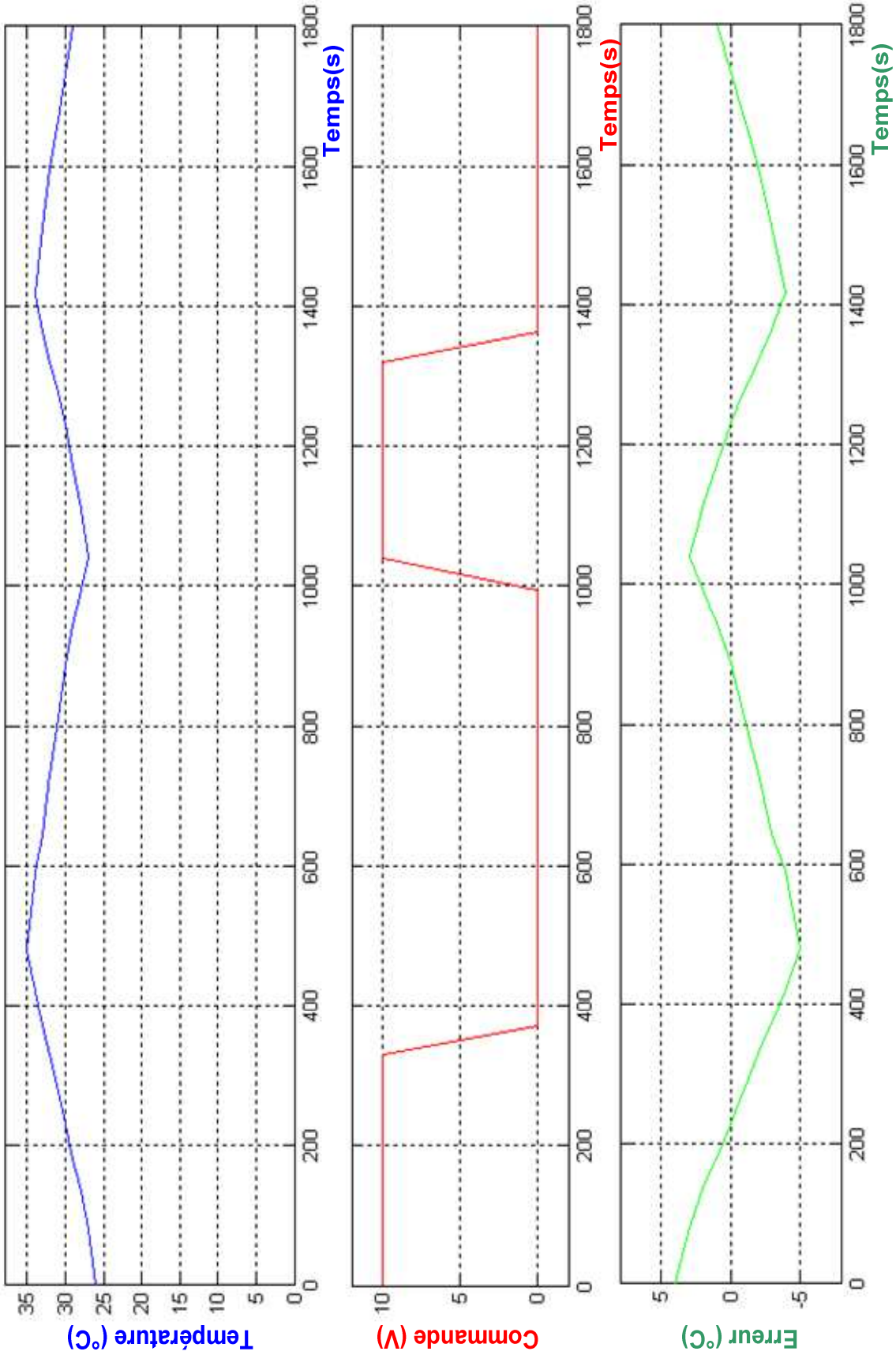


Figure III.B.27 Régulation ON-OFF

La figure B.17 montre l'évolution de la température, la commande et l'erreur pendant une durée de 30 minutes.

Bien que nous ayons borné l'erreur entre la consigne et la mesure dans l'intervalle [-3, +3] °C lors de la conception du régulateur, celle-ci varie entre -5 et +3°C, ceci est du à l'inertie du système (l'air et le liquide contenu dans le four) et au fait qu'il n'y a pas un système de refroidissement.

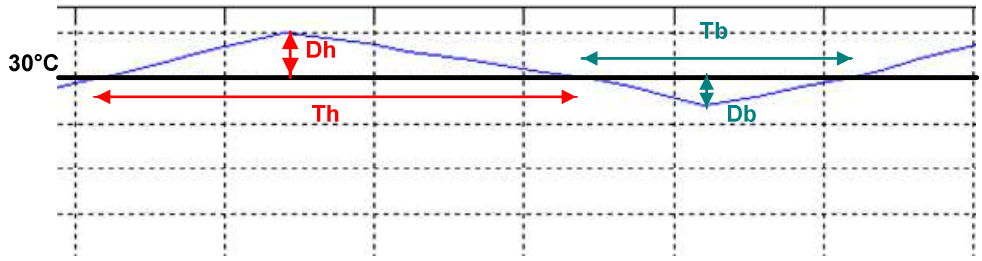


Figure III.B.28 Variation de la température autour du point de fonctionnement

D'autre part, la durée dans la quelle la sortie reste supérieur à la température consigne (Th) est plus grande que celui ou elle reste inférieur (Tb).

Pour minimiser ces intervalles et ainsi les dépassements, il faut chercher l'intervalle optimal de variation de l'erreur [-a, +a], dans le quel on assure le bon fonctionnement de l'actionneur.

On doit satisfaire deux conditions :

- Lors de la commande max : il faut que le temps Th soit largement supérieur au temps du passage de la sortie analogique d'une tension nulle à une tension max, et au temps de fermeture du triac.
- Lors de la commande min : il faut que le temps Tb soit largement supérieur au temps du passage de la sortie analogique d'une tension max à une tension nulle, et au temps d'ouverture du triac.

Les systèmes thermiques (changement de température du four) ayant une dynamique très lente par rapport aux systèmes électriques, les conditions citées en dessus sont largement satisfaits.

III.B.1.6.3. Comparaison entre la régulation ON-OFF et PID

D’après les résultats obtenu, on remarque que :

- Le correcteur ON-OFF donne une réponse plus rapide, le régime établi est obtenu après 200s (presque 3 minutes), alors qu’avec le PID le régime établi est obtenu après 300s (5 minutes).
- Le PID donne une réponse plus lisse, les oscillations sont plus importantes avec le correcteur ON-OFF.
- La commande présente des variations importantes avec le ON-OFF, avec le PID elle est plus fine.

2ème Application

Commande d'un moteur à courant continu

III.B.2. Commande d'un moteur à courant continu

Cette application a pour but de montrer les possibilités qu'offrent les automates programmables pour la commande des systèmes rapides (dynamique rapide)

C'est une commande en boucle ouverte de la vitesse d'un moteur à courant continu avec une boucle de mesure de la vitesse et de la position.

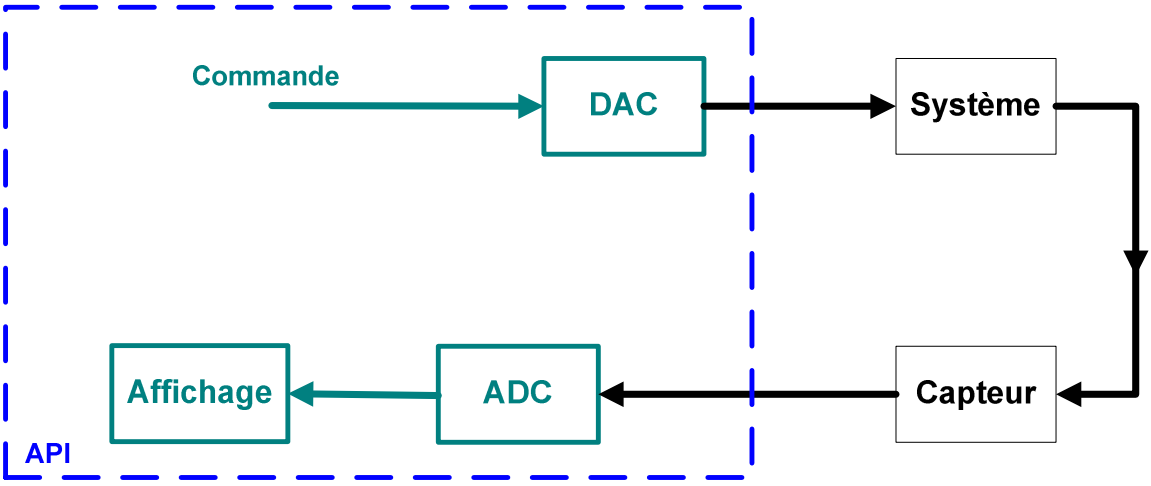
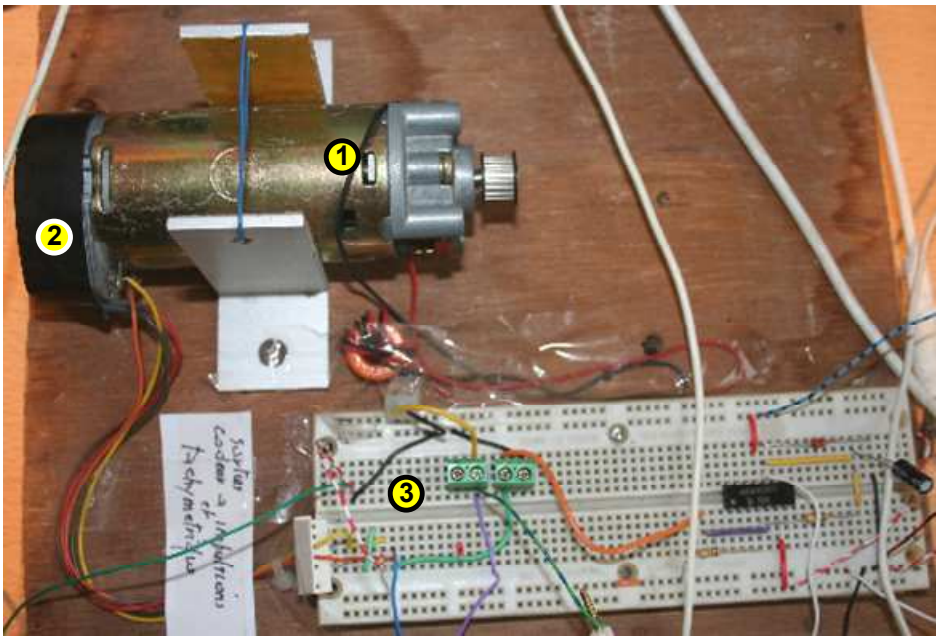


Figure III.B.29 Commande en boucle ouverte avec boucle de mesure

Le système est constitué d'un moteur, une tachymétrie et d'un circuit d'amplification et de filtrage.



(1). Le moteur
(2). Tachymètre
(3). Circuit d'amplification

Figure III.B.30 Le moteur et le circuit d'amplification

III.B.2.1. Rappels

III.B.2.1.1. Moteur à courant continu

Les moteurs à courant continu sont des machines réversibles. Il transforme l'énergie électrique en une énergie mécanique et vice versa.

Ils comportent :

- Le stator (inducteur) : c'est la partie fixe, constitué d'un aimant permanent ou d'un électroaimant, elle crée un champ magnétique dirigé vers l'axe du rotor.
- Le rotor (induit) : c'est la partie mobile, constitué d'un cylindre avec des spires à la périphérie.
- Le collecteur : c'est partie mobile constituées de lames, reliées aux spires.

L'excitation peut être réalisé de diverses manières : séparée, shunt, série, compound et par aimants permanents. Ce dernier est le plus utilisé pour les petits moteurs

Pour le moteur à aimants permanents, le schéma de fonctionnement est représenté dans la figure ci-dessous :

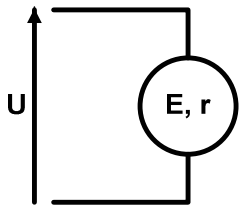


Figure III.B.31 Moteur CC à aimants permanents

U est la tension d'excitation.

$U = r \cdot I + E$ le flux étant constant, on peut écrire que $E = K \cdot \omega$

ω étant la vitesse de rotation.

III.B.2.1.2. Tachymétrie

La tachymétrie ou la génératrice tachymétrique délivre une tension proportionnelle à sa vitesse de rotation. Pour la génératrice à courant continu l'excitation est assurée par des aimants permanents.

Le schéma de principe est le suivant :

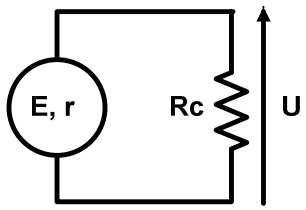


Figure III.B.32 Schéma de fonctionnement d'une génératrice tachymétrique

R_C est la résistance de charge.

$E = r \cdot I + U$ avec $E = K \cdot \omega$ et $U = R_C \cdot I$

D'où $U = \frac{K \cdot R_C}{r + R_C} \cdot \omega$ ➔ $U = K_e \cdot \omega$

K_e est la constante de la f.e.m. il est donné en $V/Tr/mn$. C'est le rapport entre la vitesse de rotation est la tension délivrée. La caractéristique tension-vitesse est donc linéaire (en régime permanent).

III.B.2.2. Description du matériel

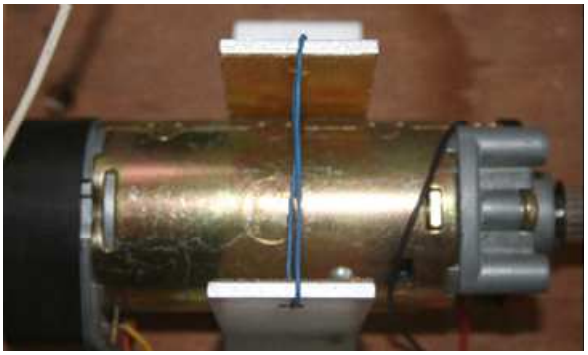


Figure III.B.33 Le moteur

C'est un moteur continu à aimants permanents, qui tourne à des vitesses angulaires entre 0 et 1400 tr/mn pour des tensions d'alimentation comprissent entre 0 et 12V. La Tachymétrie délivre une tension comprise entre 0 et 2.5V pour des vitesses angulaires entre 0 et 1400 Tr/mn

($K_e \cong 1.8 \cdot 10^{-3} V/Tr/mn$).

L'entrée analogique de l'automate accepte une tension entre 0 et 10V, on doit donc faire une amplification du signal de la tachymétrie pour avoir une meilleure précision.

Le schéma électrique du circuit d'amplification est présenté dans la figure ci-dessous :

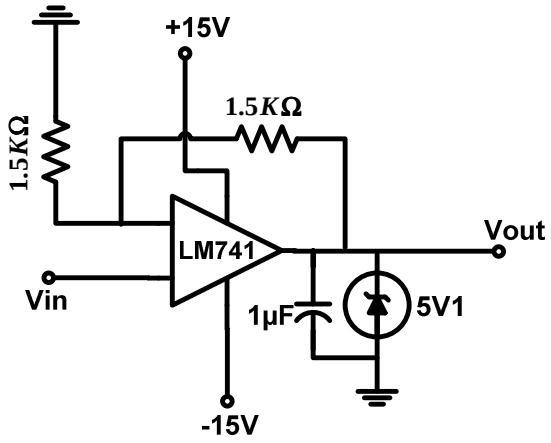


Figure III.B.34 Schéma électrique du circuit d'amplification

La tension d'entrée Vin, est amplifie avec un gain de 2, le signal de sortie est filtré grâce au condensateur et la diode ZENER sert à limiter la tension de sortie a 5.1V.

L'alimentation de ce circuit est assurée par un générateur de tension continu

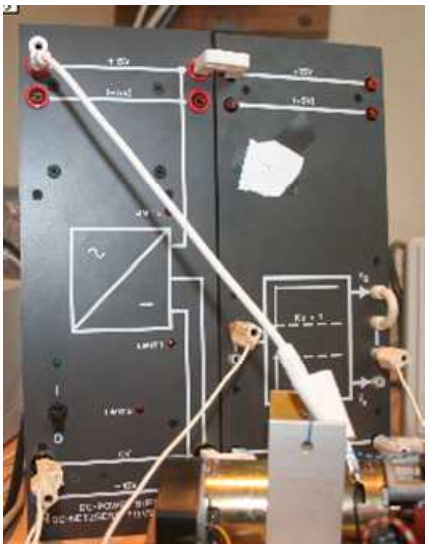


Figure III.B.35 Circuit d'alimentation

III.B.2.3. Description du programme

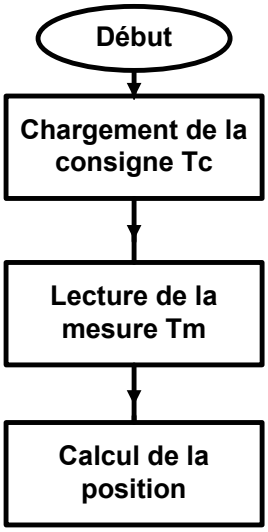


Figure III.B.36 Organigramme de la commande en boucle ouverte

Chacun de ces blocs est représenté dans le programme par une fonction FC.

Ces blocs sont appelés et exécutés dans le bloc d'organisation OB35 (Alarme Cyclique). Nous avons fixé la période d'exécution de l'OB35 à 1ms (Période d'échantillonnage) pour qu'elle soit assez petite devant la période minimale du système (presque 42ms).

III.B.2.3.1. Chargement de la consigne

La consigne est introduite manuellement dans le programme (hors ligne). Elle est donnée en Tr/mn, il faut la convertir en valeur décimale entre 0 et 27600, ce qui correspond à la sortie analogique de l'automate (DAC) à une tension entre 0 et 10V.

Voici l'organigramme ci-dessous l'organigramme de la fonction chargement consigne :

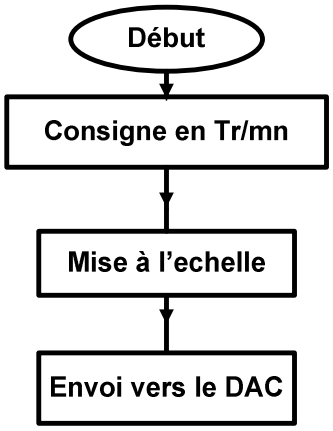


Figure III.B.37 Chargement de la consigne

III.B.2.3.2. Lecture de la vitesse

La génératrice tachymétrique délivre une tension entre 0 et 2.5V pour une vitesse angulaire entre 0 et 1400Tr/mn. Cette tension sera amplifier (x2) puis connecté à l'entrée analogique de l'automate.

Comme pour le chargement de la consigne une mise à l'échelle doit être mise en place.

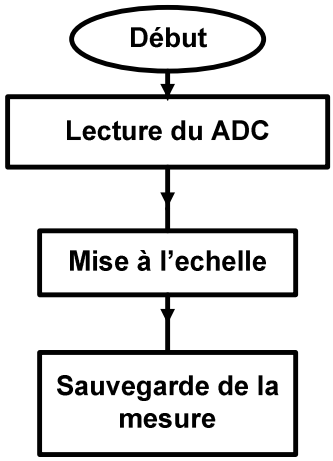


Figure III.B.38 Lecture de la mesure

III.B.2.3.3. Calcul de la position

Pour calculer la position, on suppose que la vitesse angulaire du moteur est constante entre deux instants d'échantillonnage. Ce qui nous permet d'écrire la relation suivante :

$\theta_k = \theta_{k-1} + \omega_k$. θ_k est la position à l'instant k, elle est donnée en Tr (tour). ω_k est la vitesse angulaire à l'instant k, elle est donnée en Tr/ms (tour par millisecondes)

La valeur de θ_k peut nous renseigné sur le nombre de tours effectués, la vitesse angulaire moyenne et la position angulaire actuelle :

- Le nombre de tours effectués à l'instant k : $N_k = [\theta_k]$
- La vitesse angulaire moyenne : $\omega_{moy} = \frac{\theta_k}{k} \cdot 6 \cdot 10^4\ Tr/mn$
- La position angulaire à l'instant k: $\varphi_k = (\theta_k - N_k) \cdot 360\ °$

Ces données seront très utiles pour une éventuelle commande en position du moteur.

Voici ci-dessous l'organigramme du calcul :

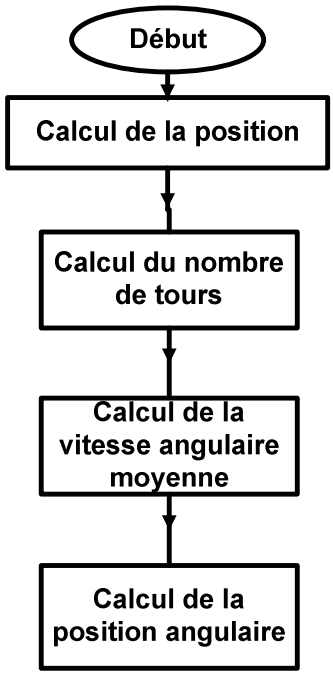


Figure III.B.39 Calcul de la position

III.B.2.4. Application de la commande

Nous avons appliqué cette commande pour une consigne en vitesse égale à 700 Tr/mn.

Le signale de commande est représenté dans la figure suivant :

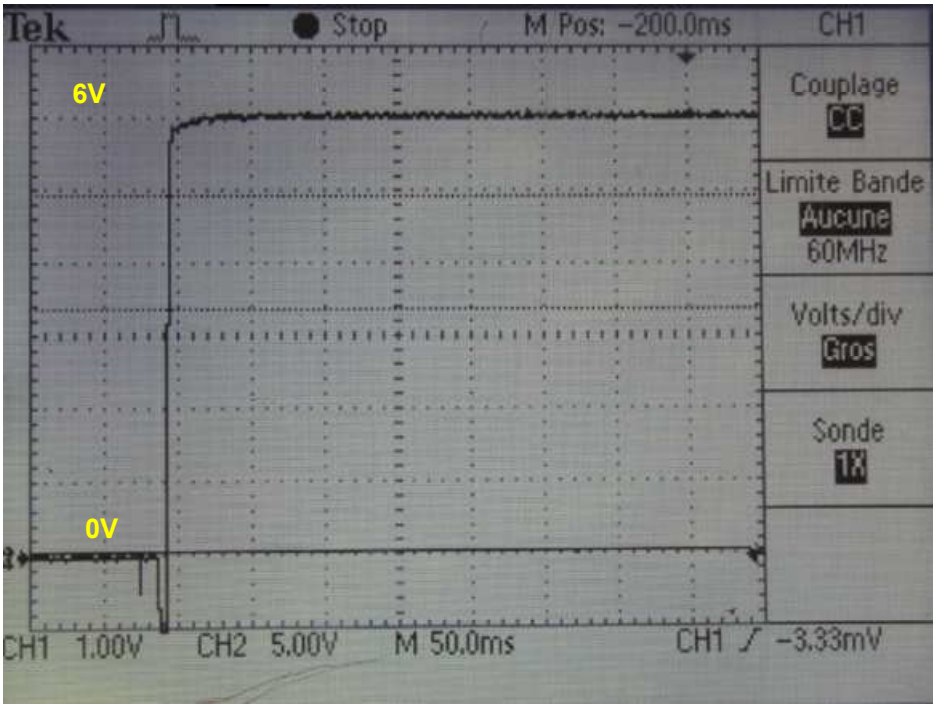


Figure III.B.40 Signal de commande

Cette figure a été prise à l'aide d'un oscilloscope branché aux bornes de la sortie analogique de l'automate.

La commande admissible du moteur étant comprise entre 0 et 12V, pour une vitesse angulaire entre 0 et 1400 Tr/mn, la consigne de 700 Tr/mn donnée à l'automate correspond parfaitement à la moitie du signal de commande maximale (6V).

La réponse du système est prélevée à l'aide d'un oscilloscope branché aux bornes de la génératrice tachymétrique. Ce signal est présenté dans la figure suivante :

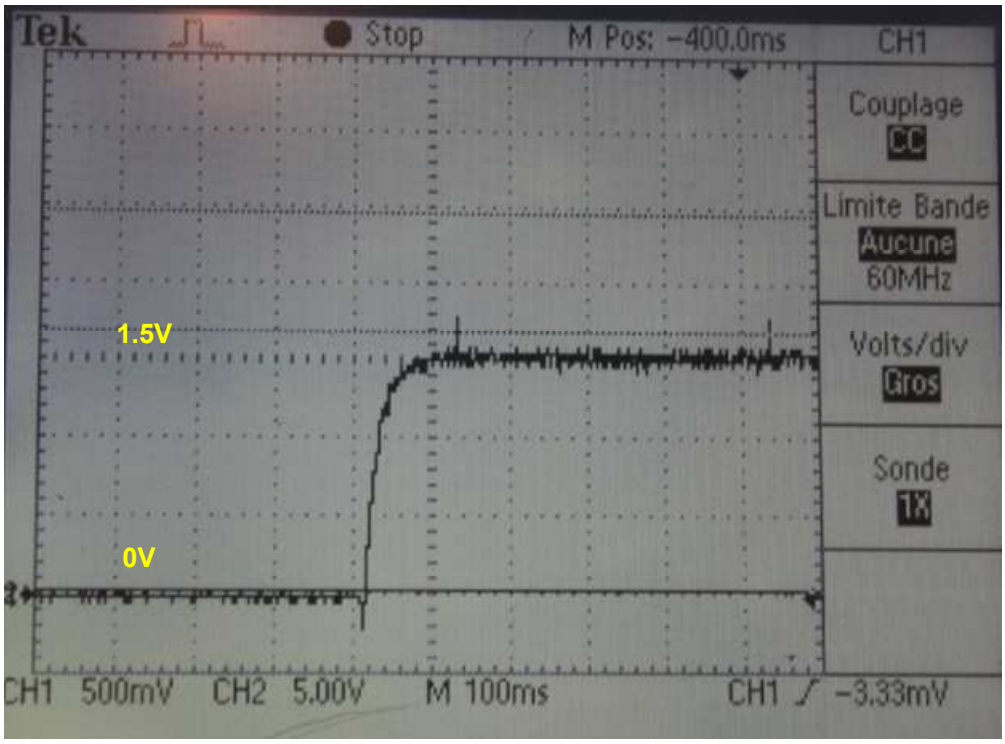


Figure III.B.41 Signal de sortie

On voit bien que le signal se stabilise autour de la tension 1.5V, ce qui correspond à une vitesse angulaire égale à 840 Tr/mn.

On remarque que l'erreur statique est un très grande 20%. Pour l'annulé il faut introduire un correcteur PI.

Après 6 secondes de marches, on a peu prélevé les données suivantes de l'automate :

- Positions $\theta = 83.6825\ Tr$
- Nombre de tours effectués : N = 83 Tr
- Position angulaire finale : $\varphi = 245.7°$
- Vitesse angulaire moyenne : $\omega_{moy} = 836.825\ Tr/mn$

CONCLUSION GENERALE

Conclusion générale

Munis d'un logiciel très performant, les automates programmables de la série S7-300 forment des unités de traitement et de commande d'une grande flexibilité. En effet simple à utiliser et dotée d'une interface graphique très intuitive, le STEP7 permet d'exploiter de manière très optimale les différentes CPU de la gamme.

Ce projet nous a permis de voir les différentes possibilités offertes par les automates programmables : simulation des systèmes continus, calcul numérique, commande des systèmes continus et séquentiels.

Les capteurs utilisés nous ont permis de voir la difficulté d'acquérir des informations fiables sur la mesure ou la sortie, des capteurs peu fiable, très sensibles aux bruits et parasites, ce qui nous a amené à introduire des filtres numérique ou physique.

La communication et le transfert d'informations via un réseau, rendront un système automatisé plus souple et plus performant par la diminution du câblage, il sera judicieux aussi de concevoir des plateformes logiciels ou matériels pour la supervision, cela permettrait de simplifier le diagnostic et le dépannage.

Il serait aussi très utile d'implémenter des algorithmes de régulation numérique, tel que la commande RST, ainsi que des algorithmes plus complexes comme la commande adaptative à modèle interne.

Seulement la question à se poser est :

- Est-ce que le temps de traitement et la mémoire offerte par cette gamme d'automate sont suffisants pour accomplir de telles taches de commande ?

Bibliographie

[B01] Jack HUGH “Automating Manufacturing Systems with PLC’s “ (1993-2005)

[B02] Michel BERTRAND « Automates programmables industriels »

École Nationale Supérieure d’Arts et Métiers ENSAM

Centre d’Enseignement et de Recherche de Lille

[B03] GILLES MICHEL « Architecture et application des automates programmable »

DUNOD, Paris (1988)

[B04] ANDRE SIMON « Automates programmables industriels : Niveau1 » EYROLLES

Paris (1991)

[B06] Technique de l’ingénieur : LE GRAFCET, Langage de programmation pour API

Norme IEC 1131-3.

[B07] Michael & William L. Luyben « Essentials of Process Control », McGraw-HILL, International Editions 1997

Manuels :

[M01] SIEMENS « Mise en route STEP7 V5.3 », Réf 6ES7810-4CA06-8CA0, SIMATIC 2004

[M02] SIEMENS « Langage CONT pour SIMATIC S7-300/400 », Réf 6ES7810-4CA06-8CR0, SIMATIC 2004

[M03] SIEMENS « Langage LOG pour SIMATIC S7-300/400 », Réf 6ES7810-4CA06-8CR0, SIMATIC 2004

[M04] SIEMENS « S7-300 and M7-300 Programmable Controllers Module Specifications », Réf 6ES7- 3988-AA03-8BA0

[M04] SIEMENS « Caractéristiques des CPU, CPU 312 IFM – 318-2 DP », Réf 6ES7 312-5AC02-0AB0

ملخص:

هدا العمل يسمح باضها ر مختلف الإمكانيات التي يعطيهSTEP7لتمثيل الأنظمة المستمرة و المتقطعة, للحساب الرقمي و التحكم التدريجي و المستمر في الوقت الحقيقي من اجل إيجاد حلول للأعمال الاوتوماتيكية و للمراقبة بواسطة آلية.

من الصنف 7 ألدى ينتمي إلى المجموعة الألمانية SIEMENS

STEP7 هو مجال تمثيلي رسمي سهل الاستعمال و البرامج فيه منظمة في علب هدا ما يسهل تقسيم المشاكل و إعادة البرمجة

إن مختلف التطبيقات أنجزت في مخبر الآليات في المعهد الوطني للمحروقات في جامعة بومرداس و هي تبين الإلية S7-313 -أنها تعطى إمكانيات كبيرة في التحكم .

كلمات مفتاحية:

STEP7آلية مبرمجة, تنظيم درجة حرارة فرن ,التحكم في محرك كهربائي .

Résumé :

Ce travail permet de montrer les différentes possibilités qu'offre le STEP7, pour la simulation des systèmes continus, le calcul numérique, la commande séquentielle et la commande continue en temps réel, afin d'implémenter des solutions et des taches d'automatisation et de contrôle sur l'automate de la série 7 de la firme allemande SIEMENS S7-313.

Le STEP7 est une interface graphique facile à utiliser, les programmes sont organisés dans des blocs, ce qui facilite la subdivisant des problèmes et la mise à jour.

Les différentes applications ont été réalisées dans un laboratoire au niveau du département d'automatisation de l'Institue Nationale d'Hydrocarbure. Elles montrent que l'automate S7-313 offre une large possibilité de commande et d'automatisation.

Mot clef : STEP7, automate programmable S7-313, régulation de la température d'un four, commande d'un moteur cc.

Abstract:

This work shows the various possibilities that the STEP7 offers, for the simulation and control of the continuous and sequential systems, in order to implement solutions of automation and control on the programmable logical controller S7-313.

The STEP7 is an easy graphical interface; the programs are organized in blocks, which facilitate the subdividing of the problems and the update.

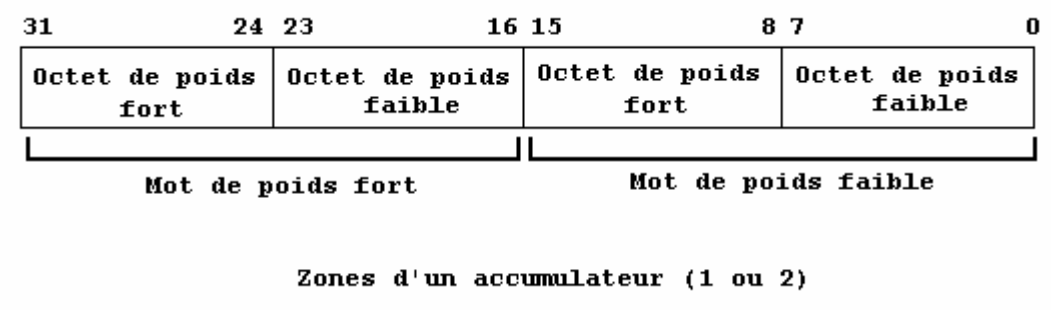
The various applications were carried out in laboratory automation in the Hydrocarbon National Institute. They show how it easy to control process with the S7-313.

Keyword: STEP7, programmable logical controller S7-313, temperature control of a furnace, orders of a DC motor.

Les Registres de la CPU

1. Les Accumulateurs

La plus part des CPU, contiennent deux accumulateurs de 32 bits, mais il existe d'autres qui contiennent quatre. Ils sont d'usage général, permettant de traiter l'octet, le mot et le double mot. On peut charger dans l'accumulateur des constantes ou des valeurs comme opérandes depuis la mémoire et les combiner. On peut aussi transférer le résultat d'une opération de l'accumulateur 1 à une adresse d'opérande.



Zones d'un accumulateur (1 ou 2)

Le mécanisme d'empilage pour la gestion des accumulateurs se présente comme suit :

- Une opération de chargement opère toujours sur l'accumulateur 1et sauvegarde l'ancien contenu dans l'accumulateur 2.
- Une opération de transfert ne modifier pas le contenu des accumulateurs 1 et 2 (à l'exception des opérations TAR1 et TAR2).
- L'opération TAK permute le contenu des deux accumulateurs.

2. Pile des parenthèses

La pile des parenthèses est un octet utilisé par les combinaisons d'expressions entre parenthèses U(, O(, X(, UN(, ON(et XN(. Ces opérations sauvegardent le résultat logique du RLG en cours dans la plie des parenthèses et commence une nouvelle séquence combinatoire.

Cette pile peut contenir jusqu'à 7 entrées. Une entrée dans la pile des parenthèses se compose des bits RLG, RB et OU du mot d'état ainsi que d'un code de fonction précisant de quelle combinaison il s'agit (U, O, X, UN, ON ou XN)

L'opération) ferme une expression entre parenthèse et exécute les fonctions suivantes :

- Elle extrait une entrée de la pile des parenthèses.
- Elle restaure les bits OU et RB.
- Elle définit le nouveau RLG en combinant le RLG en cours (c'est-à-dire le RLG de l'expression entre parenthèses) à celui de l'entrée de la pile conformément au code de fonction.

7	6	5	4	3	2	1	0
0	0	RB	RLG	OU	Code de fonction		

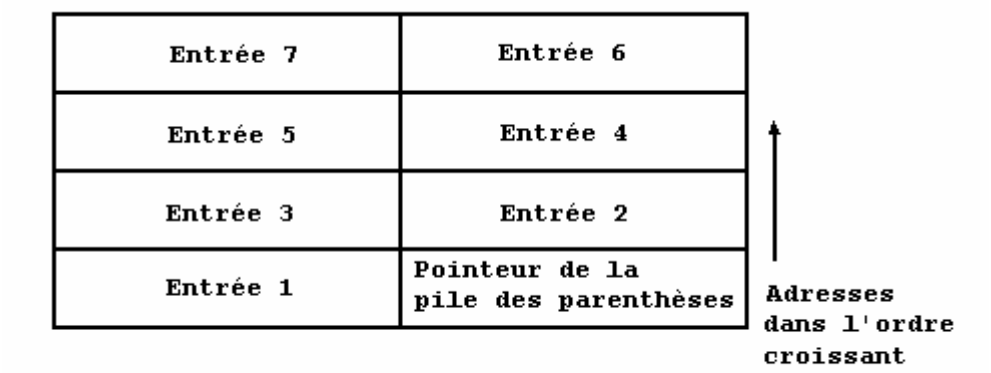
Structure d'une entrée dans la pile des parenthèse

À l'aide du code de fonction, l'opération) détermine l'opération devant être utilisée pour la combinaison du RLG de l'expression entre parenthèse et du RLG de l'entrée de la pile des parenthèses. Le tableau suivant présente les différentes combinaisons.

Fonction d'opération	Bit 2	Bit 1	Bit 0
U(	0	0	0
UN(	0	0	1
O(	0	1	0
ON(	0	1	1
X(	1	0	0
XN(	1	0	1

Code d'octet de la pile des parenthèses

Le pointeur de la pile des parenthèses indique le nombre d'entrées existant dans la pile des parenthèses.



Structure d'une pile de parenthèses avec entrées et pointeur

3. Mot d'état

Le mot d'état est un registre qui contient des bits qui donnent des informations sur l'opération précédente.

15 9	8	7	6	5	4	3	2	1	0
.....	RB	BI1	BIO	DEB	DM	OU	ETAT	RLG	/PI

Organisation du mot d'état

1. **/PI :** le bit 0 du mot d'état est appelé première interrogation, il indique si le programme va entamer une nouvelle séquence de combinaisons. Une séquence de combinaisons se termine toujours par une opération de sortie, par une opération de saut relative ou par une expression entre parenthèses. Ces opérations remettent le bit /PI à 0.
2. **RLG :** le bit 1 du mot d'état, appelé résultat logique, contient le résultat logique d'une opération combinatoire sur bits ou d'une opération de comparaison.
3. **ETAT :** ce bit est exploité pendant le test du programme, il indique l'état logique de l'opérande aux quel accède l'opération en cours. Ce bit n'a pas de signification pour les opérations qui n'accèdent pas à la mémoire, ces opérations mettent le bit d'état à 1.
4. **OU :** ce bit est nécessaire lorsque on exécute une combinaison ET avant OU à l'aide de l'opération O. Une combinaison peut contenir les opérations U, UN, U(, UN(, NOT et). le bit OU indique à ces opération qu'une combinaison ET exécutée précédemment à fournie le résultat logique 1, anticipant ainsi le résultat da la combinaison OU. Toute autre opération traitant des bits remet le bit OU à 0.
5. **DEB :** Bit de débordement, il indique s'il y a eu un débordement à la suite d'une opération arithmétique ou d'une opération de comparaison sur des nombres à virgule flottante.
6. **DM :** bit de débordement mémorisé, il est mis à 1 en même temps que le DEB, il reste à une fois l'erreur corrigée. Il indique ainsi si une erreur s'est produite dans les opérations précédentes.
7. **RB :** résultat binaire, ce bit permet d'exploiter le résultat d'une opération combinatoire sur mots comme un résultat binaire. Ainsi il protège le contenu du RLG contre toute modification apportée par une opération sur mots.
8. **BI1 et BI0 :** ces deux bits donnent des informations à la suite d'une opération arithmétique, une opération s de comparaison ou des opérations de décalage et de rotation. Les tableaux suivants présentent les différents états des bits BI1 et BI0 :

Après des opérations arithmétiques (sans débordement)

BI1	BI0	Description
0	0	Résultat = 0
0	1	Résultat < 0
1	0	Résultat > 0

Après des opérations arithmétiques sur des entiers (avec débordement)

BI1	BI0	Description
0	0	Débordement de plage négative pour l'addition
0	1	Débordement de plage négative pour la multiplication ou de plage positive pour l'addition et la soustraction.
1	0	Débordement de plage positive pour la multiplication et la division ou de plage négative pour l'addition et la soustraction.
1	1	Division par zéro.

Après des opérations arithmétiques sur des entiers (avec débordement)

BI1	BI0	Description
0	0	Dépassement bas graduel
0	1	Débordement de plage négative
1	0	Débordement de plage positive
1	1	Nombre à virgule flottante illicite

Après des opérations de comparaison

BI1	BI0	Description
0	0	Accumulateur 2 = Accumulateur 1
0	1	Accumulateur 2 < Accumulateur 1
1	0	Accumulateur 2 > Accumulateur 1
1	1	L'Accumulateur 1 ou 2 est un nombre à virgule flottante illicite

Après des opérations de rotation et de décalage

BI1	BI0	Description
0	0	Bit décalé en dernier = 0
1	0	Bit décalé en dernier = 1

Après des opérations combinatoires sur mots

BI1	BI0	Description
0	0	Résultat = 0
1	0	Résultat <> 0

5. Registre d’adresse

Il y a deux registre d’adresse 1 et 2 de 32 bits, ils contiennent les adresses intrazone ou interzone pour les opérations utilisant l’adressage indirect par registre.

LIST

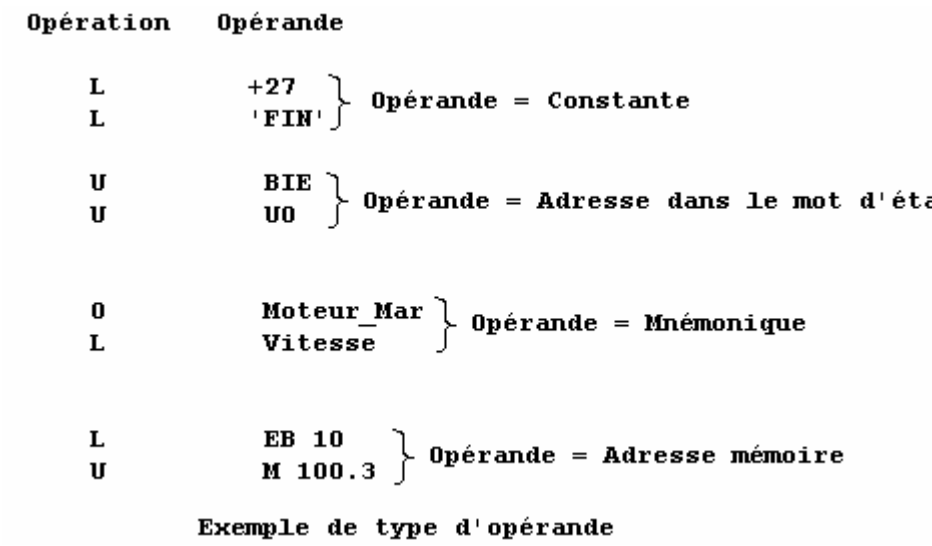
1 Structure des instructions

Selon leurs structures, les instructions appartiennent à l'un des deux groupes suivants :

- Instructions constituées d'une opération seule. (exemple SET, NOT…)
- Instructions contenant une opération et un opérande.

L'opérande d'une instruction peut être une donnée ou l'adresse de la donnée, elle peut être un nom symbolique ou une désignation absolue.

Exemple :



2 Type d'adressage

2.1 Adressage immédiat

Dans l'adressage immédiat, l'opérande fourni est la donnée elle-même.

```
Par exemple:
L    +23         // charger le nombre +23 dans l'accumulateur 1
L    'ABCD'      // charger les caractères ASCII "ABCD"
                 //  dans l'accumulateur 1

OW   W#16#A320   // OU mot
```

Adressage immédiat

2.2 Adressage direct

Dans l'adressage direct, l'opérande est constitué de deux parties :

- Un identificateur d'opérande (par exemple AW pour mot de sortie).
- Une adresse exacte à l'intérieur de la zone mémoire indiqué par l'identificateur d'opérande.

```
Par exemple:

U    E 0.0   // effectuer une combinaison ET sur le
             // bit d'entrée E 0.0
=    M 10.5  // affecter le RLG au bit de mémento 10.5
T    AW 64   // transferer le contenu de l'accumulateur 1
             // dans le double mot de sortie AW 64

                  Adressage direct
```

2.3 Adressage indirect en mémoire

Dans l'adressage indirect, l'opérande est constitué de deux parties :

- Un identificateur d'opérande (par exemple EB pour octet d'entrée).
- l'un des pointeurs suivants :
  - un mot contenant le numéro d'une temporisation (T), d'un compteur (Z), d'un bloc de donnée (DB), d'une fonction (FC) ou d'un bloc fonctionnel (FB).
  - un double mot contenant l'adresse exacte d'une valeur à l'intérieur de la zone mémoire indiqué par l'identificateur d'opérande.

Ce mot ou double mot peut se situer dans l'une des zones suivantes : mémento (M), bloc de données (DB), bloc de donnée d'instance (DI) ou donnée locale (L).

```
Par exemple :

U   E[MD2]       // effectuer une combinaison ET du bit d'entrée dont
                 // l'adresse exacte figure dans le double mot de
                 // mémento MD2
=   DIX[DBW2]    // affecter le bit RLG au bit de donnée d'instance
                 // dont l'adresse exacte figure dans le mot de
                 // donnée DBD2.

                   Adressage indirect en mémoire
```

2.4 Adressage indirecte intrazone par registre

Dans l'adressage indirect intrazone par registre, l'opérande a la forme :

Identificateur d'opérande [Code de registre d'adresse, Pointeur de décalage]

L'avantage de ce mode d'adressage est qu'on peut modifier l'opérande de l'instruction en mode dynamique (pendant le traitement du programme).

```
Par exemple :   =   A[AR1, P#1.1]

Contenu du registre d'adresse AR1: 9.7    octet  9, bit 7
+ déplacement P# :                 1.1    octet  1, bit 1

Opérande :  Octet de sortie A 11.0        octet 10, bit 8 (= 1 octet)
                                          (10 octets + 1 octet = 11 octets)
```



```
            Adressage indirect intrazone par registre
```

2.5 Adressage indirect interzone par registre

Dans l'adressage indirect interzone par registre, l'opérande a la forme :
[Code de registre d'adresse, Pointeur de décalage].

L'avantage de l'adressage indirect interzone par registre est que vous pouvez modifier l'opérande de l'instruction en mode dynamique (pendant le traitement du programme).

Un pointeur interzone en format double mot (par exemple, P#A1.0) doit déjà se trouver dans le registre d'adresse. L'adresse des données qui doivent être traitées est calculée en additionnant les deux adresses des pointeurs.

```
Exemple:

L   P#E 8.7       // charger un pointeur double mot désigant l'adresse du
                  // bit d'entrée 8.7 dans l'accumulateur 1.

LAR1              // ranger ce pointeur dans le registre d'adresse 1.

L   P#A 2.3       // charger un pointeur double mot désigant l'adresse du
                  // bit de sortie 2.3 dans l'accumulateur 1.

LAR1              // ranger ce pointeur dans le registre d'adresse 2.

U  [AR1, P#0.0]   // la CPU additionne le contenu du registre d'adresse 1
                  // au déplacement (P#0.0) et utilise l'opérande désigné
                  // par le résultat (E 8.7) pour effectuer une combinaison
                  // logique ET. Le contenu du registre d'adresse 1, reste
                  // inchangé.

=  [AR2, P#1.5]   // la CPU affecte le résultat logique (RLG) de
                  // l'opération combinatoire ET à un opérande (A 4.0).
                  // La CPU calcule cet opréande en additionnant le contenu
                  // du registre d'adresse 2 (P#A 2.3) au déplacement
                  // (P#1.5). Le contenu du registre d'adresse 1, reste
                  // inchangé.
```

Adressage indirect interzone par registre d'adresse

3. Le jeu d'instruction

3.1 Opérations sur les accumulateurs et les registres d'adresse

3.1.1 TAK : permute le contenu des accumulateurs 1 et 2.

3.1.2 PUSH : copie le contenu de l'accumulateur 1 dans l'accumulateur 2.

3.1.3 POP : copie le contenu de l'accumulateur 2 dans l'accumulateur 1.

3.1.4 INC : incrémente le contenu de l'octet de poids faible du mot de poids faible de l'accumulateur 1 de la constante de 8 bit précisé dans l'instruction.

3.1.5 DEC : décrémente le contenu de l'octet de poids faible du mot de poids faible de l'accumulateur 1 de la constante de 8 bit précisé dans l'instruction.

3.1.6 +AR1 et **+AR2 :** ajoute le contenu du mot de poids faible de l'accumulateur 1 aux registre d'adresse 1 ou 2.

3.1.7 +AR1 P#octet.bit et **+AR2 P#octet.bit:** ajoute une constante au contenu du registre d'adresse 1 ou 2.

3.1.8 NOP 0 et NOP1 : ces opérations n'exécutent aucune fonction et n'influe pas sur les bits du mot d'état. Ces opérations sont nécessaires à la décompilation.

3.1.9 LAR1 et **LAR2 :** charge le contenu de l'accumulateur 1 dans le registre d'adresse 1 ou 2.

3.1.10 TAR : permute le contenu des registres d'adresses 1 et 2.

3.1.11 TAR1 et **TAR2 :** transfère le contenu du registre d'adresse 1 ou 2 dans l'accumulateur 1.

3.2 Opérations combinatoires sur bits

3.2.1 Opérations combinatoires avec opérandes de bits

Pour les opérations combinatoires avec opérandes de bits, on dispose des opérations suivantes :

- **U** (ET logique) est sa forme inversée **UN** (ET NON),
- **O** (OU logique) est sa forme inversée **ON** (OU NON),
- **X** (OU exclusif) est sa forme inversée **XN** (OU NON exclusif).

Ces opérations exécutent les fonctions de base suivantes :

- Interrogation de l'état d'un opérande de bit « 1 » (activé) ou « 0 » (inactivé).
- Interrogation de l'état d'une cellule de temporisation ou d'un compteur « 0 » (contenu de la cellule = 0) ou « 1 » (contenu de la cellule > 0).

Le résultat logique varie en fonction du bit /PI :

- Si /PI = « 0 », le résultat de l'interrogation de l'état est rangé dans le résultat logique sans être modifié (début d'une séquence de combinaison).
- Si /PI = « 1 », le résultat de l'interrogation de l'état est combiné en fonction de la table de vérité de l'opération logique et rangé dans le résultat logique.

3.2.2 Opérations combinatoires d'expressions entre parenthèses

Les opérations U, O et X ainsi que leur forme inverse UN, ON et X**N,** permettent de combiner des parties de séquence combinatoire figurant entre parenthèses. Dans ce cas la CPU exécute les opérations à l'intérieur des parenthèses avant la combinaison qui les précède.
L'opération) combine le RLG sauvegardé dans la pile des parenthèses avec le RLG en cours selon la table de vérité de l'opération combinatoire: U(, O(, X(, UN(, ON(ou XN(

3.2.3 Opération pour front

Les opérations **FP** (Front montant) et **FN** (Front descendant) peuvent servir de contacts détecteurs de front. Ces opérations détectent les transitions dans le RLG. De cycle en cycle elle sauvegarde le RLG dans une zone mémoire indiqué par l'opérande et le compare avec le RLG du prochain cycle.

3.2.4 Fin d'une séquence combinatoire

Une séquence combinatoire est achevée à l'aide de l'une des opérations suivantes :

- **S** (mettre à 1) **:** si le RLG a été mis à 1 dans l'instruction précédente, l'opération S met à 1 l'état du signal de l'opérande.
- **R** (mettre à 0) **:** si le RLG a été mis à 1 dans l'instruction précédente, l'opération S met à 0 l'état du signal de l'opérande.
- **=** (affectation) **:** quelque soit le résultat logique RLG, la valeur du RLG est affectée à l'opérande auquel elle accède.

Une séquence combinatoire est achevée en mettant à 0 le bit de première interrogation (/PI)

3.2.5 Négation, mise à 1, mise à 0 et sauvegarde du RLG

Les opérations suivantes permettent de modifier l'état du bit de résultat logique RLG :

- **NOT :** inverse l'état logique du RLG en cours.
- **SET :** mis à 1 inconditionnel du RLG.
- **CLR :** mis à 0 inconditionnel du RLG.
- **SAVE :** sauvegarde le bit du RLG en cours dans le bit RB du mot d'état.

3.3 Opération de temporisation

Une temporisation est élément fonctionnel du STEP7, elle permet d'autoriser des temps d'attente, d'autoriser des temps de contrôle, de générer des impulsions et de mesurer le temps.

Le langage de programmation STEP7 présente les opérations de temporisations suivantes :

- Démarrer une temporisation
    - **SI :** sous forme d'impulsion.
    - **SV :** sous forme d'impulsion prolongée
    - **SE :** sous forme de retard à la montée
    - **SS :** sous forme de retard à la montée mémorisé
    - **SA :** sous forme de retard à la retombée
- **R :** remettre une temporisation à zéro
- **FR :** valider une temporisation
- Charger une temporisation
    - **L :** comme un nombre entier.
    - **LC :** comme un nombre décimale codé binaire.

Une variable de temporisation est un mot de 16 bits. Les bits de 0 à 11 contiennent la valeur du temps en codé binaire. Cette valeur de temps indique un nombre d'unités. Les bits 12 et 13 contiennent la base de temps en format binaire. Cette base de temps détermine l'intervalle auquel la valeur de temps est décrémentée d'une unité.

Base de temps	Code binaire pour la base de temps
10 ms	00
100 ms	01
1 s	10
10 s	11

Tableau Base de temps et code binaire correspondant

On peut charger une valeur de temps prédéfinie des deux façons suivantes :

- **L W#16#wxyz : w** est la base de temps et xyz est la valeur de temps en format DCB.
- **L S5T#aH_bbM_ccS_dddMS : a** = heures, **bb** = minutes, **cc** = secondes et **ddd** = millisecondes. La base de temps est sélectionnée automatiquement.

La valeur de temps maximale est 9990 secondes, soit 2H_46M_30S avec une résolution de 10 secondes.

Le tableau suivant présent les différents résolutions avec les plages correspondantes :

Base de temps	Plage
10 ms	10ms à 9s_990ms
100 ms	100ms à 1M_39s_900ms
1 s	1s à 16M_39s
10 s	10s à 2H_46M_30s

Tableau Résolution et plage de la base de temps

Choisir la bonne temporisation

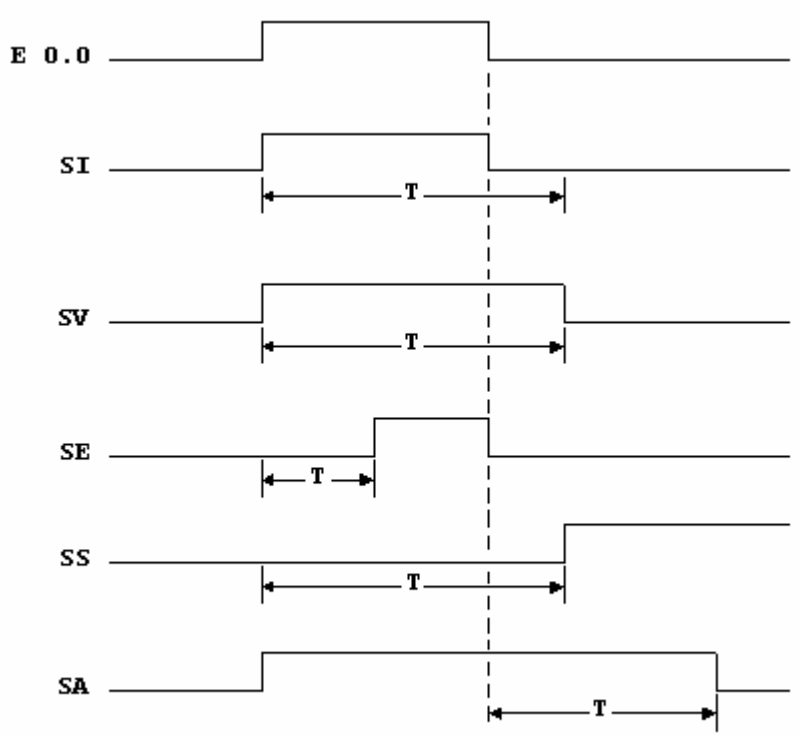
Temporisation sous forme d´impulsion SI : La durée maximale pendant laquelle le signal de sortie reste à 1 est la même que la valeur de temps « t » programmée. Le signal de sortie reste à 1 pour une durée plus courte si le signal d'entrée passe à 0.

T**emporisation sous forme d´impulsion prolongée SV:** Le signal de sortie reste à 1 pendant la durée programmée, quelle que soit la durée pendant laquelle le signal d'entrée reste à 1.

Temporisation sous forme de retard à la montée SE : Le signal de sortie est égal à 1 uniquement lorsque le temps programmé s'est écoulé et que le signal d'entrée est toujours à 1.

Temporisation sous forme de retard à la montée mémorisé SS : Le signal de sortie passe de 0 à 1 uniquement lorsque le temps programmé s'est écoulé, quel que soit la durée pendant laquelle le signal d'entrée reste à 1.

Temporisation sous forme de retard à la retombée SA : Le signal de sortie est égal à 1 lorsque le signal d'entrée est égal à 1 ou lorsque la temporisation s'exécute. La temporisation est démarrée lorsque le signal d'entrée passe de 1 à 0.



Choix de la bonne temporisation

Exemple de temporisation sous forme de retard à la retombée :

```
U   E 2.0
FR  T1                    // valider la temporisation
L   S5T#0H_28M_02S_084MS
SA  T1                    // démarrer la temporisation
U   T1
=   A 0.4                 // interroger l'état du signal de temporisation
```

3.4 Opérations de comptage

Un compteur est élément fonctionnel du STEP7, la CPU réserve un mot de 16 bits pour chaque compteur. Le langage de programmation STEP7 permet la gestion des compteurs via les opérations suivantes :

- **S :** Initialiser un compteur.
- **R :** Remettre un compteur à zéro.
- **ZV :** Incrémenter.
- **ZR :** Décrémenter.
- **FR :** Valider un compteur.
- Charger la valeur en cours du compteur :
    - **L :** comme nombre entier.
    - **LC :** comme nombre décimale codé binaire.

La programmation des compteurs doit se faire dans l'ordre suivant :

1. incrémentation
2. décrémentation
3. initialisation du compteur
4. remise du compteur a zéro
5. interrogation de l'état de signal du compteur
6. chargement de la valeur de comptage

Exemple de compteur :

```
U    E 2.0
FR   Z1          // Valider le compteur Z1
U    E 2.1
ZV   Z1          // Incrémenter le compteur Z1 d'1
U    E 2.2
ZR   Z1          // Décrémenter le compteur Z1 d'1
U    E 2.3
L    C# 3
S    Z1          // Initialiser le compteur 1
U    E 2.4
R    Z1          // Remettre à 0 le compteur 1
U    E 2.5
=    Z1          // Interroger l'état du compteur 1
L    Z1          // Charger le compteur Z1 en format binaire
T    MW10
LC   Z1          // Charger le compteur Z1 en format décimal codé  binaire
T    MW12
```

3.5 Opérations de chargement et de transfert

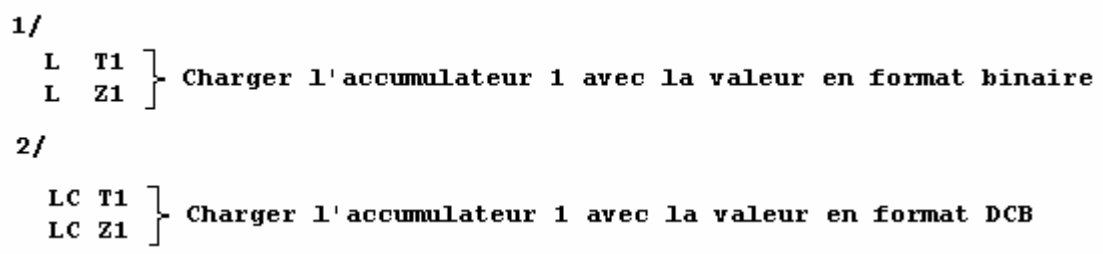
Les opérations **L** (charger) et **T** (transférer) permettent de programmer l'échange d'informations entre des modules d'E/S d'une part et des zones mémoires d'autre part, ou bien entre des zones mémoires.

L'échange d'informations avec les opérations **L** et **T,** fait appel à l'accumulateur. On peut charger et transférer des informations ayant l'une des tailles suivantes : l'octet B, le mot W et le double mot D.

De plus on peut charger et transférer le contenu du mot d'état. Par exemple :

```
L    STW    // charger les bits 0 à 8 dans le mot de poids
            // faible de l'accumulateur 1

T    STW    // transférer le contenu de l'accumulateur 1
            // dans le mot d'état
```

On peut aussi charger des temporisations et des valeurs de comptages, il y a deux formes possibles :



3.6 Opérations arithmétiques sur nombres entiers

Le tableau suivant représente les opérations LIST permettant d'additionner, de soustraire, de multiplier et de diviser des nombres entiers de 16 et 32 bits. Dans ce qui suit I désigne le mot (16 bits) et D le double mot (32 bits)

Opération	Fonction
+I, **+D**	Additionne le contenu des accumulateurs 1 et 2 et range le résultat dans 1.
-I, **-D**	Soustrait le contenu des accumulateurs 1 et 2 et range le résultat dans 1.
***I**, ***D**	Multiplie le contenu des accumulateurs 1 et 2, et range le résultat dans 1.
/I, **/D**	Divise le contenu de l'accumulateur 2 par celui de l'accumulateur 1 et range le résultat dans l'accumulateur 1.
MOD	Divise le contenu de l'accumulateur 2 par celui de l'accumulateur 1 et range le reste comme résultat dans l'accumulateur 1.

Ce tableau montre que les opérations arithmétiques et logiques combinent le contenu des accumulateurs 1 et 2 et range le résultat dans 1, l'ancien contenu de l'accumulateur 1 est transféré dans l'accumulateur 2. Pour les CPU à 4 accumulateurs, le contenu de l'accumulateur 3 est ensuite transféré dans 2, et celui de l'accumulateur 4 dans 3, l'accumulateur 4 restes inchangé.

3.7 Opérations arithmétiques sur nombres à virgule flottante

Le tableau suivant représente les instructions LIST permettant d'appliquer des opérations simples (addition, soustraction, multiplication, division) et étendus (racines carrée, logarithme…) sur des nombres à virgule flottante de 32 bits.

Opération	Fonction
+R	Additionne le contenu des accumulateurs 1 et 2 et range le résultat dans 1.

-R	Soustrait le contenu des accumulateurs 1 et 2 et range le résultat dans 1.
***R**	Multiplie le contenu des accumulateurs 1 et 2, et range le résultat dans 1.
/R	Divise le contenu de l'accumulateur 2 par celui de l'accumulateur 1 et range le résultat dans l'accumulateur 1.
ABS	Forme la valeur absolue du nombre réel contenu dans l'accumulateur 1.
SQRT	Calcule la racine carrée du contenu de l'accumulateur 1.
SQR	Calcule le carrée du contenu de l'accumulateur 1.
LN	Calcule le logarithme naturel du contenu de l'accumulateur 1.
EXP	Calcule l'exponentiel du contenu de l'accumulateur 1.
SIN	Calcule le sinus du contenu de l'accumulateur 1.
COS	Calcule cosinus du contenu de l'accumulateur 1.
TAN	Calcule la tangente du contenu de l'accumulateur 1.
ASIN	Calcule l'arc sinus du contenu de l'accumulateur 1.
ACOS	Calcule l'arc cosinus du contenu de l'accumulateur 1.
ATAN	Calcule l'arctangente du contenu de l'accumulateur 1.

3.8 Opérations de comparaison

Les opérations de comparaison permettent de comparer les paires suivantes de valeurs numériques : deux nombres entiers de 16 bits, deux nombres entiers de 32 bits et deux nombres réels (nombres à virgule flottante de 32 bits).

Critère de comparaison	Symbole des opérations
Est égal à	= =I, = =D et = =R
Est différent de	<>I, <>D et <>R
Est supérieur	>I, >D et >R
Est inférieur	<I, <D et <R
Est supérieur ou égale à	>=I, >=D et >=R
Est inférieur ou égale à	<=I, <=D et <=R

On compare toujours le contenu de l'accumulateur 2 à celui de l'accumulateur 1.

Par exemple l'opération >I vérifier si le contenu de l'accumulateur 2 est supérieur à celui de l'accumulateur 1.

3.9 Opérations de conversion

Les opérations présentées dans le tableau suivant permettent de convertir des nombres d'un format à un autre (entier – DCB, DCB – entier, réel – entier…):

Opération	Fonction
BTI	Converti une valeur décimale codé binaire en entiers 16 bits.
BTD	Converti une valeur décimale codé binaire en entiers 32 bits.
ITB	Converti un nombre entier de 16 bits en une valeur décimale codé binaire.
ITD	Converti un nombre entier de 16 bits en un nombre entier de 32 bits.
DTB	Converti un nombre entier de 32 bits en une valeur décimale codé binaire.
DTR	Converti un nombre entier de 32 bits en un nombre à virgule flottante de 32 bits.
RND	Arrondi un nombre réel au nombre entier le plus proche
RND+	Arrondi un nombre réel au nombre entier supérieur le plus proche
RND-	Arrondi un nombre réel au nombre entier inférieur le plus proche
TRUNC	Arrondi un nombre réel à sa partie entière.
TAW	Inverse l'ordre des octets de poids faible dans l'accumulateur 1.
TAD	Inverse l'ordre des octets dans l'accumulateur 1
INVI	Forme le complément à 1 d'une valeur de 16 bits.
INVD	Forme le complément à 1 d'une valeur de 32 bits.
NEGI	Forme le complément à 2 d'une valeur de 16 bits.
NEGD	Forme le complément à 2 d'une valeur de 32 bits.
NEGR	Inverse le bit de signe (bit 31) du nombre à virgule flottante (32 bits)

3.10 Opérations combinatoires sur mots

Les opérations combinatoires sur mots combinent deux mots de (16 bits) ou deux double mots de (32 bits) bit par bit selon les combinaison booléennes. Chaque mot ou double mot doit se trouver dans l'un des deux accumulateurs.

Si le résultat de la combinaison est égal à 0, le bit BI1 est mis à 0. si le résultat de cette combinaison est différent de 0, BI1 est mis à 1. Les bits BI0 et DEB du mot d'état sont remis à 0 dans tout les cas. Le tableau suivant présente ces différentes opérations, W désigne le mot (16 bits) et D le double mot (32 bits).

Opération	**Signification**
UW, UD	Combinent deux valeurs bit par bit suivant la table de vérité de la fonction ET
OW, OD	Combinent deux valeurs bit par bit suivant la table de vérité de la fonction OU
XOW, XOD	Combinent deux valeurs bit par bit suivant la table de vérité de la fonction XOR

3.11 Opérations de décalage et de rotation

3.11.1 Opérations de décalage

Les opérations de décalage permettent de décaler bit par bit le mot du poids faible ou le double mot de l'accumulateur 1 vers la droite ou vers la gauche. Le nombre de bits de décalage est précisé dans l'instruction même ou est pris dans l'octet de poids faible de l'accumulateur 2.

Les opérations suivantes décalent le contenu du mot de poids faible de l'accumulateur 1 sans signe bit par bit vers la gauche ou la droite :

- **SLW :** décalage vers la gauche d'un mot (16 bits)
- **SRW :** décalage vers la droite d'un mot (16 bits)

Les opérations suivantes décalent le contenu de l'accumulateur 1 bit par bit vers la gauche ou la droite :

- **SLD :** décalage vers la gauche d'un double mot (32 bits)
- **SRD :** décalage vers la droite d'un double mot (32 bits)

Les positions binaires libérées sont toujours complétées par des zéros.

L'opération **SSI** décale, bit par bit vers la droite, le contenu du mot de poids faible de l'accumulateur 1 (16 bits), signe inclus.

L'opération **SSD** décale, bit par bit vers la droite, le contenu entier de l'accumulateur 1 (32 bits), signe inclus. Le bit de signe est copié dans les positions de bits libérées.

3.11.2 Opérations de rotation

Les opérations de rotation permettent d'effectuer la rotation bit par bit vers la gauche ou vers la droite du contenu entier de l'accumulateur 1. Le nombre de bits de décalage est précisé dans l'instruction même ou est pris dans l'octet de poids faible de l'accumulateur 2.

On dispose des opérations suivantes :

- **RLD :** rotation vers la gauche d'un double mot (32 bits)
- **RRD :** rotation vers la droite d'un double mot (32 bits)
- **RLDA :** rotation vers la gauche de l'accumulateur 1 via BI1.
- **RRDA:** rotation vers la droite de l'accumulateur 1 via BI1.

Les positions binaires libérées sont toujours complétées par des zéros.

L'opération **SSI** décale, bit par bit vers la droite, le contenu du mot de poids faible de l'accumulateur 1 (16 bits), signe inclus.

L'opération **SSD** décale, bit par bit vers la droite, le contenu entier de l'accumulateur 1 (32 bits), signe inclus.

Le bit de signe est copié dans les positions de bits libérées.

3.12 Opérations de saut et de gestion d'exécution de programme

3.12.1 Opérations de saut

Les opérations de saut et de boucle permettent de gérer le déroulement du programme, elles interrompent le déroulement linéaire du programme afin de reprendre son exécution à un endroit différent. L'opérande d'une opération de saut ou de boucle est un repère de saut.

- Opérations de saut inconditionnel :
  - SPA : saut inconditionnel
  - SPL : saut vers liste
- Opérations de saut conditionnel selon le RLG :
  - SPB : Saut si RLG égale 1
  - SPBN : Saut si RLG égale 0
  - SPBB : Saut si RLG égale 1 avec RB
  - SPBNB : Saut si RLG égale 0 avec RB
- Opérations de saut conditionnel selon le RB ou DEB/DM :
  - SPBI : Saut si RB égale 1
  - SPBIN : Saut si RB égale 0
  - SPO : Saut si DEB égale 1

- SPS : Saut si DM égale 1

- Opérations de saut conditionnel selon le résultat dans BI1 et BI0 :
  - SPZ : Saut si égale à 0
  - SPN : Saut si différent de 0
  - SPP : Saut si plus
  - SPM : Saut si moins
  - SPMZ : Saut si inferieur ou égale à 0
  - SPPZ : Saut si supérieur ou égale à 0
  - SPU : Saut si illicite

- Boucle de programme :
  - LOOP : Saut si contenu de l'accumulateur 1 supérieur 0

Le repère de saut est composé au max de 4 caractères, le premier caractère doit être une lettre, les autres des chiffres ou des lettres (exemple F2R0)

3.12.2 Opérations de gestion de programme

3.12.2.1 Appel de fonction et de bloc fonctionnel avec CALL

L'opération CALL (appel de bloc) permet d'appeler des fonctions FC et des blocs fonctionnels FB dans le programme utilisateur. Cet appel est indépendant du résultat logique du RLG. Lors de l'appel d'un bloc fonctionnel, on doit spécifier le bloc de donnée d'instance associe.

```
Exemple:
CALL FB2, DB49  // appeler le bloc fonctionnel 40 avec
                // le bloc de donnée d'instance 49.
```

3.12.2.2 Appel de fonction et de bloc fonctionnel avec CC et UC

Les opérations CC et UC appel les fonctions et les blocs fonctionnels de la même manière que l'opération CALL, sauf qu'on n'a pas besoin de préciser le bloc de donnée d'instance dans l'opérande.

- L'opération CC appel la fonction précisée dans l'opérande si le RLG est à 1.
- L'opération UC appel la fonction précisée dans l'opérande quel que soit le résultat logique RLG.

```
Exemple:
CC FC13   // appel la fonction 13 si le RLG est à 1.
UC FC13   // appel la fonction 13 quel que soit le RLG.
```

CONT

Langage de programmation CONT (Schéma à contact) :

Ce langage de programmation fait partie du logiciel de base STEP7, il s'inspire des schémas de circuits électriques, les éléments d'un schéma de circuit sont rassemblés dans des réseaux, un ou plusieurs réseaux forment la section des instructions complète d'un bloc de code.

Dans le langage CONT, on crée le programme en utilisant un éditeur incrémental.

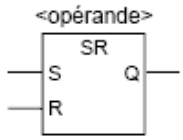
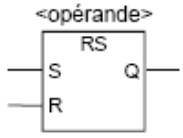
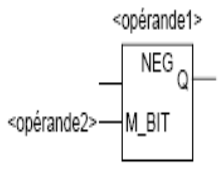
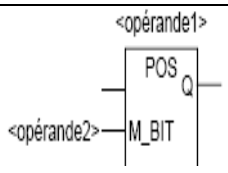
1. Opérations combinatoires sur bits :

Les opérations combinatoires sur bits utilisent deux chiffres : 1 et 0 (chiffres binaire), pour les contacts et les bobines, 1 signifie activé et 0 signifie désactivé.

Ces opérations évaluent les états de signal 1 et 0 et les combinent selon la logique booléenne, le résultat est égal soit 1 ou 0, c'est le résultat logique (RLG).

Les opérations combinatoires sur bits existantes sont données dans le tableau suivant :

Noms	**Représentation**	**Fonctionnement**
Contact à fermeture	<opérande> ---| |---	Le contact est fermé si la valeur du bit interrogé sauvegardée en <opérande> égale 1 alors RLG = l, en revanche, si l'état de <opérande> est 0, le contact est ouvert et RLG= 0.
Contact à ouverture	<opérande> ---| / |---	Le contact est fermé si la valeur du bit interrogé sauvegardée en <opérande> égale 0. Dans ce cas le RLG =1, en revanche, si l'état de <opérande> est 1, le contact est ouvert et RLG = 0.
Sauvegarder RLG dans RB	---(SAVE)	Cette opération sauvegarde le RLG dans le bit RB du mot d'état, dans ce cas le bit de première interrogation /PI n'est pas affecté, ainsi une opération ET dans le réseau suivant prendra en considération l'état du bit RB.
Bobine de sortie	<opérande> ---( )	Cette opération fonctionne comme une bobine dans un schéma à relais. Si le RLG = 1, le bit en <opérande> est mise à 1 et si RLG = 0, le bit en <opérande>est mis à 0.
Connecteur	<opérande> ---(#)---	C'est un élément d'affectation intermédiaire qui mémorise le bit RLG dans l'<opérande> précisé.
Inverseur de RLG	---|NOT|---	Cette opération inverse le bit de résultat logique (RLG).

Mettre à 1	<opérande> ---(S)	Cette opération permet d'effectuer un SET sur l'opérande si le RLG des opérations précédentes a la valeur 1, dans ce cas, l'<opérande> précisé de l'élément est mis à 1, si RLG = 0 l'opérande reste inchangé.
Mettre à 0	<opérande> ---(R)	Cette opération permet d'effectuer un RESET sur l'opérande si le RLG des opérations précédentes a la valeur 1, dans ce cas, l'<opérande> précisé de l'élément est mis à 0, si RLG = 0 l'opérande reste inchangé.
Bascule mise à 1, mise à 0		Cette opération exécute la mise à 1 si l'état de signal est 1 à l'entrée S et 0 à l'entrée R. Si l'état de signal est 0 à l'entrée S et 1 à l'entrée R, la bascule est mise à 0. Si le RLG est égal à 1 aux deux entrées, c'est l'ordre qui compte : la bascule SR exécute d'abord la mise à 1, puis la mise à 0 de l'<opérande> indiqué,les opérations S et R s'exécutent uniquement si le RLG égale 1.
Bascule mise à 0, mise à 1		Cette opération exécute la mise à 0 si l'état de signal est 1 à l'entrée R et 0 à l'entrée S. Si l'état de signal est 0 à l'entrée R et 1 à l'entrée S, la bascule est mise à 1. Si le RLG est égal à 1 aux deux entrées, c'est l'ordre qui compte : la bascule RS exécute d'abord la mise à 0, puis la mise à 1 de l'<opérande> indiqué, les opérations S et R s'exécutent uniquement si le RLG égale 1.
Détecter front descendant	<opérande> ---(N)---	Cette opération détecte le passage de 1 à 0 de l'état de signal de l'opérande et montre cette transition avec un RLG égal à 1 après cette opération.
Détecter front montant	<opérande> ---(P)---	L'opération détecte le passage de 0 à 1 de l'état de signal de l'opérande et montre cette transition avec un RLG égal à 1 après opération.
Détecter front descendant de signal		Cette opération compare l'état de <opérande1> à celui provenant de l'interrogation d'état de signal précédent figurant dans <opérande2>.
Détecter front montant de signal		Cette opération compare l'état de signal de <opérande1> à celui provenant de l'interrogation d'état de signal précédent figurant dans <opérande2>.

2. Opérations de comparaison :

Les opérations de comparaison comparent les entrées IN1 et IN2, si la comparaison est vraie, son résultat logique (RLG) est 1 ; sinon, il est égal à 0, dans le langage CONT on possède les opérations de comparaison suivantes :

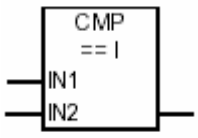
1. CMP ? I : Comparer entiers de 16 bits.
2. CMP ? D : Comparer entiers de 32 bits.
3. CMP ? R : Comparer nombres réels

Les types de comparaison sont :

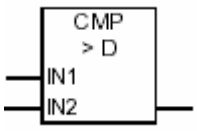
1. == IN1 égal à IN2
2. <> IN1 différent de IN2
3. > IN1 supérieur à IN2
4. < IN1 inférieur à IN2
5. >= IN1 supérieur ou égal à IN2
6. <= IN1 inférieur ou égal à IN2

Leurs représentations se fait par des blocs, ou à l'intérieur on mentionne l'opération et le type de comparaison par les symboles qu'on a ci-dessus.

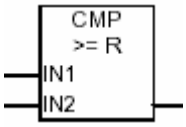
Exemples :



Egalité entre 2 entiers 16 bits



comparaison entre 2 entiers 32 bits : IN1 supérieur à IN2

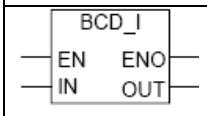
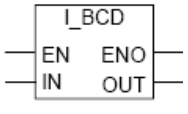
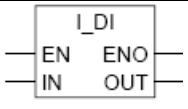


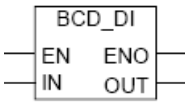
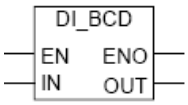
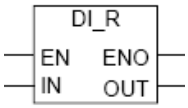
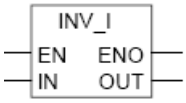
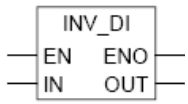
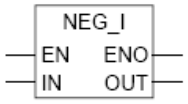
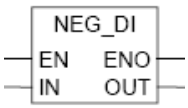
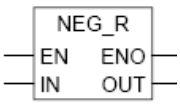
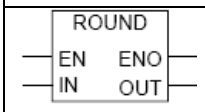
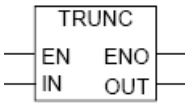
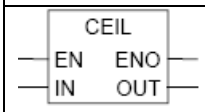
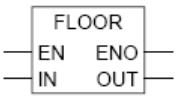
comparaison entre 2 réel : IN1 supérieur ou égal à IN2

3. Opérations de conversion :

Les opérations de conversion lisent le contenu du paramètre d'entrée IN, le convertissent ou en changent le signe. Le résultat est rangé dans le paramètre de sortie OUT.

Les opérations de conversion sont données dans le tableau suivant :

Représentation	**Fonctionnement**
	Cette opération permet de convertir un nombre en BCD en un nombre entier de 16 bits
	Cette opération permet de convertir nombre entier sur 16 bits en un nombre en BCD.
	Cette opération permet de convertir nombre entier sur 16 bits en un entier sur 32 bits

	Cette opération permet de convertir un nombre en BCD en un nombre entier de 32 bits
	Cette opération permet de convertir nombre entier sur 32 bits en un nombre en BCD.
	Cette opération permet de convertir nombre entier sur 32 bits en un nombre réel.
	Cette opération permet de donner le complément à 1 d'un entier sur 16 bits
	Cette opération permet de donner le complément à 1 d'un entier sur 32 bits
	Cette opération permet de donner le complément à 2 d'un entier sur 16 bits
	Cette opération permet de donner le complément à 2 d'un entier sur 32 bits
	Cette opération permet d'inverser le signe d'un nombre réel
	Cette opération permet d'arrondir un réel en un entier de 32 bits.
	Cette opération permet de prendre la valeur entière d'un réel et la mettre sous forme d'un entier de 32 bits
	Cette opération permet de convertir un nombre réel en un nombre entier supérieur le plus proche
	Cette opération permet de convertir un nombre réel en un nombre entier inférieur le plus proche

4. Opérations sur blocs de données :

---(OPN) Ouvrir bloc de données :

Cette opération ouvre un bloc de données (DB) ou un bloc de données d'instance (DI), le numéro du bloc de données est transféré au registre DB ou DI.

5. Opérations de saut :

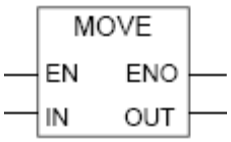
L'opérande d'une opération de saut est un repère de saut qui indique la destination où doit sauter le programme, il doit être indiqué au-dessus de la bobine de saut.

Le repère de destination de saut doit se trouver au début du réseau pour l'indiquer on sélectionne LABEL dans la boîte de sélection CONT puis spécifie le nom du repère.
On dispose des opérations de saut suivantes :

- **Saut inconditionnel** ---(JMP)--- : S'il n'y a aucun élément CONT entre la barre d'alimentation gauche et l'opération cette opération fonctionne comme un saut inconditionnel, les opérations entre l'opération de saut et le repère de saut n'est pas exécuté.
- **Saut à l'intérieur d'un bloc si 1 (conditionnel)** ---(JMP)--- : Cette opération fonctionne comme un saut conditionnel si le RLG de la combinaison précédente est égal à 1, les opérations entre l'opération de saut et le repère de saut ne sont pas exécutées.
- **Saut à l'intérieur d'un bloc si 0 (conditionnel)** ---(JMPN) : Elle fonctionne comme un saut conditionnel si le RLG de la combinaison précédente est égal à 0, les opérations entre l'opération de saut et le repère de saut ne sont pas exécutées.
- **LABEL Repère de saut :** Il identifie la destination d'une opération de saut. Il est constitué de 4 caractères alphanumériques, le premier devant être une lettre. Pour chaque opération ---(JMP) ou ---(JMPN), il doit exister un repère de saut.

6. Opération de transfert :

Affecter valeur (MOVE):



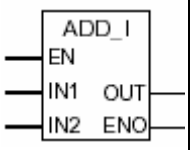
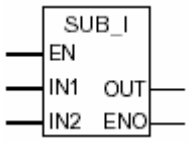
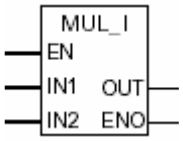
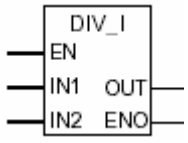
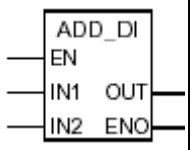
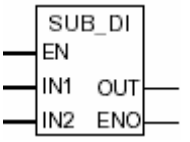
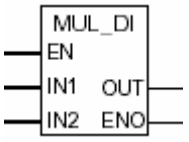
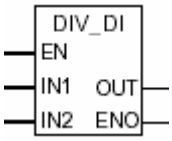
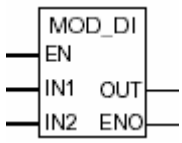
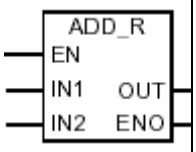
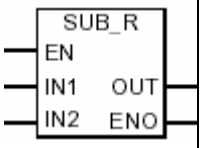
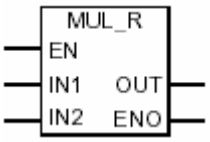
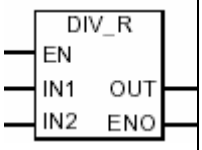
Si l'entrée de validation EN=1 l'opération de transfert est activé et dans ce cas la valeur indiquée dans l'entrée IN est copiée des octets, des mots ou doubles mots à l'adresse précisée dans la sortie OUT ; L'état de signal de ENO est identique à celui de EN.

Pour copier des types de données utilisateur tels que des tableaux ou des structures, on doit faire appel à la fonction système "BLKMOV" (SFC 20).

7. Opérations arithmétiques :

Les opérations arithmétiques sont effectuées entre les entrées IN1 et IN2 qui sont des réels ou des entiers sur 16 bits ou sur 32 bits.

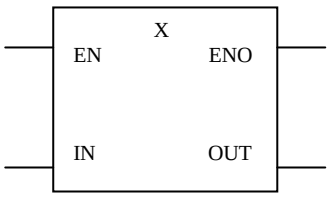
Si l'état de signal est 1 à l'entrée de validation EN, le résultat est rangé dans la sortie OUT et si ce résultat est hors de la plage autorisée pour un nombre entier de 16 bits, les bits DEB et DM du mot d'état sont mis à 1 et la sortie ENO est à mise à 0. Ainsi, les opérations suivant les opérations arithmétiques et qui y sont connectées par ENO (cascade) ne sont pas exécutées.

opérations	Addition	Soustraction	Multiplication	Division	Reste de division
Entiers sur 16 bits					–
Entiers sur 32 bits					
Réels					–

Les opérations arithmétiques sur nombres à virgule flottante permettent d'exécuter les fonctions arithmétiques suivantes sur un nombre réel de 32 bits :

- Valeur absolue (ABS) d'un nombre réel :
- Carré (SQR) ou Racine carrée (SQRT) d'un nombre réel
- Logarithme naturel (LN) d'un nombre réel
- Valeur exponentielle (EXP) sur la base e (= 2,71828) d'un nombre réel
- Fonctions trigonométriques d'angles représentés sous forme de nombres réels :
    1. Sinus (SIN) et Arc sinus (ASIN).
    2. Cosinus (COS) et Arc cosinus (ACOS).
    3. Tangente (TAN) et Arc tangente (ATAN).

Pour les opérations sur un nombre la représentation générale est donnée par le bloc suivant :



Remarque :

X désigne la représentation les opération sur nombre : ABS, SQR, SQRT, LN, EXP, SIN, ASIN, COS, ACOS, TAN et ATAN.

Pour les opérations ASIN et ACOS la valeur d'entrée doit être entre -1 et 1, et le résultat est un angle en radian compris entre -90° et 90°.

8. Opérations combinatoires sur mots :

Les opérations combinatoires sur mots combinent deux mots (16 bits) ou deux doubles mots (32 bits), bit par bit, selon les combinaisons booléennes. Ces opérations sont activées si l'état de signal est 1 à l'entrée de validation EN.

Si le résultat à la sortie OUT est différent de 0, le bit BI1 du mot d'état est mis à 1.

Si le résultat à la sortie OUT égale 0, le bit BI1 du mot d'état est mis à 0.

On dispose des opérations combinatoires sur mots suivantes :

- WAND_W : ET mot
- WOR_W : OU mot
- WXOR_W : OU exclusif mot
- WAND_DW : ET double mot
- WOR_DW : OU double mot
- WXOR_DW : OU exclusif double mot

9. Opérations de décalage et de rotation :

Opérations de décalage :

Les opérations de décalage permettent de décaler bit par bit le contenu de l'entrée IN vers la gauche ou vers la droite, Les positions binaires libérées par l'opération de décalage sont soit remplies par des zéros, soit par l'état de signal du bit de signe.

Le nombre de bits de décalage est précisé dans le paramètre d'entrée N. L'état de signal du bit décalé en dernier est chargé dans le bit BI1 du mot d'état. Les bits BI0 et DEB du mot d'état sont remis à 0.

On dispose des opérations de décalage suivantes :

- SHR_I : Décalage vers la droite d'un entier de 16 bits.
- SHR_DI : Décalage vers la droite d'un entier de 32 bits.
- SHL_W : Décalage vers la gauche d'un mot.
- SHR_W : Décalage vers la droite d'un mot.
- SHL_DW : Décalage vers la gauche d'un double mot.
- SHR_DW : Décalage vers la droite d'un double mot.

Opérations de rotation :

Les opérations de rotation permettent d'effectuer la rotation bit par bit vers la droite ou vers la gauche du contenu entier de l'entrée IN de N bits, et le résultat est mis dans OUT.

Les positions binaires libérées sont complétées par l'état de signal des bits qui ont été décalés hors de l'entrée IN.

On dispose des opérations de rotation suivantes :

- ROL_DW Rotation vers la gauche d'un double mot.
- ROR_DW Rotation vers la droite d'un double mot.

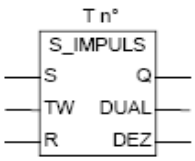
10. Opérations de gestion d'exécution de programme :

Les principales fonctions de gestion de programme sont données dans le tableau suivant :

L'opération	**Représentation**
CALL Appeler FC/SFC sans paramètre	<FC/SFC n°> **---(CALL)**
CALL_FB (Appeler FB)	<DB n°> FB n° EN ENO
CALL_FC (Appeler FC)	FC n° EN ENO
CALL_SFB (Appeler SFB)	<DB n°> SFB n° EN ENO
CALL_SFC (Appeler SFC)	SFC n° EN ENO
Appeler multi-instances	#Nom-variable EN ENO

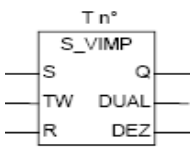
11. Opérations de temporisation :

1. **S_IMPULS temporisation sous forme d'impulsion :**



La temporisation précisée est démarrée en cas de front montant à l'entrée de démarrage S et tant que S=1 la valeur de temps indiquée à l'entrée TW s'écoule et la sortie Q=1. En cas de passage de 1 à 0 à l'entrée S avant la fin du temps, la temporisation s'arrête et la sortie Q est mise à 0. Si l'état de l'entrée R passe de 0 à 1 avant la fin de la temporisation, cette dernière est remise à 0 au même moment que la valeur de temps en cours et la base de temps.

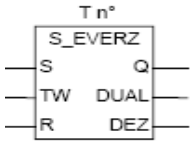
2. **S_VIMP temporisation sous forme d'impulsion prolongée :**



En cas de front montant à l'entrée S la temporisation démarre et le temps indiqué en TW s'écoule même si l'état de l'entrée S passe à 0 avant la fin du temps, tant que la temporisation s'exécute, l'état de signal à la sortie Q égale 1. La temporisation est redémarrée avec la valeur de temps prédéfinie si l'entrée S passe de 0 à 1 avant

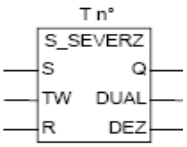
la fin de la temporisation. En cas de passage de 0 à 1 à l'entrée R pendant que la temporisation s'exécute, elle est remise à 0 au même moment que la valeur de temps en cours et la base de temps.

3. S_EVERZ temporisation sous forme de retard à la montée :



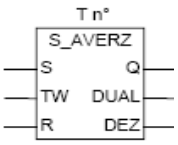
Un front montant à l'entrée S fait démarrer la temporisation, le temps indiqué à l'entrée TW s'écoule et l'état de signal à la sortie Q=1 lorsque la temporisation s'est exécutée sans erreur et que l'état de signal à l'entrée S est à 1. La temporisation s'arrête et sera mise à 0 si l'entrée S passe de 1 à 0 alors que la temporisation s'exécute. Si R passe de 0 à 1 alors que la temporisation s'exécute la sortie Q=0 et la temporisation est remise à 0 au même temps que la valeur de temps en cours et la base de temps.

4. S_SEVERZ temporisation sous forme de retard à la montée mémorisé :



La temporisation démarre si un front montant est appliqué à l'entrée S dans ce cas et le temps indiqué en TW s'écoule même si l'état de l'entrée S passe à 0 avant la fin de la temporisation et Q=1 lorsque la temporisation sera fini, quel que soit l'état de S, en revanche si S passe de 0 à 1 alors que la temporisation s'exécute, cette dernière est redémarrée avec la valeur de temps de TW et en cas de passage de 0 à 1 à l'entrée R, la temporisation est remise à 0 quel que soit le RLG à l'entrée S et Q=0.

5. S_AVERZ temporisation sous forme de retard à la retombée :

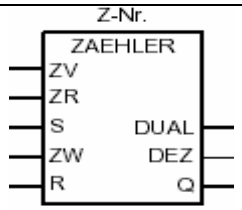
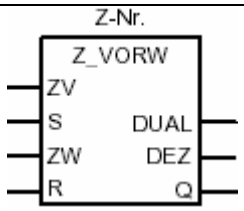
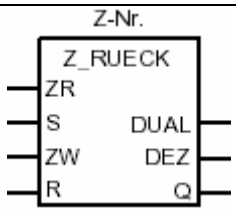


Cette opération démarre la temporisation précisée en cas de front descendant à l'entrée S, l'état de signal à la sortie Q égale 1 lorsque l'entrée S est 1 ou lorsque la temporisation s'exécute. La temporisation est remise à 0 lorsque l'état de l'entrée S passe de 0 à 1 alors que la temporisation s'exécute. La temporisation n'est redémarrée que lorsque l'état de S repasse de 1 à 0, en cas de passage de 0 à 1 à l'entrée R pendant que la temporisation s'exécute, cette dernière est remise à 0.

12. Opérations de comptage :

Une zone de mémoire est réservée aux compteurs dans la CPU qui n'est accédée que par les opérations de comptage, ou un mot de 16 bits y est réservé pour chaque compteur.
La valeur de comptage est contenue dans les bits 0 à 9 du mot de comptage, lorsque le compteur est mis à 1, la valeur qu'on définie y est placée par l'accumulateur, la plage de la valeur de comptage est comprise entre 0 et 999.

On peut modifier cette valeur en utilisant les opérations suivantes :

Type de compteur	Incrémental/décrémental	Incrémental	décrémental
Représentation			
Fonctionnement	Un front montant sur l'entrée S initialise le compteur avec la valeur ZW, un front montant sur l'entrée ZV incrémente le compteur de 1 et s'il contient 999 il repasse à 0, un front montant sur l'entrée ZR décrémente le compteur de 1 et s'il contient 0 le compteur passe à 999. S'il y a front montant sur les deux entrées, la valeur reste inchangée, la sortie Q donne un résultat égal à 1 si la valeur du compteur est différente dei 0		

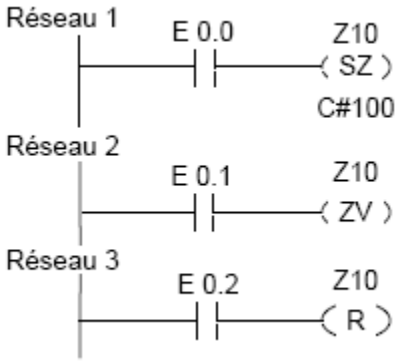
Il y a des opérations plus simples qui permettent de gérer les compteurs et sont :

---(SZ) Initialiser compteur : Elle ne s'exécute que si le RLG présente un front montant. La valeur prédéfinie est alors transférée au compteur indiqué.

---(ZV) Incrémenter : Cette opération incrémente de 1 la valeur du compteur précisé si le RLG présente un front montant et si la valeur du compteur est inférieure à 999 le compteur reste inchangée en l'absence de front montant au RLG ou si le compteur est déjà égal à 999.

---(ZR) Décrémenter : Cette opération décrémente d'un la valeur du compteur précisé si le RLG présente un front montant et si la valeur du compteur est supérieure à 0, et en l'absence de front montant au RLG ou si le compteur est déjà égal à 0, la valeur du compteur reste inchangée.

Exemple :



Si l'entrée E 0.0 passe de 0 à 1 le compteur Z10 est initialisé avec la valeur 100 et si l'entrée E 0.1 passe de 0 à 1 la valeur de comptage du compteur Z10 est incrémentée d'un, s'il n'est pas à 999. Si l'état de signal à l'entrée E 0.2 est égal à 1, le compteur est mis à zéro.

LOG

1. Opérations combinatoires sur bits :

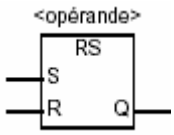
Les opérations de combinaison sur bits évaluent les états de signal et les combinent selon la logique booléenne. La bibliothèque du langage LOG comprend les opérations combinatoires sur bits de base : Porte ET (&), Porte OU **(>=1),** Porte XOR, ligne Inverseur, Affectation (**=)…** Ainsi que d'autre opérations comme :

1- **# (Connecteur) :** Il permet de sauvegarder un résultat intermédiaire dans l'opérande associé.

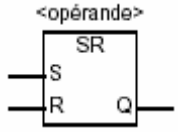
2- **R (RESET) :** L'opération Mettre à 0 met son opérande à 0, seul SET le remet à 1, elle ne s'exécute que si le RLG a la valeur 1.

3- **S (SET) :** L'opération Mettre à 1 met son opérande à 1, seul RESET le remet à 0, elle ne s'exécute que si le RLG a la valeur 1.

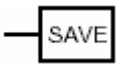
4- **Bascule RS (Bascule mise à 0, mise à 1) :** La bascule RS est mise à 1 si l'état de signal est 1 à l'entrée R et 0 à l'entrée S. Si l'état de signal est 0 à l'entrée R et 1 à l'entrée S, la bascule est mise à 0, cela ne se fait que lorsque le RLG est égal à 1.



5- **Bascule SR (Bascule mise à 1, mise à 0) :** La bascule SR est mise à 1 si l'état de signal est 1 à l'entrée S et 0 à l'entrée R. Si l'état de signal est 1 à l'entrée R et 0 à l'entrée S, la bascule est mise à 0, cela ne se fait que lorsque le RLG est égal à 1.

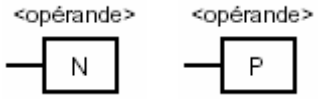


6- **SAVE (Sauvegarder RLG dans RB) :**L'opération Sauvegarder RLG dans RB permet de sauvegarder le résultat logique dans le bit RB du mot d'état. ceci est intéressant lors de la sortie d'un bloc .En effet, la sortie ENO d'un bloc est égale au bit RB, une opération SAVE permet donc de quitter un bloc avec un résultat 1, ceci permet de gérer les erreurs de traitement d'un bloc.



7- Opérations de détection de front :

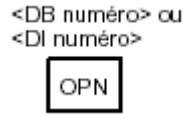
1- **N (Détecter front descendant) et P (Détecter front montant) :** Elles donnent des sorties égales à1 si des fronts sur RLG qui précède les blocs sont détectés, le RLG du cycle précédent est sauvegardé dans <opérande>.



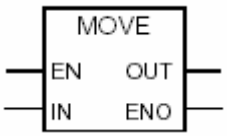
2- **NEG (Détecter front descendant de signal) :** L'opération Détecter front descendant de signal compare l'état de signal de <opérande1> à celui provenant de l'interrogation précédente figurant dans le paramètre M_BIT. En cas de passage de 1 à 0, la sortie Q est mise à 1 ; dans tous les autres cas, elle est mise à 0.

3- **POS (Détecter front montant de signal) :** L'opération Détecter front montant de signal compare l'état de signal de <opérande1> à celui provenant de l'interrogation précédente figurant dans le paramètre M_BIT. En cas de passage de 0 à 1, la sortie Q est mise à 1 ; dans tous les autres cas, elle est mise à 0.

2. Opérations d'ouverture d'un bloc de données : Cette opération ouvre un bloc de données (DB) ou un bloc de données d'instance (DI) pour lire ou modifier son contenu.



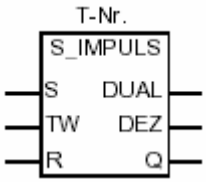
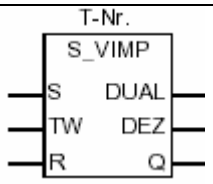
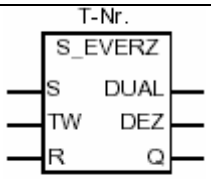
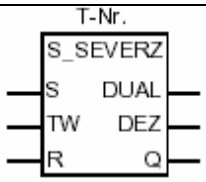
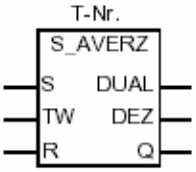
3. Opérations de transfert MOVE : L'opération MOVE permet d'initialiser des variables avec des valeurs indiquées à l'entrée IN sont copiées dans l'opérande précisé dans la sortie OUT, la boîte MOVE permet de copier tous les types de données élémentaires de 8, 16 ou 32 bits. Pour copier des types de données utilisateur tels que des tableaux, on fait appel à la fonction système SFC20 "BLKMOV".



4. Opérations de temporisation :

La programmation en langage LOG permet d'utiliser jusqu'à 256 temporisations, on rappelle que les opérations de temporisations possibles sous le langage LOG sont les mêmes que celles sous le langage CONT.

Les différents blocs de temporisation sont indiqués dans le tableau suivant

Type	Représentation	Description
Temporisation sous forme d'impulsion		Démarre la temporisation lorsque l'état du signal S passe de 0 à 1, la sortie Q se met à1, en cas de passage de 1 à 0 de l'entrée S avant l'écoulement du temps, la temporisation s'arrêt, et la sortie Q se met à 0.
Temporisation sous forme d'impulsion prolongée		La temporisation démarre lorsque l'état de S passe de 0 à 1, dans ce cas la sortie Q se met à1.La valeur de temps indiquée à l'entrée TW continue à s'écouler même si l'état de l'entrée S passe à 0 avant expiration du temps. La temporisation se met à 0 si R passe de 0 à 1.
Temporisation sous forme de retard à la montée		La temporisation démarre lorsque l'état S passe de 0 à 1, le temps s'écoule tant que l'état de S est à 1, dans ce cas la sortie Q se met à1lorsque le temps expire, si R passe de 0 à 1, Q se met à 0.
Temporisation sous forme de retard à la montée mémorisé		La temporisation démarre en cas de front montant sur l'entrée S, et le temps s'écoule même si S passe à 0, la sortie Q se met à1 après l'écoulement du temps. La temporisation est remise à 0 si R passe de 0 à 1.
Temporisation sous forme de retard à la retombée		La temporisation démarre lorsque l'état de S de1 à 0, la sortie Q est à 1 lorsque l'état de S est 1 ou lorsque la temporisation s'écoule. La temporisation est remise à zéro lorsque S passe de 1 à 0 ou R passe 0 à1.

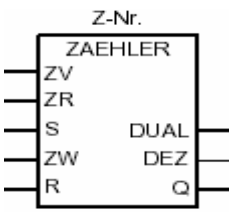
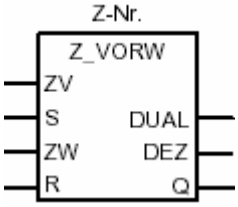
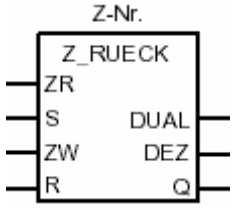
5. Opérations de saut :

Dans STEP 7 on dispose des opérations de sauts conditionnels ou inconditionnelles, elles réagissent à une interrogation à 1 ou à 0 selon la commande choisie.

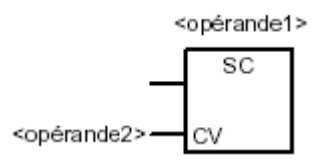
1. **Saut inconditionnel :** Cet élément lorsqu'il est utilisé directement dans un réseau (sans combinaison logique qui le précède) effectue un saut inconditionnel vers l'étiquette définit dans l'opérande.

2. **Saut conditionnel :** Les sauts conditionnels doivent être précédés de combinaison logique qui représentera la condition, il existe 2 types de saut conditionnel : JMP (Saut si 1) et JMPN (Saut si 0)
3. **Étiquette :** Sont les repères de saut qui identifie la destination d'une opération de saut

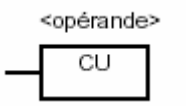
6. Opérations de comptage :

Type de compteur	Incrémental/décrémental	Incrémental	décrémental
Représentation			
Fonctionnement	Le domaine des valeurs que peut prendre le compteur va de 0 à 999. Un front montant sur l'entrée S initialise le compteur avec la valeur ZW, un front montant sur l'entrée ZV incrémente le compteur de1 sauf s'il contient 999 : dans ce cas le compteur repasse à 0. Un front montant sur l'entrée ZR décrémente le compteur de1 sauf s'il contient 0 : dans ce cas le compteur passe à 999. S'il y a front montant sur les deux entrées, la valeur reste inchangée. La sortie Q donne un résultat égal à 1 si la valeur du compteur est différente dei 0, sinon elle donne 0.		

Il existe des fonctions plus basiques permettant de gérer les compteurs qui sont :



Initialiser compteur



Incrémenter compteur

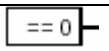
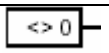
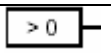
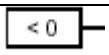
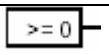
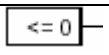
Décrémenter compteur

Le bloc d'initialisation opère sur le compteur <opérande1> lors d'un front montant sur l'entrée, le compteur prend alors la valeur <opérande2>.

L'incrémentation et la décrémentation sont effectuées s'il y a front montant sur l'entrée de validation.

Opérations sur bits d'état :

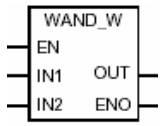
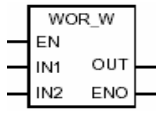
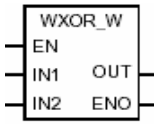
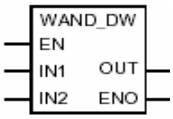
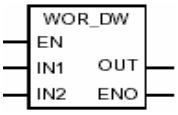
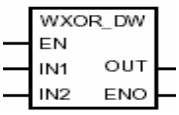
1. **OV (Débordement) :** L'opération Bit d'anomalie Débordement permet de détecter un débordement dans une opération arithmétique, alors après une opération arithmétique, si un débordement est détecté le bit de débordement (DEB) du mot d'état se met à 1, et il est remis à 0 lorsque l'erreur est corrigée.
2. **OS (Débordement mémorisé) :** Cet élément permet de détecter un débordement mémorisé (DM) dans une opération arithmétique, le bit interrogé est le bit DM du mot d'état qui se met à 1 si un débordement est détecté. Contrairement au bit DEB, le bit DM reste à 1 même après que l'erreur soit corrigée.
3. **UO (Opération illicite) :** Ce test permet dei déterminer si une opération à virgule flottante est illicite, c'est-à-dire que tous les opérations sont des nombres à virgule flottante. Ce sont les bits BI1 et BI0 qui sont testés pour obtenir le résultat.
4. **BIE (Registre RB) :** L'opération Bit d'anomalie Registre RB permet d'interroger l'état du bit de résultat binaire RB du mot d'état.
5. **(Bits de résultat) :** Les opérations Bits de résultat permettent de comparer le résultat d'une opération arithmétique par rapport à 0, pour cela les éléments suivants interrogent les bits BI1 et BI0

Elément	Description
	L'opération Bit de résultat pour égal à 0 détermine si le résultat d'une opération arithmétique est ou non égal à 0.
	L'opération Bit de résultat pour différent de 0 détermine si le résultat d'une opération arithmétique est ou non différente de 0.
	L'opération Bit de résultat pour supérieur à 0 détermine si le résultat d'une opération arithmétique est ou non supérieur à 0.
	L'opération Bit de résultat pour inférieur à 0 détermine si le résultat d'une opération arithmétique est ou non inférieur à 0.
	L'opération Bit de résultat pour supérieur ou égal à 0 détermine si le résultat d'une opération arithmétique est ou non supérieur ou égal à 0.
	L'opération Bit de résultat pour inférieur ou égal à 0 détermine si le résultat d'une opération arithmétique est ou non inférieur ou égal à 0.

Opérations combinatoires sur mots :

Les opérations combinatoires sur mots combinent deux mots (16 bits) ou deux doubles mots (32 bits) indiqués dans les entrées IN1 et IN2, selon les combinaisons booléennes, Le résultat est donné par la sortie OUT. Ces opérations sont activées si l'état de signal est 1 à l'entrée de validation EN.

Les opérations combinatoires sur mots sont :

Opération	**Représentation**	**Fonctionnement**
WAND_W		Cette opération combine, bit par bit selon la table de vérité ET, les deux mots indiqués dans les entrées IN1 et IN2.
WOR_W		opération combine, bit par bit selon la table de vérité OU, les deux mots indiqués dans les entrées IN1 et IN2.
WXOR_W		Cette opération combine, bit par bit selon la table de vérité OU exclusif, les deux mots indiqués dans les entrées IN1 et IN2.
WAND_DW		Cette opération combine, bit par bit selon la table de vérité ET, les deux doubles mots indiqués dans les entrées IN1 et IN2.
WOR_DW		Cette opération combine, bit par bit selon la table de vérité OU, les deux doubles mots indiqués dans les entrées IN1 et IN2.
WXOR_DW		Cette opération combine, bit par bit selon la table de vérité OU exclusif, les deux doubles mots indiqués dans les entrées IN1 et IN2.

Remarque :

Les blocs d'opérations arithmétique et de comparaison ainsi que celles de gestion de programme sont les mêmes que pour le langage CONT.

S7-GRAPH

Langage de programmation S7-GRAPH :

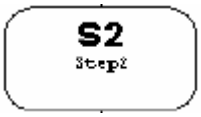
Un graphe séquentiel est une suite d'étapes sont activées, puis désactivées dans un ordre déterminé, en fonction des réceptivités exprimées dans les transitions.
Le langage de programmation S7-GRAPH permet de programmer graphiquement les commandes séquentielles de manière rapide et facile.
Le processus est subdivisé en étapes au nombre de fonctions limité, la séquence est représentée graphiquement et les actions à exécuter sont associées aux étapes, tandis que des transitions règlent l'évolution entre deux étapes successives (réceptivités).

Eléments d'un graphe séquentiel :

Les éléments dont peut se composer un graphe séquentiel sont :

- Etape + transition
- Saut
- Ouvrir branche OU (divergence en OU)
- Fermer branche OU (convergence en OU)
- Ouvrir branche ET (divergence en ET)
- Fermer branche ET (convergence en ET)
- Fin de graphe
- Nouveau graphe

Etape : La tâche de commande est divisée en plusieurs étapes. C'est dans les étapes que sont formulées les actions exécutées par la commande séquentielle.



Etape active : Une étape active est une étape dont les actions sont en cours d'exécution, et ceci quand :

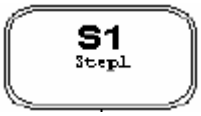
- la réceptivité de la transition qui la précède est vraie (conditions de franchissement remplies)
- elle est définie comme étape initiale et que le graphe séquentiel a été initialisé ou encore elle est appelée par une action déclenchée par un événement.

Paire étape/transition : Par défaut, le FB S7-GRAPH contient déjà une paire étape/transition vide à laquelle on peut ajouter d'autres paires, elles reçoivent automatiquement un numéro lors de l'insertion.

Etape initiale : L'étape initiale est l'étape d'un graphe séquentiel qui est d'abord activée au premier appel d'un FB S7-GRAPH, sans tenir compte de l'état des conditions. Elle n'est pas obligatoirement la première étape du graphe.

Durant l'exécution cyclique du graphe séquentiel, cette étape n'est activée, comme toute autre étape, que lorsque la réceptivité de la transition précédente est vraie.

Quand le paramètre de FB INIT_SQ est à 1, le graphe séquentiel est initialisé, c'est-à-dire qu'il démarre à l'étape initiale.



Saut : Un saut permet de passer d'une transition à une étape quelconque dans un graphe séquentiel il est toujours placé après une transition et met fin, à cette position, au graphe séquentiel ou à la séquence d'une branche.

Contrairement à la fin de graphe, le saut provoque la reprise du traitement du graphe séquentiel ou de parties du graphe.

Branche OU : Une branche OU se compose de plusieurs séquences parallèles (125 au plus). Chaque séquence OU commence par une transition. La seule séquence exécutée est celle dont la transition est franchie en premier. La branche OU est donc un aiguillage traduisant le choix conditionnel entre plusieurs séquences dont une seule peut être active.

Branche ET : Une branche ET se compose de plusieurs séquences parallèles (249 max.) commençant chacune par une étape. Ces séquences sont exécutées simultanément. La branche ET correspond à une séquence simultanée. La transition précédant la branche ET active les premières étapes des deux différentes branches ET.

Quand plusieurs séquences ET sont regroupées sur la même transition, celle-ci ne sera franchie qu'une fois toutes les séquences ET actives entièrement traitées.

Fin de graphe : Une fin de graphe, placée à la fin d'un graphe séquentiel linéaire ou à la fin d'une séquence dans une branche OU, met fin au graphe. Dans ce cas, le graphe séquentiel n'est pas traité de façon cyclique. Une fin de graphe à la fin d'une branche ET ne met fin qu'à cette branche. L'exécution des autres branches se poursuit.

Principe du traitement d'un graphe séquentiel :

Le traitement d'un graphe séquentiel commence toujours par :

- une étape initiale ou plusieurs étapes initiales pouvant se trouver à un endroit quelconque du graphe séquentiel.

Une étape active est désactivée :

- quand tous les défauts éventuellement signalés ont disparu ou ont été acquittés, et que la réceptivité de la transition suivant l'étape est vraie.

A la fin d'un graphe séquentiel, on peut trouver :

- un saut à une étape quelconque de ce graphe ou d'un autre graphe du FB, ce qui rend l'exécution cyclique du graphe séquentiel possible.
- une fin de graphe qui met fin à l'exécution.

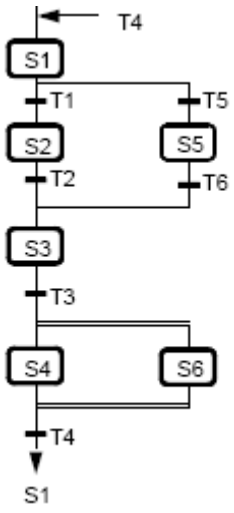
Structures d'un graphe séquentiel :

La structure la plus simple présentée par un graphe séquentiel est une suite linéaire d'étapes et de transitions sans aucune branche.

Un graphe linéaire débute par une étape et se termine par une transition qui peut être suivie soit par un saut à une étape quelconque, soit par une fin de graphe.

Cette suite linéaire peut être complétée par :

- Des branches (divergence et convergence en ET, en OU).
- Des sauts à des étapes quelconques.
- D'autres graphes séquentiels dépendant du premier graphe ou se déroulant de façon entièrement autonome.
- Des opérations permanentes placées en amont ou en aval du graphe séquentiel.



GRAPH avec graphe séquentiel comportant une branche OU (divergence en OU et convergence en OU) et une branche ET (divergence en ET et convergence en ET)

Règles de structuration d'un graphe séquentiel :

La structure d'un graphe séquentiel doit obéir aux règles suivantes :

- Un FB S7-GRAPH peut contenir jusqu'à 250 étapes ou transitions. Les étapes et transitions ne peuvent être insérées que par paires

Les graphes séquentiels démarrent à l'appel du FB S7-GRAPH.

- par la première étape du graphe respectif
- par une étape initiale

Un graphe séquentiel peut contenir au maximum 256 branches composées de :

- 125 branches OU au plus.
- 249 branches ET au plus.

Mais pour des raisons de durée d'exécution, il est recommandé de se limiter à un nombre de séquences compris entre 20 et 40 par CPU.

Une branche ne peut être refermée (convergence) que sur une séquence située à la gauche de la séquence à fermer.

Un saut peut être inséré après une transition à la fin d'une branche. Il aboutira devant une étape soit du même graphe, soit d'un autre graphe du FB en cours.

Une fin de graphe peut être ajoutée derrière une transition à la fin d'une branche et interrompra alors le traitement de cette branche.

Des opérations permanentes peuvent être définies dans la zone prévue à cet effet avant ou après le graphe séquentiel. Elles seront appelées une fois par cycle.

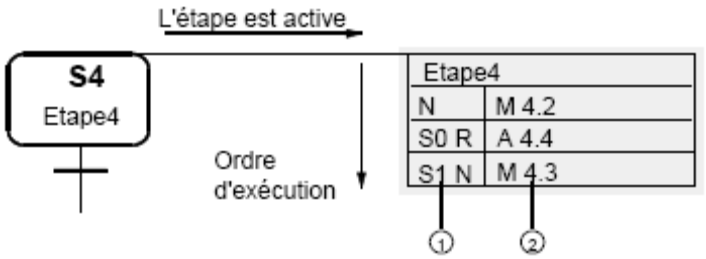
Programmation des actions et des conditions

Actions : On programme dans les étapes des actions qui activent ou désactivent entre autres des entrées sorties et des mémentos, des étapes du graphe séquentiel ou appellent des blocs. Les actions contiennent donc des instructions destinées à la commande du processus. Elles sont exécutées de "haut" en "bas" quand l'étape est active.

Composants d'une action : Une action se compose de

1 - un événement facultatif (ex. : S1) et une opération (ex. : N).

2 - un opérande (ex. : M4.3) ou une affectation (ex. : A:=B+C).



Types d'action : On distingue :

- Les actions standard : avec et sans Interlock
- Les actions déclenchées par un événement : avec et sans Interlock pour activer et désactiver des étapes, les compteurs, temporisations et expressions arithmétiques dans des actions.

Etape vide (étapes sans actions) : L'étape pour laquelle aucune action n'a été programmée est une étape vide, elle se comporte comme une étape active. La transition qui la suit est immédiatement validée.

Conditions : Les conditions sont des états logiques du processus qui, en tant qu'éléments CONT ou LOG (Contact à fermeture, contact à ouverture, comparateur, boîte ET, boîte OU, comparateur), peuvent être combinés entre eux dans le schéma à contacts ou dans le logigramme. Les fonctions combinatoires sont programmées en CONT ou en LOG.Les conditions sont :

- des événements (par exemple Fin de l'étape active).
- des états (par exemple Mise à 1 de l'entrée E2.1).

Champs pouvant contenir des conditions : Les conditions apparaissent dans les champs suivants :

- Transition (réglant l'évolution entre deux étapes successives).
- Interlock (verrouillage d'étape).
- Supervision (surveillance d'étape)
- Opérations permanentes (conditions et appels de bloc au début et/ou à la fin du graphe séquentiel).

Transition : Dans une transition, on programme des conditions qui commandent l'évolution entre deux étapes successives, elle s'affiche et se programme en représentation étape.
La transition est franchie lorsque la fonction combinatoire des conditions de franchissement (réceptivité) est vraie, c'est-à-dire donne le résultat 1.
L'étape immédiatement suivante est alors activée.
La transition n'est pas franchie lorsque la réceptivité n'est pas vrai, c'est-à-dire donne le résultat 0.L'étape qui était active le reste.

Particularités dans la programmation d'un graphe séquentiel

Initialisation : Un graphe séquentiel est initialisé au moyen du paramètre INIT_SQ. Quand ce dernier passe à "1", toutes les étapes repérées comme étape initiale sont activées. Toutes les autres étapes sont désactivées.
Une fois la commande mise en circuit, le graphe séquentiel se retrouve dans le mode de fonctionnement qui était le sien avant la mise hors circuit. S'il était en mode SW_AUTO avant la

mise hors circuit, il sera dans le même mode après la remise en circuit. Si le programme requiert un autre mode, il faut commuter l'automate explicitement dans le mode de fonctionnement souhaité (par exemple SW_MAN) après la mise en circuit, par exemple à l'aide du signal d'initialisation. Si l'état du processus (installation ou machine) ne correspond pas à l'état initial du graphe séquentiel, on a le choix entre deux solutions :

- Mettre le processus en position initiale au moyen d'une commande ; donc créer un graphe séquentiel particulier.
- Synchronisez le graphe séquentiel avec l'état du processus à l'aide de la fonction de synchronisation.

Verrouillages : Il est possible dans S7-GRAPH de définir pour chaque étape des conditions de verrouillage comme "Interlock". Toutes les actions dont l'exécution est conditionnelle ne seront activées que si les conditions de l'Interlock sont remplies.

Surveillances : On peut dans S7-GRAPH de définir dans chaque étape des conditions de surveillance comme "Supervision". C'est le plus souvent la durée d'exécution des actions qui est surveillée.

Arrêt anormal : On parle d'arrêt anormal quand une situation critique provoque un "Stop immédiat" ou un "Arrêt d'urgence" ou quand il y a un défaut sur machine. Les actions suivantes sont alors exécutées :

- Les graphes séquentiels sont arrêtés par commutation sur SW_MAN.
- Au niveau des machines, les signaux de sortie sont désactivés par inhibition du mode automatique.

Synchronisation : Il est nécessaire de synchroniser le graphe séquentiel avec l'état de l'installation ou de la machine, dans les cas suivants.

Cas 1 : une modification du programme durant la phase de mise en service a entraîné une nouvelle génération du DB d'instance.

Cas 2 : en mode manuel, les machines ont été actionnées manuellement, de sorte que l'état du graphe ne correspond plus à celui de l'installation ou de la machine ; après l'intervention manuelle, il faut reprendre le fonctionnement automatique sur la base de l'état en cours de l'installation ou de la machine. Le graphe séquentiel doit donc être synchronisé sur le processus.

Modifications durant la mise en service : Si la structure du graphe séquentiel est modifiée lors de la mise en service, par l'ajout d'une étape par exemple, ou sa suppression ou renumérotation, ou encore la suppression ou l'ajout d'une transition, un nouveau DB d'instance sera généré à la compilation. Le chargement de ce DB d'instance dans l'automate programmable a le même effet que l'initialisation au moyen de l'entrée INIT_SQ.

SCL

1-Langage de programmation SCL :

SCL (Structured Control Language) est un langage de programmation évolué proche du PASCAL, il contient outres les éléments de langage évolué des éléments typiques des automates programmables comme les entrées, les sorties, temporisations, appels de bloc, etc.
Les langages de programmation de STEP 7 et SCL se complètent, ainsi un bloc programmé en SCL peut en appeler un autre programmé en LIST, CONT ou LOG, et vice versa.

2-Opérations, opérandes et expressions:

2-1-Opérations :

La plupart des opérations de SCL sont binaires en combinant deux opérandes (exemple A+B), les autres opérations n'utilisant qu'un opérande sont des opérations unaires (exemple -B), l'ordre de calcul des opérations est déterminé par leurs priorités qui sont données par le tableau suivant : ("1" correspond à la priorité la plus élevée)

Classe	**Opération**	**Représentation**	**Priorité**
Opération d'affectation :	Affectation	: =	11
Opérations arithmétiques :	Puissance	**	2
	Plus unaire	+	3
	Moins unaire	-	3
	Opérations arithmétiques de base :		
	Multiplication	*	4
	Division	/	4
	Fonction modulo	MOD	4
	Division entière	DIV	4
	Addition	+	5
	Soustraction	-	5
Opérations de comparaison :	Infériorité	<	6
	Supériorité	>	6
	Infériorité ou égalité	<=	6
	Supériorité ou égalité	>=	6
	Egalité	=	7
	Différence	<>	7
Opérations logiques :	Et	AND ou &	8
	Ou exclusif	XOR	9
	Ou	OR	10
	Négation	NOT	3
Caractères de parenthèse :	Parenthèses	( )	1

2-2-Opérandes :

Les opérandes sont des objets avec lesquels on peut former une expression, les éléments autorisés pour former des opérandes sont :

- Les constantes.
- Les variables étendues.

- Les expressions.

2-2-1-Constantes : On peut utiliser des constantes sous forme de valeur numérique, de mnémonique ou de chaîne de caractère, comme exemple de constante autorisée on peut citer : 4_711, 'CARACTERE',…

2-2-2Variables étendues : Les variables étendues sont des termes génériques désignant une série de variables, les types de variables existant sont : Variables simples, variables absolues, variables dans un DB, variables dans une instance locale et les appelles de FC, et comme exemples de variable E100.5 //variable absolue

Remarques :

- Un opérande placé entre deux opérations de priorités différentes est toujours lié à celle qui a la plus haute priorité.
- Les opérations de même priorité sont traitées de gauche à droite.
- Les opérations arithmétiques ne peuvent pas venir immédiatement à la suite l'une de l'autre, ainsi l'expression a * - b n'est pas autorisée, par contre a*(-b) l'est.
- Les expressions entre parenthèses sont considérées comme des opérandes uniques et sont toujours traitées en premier.

2-3-Les expressions : Une expression représente une valeur qui sera calculée durant l'exécution du programme, elle comporte des opérandes (par exemple des constantes, des variables ou des appels de fonction) et des opérations (par exemple *, /, + ou -), le type de l'expression est déterminé par les types de données des opérandes et les opérations utilisées, le résultat d'une expression peut être affecté à une variable, utilisé comme condition d'une instruction de contrôle ou utilisé comme paramètre pour l'appel d'une fonction ou d'un bloc fonctionnel.
SCL distingue les :

- Expressions arithmétiques.
- Expressions logiques.
- Expressions de comparaison.

2-3-1-Expressions arithmétiques : Il s'agit d'expressions formées d'opérations arithmétiques, elles permettent de traiter des types de données numériques et non pas des caractères ou des données logique, le tableau ci-après regroupe les opérations possibles et indique à quel type il faut associer le résultat, en fonction des opérandes utilisés:

Opération	**Identificateur**	**1er opérande**	**2nd opérande**	**Résultat**	**Priorité**
Puissance	**	ANY_NUM	ANY_NUM	REAL	2
Plus unaire	+	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	3

Moins unaire	-	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	3
Multiplication	*	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY_INT	TIME	4
Division	/	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY_INT	TIME	4
Division entière	DIV	ANY_INT	ANY_INT	ANY_INT	4
		TIME	ANY_INT	TIME	4
Division modulo	MOD	ANY_INT	ANY_INT	ANY_INT	4
Addition	+	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	5
		TOD	TIME	TOD	5
		DT	TIME	DT	5
Soustraction	-	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	5
		TOD	TIME	TOD	5
		DATE	DATE	TIME	5
		TOD	TOD	TIME	5
		DT	TIME	DT	5
		DT	DT	TIME	5

ANY_INT : Désigne les types de données INT, DINT.

ANY_NUM : Désigne les types de données INT, DINT et REAL.

Remarques :

- Pour les divisions avec deux opérandes entiers de type INT, les opérations "DIV" et "/" fournissent le même résultat.
- Pour les opérations de division "/", "MOD" et "DIV", le second opérande doit être différent de zéro.
- Lorsque l'un des opérandes est du type INT et le second du type REAL le résultat est de type REAL.

Exemples :

VALEUR1 := 3 + 3 * 4 / 2 - (7+3) / (-5) ; // La VALEUR de l'expression suivante est 1

2-3-2-Expressions logiques :

Il s'agit des expressions formées d'opérations logiques, le résultat de l'expression est TRUE ou FALSE lorsque on combine des opérandes booléens, soit un profil binaire après combinaison binaire des deux opérandes.

Les expressions logiques disponibles ainsi que le type de données du résultat et des opérandes sont indiqués dans le tableau suivant :

Opération	**Identificateur**	**1er opérande**	**2nd opérande**	**Résultat**	**Priorité**
Négation	NOT	ANY_BIT	-	ANY_BIT	3

Intersection	AND	ANY_BIT	ANY_BIT	ANY_BIT	8
OU exclusif	XOR	ANY_BIT	ANY_BIT	ANY_BIT	9
OU inclusif	OR	ANY_BIT	ANY_BIT	ANY_BIT	10

ANY_BIT : Désigne les types de données BOOL, BYTE, WORD, DWORD.

2-3-3-Expressions de comparaison :

Les expressions de comparaison comparent les valeurs de deux opérandes et fournissent une valeur booléenne TRUE si la comparaison s'avère juste, et FALSE dans le cas contraire.

Remarques :

- Toutes les variables peuvent être comparées au sein des classes de types suivantes : INT, DINT, REAL, BOOL, BYTE, WORD, DWORD, CHAR et STRING.
- La comparaison de caractères s'effectue conformément à la chaîne de caractères ASCII.

3-Les instructions :

Une instruction est la plus petite unité autonome du programme utilisateur, elle représente une tâche que doit exécuter le processeur. Les types d'instruction existants sur SCL sont :

- Affectation de valeur permettant d'attribuer à une variable soit une valeur, soit le résultat d'une expression ou encore la valeur d'une autre variable.
- Instruction de contrôle (de branchement) servant à répéter des instructions ou des groupes d'instructions ou à orienter la suite du déroulement d'un programme.
- Exécution de sous-programmes destinés à appeler des fonctions et des blocs fonctionnels.

3-1-Affectations de valeur (assignation) :

Une affectation de valeur remplace la valeur momentanée d'une variable par une nouvelle valeur, indiquée sous forme d'expression qui peut contenir des identificateurs de fonctions qui s'en trouvent activées et fournissent des valeurs correspondantes (valeurs de retour).

Syntaxe de l'affectation de valeur : Variable := Expression

L'expression figurant dans la partie droite de l'affectation de valeur est évaluée et le résultat est mémorisé dans la variable dont le nom figure à gauche du signe d'affectation.

3-1-1-Affecter une valeur à une variable de type de données simple :

Toute expression et toute variable de type de données simple peuvent être affectées à une autre variable de même type.

Identificateur := expression ;

3-1-2-Affecter une valeur à des variables de type STRUCT et UDT :

Les variables de type STRUCT et UDT sont des variables complexes qui représentent soit une structure complète, soit un composant de cette structure, voici des indications correctes de variables complexes :

Image //Identificateur d'une structure

Image.tableau [2,5] //Identificateur d'un composant de tableau au sein d'une structure

3-1-3-Affectation d'une structure complète :

Une structure complète ne peut être affectée à une autre structure que si les types de données et les noms des composants sont identiques, voici une affectation correcte :

nomstruct_1 := nomstruct_2;

3-1-4-Affectation de composants de structure :

On peut affecter à tout composant d'une structure une variable ou une expression de type compatible ou encore un autre composant de structure.

Pour spécifier un composant de structure, on indique l'identificateur de la structure suivi d'un point et de l'identificateur du composant, voici des exemples d'affectations correctes :

nomstruct_1.element1 := 20.0 ;

nomstruct_1.nomtableau [10] := 100 ;

3-1-5-Affecter une valeur à une variable de type ARRAY :

Un tableau se compose de 1 à 6 dimensions au maximum et contient des éléments qui sont tous du même type, deux possibilités s'offrent à nous pour affecter un tableau à une variable,on peut référencer des tableaux complets ou des tableaux partiels.

3-1-5-1-Affectation d'un tableau complet :

Pour qu'on puisse affecter un tableau complet à un autre tableau, il faut que les types de données des composants et les limites du tableau (plus petit et plus grand indice) soient identiques, si tel est le cas, on indique l'identificateur du tableau à la suite du signe d'affectation.

3-1-5-2-Affectation d'un composant de tableau :

Pour adresser un élément particulier du tableau, on doit indiquer le nom du tableau suivi des valeurs appropriées des indices entre crochets, ils sont séparés par des virgules et placés entre crochets, un indice doit être une expression arithmétique de type INT.

Pour affecter une valeur à un tableau partiel, on supprime des indices entre les crochets qui suivent le nom du tableau, en commençant par la droite, on adresse ainsi un tableau partiel dont le nombre de dimensions correspond au nombre d'indices omis.

3-1-6-Affecter une valeur à une variable de type STRING :

Une variable de type STRING contient une chaîne de 254 caractères au maximum, on peut lui affecter une autre variable de même type.

3-1-7-Affecter une valeur à une variable de type DATE_AND_TIME :

Le type de données DATE_AND_TIME définit une zone de 64 bits (8 octets) pour indiquer la date et l'heure. A toute variable de type DATE_AND_TIME, on peut affecter une autre variable de même type ou une constante.

3-2-Les instructions de contrôle :

3-2-1-Instructions de sélection :

Une instruction de sélection vous permet d'aiguiller le déroulement du programme vers différentes séquences d'instructions en fonction des conditions.

3-2-1-1-Instruction IF :

Il s'agit d'une instruction conditionnelle, elle permet d'aiguiller le déroulement du programme vers l'une de deux alternatives, en fonction d'une condition qui est soit vraie, soit fausse.

Syntaxe : IF Expression THEN Instruction

ELSEIF Expression THEN Instruction

ELSE Instruction

END IF

Remarque : Le nombre d'instructions ELSIF est illimité.

3-2-1-2-Instruction CASE :

L'instruction CASE permet d'orienter le déroulement du programme vers l'une parmi n variantes, en faisant prendre à une variable une valeur parmi n valeurs possibles.

Syntaxe : CASE Expression OF

Liste de valeur : Instruction

ELSE : Instruction

END CASE

3-2-2-Instruction de boucles (itération):

On peut programmer l'exécution de boucles en utilisant des instructions d'itération, une instruction d'itération indique quelles parties d'un programme doivent être répétées en fonction de certaines conditions.

3-2-2-1-Instruction FOR :

L'instruction FOR sert à répéter une suite d'instructions aussi longtemps qu'une variable de contrôle se trouve dans la plage de valeurs spécifiée.

Syntaxe: FOR valeur initiale TO valeur finale

BY valeur de l'incrément DO instruction

END FOR

Remarques:

1- L'indication de BY est facultative. En son absence, la valeur utilisée est +1.

2- Valeur initiale, valeur finale et incrément sont des expressions.

3-2-2-2-Instruction WHILE :

L'instruction WHILE permet de reprendre l'exécution d'une suite d'instructions tant qu'une condition d'exécution est remplie.

Syntaxe : WHILE expression DO instruction

END WHILE

3-2-2-3-Instruction REPEAT :

L'instruction REPEAT provoque l'exécution répétée de la suite d'instructions figurant entre REPEAT et UNTIL, jusqu'à ce que la condition d'abandon soit remplie.

Syntaxes: REPEAT instruction UNTIL expression

END REPRAT

Remarques : l'expression représente la condition d'abondons.

3-2-3-Saut dans le programme (branchement) :

Un saut dans le programme provoque le saut immédiat à un repère spécifié et donc à une autre instruction dans le même bloc.

3-2-3-1-Instruction CONTINUE :

L'instruction CONTINUE sert à abandonner l'exécution du parcours momentané d'une instruction d'itération (FOR, WHILE ou REPEAT).

Syntaxes: CONTINUE;

3-2-3-2-Instruction EXIT :

L'instruction EXIT sert à quitter une boucle (FOR, WHILE ou REPEAT) à un endroit quelconque, que la condition d'abandon soit remplie ou pas.

Syntaxes: EXIT;

3-2-3-3-Instruction GOTO :

L'instruction GOTO permet de réaliser un branchement dans le programme, elle provoque le saut immédiat à un repère de saut spécifié.

L'instruction GOTO ne doit être utilisée qu'à titre exceptionnel, par exemple pour le traitement des erreurs.

Syntaxe : GOTO Identificateur (Repère de saut)

3-2-3-4-Instruction RETURN :

L'instruction RETURN permet de quitter un bloc en cours d'exécution (OB, FB, FC) et de revenir au bloc appelant ou au système d'exploitation, si c'est un OB qui est quitté.

Syntaxe : RETURN

3-3-Appel de fonctions et de blocs fonctionnels :

Dans un bloc SCL, on a la possibilité d'appeler d'autres FC ou FB, à savoir :

- Les blocs fonctionnels et fonctions créés dans SCL.
- Les blocs fonctionnels et fonctions créés dans d'autres langages de programmation de STEP7.

- Des fonctions système (SFC) et blocs fonctionnels système (SFB), disponibles dans le système d'exploitation de la CPU.

Lorsque on appel une fonction ou un bloc fonctionnel, un échange de données aura lieu entre le bloc appelant et le bloc appelé, les paramètres à transmettre doivent être indiqués sous forme de liste de paramètres lors de l'appel, ils doivent figurer entre parenthèses et être séparés par des virgules.

3-3-1-Appel de blocs fonctionnels :

Pour l'appel d'un bloc fonctionnel, on peut utiliser un bloc de données d'instance global ou une zone d'instance locale du bloc de données d'instance actif, ça dépend de l'endroit où les données seront enregistrées, dans le premier cas, les données sont stockées dans un DB séparé, et dans le deuxième cas, les données sont stockées dans le bloc de données d'instance du FB appelant.

3-3-1-1-Appel comme instance globale :

L'appel d'une instance globale peut être absolu ou symbolique, et on indique dans l'instruction d'appel :

- Le nom du bloc fonctionnel ou du bloc fonctionnel système (FB ou de SFB).
- Le bloc de données d'instance (désignation de DB).
- Les valeurs des paramètres (paramètres du FB)

3-3-1-2-Appel comme instance locale :

L'appel d'une instance locale est toujours symbolique, on doit déclarer le mnémonique correspondant dans la section de déclaration du bloc appelant, dans l'instruction on indique :

- Le nom de l'instance locale (identificateur).
- Les valeurs des paramètres (paramètres du FB).

Remarque :

A l'appel d'un bloc fonctionnel (comme instance globale ou locale) on doit définir les paramètres d’entrée et d’entrée/sortie dans la liste des paramètres.

3-3-2-Appel de fonctions :

Pour appeler une fonction, on indique son nom absolue ou symbolique (désignation de FC, de SFC ou Identificateur) ainsi que la liste des paramètres.

FC31 (X1:=5, Q1:=total_horizontal) // sous forme absolue

ECART (X1:=5, Q1=:total_horizontal) // sous forme symbolique

Après l'appel, les résultats de la fonction sont disponibles comme valeurs de retour ou comme paramètres de sortie et d'entrée/sortie et peuvent être utilisée dans les éléments d'une FC ou d'un FB comme une affectation de valeur, une expression logique, arithmétique ou de comparaison ou comme paramètre pour un autre appel de bloc fonctionnel ou de fonction.

Remarque :

L'appel dans SCL d'une fonction dont la valeur de retour n'a pas été définie peut entraîner une exécution erronée du programme utilisateur.

4-Les compteurs et les temporisations :

4-1-Fonctions de temporisations :

Les temporisations sont des éléments fonctionnels de programme permettant d'exécuter et de contrôler des séquences déclenchées par horloge.

Le tableau suivant indique les paramètres pour une fonction de temporisation.

Paramètre	**Type de données**	**Description**
T_NO	TIMER INTEGER	Numéro d'identification de la temporisation ; la plage dépend de la CPU.
S	BOOL	Entrée de démarrage
TV	S5TIME	Valeur par défaut de la durée (format DCB)
R	BOOL	Entrée de remise à 0
Q	BOOL	Etat de la temporisation
BI	WORD	Durée restante (binaire)

4-1-1-Les types de temporisation :

STEP 7 propose diverses fonctions standard de temporisation qu'on peut utiliser avec SCL.

4-1-1-1-S_PULSE : une temporisation sous forme d'impulsion.

La durée maximale pendant laquelle le signal de sortie reste à "1" correspond à la valeur de temps programmée. Si l'entrée prend l'état logique "0" durant l'exécution de la temporisation, celle-ci est mise à "0". Ceci représente un arrêt prématuré de l'exécution.

Fonctionnement :

La temporisation démarre lorsque l'état logique de l'entrée de démarrage (**S**) passe de "0" à "1" et continue à courir avec la valeur indiquée à l'entrée **TV** et l'interrogation à "1" de la sortie **Q** fournit le résultat "1", jusqu'à ce que la durée programmée soit écoulée tant que l'entrée **S** = 1, si **S** passe de "1" à "0" avant que la valeur de temps ne soit écoulée, la temporisation s'arrête, elle est remise à "0" lorsque l'entrée (**R**) passe de "0" à "1" pendant que la temporisation court, la valeur de temps et la base de temps sont alors également remises à zéro, on peut interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_PULSE.

4-1-1-2-S_PEXT : une temporisation sous forme d'impulsion prolongée.

Le signal de sortie reste à "1" durant le temps (t) programmé, quel que soit le temps pendant lequel le signal d'entrée reste à "1". Si l'impulsion de démarrage est à nouveau déclenchée, la valeur de temps s'écoule de nouveau, de sorte que l'impulsion de sortie est prolongée dans le temps (redémarrage).

Fonctionnement :

L'opération "Démarrer une temporisation sous forme d'impulsion prolongée" démarre la temporisation indiquée lorsque l'état logique de l'entrée (**S**) passe de "0" à "1", si l'état logique de l'entrée **S** passe à nouveau à "1" durant l'exécution, la temporisation est redémarrée avec la valeur de temps indiquée à l'entrée **TV** jusqu'à ce que la durée programmée soit écoulée, Tant que la temporisation court, l'interrogation à "1" de la sortie **Q** fournit le résultat "1", la temporisation est remise à 0 lorsque l'entrée (**R**) passe de "0" à "1" pendant que la temporisation court.

On peut interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_PEXT.

4-1-1-3-S_ODT : une temporisation sous forme dei retard à la montée.

Le signal de sortie ne passe de "0" à "1" que lorsque la temporisation programmée s'est écoulée et que le signal d'entrée est toujours "1". Ceci signifie que la sortie est activée avec un retard. Les signaux d'entrée dont la durée est plus courte que celle de la temporisation programmée n'apparaissent pas à la sortie.

Fonctionnement :

L'opération "Démarrer une temporisation sous forme de retard à la montée" démarre la temporisation indiquée lorsque l'état logique de l'entrée de démarrage (**S**) passe de "0" à "1", et continue de courir avec la valeur indiquée à l'entrée **TV** tant que l'état logique de l'entrée S = 1, si l'état logique de l'entrée **S** passe de "1" à "0" pendant que la temporisation court, celle-ci est arrêtée, elle est remise à 0 si (**R**) passe de "0" à "1" pendant que la temporisation court, la temporisation est aussi remise à zéro lorsque **R** = 1 pendant qu'elle ne court pas.

Une interrogation à "1" de la sortie **Q** fournit le résultat "1" lorsque la temporisation s'est écoulée correctement et que l'entrée **S** est toujours à "1", si la temporisation a été arrêtée, l'interrogation à "1" donne toujours "0".

Une interrogation à "1" de la sortie **Q** fournit également le résultat "0" lorsque la temporisation ne court pas et que l'état logique de l'entrée **S** est toujours "1".

On peut interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_ODT.

4-1-1-4-S_ODTS : une temporisation sous forme de retard à la montée mémorisé.

Le signal de sortie ne passe de "0" à "1" que lorsque la temporisation programmée s'est écoulée, quel que soit le temps pendant lequel le signal d'entrée reste à "1".

Fonctionnement :

Une temporisation sous forme de retard à la montée mémorisé démarre lorsque l'état logique de l'entrée de démarrage (**S**) passe de "0" à "1", elle est redémarrée avec la valeur indiquée lorsque

l'entrée **S** passe de "0" à "1" pendant que la temporisation court avec la valeur indiquée à l'entrée **TV** même si l'état logique de **S** passe à "0" avant que la temporisation ne soit écoulée, quel que soit l'état logique de l'entrée **S**, la temporisation est remise à 0 lorsque l'entrée (**R**) passe de "0" à "1" et l'interrogation à "1" de la sortie **Q** fournit le résultat "1" lorsque la temporisation s'est écoulée.

On peut interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_ODTS.

4-1-1-5-S_OFFDT : une temporisation sous forme de retard à la retombée.

Lorsque l'état logique de l'entrée S passe de "0" à "1", la sortie Q prend l'état logique "1". Lorsque l'état logique de l'entrée de démarrage passe de "1" à "0", la temporisation est démarrée. Ce n'est qu'une fois la durée écoulée que la sortie prend l'état logique "0". La sortie est donc désactivée avec un retard.

Fonctionnement :

La temporisation sous forme de retard à la retombée démarre lorsque l'état logique de l'entrée de démarrage (**S**) passe de "1" à "0" et elle court avec la valeur indiquée à l'entrée **TV,** elle est redémarrée lorsque l'état logique de l'entrée **S** passe à nouveau de "1" à "0".

La temporisation est remise à 0 lorsque l'entrée de remise à zéro (**R**) passe de "0" à "1" pendant que la temporisation court, et une interrogation à "1" de la sortie **Q** fournit le résultat "1" lorsque l'état logique de l'entrée **S** = 1 ou lorsque la temporisation court.

On peut interroger la valeur de temps actuelle à la sortie **BI** à l'aide de la valeur de la fonction S_OFFDT.

Remarques :

1- Un changement d'état logique de l'entrée de démarrage (**S**) est toujours requis pour valider la temporisation.
2- lorsque l'entrée de remise à zéro (**R**) passe de "0" à "1" la valeur de temps et la base de temps sont alors également remises à zéro pour les temporisations S_PULSE, S_PEXT et S_ODT.

4-1-2-Appel d'une fonction de temporisation :

Les fonctions de temporisation sont appelées comme des fonctions, la valeur de retour de la fonction fournie à la position appelante, est une durée du type de données S5TIME.

On peut entrer dans l'appel une valeur absolue de type TIMER comme numéro de la temporisation (par ex. T_NO:=T10), mais il faut savoir qu'une telle valeur n'est plus modifiable à l'exécution c'est l'appel absolue, il y a aussi l'appel dynamique qui utilise des variables ou des constantes de type données INT ou TIMER à la place des valeurs absolues. **Remarque :**

A l'appel, il faut affecter une valeur au paramètre T_NO qui désigne la temporisation, que ce soit un numéro absolu de la temporisation ou une variable du type de données INT ou un paramètre d'entrée du type TIMER, la valeur de résultat en format S5TIME est toujours la valeur de la fonction.

4-2-Les compteurs:

STEP 7 propose diverses fonctions standard de comptage qu'on peut utiliser dans le programme SCL sans devoir les déclarer au préalable, il nous suffit de leur affecter les paramètres requis, les fonctions de comptage disponibles sont :

4-2-1-Compteur incrémental (Counter Up) S_CU :

Le compteur incrémental (S_CU) nous permet uniquement d'exécuter des opérations de comptage incrémental. Le tableau suivant indique comment le compteur fonctionne :

Opération	**Fonctionnement**
Incrémenter	La valeur du compteur est augmentée de 1 lorsque l'état logique de l'entrée **CU** passe de "0" à "1" et si la valeur de comptage est inférieure à 999.
Initialiser le compteur	Lorsque l'état logique de l'entrée **S** passe de "0" à "1", le compteur est initialisé avec la valeur de l'entrée **PV**. Un tel changement d'état logique est toujours requis pour initialiser un compteur.
Mettre le compteur à 0	Le compteur est mis à 0 lorsque l'entrée **R** = 1. A la mise à 0 du compteur, la valeur de comptage prend la valeur 0.
Interroger le compteur	L'interrogation de l'état logique de la sortie **Q** fournit la valeur "1" si la valeur de comptage est >0, et fournit "0" si la valeur de comptage est = 0.

4-2-2-Compteur décrémental (Counter Down) S_CD :

Le compteur décrémental (S_CD) nous permet uniquement d'exécuter des opérations de comptage décrémental, son fonctionnement est indiqué dans le tableau suivant :

Opération	**Fonctionnement**
Décrémenter	La valeur du compteur est diminuée de 1 lorsque l'état logique de l'entrée **CD** passe de "0" à "1" et si la valeur de comptage est supérieure à 0.
Initialiser le compteur	Lorsque l'état logique de l'entrée **S** passe de "0" à "1", le compteur est initialisé avec la valeur de l'entrée **PV**. Un tel changement d'état logique est toujours requis pour initialiser un compteur.
Mettre le compteur à 0	Le compteur est mis à 0 lorsque l'entrée **R** = 1. A la mise à 0 du compteur, la valeur de comptage prend la valeur 0.
Interroger le compteur	L'interrogation de l'état logique de la sortie **Q** fournit la valeur "1" si la valeur de comptage est >0, et fournit "0" si la valeur de comptage est =0.

4-2-3-Compteur incrémental/ décrémental (Counter Up Down) S_CUD :

Le compteur incrémental et décrémental (S_CUD) nous permet d'exécuter des opérations de comptage aussi bien incrémental que décrémental, le tableau suivant indique comment le compteur fonctionne :

Opération	**Fonctionnement**
Incrémenter	La valeur du compteur est augmentée de 1 lorsque l'état logique de l'entrée **CU** passe de "0" à "1" et si la valeur de comptage est inférieure à 999.
Décrémenter	La valeur du compteur est diminuée de 1 lorsque l'état logique de l'entrée **CD** passe de "0" à "1" et si la valeur de comptage est supérieure à 0.
Initialiser le compteur	Lorsque l'état logique de l'entrée **S** passe de "0" à "1", le compteur est initialisé avec la valeur de l'entrée **PV**. Un tel changement d'état logique est toujours requis pour initialiser un compteur.
Mettre le compteur à 0	Le compteur est mis à 0 lorsque l'entrée **R** = 1. A la mise à 0 du compteur, la valeur de comptage prend la valeur 0.
Interroger le compteur	L'interrogation de l'état logique de la sortie Q fournit la valeur 1 si la valeur de comptage est >0, et fournit 0 si la valeur de comptage est égale =0.

4-2-4-Définition des paramètres d'une fonction de comptage

Le tableau suivant indique les paramètres des fonctions de comptage :

Paramètre	**Type de données**	**Description**
C_NO	COUNTER INT	Numéro du compteur (DESIGNATION DU COMPTEUR) ; la plage dépend de la CPU.
CD	BOOL	Entrée CD : décrémenter
CU	BOOL	Entrée CU : incrémenter
S	BOOL	Entrée d'initialisation du compteur
PV	WORD	Valeur comprise entre 0 et 999 pour initialiser le compteur (saisie sous la forme 16#<valeur>, la valeur étant en format DCB)
R	BOOL	Entrée de remise à "0" du compteur
Q	BOOL	Sortie : état du compteur
CV	WORD	Sortie : état du compteur en binaire

4-2-5-Appel d'une fonction de comptage :

Les fonctions de comptage sont appelées comme des fonctions, la valeur de la fonction (valeur de retour), fournie à la position appelante, est la valeur de comptage actuelle (format DCB) dans le type de données WORD.

Pour appeler un compteur il y a 2 possibilités en SCL, soit un appel absolu et dans ce cas on entre une valeur absolue comme numéro de compteur, soit un appel dynamique et dans ce cas de figure on indique une variable ou une constante de type données INT ou COUNTER a la place du numéro de compteur.

LES PROGRAMMES

1. Programme Simulation numérique

Ce programme est partagé en 2 blocs d'organisation et 2 fonctions

1.1 Programme dans l'OB100

Dans cette partie de programme, on initialise les variables, et on calcule le nombre d'itérations

```
OB100 :  "Complete Restart"
Réseau  1: Titre :
          L     0
          T     "i"

     // le pas de descrétisation
          L     1.000000e+000
          T     "h"

     // y initiale
          L     1.000000e+000
          T     "yi"
     // w initiale
          L     2.000000e+000
          T     "wi"

     // x finale
          L     1.000000e+001
          T     #xf

     // x initiale
          L     0.000000e+000
          T     #xo
          T     "xi"

     // calcul du nombre de valeurs requises
          -R
          L     "h"
          /R
          RND
          T     "n"
```

1.2 Programme dans OB35

L'OB35 est constitué de 9 réseaux

```
OB35 :  "Cyclic Interrupt"
Réseau  1: Titre :
          L     "i"
          L     "n"
          <=I
          SPBN  fin
```

Réseau 2 : Titre :

```
// calcul de k1 & v1
      CALL  "Fonction Y"
       xi:="xi"
       yi:="yi"
       wi:="wi"
       ki:=#k1

      CALL  "Fonction W"
       xi:="xi"
       yi:="yi"
       wi:="wi"
       vi:=#v1
```

Réseau 3 : Titre :

```
// calcul de k2 & v2
      L     "h"
      L     5.000000e-001
      *R
      L     "xi"
      +R
      T     #x0
      L     #k1
      L     5.000000e-001
      *R
      L     "yi"
      +R
      T     #y0
      L     #v1
      L     5.000000e-001
      *R
      L     "wi"
      +R
      T     #w0

      CALL  "Fonction Y"
       xi:=#x0
       yi:=#y0
       wi:=#w0
       ki:=#k2

      CALL  "Fonction W"
       xi:=#x0
       yi:=#y0
       wi:=#w0
       vi:=#v2
```

Réseau 4 : Titre :

```
// calcul de k3 & v3
      L     "h"
      L     5.000000e-001
      *R
      L     "xi"
      +R
      T     #x0
      L     #k2
      L     5.000000e-001
      *R
      L     "yi"
      +R
      T     #y0
      L     #v2
      L     5.000000e-001
      *R
      L     "wi"
      +R
      T     #w0

      CALL  "Fonction Y"
       xi:=#x0
       yi:=#y0
       wi:=#w0
       ki:=#k3

      CALL  "Fonction W"
       xi:=#x0
       yi:=#y0
       wi:=#w0
       vi:=#v3
```

Réseau 5 : Titre :

```
// calcul de k4 & v4
      L     "h"
      L     "xi"
      +R
      T     #x0
      L     #k3
      L     "yi"
      +R
      T     #y0
      L     #v3
      L     "wi"
      +R
      T     #w0

      CALL  "Fonction Y"
       xi:=#x0
       yi:=#y0
       wi:=#w0
       ki:=#k4

      CALL  "Fonction W"
       xi:=#x0
       yi:=#y0
       wi:=#w0
       vi:=#v4
```

```
Réseau  6 : Titre :
    // calcul de y(i+1)
          L     #k2
          L     #k3
          +R
          L     2.000000e+000
          *R
          L     #k1
          +R
          L     #k4
          +R
          L     1.666700e-001
          *R
          L     "h"
          *R
          L     "yi"
          +R
          T     "yi"

Réseau  7 : Titre :
    // calcul de w(i+1)
          L     #v2
          L     #v3
          +R
          L     2.000000e+000
          *R
          L     #v1
          +R
          L     #v4
          +R
          L     1.666700e-001
          *R
          L     "h"
          *R
          L     "wi"
          +R
          T     "wi"

Réseau  8 : Titre :
    // calcul de xi
          L     "xi"
          L     "h"
          +R
          T     "xi"

Réseau  9 : Titre :
    // actualisation du nombre d'iteration
          L     "i"
          L     1
          +I
          T     "i"

    fin:  NOP   0
```

Les fonctions FC1 & FC2 traduisent les deux fonctions de Runge-Kutta y et w

2. Programme de la commande du four

Programme dans l'OB1 :

```
OB1 : Titre :
Réseau  1: Titre :
          CALL  "Le_Filtre"
           Mesure        :="T_Mesure"
           Mesure_Filtree:="T_Filtree"

          CALL  "Max"
           Mesure  :="T_Mesure"
           compteur:="Compteur"
           Max     :="T_Max"

          L     "Compteur"
          L     L#1
          +D

          T     "Compteur"
```

Programme dans l'OB100

```
OB100 :  "Complete Restart"
Réseau  1: Choix de la boucle de commande
          CALL  "Boucle de Commande"
           Boucle_CMD    :=FALSE
           Boucle_Ouverte:="Boucle_Ouverte"

Réseau  2: Chargement de la consigne
          CALL  "Chargement Consigne"
           Port          :="ADC2"
           Reference     :=30
           Consigne_Man  :=TRUE
           Boucle_Ouverte:="Boucle_Ouverte"
           Consigne      :="DAC"
           Consigne_2    :="T_Consigne"


Réseau  3: Choix de la regulation
          L     3
          T     "Selection_Regulation"
```

Réseau 4 : Initialisation des zones mémoires de travail

```
      L     0
      T     "T_Filtree"
      T     "T_Mesure"
      T     "L_Ecart"
      T     "Ecart_Prec"
      T     "T_sav"
      T     "T_Max"
      T     "u"
      T     "v"
      T     "es"

      L     L#0
      T     "Temps"
      T     "Compteur"
      T     "Somme"

      L     1000
      T     "n"
```

Réseau 5 : Simulateur

```
// le pas de descrétisation
      L     1.000000e-002
      T     "h"

// y initiale
      L     0.000000e+000
      T     "yi"

// to = RC la consatante de temps
      L     2.200000e+001
      T     "RC"
```

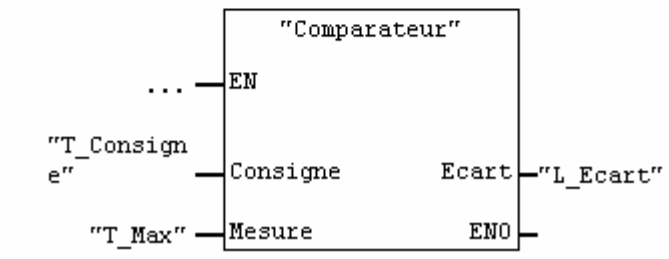
Programme dans OB35

OB35 : "Cyclic Interrupt"

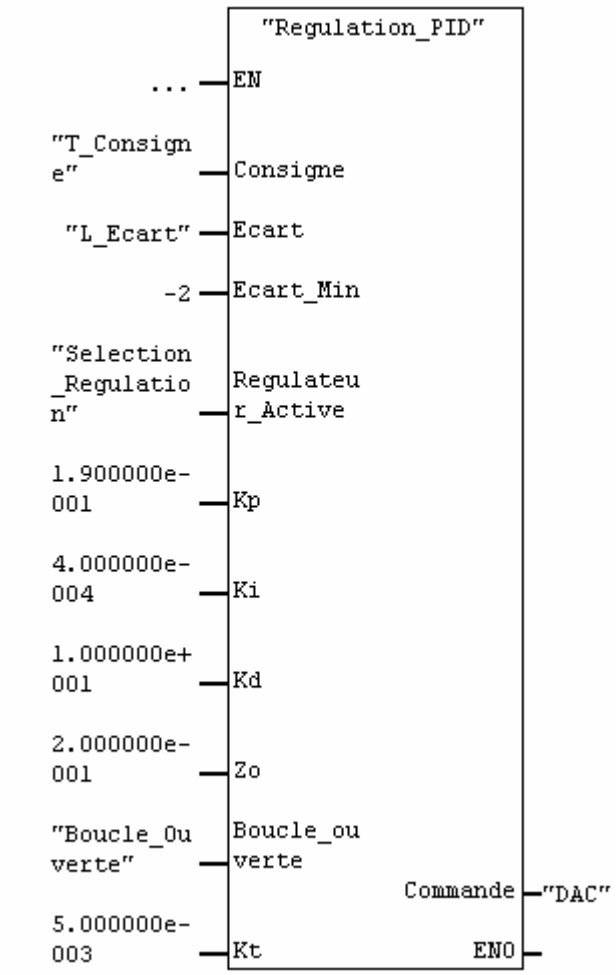
Réseau 1 : Titre :

```
      L     "Temps"
      L     L#1
      +D
      T     "Temps"


      L     "Temps"

      L     "T_Max"
      L     "T_Filtree"
```

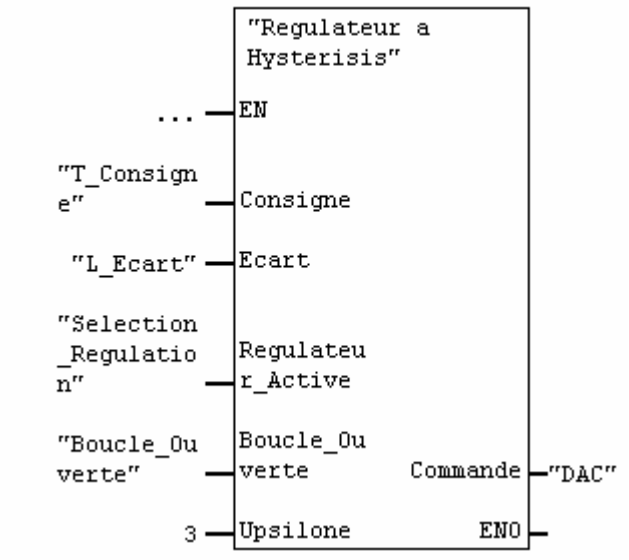
Réseau 2 : Comparateur



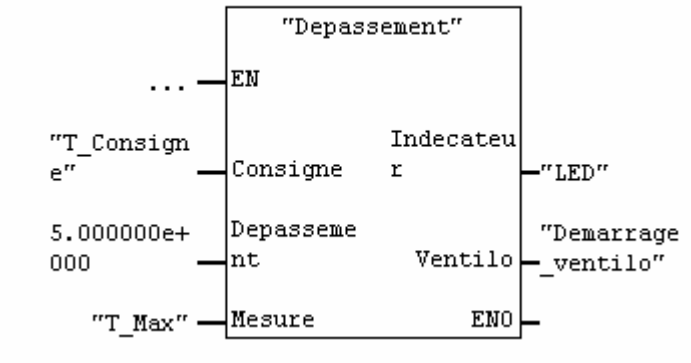
Réseau 3 : calcul de la commande "PID"



Réseau 4 : Calcul de la commande ON-OFF



Réseau 5 : Indecateur de dépassement



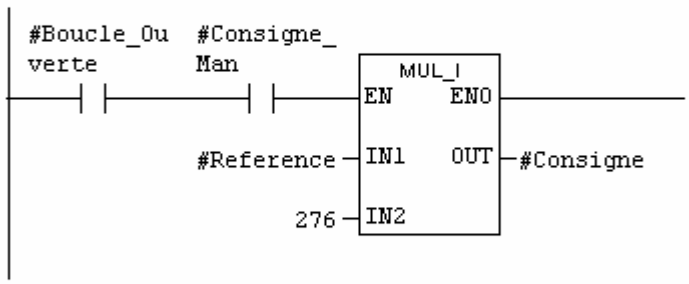
Programme dans FC2 lcture de la mesure

```
FC2 : Titre :
Réseau 1: Titre :
          L     #Port
          ITB
          L     276
          /D
          T     #Mesure
```

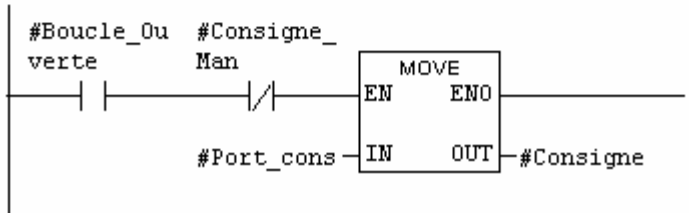
Programme dans FC3 Chargement de la consigne

```
FC3 : Titre :
Réseau 1 : Titre :
          L     #Port
          ITB
          T     #Port_cons
          NOP   0
```

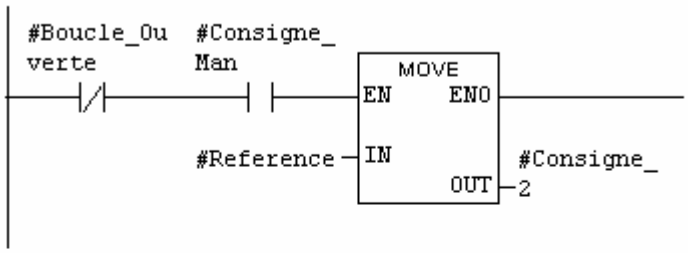
Réseau 2 : Titre :



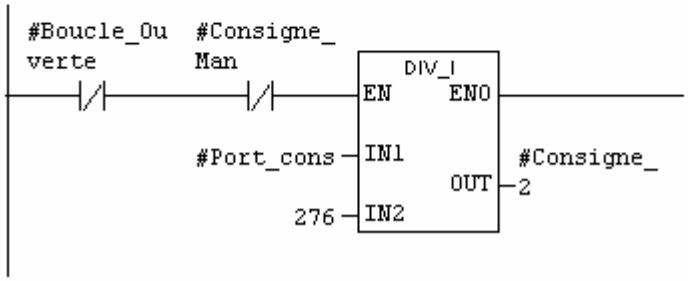
Réseau 3 : Titre :



Réseau 4 : Titre :



Réseau 5 : Titre :



Programme dans FC4 Le comparateur

```
FC4 : Titre :
Réseau  1: Titre :
          L     #Consigne
          L     #Mesure
          -I
          T     #Ecart
```

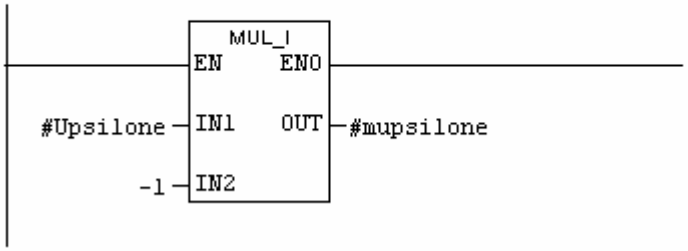
Programme dans FC5 Depassement

```
FC5 : Indécateur de dépassement
Réseau  1: Titre :
          L     #Consigne
          DTR
          L     #Depassement
          *R
          L     1.000000e+002
          /R
          RND
          L     #Consigne
          +I
          L     #Mesure
          <I
          =     #Indecateur
          =     #Ventilo
```

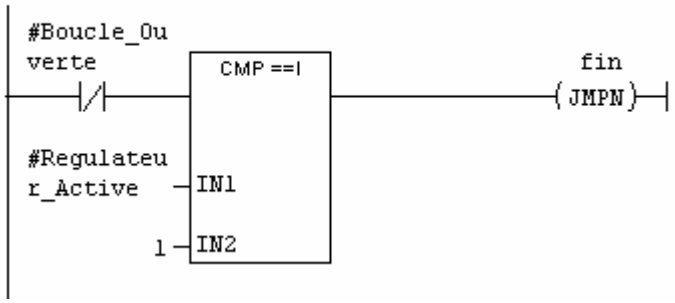
Programme dans FC8 Régulateur ON-OFF

FC8 : Régulateur à hysterisis

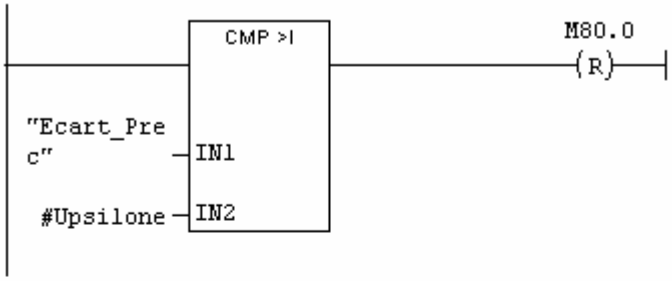
Réseau 1 : Titre :



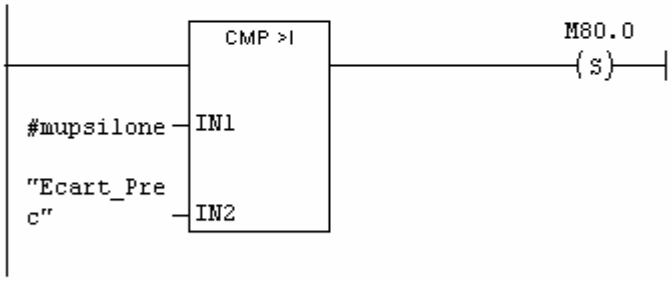
Réseau 2 : Titre :



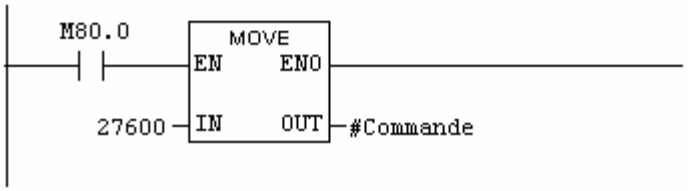
Réseau 3 : Titre :



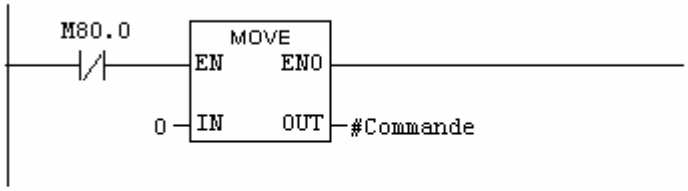
Réseau 4 : Titre :



Réseau 5 : Titre :



Réseau 6 : Titre :



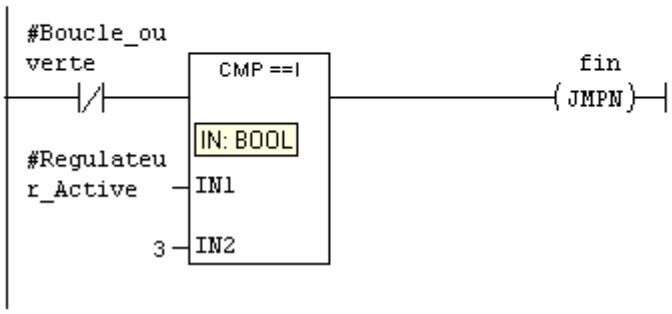
Réseau 7 : Titre :

```
    fin:  NOP   0
```

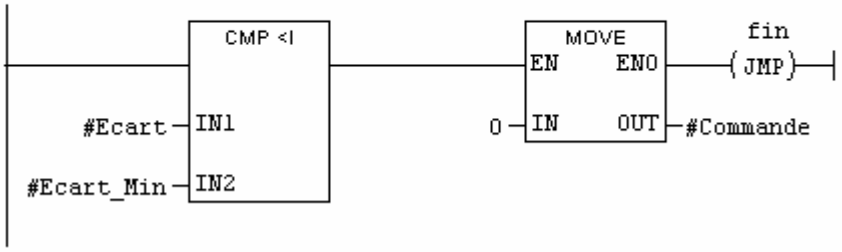
Programme dans FC9 Régulateur PID

FC9 : Régulation PID

Réseau 1 : Titre :



Réseau 2 : Titre :



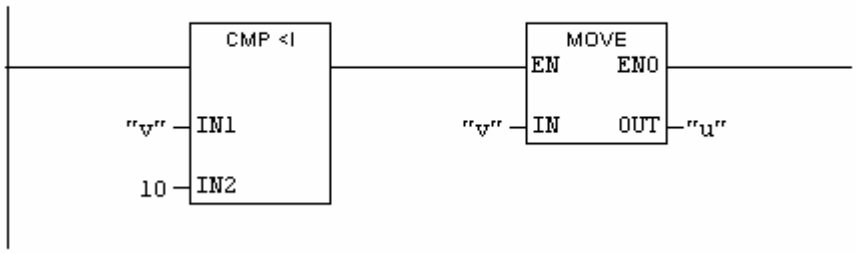
Réseau 3 : Titre :

```
    // calcul de Kpid
          L     #Kp
          L     #Ki
          +R
          L     #Kd
          +R
          T     #KPID
    // calcul de Kd
          L     1.000000e+000
          L     #Zo
          -R
          L     #Kd
          *R
          T     #Kd2
```

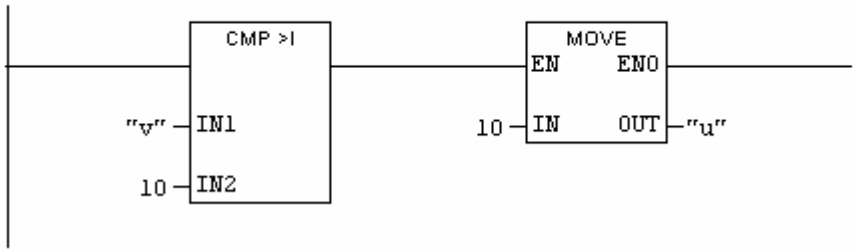
Réseau 4 : Titre :

```
      L     "Ecart_Prec"
      DTR
      L     #Kd2
      *R
      T     #Resultl
      L     #Ecart
      DTR
      L     #KPID
      *R
      L     #Resultl
      -R
      T     #Resultl
      L     "Somme"
      DTR
      L     #Ki
      *R
      L     #Resultl
      +R
      RND
      T     #cmd
      L     "es"
      DTR
      L     #Kt
      *R
      RND
      L     #cmd
      +I
      T     "v"
```

Réseau 5 : Titre :



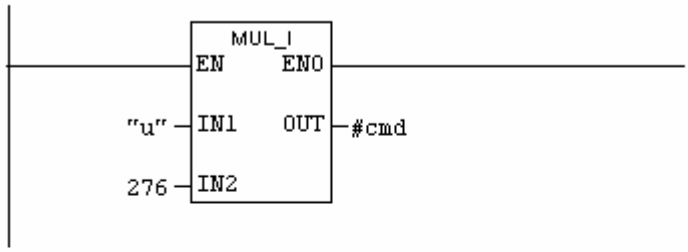
Réseau 6 : Titre :



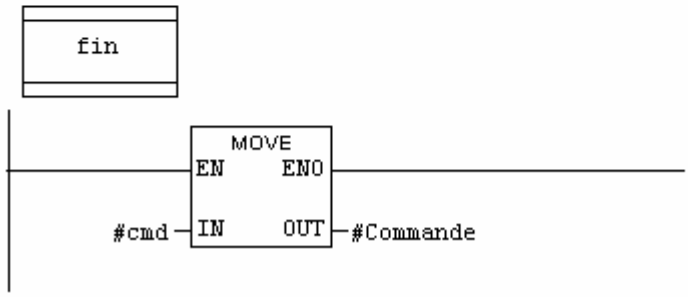
Réseau 7 : Titre :

```
    // actualisation de es
      L     "es"
      L     "u"
      +I
      L     "v"
      -I
```

Réseau 8 : Titre :



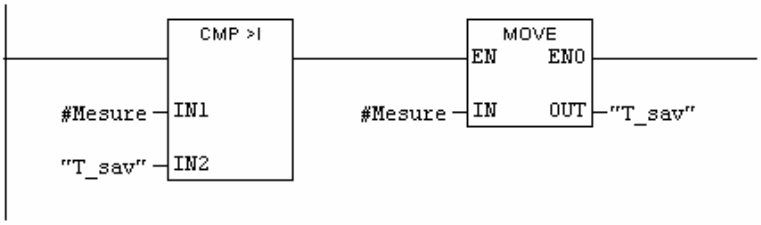
Réseau 9 : Titre :



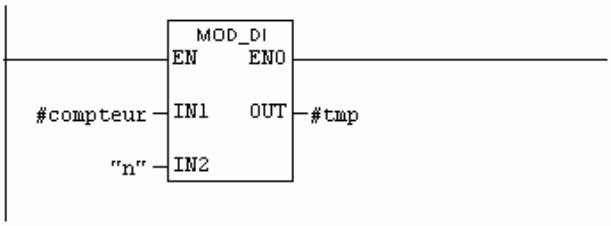
Programme dans FC11 Recherche du Max

FC11 : Titre :

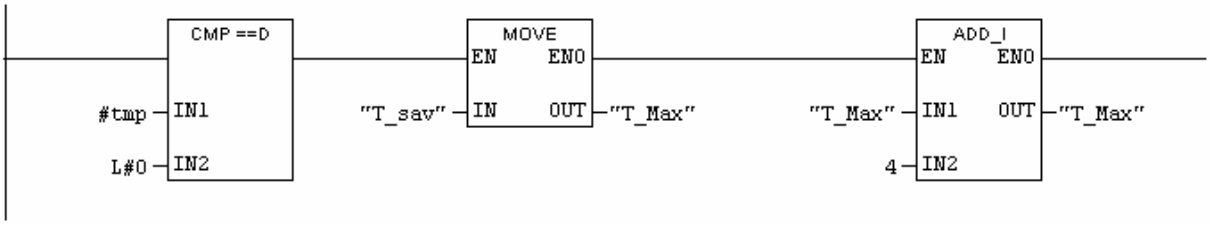
Réseau 1 : Titre :



Réseau 2 : Titre :



Réseau 3 : Titre :



3. Programme de la commande du moteur :

Programme dans l'OB100 :

```
OB100 :  "Complete Restart"
Réseau  1: Initialisation des zones mémoires de travail
          L     0
          T     "Vitess_mesurée"
          T     "L_Ecart"
          T     "Ecart_Prec"
          T     "u"
          T     "v"
          T     "es"

          L     L#0
          T     "Somme"

Réseau  2 : Titre :
          CALL  "Chargement Consigne"
           Reference:=700
           Consigne :="DAC"
```

Programme dans OB35 :

```
OB35 :  "Cyclic Interrupt"
Réseau  1: Titre :
          CALL  "Lecture de la Mesure"
           Port  :="ADC"
           Mesure:="Vitess_mesurée"

Réseau  2 : Titre :
          CALL  "Comparateur"
           Consigne:="Vitesse_Consigne"
           Mesure  :="Vitess_mesurée"
           Ecart   :="L_Ecart"

Réseau  3 : affichage
          L     "Vitess_mesurée"
          L     "Vitesse_Consigne"
          L     "L_Ecart"
```

Programme dans FC2 Lecture de la mesure :

```
FC2 : Titre :
Réseau  1: Titre :
          L     #Port
          ITB
          DTR
          L     1.084000e+001
          /R
          RND
          T     #Mesure
```

Programme dans FC3 Chargement consigne :

```
FC3 : Titre :
Réseau 1: Titre :
          L     #Reference
          DTR
          L     2.367000e+001
          *R
          RND
          T     #Consigne
```

Programme dans FC4 Le comparateur :

```
FC4 : Titre :
Réseau 1: Titre :
          L     #Consigne
          L     #Mesure
          -I
          T     #Ecart
```