République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

ECOLE NATIONALE POLYTECHNIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

DEPARTEMENT DU GENIE ELECTRIQUE
Option Automatique

Mémoire de projet de fin d'études
Pour l'obtention du diplôme d'Ingénieur d'Etat en Automatique

Thème

Mise en œuvre de l'automate S7-314 IFM et du logiciel STEP7

Proposé et dirigé par :
Mr E.M. BERKOUK

Etudié par :
M^lle^ BOULHARES Sara
Mr BOUROUBI Riadh

Promotion 2003/2004

E.N.P 10, Avenue Hassen Badi, B.P.182 EL HARRACH, ALGER

Je dédicace ce modeste travail tout d'abord à mes parents sans lesquels je ne serais pas ce que je suis.
A mon frère Moncef qui devrait se couper les cheveux. (Je te souhaite toute la réussite du monde).
A tata Samia et tonton Youcef.
A tonton Kader (Tu resteras dans mémoire de tous)
Ainsi qu'à toute ma famille.

Maintenant je m'occupe des autres :
Sara, je t'ai trop vu. Je rigole.
Samir, si t'as pas ton année je t'enlève ton PC.
Nabil, sept années de médecine, le pauvre...
Pechpoucha, elle se reconnaîtra (3 c'est déjà trop arrête).
Tchang, mieux vaut tard que jamais, je crois que tu l'as compris l'année dernière (dis le à Sam et Mus)
Ah , Mus, tu devrais avoir honte.
Malya, je te souhaite bien du courage pour la suite.
Nadjib et Nassima, pas de soucis.
A la R9 qui est « presque » la meilleure voiture du monde. A toi aussi Adel, ne sois pas jaloux.
Esma fais gaffe à ton bébé
Naïma take it easy et merci beaucoup.
Soumeya, t'es très courageuse, ça fait plaisir.

C'est pas que je ne me souviens pas des autres, mais la page est bientôt finie.

Merci à toutes les personnes qui m'ont aidé et mêmes celles qui ne m'ont pas aidé (peut être qu'elles le feront un jour)

Une dernière chose :
« L'homme ne vaut pas forcément la valeur que lui donne la société... »

BOUROUBI Riadh

Je dédicace ce travail :

A la mémoire de mon père, homme très bon et très courageux, il a toujours été mon idole et il le restera pour toujours.

A ma chère maman et à mes trois sœurs qui m'ont toujours aidée et soutenue. Je les remercie de leur patience.

A mon binôme qui m'a supporté durant quatre années .

A tous mes amis de l'école polytechnique et mes amis de Koléa, qui ont su être à mes cotés dans les moments difficiles, Grand Merci .

A mes copines de chambre, avec qui j'ai partagé des moments inoubliables de ma vie.

A Mounir, qui est toujours aussi adorable avec moi.

A toute ma famille , à mes voisines.

A tous mes enseignants durant mon cursus scolaire, du primaire à l'université.

A tous les gens que j'aime.

BOULHARES Sara

Remerciements

Nous remercions en premier lieu **Mr E.M. BERKOUK**, notre encadreur, de nous avoir aidé à réaliser le travail demandé dans ce projet de fin d'études.

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

Nous témoignons notre gratitude à **Mr M. MOKRANI**, ingénieur formateur à SIEMENS, pour la disponibilité et les conseils précieux qu'il nous a accordé.

Nos sincères remerciements également à tous nos professeurs de l'Ecole Nationale Polytechnique pour l'enseignement et les connaissances qu'ils nous ont apportés.

Merci à Toutes et à Tous.

BOULHARES Sara
BOUROUBI Riadh

Sommaire :

Section B : Description de la CPU S7-314 IFM

CHAPITRE II : Description de STEP

CHAPITRE III : Les fonctions intégrées de la CPU S7-314 IFM

CHAPITRE IV : Applications

Section A : Circuits de simulation.

Section B : Exemples de programmation.

Sommaire des tableaux :

Sommaire des figures :

Introduction

Dans le monde industriel, les exigences attendues de l'automatisation ont bien évolué. Avec la progression continuelle de la technologie, les critères demandés ne s'arrêtent pas uniquement à l'augmentation de la productivité, l'amélioration de la qualité du produit ou la diminution des coûts de production, mais concernent aussi l'amélioration des conditions de travail, l'accroissement de la sécurité et la suppression des tâches pénibles et répétitives.

L'automate programmable industriel apporte alors la solution sur mesure pour les besoins d'adaptation et de flexibilité de nombre d'activités économiques actuelles. Il est devenu aujourd'hui le constituant le plus répandu des installations automatisées. On le trouve non seulement dans tous les secteurs de l'industrie, mais aussi dans les services (gestion de parking, accès à des bâtiments, sécurité etc.) et dans l'agriculture (composition et délivrance de rations alimentaires dans les élevages).

L'API, machine programmable, placé dans un procédé industriel, fait partie intégrante de la boucle de réglage. Il a pour tâche principale de récolter des informations à partir des capteurs via ses interfaces d'entrées, de traiter ces informations pour prendre une décision et ainsi commander les actionneurs avec des signaux via ses interfaces de sorties.

L'API doit en outre communiquer avec son environnement (humains, autres API ou ordinateurs).

Le but de ce travail est la création et l'implémentation de programmes qui solutionnent des problèmes d'automatisation.

De nombreux constructeurs d'automates programmables existent, mais la firme allemande SIEMENS offre l'une des plus grandes gammes de produits. Et notre choix s'est porté sur la CPU S7-314 IFM car en plus des fonctions que possèdent les autres automates de la même série, celle-ci intègre un module fonctionnel.

Ce document englobe l'étude de l'automate SIMATIC S7-314 IFM, son logiciel de mise œuvre : STEP7, ainsi que l'implémentation de quelques tâches d'automatisation.

Le logiciel ne permettant pas la simulation de toutes les entrées de l'automate, la réalisation de circuits de simulation des entrées TOR et analogiques a été nécessaire.

Chapitre

1

Les automates programmables industriels.

Section

A

Généralités sur les automates programmables industriels

1. Historique:

Au début des années 50, les ingénieurs étaient déjà confrontés à des problèmes d'automatismes, les composants de base de l'époque étaient les relais électromagnétiques à un ou plusieurs contacts. Les circuits conçus comportaient des centaines voire des milliers de relais. Le transistor n'était connu que comme un composant d'avenir et les circuits intégrés étaient inconnus.

Vers 1960, les semi-conducteurs (transistors, diodes) sont apparus dans les automatismes sous forme de circuits digitaux. Ce n'est que quelques années plus tard, que l'apparition des circuits intégrés a amorcé une révolution dans la façon de concevoir les automatismes. Ceux-ci étaient très peu encombrants et leur consommation était des plus réduite. On pouvait alors concevoir des fonctions de plus en plus complexes à des coûts toujours décroissants.

C'est en 1969 que les constructeurs américains d'automobiles (General Motors en particulier) ont demandé aux firmes fournissant le matériel d'automatisme des systèmes plus évolués et plus souples pouvant être modifiés simplement sans coûts exorbitants.

Les ingénieurs américains ont résolu le problème en créant un nouveau type de produit nommé automates programmables. Ils n'étaient rentables que pour des installations d'une certaine complexité, mais la situation a très vite changée, ce qui a rendu les systèmes câblés obsolètes.

De nombreux modèles d'automates sont aujourd'hui disponible ; depuis les nano automates bien adaptés aux machines et aux installations simples avec un petit nombre d'entrées/sorties, jusqu'aux automates multifonctions capables de gérer plusieurs milliers d'entrées/sorties et destinés au pilotage de processus complexes.

2. Définition générale :

Un automate programmable industriel (API) est une machine électronique spécialisée dans la conduite et la surveillance en temps réel de processus industriels et tertiaires. Il exécute une suite d'instructions introduites dans ses mémoires sous forme de programmes, et s'apparente par conséquent aux machines de traitement de l'information.

Trois caractéristiques fondamentales le distinguent des outils informatiques tels que les ordinateurs utilisés dans les entreprises et le tertiaire:

- Il peut être directement connecté aux capteurs et pré-actionneurs grâce à ses entrées/sorties industrielles.
- Il est conçu pour fonctionner dans des ambiances industrielles sévères (Température, vibrations, micro-coupures de la tension d'alimentation, parasites, etc...).
- Enfin, sa programmation à partir de langages spécialement développés pour le traitement de fonctions d'automatisme facilite son exploitation et sa mise en œuvre.*

Selon la norme française EN 61131-1, un automate programmable est un:

« Système électronique fonctionnant de manière numérique, destiné à être utilisé dans un environnement industriel, qui utilise une mémoire programmable pour le stockage interne des instructions orientées utilisateur aux fins de mise en oeuvre des 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 Tout ou Rien ou analogiques divers types de machines ou de processus. L'automate programmable et ses périphériques associés sont conçus pour pouvoir facilement s'intégrer à un système d'automatisme industriel et être facilement utilisés dans toutes leurs fonctions prévues. »**

* Automates Nano et Plate-forme d'automatisme Micro [104] Schneider Electric 1999.
** M.Bertrand , Automates programmables industriels S 8 015, Techniques de l'ingénieur

3. Architecture des API :

3.1. Le processeur :

Le processeur a pour rôle principal le traitement des instructions qui constituent le programme de fonctionnement de l'application. Mais en dehors de cette tâche de base, il réalise également d'autres fonctions :

- Gestion des entrées/sorties.
- Surveillance et diagnostique de l'automate par une série de tests lancés à la mise sous tension ou cycliquement en cours de fonctionnement.
- Dialogue avec le terminal de programmation aussi bien pour l'écriture et la mise au point du programme qu'en cours d'exploitation pour des réglages ou des vérifications de données.*

Le processeur est organisé autour d'un certain nombre de registres, ce sont des mémoires rapides permettant la manipulation des informations qu'elles retiennent, ou leur combinaison avec des informations extérieures .

Les principaux registres existants dans un processeur sont :

3.1.1. L'accumulateur :

C'est le registre où s'effectuent les opérations du jeu d'instruction, les résultats sont contenus dans ce registre spécial.

3.1.2. Le registre d'instruction :

Il reçoit l'instruction à exécuter et décode le code opération. Cette instruction est désignée par le pointeur

3.1.3. Le registre d'adresse :

Ce registre reçoit, parallèlement au registre d'instruction, la partie opérande de l'instruction. Il désigne le chemin par lequel circulera l'information lorsque le registre d'instruction validera le sens et ordonnera le transfert.

3.1.4. Le registre d'état :

C'est un ensemble de positions binaires décrivant, à chaque instant, la situation dans laquelle se trouve précisément la machine.

* Automates Nano et Plate-forme d'automatisme Micro [104] Schneider Electric 1999.

3.1.5. Les piles :

Une organisation spéciale de registres constitue une pile, ces mémoires sont utilisées pour contenir le résultat de chaque instruction après son exécution. Ce résultat sera utilisé ensuite par d'autres instructions, et cela pour faire place à la nouvelle information dans l'accumulateur.

3.2. Les mémoires :

Un système à processeur est toujours accompagné d'un ou de plusieurs types de mémoires. Les automates programmables industriels possèdent pour la plupart les mémoires suivantes :

3.2.1. Mémoire de travail :

La mémoire de travail (mémoire vive) contient les parties du programme significatives pour son exécution. Le traitement du programme a lieu exclusivement dans la mémoire de travail et dans la mémoire système.

3.2.2. Mémoire système :

La mémoire système (mémoire vive) contient les éléments de mémoire que chaque CPU met à la disposition du programme utilisateur comme, par exemple, mémoire image des entrées, mémoire image des sorties, mémentos, temporisations et compteurs. La mémoire système contient, en outre, la pile des blocs et la pile des interruptions.

Elle fournit aussi la mémoire temporaire allouée au programme (pile des données locales).

3.2.3. Mémoire de chargement :

La mémoire de chargement sert à l'enregistrement du programme utilisateur sans affectation de mnémoniques ni de commentaires (ces derniers restent dans la mémoire de la console de programmation). La mémoire de chargement peut être soit une mémoire vive (RAM), soit une mémoire EPROM.

3.2.4. Mémoire RAM non volatile :

Zone de mémoire configurable pour sauvegarder des données en cas de défaut d'alimentation.

3.2.5. Mémoire ROM :

Contient le système d'exploitation qui gère la CPU.

3.3. Les modules d'entrée/sortie :

Ils traduisent les signaux industriels en informations API et réciproquement, appelés aussi coupleurs.

Beaucoup d'automates assurent cet interfaçage par des modules amovibles qui peuvent être modulaires par carte ou par rack. D'autres automates ont une structure monobloc, avec des modules intégrés dans un châssis de base, (cas des automates de Télémécanique TSX17 et SIMATIC S7-314 IFM).

Le nombre total de modules est évidement limité, pour des problèmes physiques:

- Alimentation électrique.
- Gestion informatique.
- Taille du châssis.

Différents types de modules sont disponibles sur le marché selon l'utilisation souhaitée, les plus répandus sont:

3.3.1. Entrées/Sorties TOR (Tout ou Rien):

La gestion de ce type de variables constituant le point de départ historique des API reste une de leurs activités majeures.

Leur nombre est en général de 8,16, 24 ou 32 entrées/sorties, qui peuvent fonctionner :

- en continu: 24V, 48V.
- en alternatif: 24V, 48V, 100/120V, 220/240V.

3.3.2. Entrées/Sorties analogiques :

Elles permettent l'acquisition de mesures (entrées analogiques), et la commande (sorties analogiques). Ces modules comportent un ou plusieurs convertisseurs Analogique/Numérique (A/N) pour les entrées et Numérique/Analogique (N/A) pour les sorties dont la résolution est de 8 à 16 bits.

Les standards les plus utilisés sont: ±10V, 0-10V, ±20mA, 0-20mA et 4-20mA.

Ces modules sont en général multiplexés en entrée pour n'utiliser qu'un seul convertisseur A/N, alors que les sorties exigent un convertisseur N/A par voie pour pouvoir garder la commande durant le cycle de l'API.

3.3.3. Modules spécialisés:

Ils assurent non seulement une liaison avec le monde extérieur, mais aussi une partie du traitement pour soulager le processeur et donc améliorer les performances. Ces modules peuvent posséder un processeur embarqué ou une électronique spécialisée.

On peut citer:

3.3.3.1. Les cartes de comptage rapide:

Elles permettent de saisir des événements plus courts que la durée du cycle, travaillant à des fréquences qui peuvent dépasser 10kHz.

3.3.3.2. Les Entrées/Sorties déportées:

Leur intérêt est de diminuer le câblage en réalisant la liaison avec détecteurs, capteurs ou actionneurs au plus près de ceux-ci, ce qui a pour effet d'améliorer la précision de mesure.

La liaison entre le boîtier déporté et l'unité centrale s'effectue par le biais d'un réseau de terrain suivant des protocoles bien définis. L'utilisation de la fibre optique permet de porter la distance à plusieurs kilomètres.

3.4. L'alimentation électrique :

Elle a pour rôle de fournir les tensions continues nécessaires aux composants avec de bonnes performances, notamment face aux micro-coupures du réseau électrique qui constitue la source d'énergie principale.

La tension d'alimentation peut être de 5V, 12V ou 24V.

D'autres alimentations peuvent être nécessaires pour les châssis d'extension et pour les modules entrées/sorties.

Un onduleur est nécessaire pour éviter les risques de coupures non tolérées.

3.5. Les liaisons :

Elles s'effectuent :

- Avec l'extérieur par des **borniers** (à visser, à clipser,...), sur lesquels arrivent des câbles transportant les signaux électriques.
- Avec l'intérieur par des **bus**, liaisons parallèles entre les divers éléments. Il existe plusieurs types de bus, car on doit transmettre des données, des états, des adresses.

3.6. Eléments auxiliaires :

- Un ventilateur est indispensable dans les châssis comportant de nombreux modules ou dans le cas où la température ambiante est susceptible de devenir assez élevée.
- Un support mécanique : il peut s'agir d'un rack, l'automate se présente alors sous forme d'un ensemble de cartes, d'une armoire, d'une grille, et des fixations correspondantes.
- Des indicateurs d'état : concernant la présence de tension, la charge de la batterie, le bon fonctionnement de l'automate etc....

4. Protection de l'automate :

La protection des circuits d'entrée contre les parasites électriques est souvent résolue par découplage optoélectronique. Le passage des signaux par un stade de faisceau lumineux assure en effet une séparation entre les circuits internes et externes.

Du coté sorties, on doit assurer le même type de protection, mais aussi une amplification de puissance, avec au final un courant continu ou alternatif selon les cas.

Deux types de cartes électroniques sont utilisés:

4.1. Les modules à sorties statiques:

Relais statiques intégrant des composants spécialisés: transistors bipolaires, thyristors. Ces composants n'ont aucune usure mécanique et leurs caractéristiques de commutation se maintiennent dans le temps.

4.2. Les modules à relais électromagnétiques:

Où le découplage résulte de l'existence de deux circuits électriques (bobines d'excitation, circuits de puissance), Ces relais électromagnétiques ont l'avantage d'avoir une faible résistance de contact, une faible capacité de sortie et surtout un faible coût, mais ont une durée de vie et une vitesse de commutation inférieures aux sorties statiques.

5. Environnement :

Dans le cadre d'une évolution conduisant à une automatisation de plus en plus globale, l'automate est de moins en moins acheté « nu ». Et même si c'est le cas, il doit pouvoir se connecter à d'autres matériels à processeur, et dialoguer avec les agents d'exploitation.

Les types de communication supportés par les API modernes sont :

- La communication avec un opérateur par un pupitre ou un terminal industriel : Ils permettent une communication homme-machine, et ce dans les deux sens (clavier

alphanumérique, écran à affichage graphique). Ils offrent des protections telles des claviers étanches pour une utilisation en ambiance industrielle.

- Les échanges d'informations avec une supervision dont le rôle dépasse largement la communication entre l'API et l'opérateur. Les postes de supervision constituent un outil de communication à distance pour recevoir des informations de l'automate, lui donner des ordres, voire changer certains de ses paramètres.
- Les échanges d'informations avec des capteurs et actionneurs intelligents.
- Les échanges d'informations avec un processeur maître, ou, au contraire, avec des esclaves, dans le cadre d'un réseau.

Section

B

Description de l'automate

S7-314IFM

1. Présentation de la CPU :

L'automate programmable utilisé dans ce projet est un S7-314 IFM. Sa caractéristique principale est l'intégration de modules comportant entre autres des fonctions intégrées.

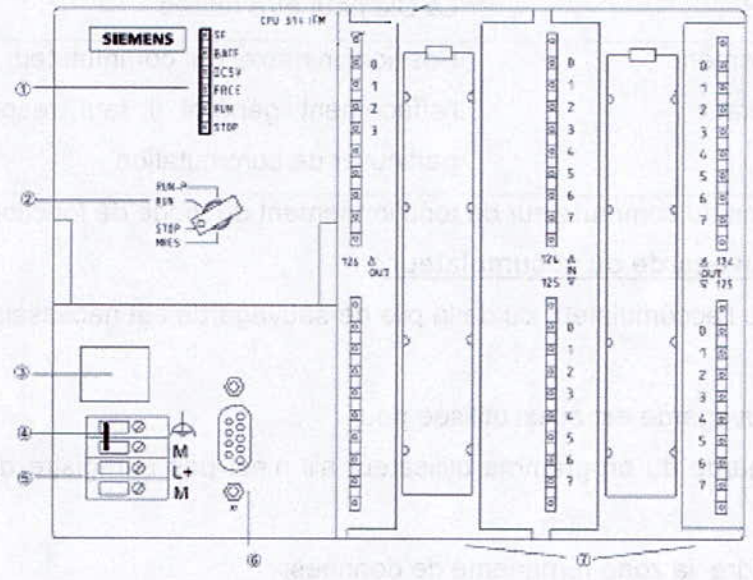


Figure1.1 Vue générale de la CPU S7-314 IFM.

Les CPU de la série S7-300 sont composées des éléments suivants :

1.1. LED de visualisation d'état et de défaut :

(rouge) SF	Défaut de matériel ou de logiciel.
(rouge) BATF	Défaillance de la pile.
(verte) 5V cc	L'alimentation 5V cc est correcte .
(jaune) FRCE	Le forçage permanent est actif.
(verte) RUN	CPU en RUN.
(jaune) STOP	CPU en STOP ou ATTENTE ou en démarrage.

Figure1.2 LED de visualisation d'état de la CPU

1.2. Commutateur de mode de fonctionnement :

Le changement de mode se fait à l'aide d'une clé :

Position	Signification	Explication
RUN –P	Mode de fonctionnement RUN-PROGRAM	La CPU traite le programme utilisateur. Le programme peut être modifié. Dans cette position la Clé ne peut pas être retirée.
RUN	Mode de fonctionnement RUN	La CPU traite le programme utilisateur. Le programme ne peut être modifié qu'avec légitimation par mot de passe

		La Clé peut être retirée.
Position	**Signification**	**Explication**
STOP	Mode de fonctionnement STOP	La CPU ne traite aucun programme utilisateur. La Clé peut être retirée.
MRES	Effacement Général	Position instable du commutateur, pour effectuer l'effacement général il faut respecter un ordre particulier de commutation

Tableau1.1 Positions du commutateur de fonctionnement du mode de fonctionnement.

1.3. pile de sauvegarde ou accumulateur :

L'utilisation de l'accumulateur ou de la pile de sauvegarde est nécessaire pour l'horloge temps réel.

La pile de sauvegarde est aussi utilisée pour :

- la sauvegarde du programme utilisateur s'il n'est pas enregistré dans la mémoire morte.
- pour étendre la zone rémanente de données.

L'accumulateur est rechargé à chaque mise sous tension de la CPU. Son autonomie est de quelques jours voir quelques semaines au maximum. La pile de sauvegarde n'est pas rechargeable mais son autonomie peut aller jusqu'à une année.

1.4. Carte mémoire :

La CPU S7-314 IFM utilisée n'est pas dotée d'une carte mémoire mais il faut savoir que la plupart des CPU en possèdent, son rôle est de sauvegarder le programme utilisateur, le système d'exploitation et les paramètres qui déterminent le comportement de la CPU et des modules en cas de coupure de courant.

1.5. Interface MPI : (interface multipoint)

L'interface MPI est l'interface de la CPU utilisée pour la console de programmation (PG), le pupitre opérateur (OP) ou pour la communication au sein d'un sous réseau MPI. La vitesse de transmission typique est de 187,5 k bauds.

2. Caractéristiques techniques de la CPU :

La CPU S7-314 IFM possède des entrés/sorties intégrées, leur câblage se fait par le biais de connecteurs frontaux 20 ou 40 points.

2.1. Tableaux techniques :

Les tableaux suivants résument les principales caractéristiques de la CPU S7-314 IFM

Mémoires	
Mémoire de travail intégrée uniquement	32ko
Mémoire de chargement intégrée	48Ko de RAM 48Ko de FEPROM
Impossibilité d'extension pour la mémoire de travail ainsi que la mémoire de chargement	
Mémentos	
Nombre	2048 bits
Rémanence : réglable par défaut	de MB 0 à MB 143 de MB 0 à MB 15
Mémentos de cadence	Un octet de mémento
Blocs de données	
Nombre	Maximum 127 (DB 0 réservé)
Taille	Maximum 8 Ko
Rémanence : réglable par défaut	Maximum 2 DB, 144 octets de données. Pas de rémanence
Blocs	
Blocs d'organisation (OB)	13
Taille	Maximum 8 Ko
Profondeur d'imbrication : Par classe de priorité Supplémentaire à l'intérieur d'un OB d'erreur	 8 4
Blocs fonctionnels (FB)	128
Taille	Maximum 8 Ko
Fonctions (FC)	128
Taille	Maximum 8Ko
Temporisations /compteurs	
Compteurs S7 Rémanence par défaut Rémanence réglable Plage de comptage	64 Z0 àZ7 Z0 à Z63 0 à 999
Temporisations S7 Rémanence par défaut Rémanence réglable Plage de temps	128 Aucune temporisation rémanente T0 à T7 10 ms à 9990 s

Zones d'adresses (entrées/sorties)	
Numériques	0.0 à 123.7/0.0 à 123.7
Numériques intégrées	124.0 à 125.7/124.0 à 125.7
Spéciales	126.0 à 126.3/124.0 et 124.1
Analogiques	256 à 751/256 à 751
Analogiques intégrées	128 à 135/128 à 129
Mémoire image (non réglable)	128 octets/128 octets
Sauvegarde	
Avec pile	Toutes les données
Sans pile	144 octets

Tableau1.2 Zones de mémoire et de périphérie de la CPU*

Fonction de test et de diagnostic	
Etat/forçage de variables	Oui
Variable	Entrées, sorties, DB, temporisations, compteurs, mémentos.
Nombre	
Etat de variables	Maximum 30
Forçage de variables	Maximum 14
Forçage permanent	Oui
Variables	Entrées, sorties
Nombre	Max 10
Nombre de points d'arrêt	2
Tampon de diagnostic	Oui
Nombre d'entrées (non réglables)	100

Tableau1.3 Fonction de test et de diagnostic*

Interface de communication MPI	
Vitesse de transmission	19.2 , 187.5 k bauds

Tableau1.4 Interface de communication MPI*

Tensions, Courants	
Tension d'alimentation	24V cc
Plage admissible	20,4 à 28,8 V
Consommation (en marche à vide)	Typique 1.0 A

Tableau1.5 Tensions et courants*

*Automates programmables S7-300 caractéristiques techniques des CPU [62] SIMATIC 2001

Fonctions intégrées	
Compteur	1 ou 2 selon la configuration utilisateur
Fréquencemètre	Maximum 10 KHz
Positionnement	1 voie

Tableau1.6 Fonctions intégrées de la CPU*

3. Les registres de la CPU :

3.1. Le mot d'état :

C'est un registre composé de 9 bits qui nous renseignent sur l'état de la CPU à chaque instant :

15	14	.....					8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI

3.1.1. Première interrogation /PI :

Le fonctionnement de ce bit est le suivant :

- L'état de /PI est interrogé au même moment que l'état de l'opérande en cours,
- Si /PI est à 0, la CPU exécute la séquence comme étant une nouvelle, et met le bit /PI à 1, seul le résultat de l'interrogation de l'opérande est mémorisé dans le RLG,
- Tant que /PI est à 1, le résultat de l'interrogation de l'opérande en cours est comparé, selon l'opération combinatoire effectuée, à celui mémorisé précédemment dans le RLG ,
- La fin d'une séquence ou une instruction de saut conditionnel remettent le bit /PI à 0.

3.1.2. Le bit du résultat logique RLG :

Il contient le résultat d'une opération combinatoire sur bits, ou le résultat d'une comparaison.

Dans une séquence combinatoire, le résultat d'une interrogation est toujours combiné avec le RLG, suivant la règle booléenne établie, à condition que /PI soit à 1. Si ce dernier est à 0, c'est le contenu de l'opérande en cours qui lui est affecté.

3.1.3. Le bit d'état ETAT :

Contient la valeur du bit en accès, il est utilisé uniquement pour les opérations combinatoires ayant accès à la mémoire. Pour les opérations n'ayant pas accès aux mémoires, ce bit est à 1 et n'a pas de signification.

3.1.4. Le bit OU :

Ce bit est utilisé lors de l'utilisation de l'opération ET avant OU,

Le RLG d'une séquence interne est transféré vers ce bit, pour pouvoir enregistrer le nouveau résultat dans le bit RLG.

* Automates programmables S7-300 caractéristiques techniques des CPU [63] SIMATIC 2001

3.1.5. Le bit de débordement DEB :

Il est mis à 1 par une opération arithmétique, une opération de conversion ou une opération de comparaison de nombres à virgule flottante lorsqu'il y a débordement.

3.1.6. Le bit de débordement mémorisé DM :

Il est mis à 1 au même moment que DEB, et il le reste après la correction de l'erreur, il indique donc si une erreur s'est produite dans l'une des opérations exécutées précédemment. L'opération SPS le remet à 0.

3.1.7. Les bits indicateurs BI1 et BI0 :

Ils donnent des informations sur les résultats des opérations suivantes, avec ou sans débordement :

- Le résultat d'une opération arithmétique.
- Le résultat d'une opération de comparaison.
- Le résultat d'une opération combinatoire sur mots.
- Les bits décalés par une opération de rotation ou de décalage.

3.1.8. Le bit du résultat binaire RB :

Il constitue un lien entre le traitement d'opérations combinatoires sur bits et sur mots. En effet, il permet d'utiliser le résultat d'une opération sur mots, comme étant un résultat binaire, et l'intégrer à une séquence combinatoire sur bits.

Il correspond aussi à la sortie de validation ENO les FB et les FC.

3.2. Accumulateur1 et Accumulateur 2 :

Registres sur 32 bits, qui permettent de traiter des octets, des mots ou des doubles mots. Ils sont utilisés pour le chargement des opérandes. Le résultat d'une opération se trouve toujours dans l'accumulateur 1.

3.3. Registres d'adresses : AR1 et AR2 :

Deux registres sur 32 bits, renfermant les adresses des opérandes en cours d'utilisation.

3.4. Pile des parenthèses :

Octet de mémoire utilisé pour les combinaisons d'expressions entre parenthèses, on peut avoir jusqu'à 7 niveaux de parenthèses, appelés « entrées », chaque entrée englobe les bits du mot d'état suivants ;RLG, RB, OU.

L'opération fermer parenthèse ")" ferme l'expression entre parenthèses et extrait une entrée de la pile, puis définit le nouveau RLG qui est le résultat de la combinaison du RLG en cours avec celui mis dans la pile des parenthèses.

Chapitre

2

Description de STEP 7

1. Qu'est-ce que STEP 7 ?

STEP7 fait partie de l'industrie logicielle SIMATIC. Il représente le logiciel de base pour la configuration et la programmation de systèmes d'automatisation.

Les tâches de bases qu'il offre à son utilisateur lors de la création d'une solution d'automatisation sont :

- La création et gestion de projets.
- La configuration et le paramétrage du matériel et de la communication.
- La gestion des mnémoniques.
- La création des programmes.
- Le chargement de programmes dans les systèmes cibles.
- Le test de l'installation d'automatisation.
- Le diagnostic lors des perturbations dans l'installation.

Il s'exécute sous les systèmes d'exploitation de MICROSOFT à partir de la version Windows 95. Et s'adapte par conséquent à l'organisation graphique orientée objet qu'offre ces systèmes d'exploitation.

2. Applications de STEP7 :

Il est constitué de deux types d'application ; applications de base et applications en options.

2.1. Applications du logiciel de base STEP7 :

Le logiciel STEP7 met à disposition les applications de base suivantes :

- Le gestionnaire de projets.
- La configuration du matériel.
- L'éditeur de mnémoniques.
- L'éditeur de programmes CONT, LOG, LIST.
- La configuration de la communication NETPRO.
- Le diagnostic du matériel.

2.1.1. Gestionnaire de projets SIMATIC Manager :

Le gestionnaire de projets SIMATIC Manager gère toutes les données relatives à un projet d'automatisation, il démarre automatiquement les applications requises pour le traitement des données sélectionnées.

Le schéma suivant représente la fenêtre qui apparaît à l'ouverture du SIMATIC

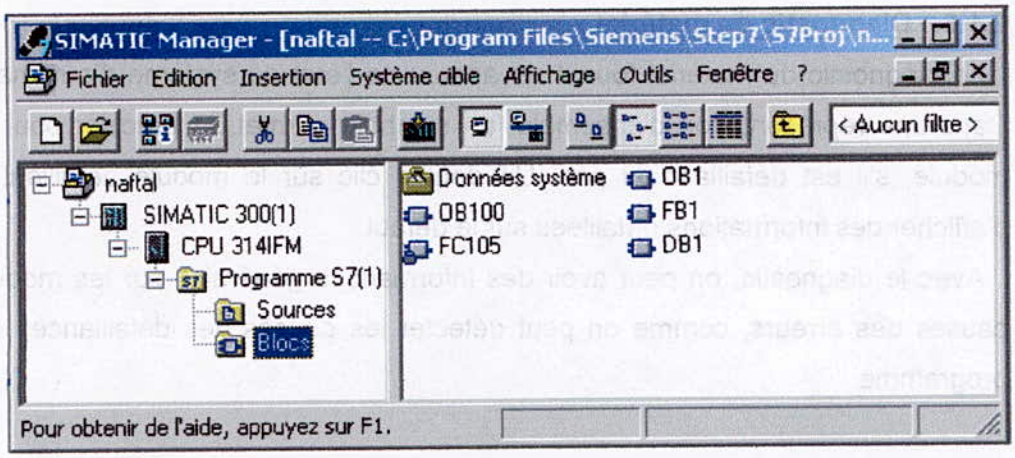


Figure 2.1 Fenêtre du SIMATIC manager.

2.1.2. Configuration du matériel HW Config :

HW Config est utilisé pour configurer et paramétrer le support matériel dans un projet d'automatisation.

2.1.3. Editeur de mnémoniques :

Il permet la gestion de toutes les variables globales. En effet, il définit des désignations symboliques et des commentaires pour les signaux du processus (Entrées/Sorties), les mémentos, les blocs de données, les temporisations et les compteurs.

La table des mnémoniques qui en résulte est mise à disposition de toutes les applications. La modification de l'un des paramètres d'un mnémonique est de ce fait reconnue automatiquement par toutes les applications.

2.1.4. Editeur de programme :

Les langages de base proposés sont :

- Le schéma à contact (CONT), langage graphique similaire aux schémas de circuits à relais, il permet de suivre facilement le trajet du courant.
- La liste d'instruction (LIST), langage textuel de bas niveau, à une instruction par ligne, similaire au langage assembleur.
- Le logigramme (LOG), langage de programmation graphique qui utilise les boîtes de l'algèbre de BOOLE pour représenter les opérations logiques.

L'éditeur de programmes permet aussi la visualisation et forçage de variables,

2.1.5. Configuration de la communication Net Pro :

La configuration et le paramétrage de réseaux se font à l'aide de l'application Net Pro. Elle permet de :

- Créer une vue graphique du réseau en question ainsi que les sous-réseaux qui le constituent.
- Déterminer les propriétés et les paramètres de chaque sous-réseau.

2.1.6. Diagnostic du matériel :

Le diagnostic du matériel 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. Un double clic sur le module défaillant permet d'afficher des informations détaillées sur le défaut.

Avec le diagnostic, on peut avoir des informations générales sur les modules, les causes des erreurs, comme on peut détecter les causes des défaillances dans un programme.

2.2. Possibilité d'extension du logiciel de base STEP7 :

Cette extension est réalisée à l'aide de logiciels optionnels, regroupés dans trois catégories :

2.2.1. Applications techniques :

Ensemble de langages de programmation évolués pour programmeur, des langages graphiques pour l'ingénieur en technologie et enfin des logiciels complémentaires pour le diagnostic, la simulation, la maintenance à distance, et la documentation de l'installation.

2.2.2. Logiciels exécutables :

Solutions finies programmées pouvant être appelées dans le programme utilisateur, et d'être directement intégrées dans la solution d'automatisation. Exemple :

- Le contrôle PID qui permet l'intégration de régulateurs à action continue et de régulateurs à impulsion dans le programme utilisateur. L'application de paramétrage à laquelle la définition du régulateur est intégrée, permet le paramétrage rapide et le réglage optimal du régulateur.
- Le contrôle avec logique floue. Utilisé lorsque des processus ne peuvent pas ou peuvent difficilement être décrits mathématiquement, lorsque le déroulement de mécanismes et de processus est imprévisible, lorsque des comportements non linéaires surviennent alors que l'on dispose d'une connaissance acquise par expérience du processus.

2.2.3. Interfaces homme/machine :

Ce sont des logiciels spécifiques au contrôle-commande dans SIMATIC.

On peut citer :

- Le système de visualisation du processus SIMATIC WinCC qui est un système de base indépendant des branches et technologies d'utilisation et qui comporte toutes les fonctions indispensables au contrôle-commande.

- ProAgent qui permet un diagnostic précis et rapide du processus dans les installations et les machines en fournissant des informations relatives à la localisation et à la cause des erreurs.

3. Premier pas vers STEP7 :

Pour concevoir un projet avec STEP7, il existe 2 approches :

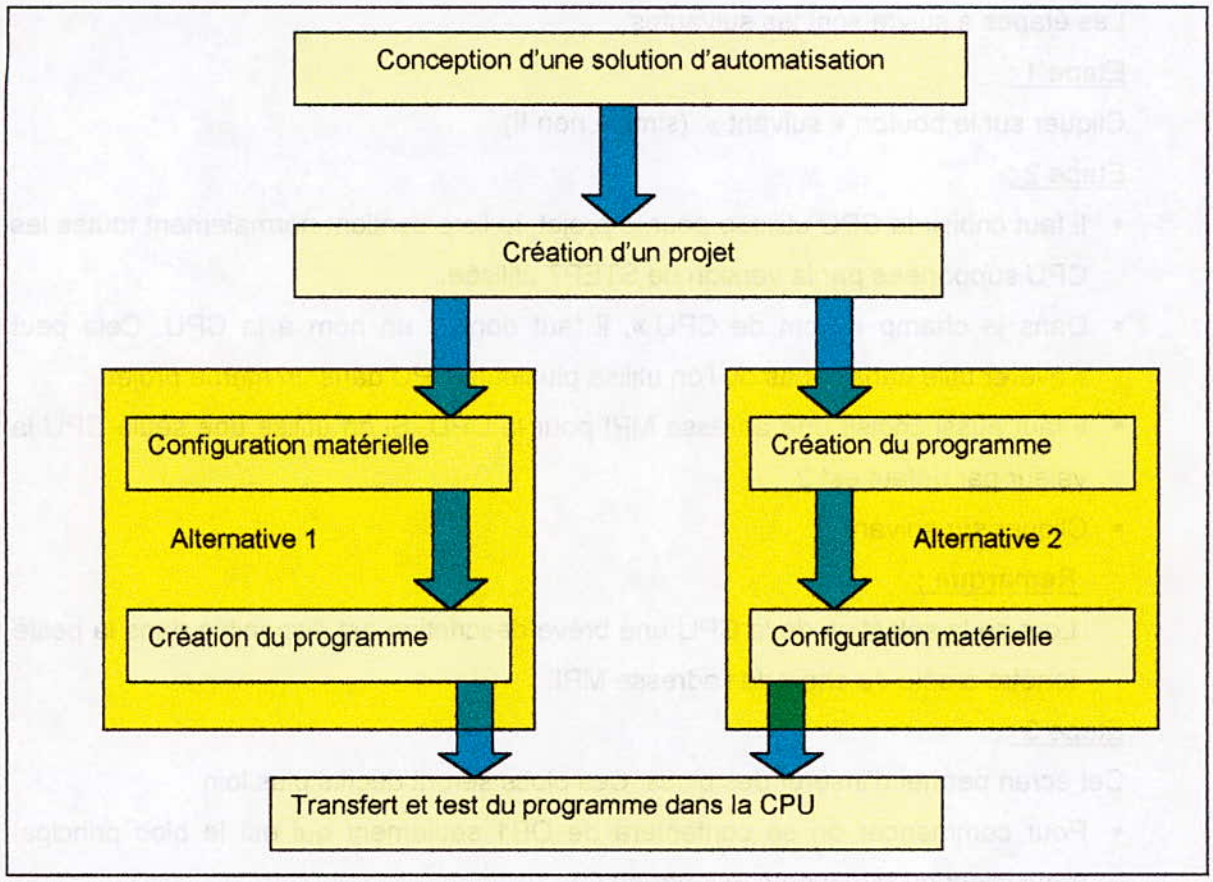


Figure 2.2 organigramme pour la création de projets sous STEP 7.*

Il faut noter que pour un système contenant beaucoup de variables, la seconde alternative n'est pas très pratique.

Nous optons donc pour la première approche.

3.1. Création du projet avec STEP7 :

Pour créer un projet avec STEP 7, on peut lancer l'assistant de création de projet de STEP7, ou créer directement un projet que l'on configurera soi même.

* Mise en route de STEP 7 5.2 6ES7810-4CA06-8CA0 [10] SIMATIC 2002

3.1.1. Utilisation de l'assistant de création de projets :

Par défaut l'assistant de création de projets apparaît à chaque démarrage du SIMATIC Manager, si ce n'est pas le cas, son lancement se fait en passant par le menu Fichier>Assistant 'nouveau projet'.

Cet assistant permet de créer un projet avec une interface simple.

Les étapes à suivre sont les suivantes :

Etape 1 :

Cliquer sur le bouton « suivant » (simple non !!)

Etape 2 :

- Il faut choisir la CPU utilisée pour le projet, la liste contient normalement toutes les CPU supportées par la version de STEP7 utilisée.
- Dans le champ « Nom de CPU », il faut donner un nom à la CPU. Cela peut s'avérer utile dans le cas où l'on utilise plusieurs CPU dans un même projet.
- Il faut aussi choisir une adresse MPI pour la CPU. Si on utilise une seule CPU la valeur par défaut est 2.
- Cliquer sur suivant.

Remarque :

Lors de la sélection de la CPU une brève description est disponible dans la petite fenêtre à côté du choix de l'adresse MPI.

Etape 3 :

Cet écran permet d'insérer des blocs. Ces blocs seront décrits plus loin.

- Pour commencer on se contentera de OB1 seulement qui est le bloc principal (équivalent du MAIN pour le langage C).
- On doit aussi choisir un langage de programmation parmi les trois proposés (LIST, CONT ou LOG).
- Cliquer sur suivant.

Etape 4 :

Nommer le projet et cliquer sur Créer.

Le projet est maintenant créé, on peut voir une arborescence à gauche de la fenêtre qui s'est ouverte.

3.1.2. Création d'un nouveau projet sans l'assistant de création de projets :

Cette méthode est un peu plus compliquée, mais permet de mieux gérer le projet.

Dans la fenêtre SIMATIC Manager, cliquer sur *Fichier>Nouveau* (ou encore CTRL+N), une fenêtre demandant un nom de projet s'ouvre. Il faut donc donner un nom au projet puis valider par OK.

La fenêtre du projet s'ouvre.

Le projet est vide, il faut lui insérer une station SIMATIC, cela est possible en cliquant sur le projet avec le bouton droit puis *Insérer un nouvel objet>station SIMATIC 300.*

La station SIMATIC n'est toujours pas configurée, il faut passer à l'étape de configuration matérielle.

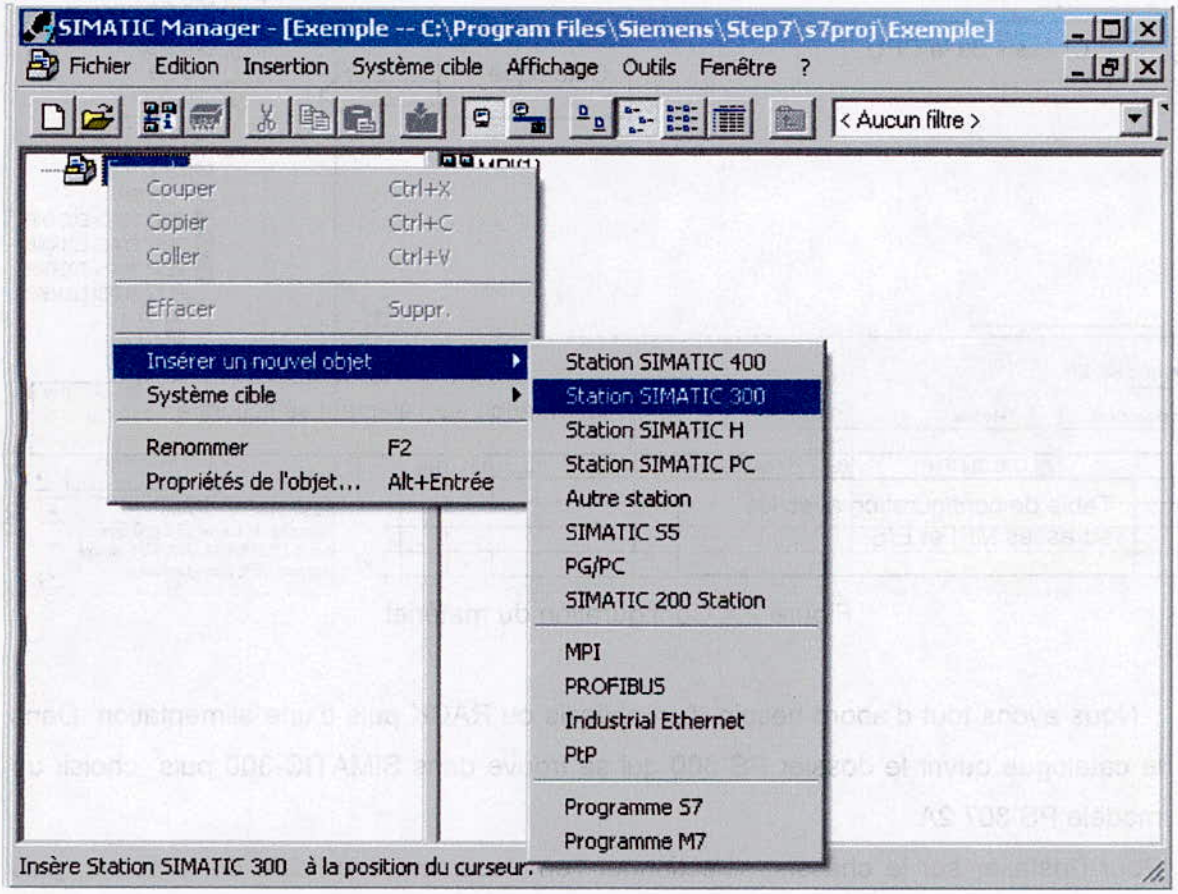


Figure 2.3 Création d'un projet sans l'assistant.

3.2. Configuration matérielle :

La configuration matérielle est une étape très importante, elle permet de reproduire à l'identique le système utilisé (alimentation, CPU, modules etc..).

Pour effectuer cette configuration, il faut aller sur l'icône Station SIMATIC 300.

Sur la fenêtre de droite s'affichent deux icônes : « Matériel » et le nom de la CPU.

Il faut ouvrir l'icône matériel : La fenêtre HW Config s'ouvre.

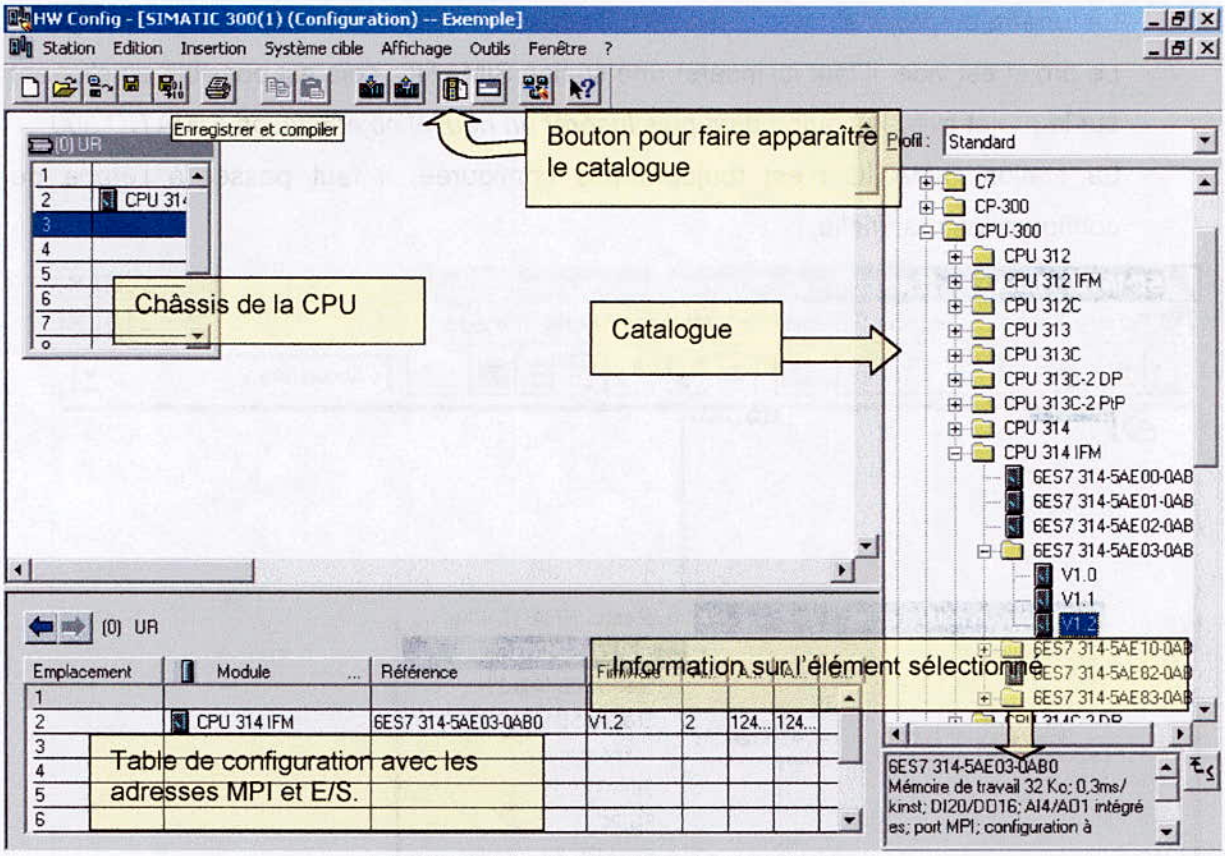


Figure 2.4 Configuration du matériel.

Nous avons tout d'abord besoin d'un châssis ou RACK puis d'une alimentation. Dans le catalogue ouvrir le dossier PS-300 qui se trouve dans SIMATIC-300 puis choisir un modèle PS 307 2A.

Pour l'installer sur le châssis, sélectionner l'emplacement 1 à l'aide de la souris, puis double-cliquer sur l'alimentation.

On peut de la même manière insérer des composants sur le châssis en fonction de la configuration réelle. Dans notre cas, l'insertion de la CPU S7 314 IFM de référence 6ES7314-5EA03-0AB0 se fera dans l'emplacement 2.

Le paramétrage de la CPU, à l'aide de menu, permet de définir des caractéristiques, telles que : le comportement à la mise en route, la surveillance du temps de cycle ainsi que l'activation et la désactivation des plages de rémanences et les fonctions intégrées, ces données sont enregistrées dans les blocs de données systèmes.

Le paramétrage des modules est réalisé automatiquement au démarrage de la CPU. Ainsi, le remplacement d'un module est ainsi possible sans nouveau paramétrage.

A la fin de la configuration, il suffit de cliquer sur *Station>Enregistrer et compiler* pour valider les changements apportés au châssis. De cette manière les changements seront pris en compte dans le reste du projet.

3.3. Définition des mnémoniques :

Il faut maintenant définir les variables qui vont être utilisées lors des étapes de programmation. L'utilisation de noms communs est plus aisée que la manipulation de chiffres (ex : utiliser « moteur » au lieu du bit de sortie A0.0).

Pour accéder à la table des mnémoniques : cliquez sur le dossier *programme* dans la fenêtre du projet, puis sur l'icône mnémoniques.

L'utilisation de cette table consiste à :

- Donner un nom à la mnémonique dans la première colonne.
- Donner la variable associée à cette mnémonique dans la seconde colonne.
- Le type de la donnée est automatiquement généré par STEP 7.
- Ecrire éventuellement un commentaire dans la colonne prévue à cet effet.

Après avoir défini toutes les mnémoniques, il suffit d'enregistrer pour que les changements soient pris en compte dans le reste du projet.

Editeur de mnémoniques - [Programme S7(1) (Mnémoniques) -- chariot_compteur\SIMATIC 300(1)...

Table Edition Insertion Affichage Outils Fenêtre ?

Tous les mnémoniques

	Etat	Mnémonique	Opérande	Type de don	Commentaire
1		A	E 124.1	BOOL	Détecteur A
2		B	E 124.2	BOOL	Détecteur B
3		C	E 124.3	BOOL	Détecteur C
4		COMPLETE RESTART	OB 100	OB 100	Complete Restart
5		D	E 124.4	BOOL	Détecteur D
6		FB1OK	A 125.5	BOOL	Sortie signalant le bon fonctionnement du bloc de fonct...
7		M	E 124.0	BOOL	Bouton de mise en marche du système
8		M1	A 125.0	BOOL	Sortie de commande du moteur M1
9		M2	A 125.1	BOOL	Sortie de commande du moteur M2
10		SENS	A 125.4	BOOL	Sortie commandant le sens de rotation des moteurs
11		TEMPO1	A 125.2	BOOL	Sortie indiquant l'état de la première temporisation
12		TEMPO2	A 125.3	BOOL	Sortie indiquant l'état de la seconde temporisation
13					

Pour obtenir de l'aide, appuyez sur F1. NUM

Figure 2.5 Vue sur la fenêtre d'édition de mnémoniques.

Si on a besoin d'insérer de nouveaux objets dans le projet (ex : d'autre blocs de programme) il suffit de cliquer avec le bouton droit de la souris sur le dossier ou l'on veut

- Standardiser certaines parties du programme.
- Simplifier l'organisation du programme.
- Modifier facilement le programme.
- Simplifier le test du programme, car on peut l'exécuter section par section.
- Faciliter la mise en service.

4.1. hiérarchisation dans un projet :

Chaque programme utilisateur est structuré sous forme de projet. Un projet représente l'ensemble des données et programmes d'une solution d'automatisation et se trouve à la tête d'une hiérarchie d'objets qui sont :

- objet projet.
- objet Station.
- objet Modules programmables.
- objet Programme S7/M7.
- objet dossier Sources.
- objet dossier Blocs.

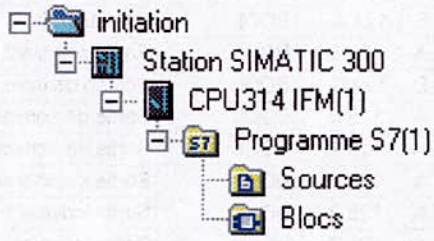


Figure 2.6 Hiérarchisation d'un projet STEP 7.

4.1.1. Objet station :

Une station SIMATIC 300/400 représente une configuration matérielle S7. Elle comporte un ou plusieurs modules programmables.

4.1.2. Objet modules programmables :

Représente les données de paramétrage d'un module comme les CPU et les modules fonctionnels.

4.1.3. Objet programme S7/M7 :

Un programme (S7/M7) est un dossier contenant les logiciels pour les modules programmables. Dans un programme S7/M7 figurent déjà :

Une table des mnémoniques, un dossier "Blocs" et un dossier "Sources".

4.1.4. Objet dossier sources :

Il contient les programmes source sous forme de texte. On peut saisir une partie ou tout le programme sous forme de source que l'on compilera ensuite en blocs.

Les avantages de la création de programmes sous forme de source sont :

- La possibilité de programmer plusieurs blocs dans une même source.
- L'avantage d'enregistrer la source malgré la présence éventuelle d'erreurs de syntaxe, ce qui n'est pas possible lors de la création de blocs de code avec vérification de syntaxe. Cela signifie que les erreurs de syntaxe ne seront signalées que lors de la compilation de la source.

Elle peut être créée et traitée avec le choix de l'éditeur ASCII, puis importée et compilée en blocs individuels. La compilation entraîne la génération des différents blocs et leur sauvegarde dans le programme utilisateur S7.

La source doit être écrite en utilisant la syntaxe du langage de programmation "liste d'instructions (LIST)", et nécessite par conséquent l'application de règles précises.

4.1.5. Objet dossier Blocs :

Le dossier Blocs contient les blocs que l'on doit charger dans la CPU pour réaliser la tâche d'automatisation.

Il englobe les blocs de code (OB, FB, SFB, FC, SFC) qui contiennent les programmes qu'on doit charger dans la CPU, et les blocs de données (DB d'instance et DB globaux) qui contiennent les paramètres du programme.

4.1.5.1. Les blocs d'organisation (OB) :

Ils constituent l'interface entre le système d'exploitation et le programme utilisateur. Ils sont appelés par le système d'exploitation et gèrent le traitement de programme cyclique et déclenché par alarme, ainsi que le comportement à la mise en route de l'automate programmable et le traitement des erreurs. On peut programmer les blocs d'organisation et déterminer ainsi le comportement de la CPU.

Le bloc d'organisation **OB1** est généré automatiquement lors de la création d'un projet, il représente le programme principal (MAIN dans le langage C). En effet, c'est le programme appelé cycliquement par le système d'exploitation.
Les autres blocs, existants dans le projet seront exécutés à leur appel par l'OB1.

4.1.5.2. Les blocs fonctionnels (FB) :

- **Le FB :**

C'est un sous programme écrit par l'utilisateur, il facilite la programmation de fonctions complexes souvent utilisées. Il est exécuté par l'appel d'autre bloc de code.
Un bloc de données d'instance, qui constitue sa mémoire, lui est associé. Ce dernier contient les paramètres transmis au FB ainsi que les variables statiques.

- **Le bloc fonctionnel système (SFB):**

C'est un bloc fonctionnel intégré à la CPU S7. Les SFB font partie du système d'exploitation, par conséquent, ils ne sont pas chargés en tant que partie du programme. Comme les FB, les SFB sont des blocs avec mémoire. On doit donc également créer pour les SFB des blocs de données d'instance que l'on charge dans la CPU en tant que partie du programme.

Ils sont utilisés pour des fonctions spéciales intégrées de la CPU 314 IFM, comme ils peuvent être utilisés pour la communication via des liaisons configurées.

4.1.5.3. Les fonctions (FC) :

- **La FC :**

Elle contient des routines pour les fonctions fréquemment utilisées, comme le renvoi d'une valeur au bloc appelant. Elle est sans mémoire et contient uniquement des variables temporaires qui sont sauvegardées dans la pile de données locales et perdues à l'achèvement de cette fonction.
Mais elle peut faire appel à des blocs de données globaux pour la sauvegarde de ses données.

- **La fonction système (SFC) :**

C'est une fonction intégrée dans la CPU S7, pré-programmée et testée. Elle est appelée à partir du programme. Comme ces fonctions font partie du système

d'exploitation, elles ne sont pas chargées en tant que partie du programme. Comme les FC, les SFC constituent des blocs sans mémoire.

Parmi les fonctionnalités qu'elles proposent : Le contrôle du programme, la gestion des alarmes horaires et temporisées, la mise à jour de la mémoire image du processus, l'adressage de modules et la création de messages relatifs aux blocs.

4.1.5.4. les blocs de données d'instance (DB d'instance) :

Associé à chaque bloc fonctionnel, il contient les paramètres effectifs et les données statiques du FB.

On peut utiliser plusieurs DB pour un même FB ; par exemple, un FB pour la commande de plusieurs moteurs, les données de chaque moteur sont sauvegardées dans différents DB.

4.1.5.5. les blocs de données globaux (DB):

A l'opposé des DB d'instance qui ne sont associés qu'aux blocs fonctionnels, les DB globaux servent à l'enregistrement de données utilisateur pouvant être utilisées par tous les autres blocs de code.

Remarque :

A l'appel d'un FB, le DB d'instance lui correspondant est automatiquement généré une fois qu'il est inséré à son emplacement.

Mais si on veut créer un bloc de données à partir du dossier bloc, on procède comme suit :

- On clique avec le bouton droit de la souris, puis sur *Insérer un nouvel objet> Bloc de données*,
- Ou Dans la fenêtre SIMATIC Manager, on clique sur le menu *Insertion>Bloc S7>bloc de données*.

Dans les deux cas, le type du bloc de données sera demandé (DB d'instance ou DB global).

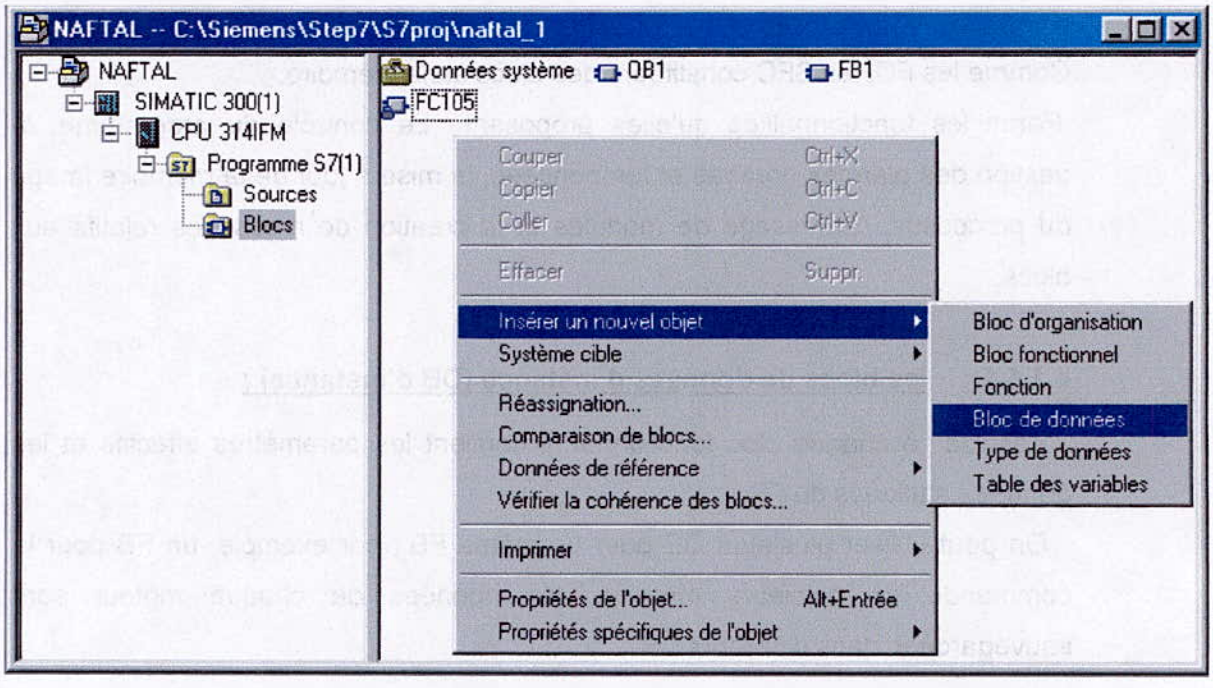


Figure2.7 Création de blocs de données

5. Blocs d'organisation :

Ils permettent l'organisation totale du programme utilisateur, en déclenchant l'exécution conditionnelle de certaines parties du programme, ceci se fait :

- A la mise en route de la CPU.
- A des horaires précis ou cycliquement.
- A l'apparition d'erreurs.
- A l'apparition d'alarmes de processus.

5.1. Blocs d'organisation dans la CPU S7-314 IFM :

Les blocs d'organisation qui existent dans la CPU S7-314 IFM sont :

5.1.1. L'OB1 (Programme cyclique):

Comme dans toutes les CPU S7, l'OB1 est exécuté de manière cyclique.

Il peut être interrompu par tous les autres OB car il dispose de la classe de priorité la plus basse. Les événements suivants provoquent son appel par le système d'exploitation :

- La fin du traitement de la mise en route.
- La fin du traitement de son cycle précédent.

5.1.2. L'OB100 (Mise en route) :

Il contient le programme à exécuter lors d'une mise en route.

5.1.3. Les OB d'alarmes :

On appelle alarmes les événements qui déclenchent l'appel d'un OB donné.

- **OB10 (Alarme horaire):**

A l'appel de cet OB, le programme qui s'y trouve ne s'exécute qu'une seule fois ou de manière cyclique toutes les minutes, toutes les heures, tous les jours, toutes les semaines, tous les mois, tous les ans, ou à la fin du mois.

Pour appeler l'OB10, il faut qu'il soit activé et paramétré, ceci se fait en utilisant :

- STEP7 : Dans HW Config, activer et paramétrer l'alarme horaire en accédant aux propriétés de la CPU.
- Le programme utilisateur : activer l'alarme en appelant le SFC30 et la paramétrer avec la SFC28.

- **OB20 (Alarme temporisée):**

Cet OB est appelé après l'écoulement d'un retard paramétré. Pour ce faire, il faut procéder comme suit :

- Appeler la fonction système SFC32,
- Chargez l'OB d'alarme temporisée dans la CPU comme partie du programme.

- **OB35 (Alarme cyclique) :**

L'OB est appelé à des intervalles de temps égaux qui peuvent être de 1ms à 60000 ms. La valeur de cycle par défaut est de 100 ms, mais on peut la changer en accédant aux propriétés de la CPU dans HW Config.

- **OB40 (Alarme processus) :**

Quand un module déclenche une alarme de processus, le système d'exploitation identifie de quel module il s'agit, et déclenche l'OB40, qui une fois exécuté, permet l'acquittement de cette voie particulière.

Si une autre alarme de processus sur le même module est détectée entre l'identification et l'acquittement de l'alarme en cours, elle n'est pas prise en considération.

Dans le cas de la CPU 314 IFM, les alarmes de processus peuvent être déclenchées par les fonctions intégrées, si elles sont validées lors de la configuration de ces fonctions.

5.1.4. Les OB d'erreurs :

- **OB82 (diagnostic des modules):**

Le diagnostic signale les erreurs survenant sur les modules et permet alors l'exécution de l'OB82. Cet OB contient, dans ses variables locales, l'adresse de base logique du module erroné ainsi que des informations de diagnostic de quatre octets de long.

Si l'OB82 n'a pas été programmé, la CPU passe à l'état arrêt à la survenue d'une erreur.

- **OB80 (Erreur asynchrone):**

Il est appelé si l'une des erreurs suivantes se produit lors de l'exécution d'un OB :

- Dépassement du temps de cycle.
- Erreur d'acquittement lors de l'exécution d'un OB.
- Saut de l'heure de déclenchement d'un OB (horloge avancée).

Si l'OB80 n'a pas été programmé, la CPU passe à l'état arrêt à la survenue d'une erreur.

- **OB81 (Erreur d'alimentation):**

Cet OB est exécuté si un événement provoque une erreur d'alimentation ; par exemple, absence ou rétablissement de la tension de sauvegarde dans la CPU.

Si l'OB81 n'a pas été programmé, la CPU ne passe pas à l'état arrêt à la survenue d'une erreur.

- **OB85 (Erreur d'exécution de programme)**

Cet OB est exécuté si l'un des événements suivants se produit :

- Absence d'un OB dans la CPU lors de son appel.
- Erreur lors de l'accès du système d'exploitation à un bloc.
- Erreur d'accès à la périphérie lors de l'actualisation de la mémoire image des entrées.
- Erreur d'accès à la périphérie lors du transfert de la mémoire image aux modules de sorties.

Si l'OB85 n'a pas été programmé, la CPU passe à l'état arrêt à la survenue d'une erreur.

- **OB87 (Erreur de communication):**

Le programme de cet OB est exécuté, si une erreur de communication se produit.

Par exemple, l'état d'ensemble des données globales ne peut pas être écrit dans le bloc de données.

Si l'OB87 n'a pas été programmé, la CPU passe à l'état arrêt si une erreur survient.

- **OB121 (Erreur de programmation):**

L'apparition d'erreur lors du traitement du programme utilisateur, déclenche l'appel de ce bloc. Cette erreur peut être :

- Une erreur de conversion BCD.
- Un numéro de temporisation erroné.
- Une erreur de numéro de compteur.
- Une erreur d'écriture à l'accès au DB.
- Une erreur de numéro de bloc à l'ouverture d'un DB.
- Une erreur de numéro de bloc à l'appel d'une FC.
- Une erreur de numéro de bloc à l'appel d'un FB.
- Un DB non chargé.
- Un FC non chargée.
- un FB non chargé.

Si l'OB121 n'a pas été programmé, la CPU passe à l'état arrêt à la survenue d'une erreur.

- **OB122 (Erreur d'accès à la périphérie):**

Cet OB se déclenche à l'apparition d'erreur d'accès à la périphérie en lecture ou en écriture.

Si l'OB122 n'a pas été programmé, la CPU passe à l'état arrêt à la survenue d'une erreur.

Remarque

Les erreurs résolues par l'appel des deux blocs OB121et OB122 peuvent être masquées, démasquées ou lues, par les fonctions systèmes suivantes :

- La SFC36 : masque certains codes d'erreurs.
- La SFC37 : démasque les codes d'erreurs qui ont été masquées par la SFC36.
- La SFC38 : lit le registre d'erreur.

5.2. Classes de priorité des blocs d'organisation :

Les blocs d'organisation définissent l'ordre dans lequel les différentes parties du programme sont traitées. L'exécution d'un OB peut être interrompue par l'appel d'un autre OB.

Cette interruption se fait selon la priorité : les OB de priorité plus élevée interrompent les OB de priorité plus faible.

OB	Classes de priorité
OB1	1
OB10	2
OB20	3
OB35	12
OB40	16
OB80	26
OB81	26
OB82	26
OB85	26
OB100	27
OB121	Selon l'OB responsable de l'interruption
OB122	

Tableau 2.1 Classes de priorité des blocs d'organisation

Remarque :

STEP7 permet la modification des classes de priorité pour les alarmes horaires, alarmes temporisées, alarmes cycliques et alarmes de processus. Mais cette propriété n'est pas possible dans les CPU S7 300.

6. Déroulement d'un programme :

6.1. Les programmes existants dans la CPU:

Dans une CPU, s'exécutent deux programmes ; le système d'exploitation et le programme utilisateur.

6.1.1. Le système d'exploitation :

Il regroupe toutes les fonctions et procédures qui ne sont pas liées à la tâche de programmation, ces fonctions et procédures sont :

- La mise en route.
- L'actualisation de la mémoire image des entrées MIE et l'émission de la mémoire image des sorties MIS.
- L'appel du programme utilisateur.
- L'enregistrement des alarmes et l'appel des OB d'alarme.
- La détection et le traitement des erreurs.

- La gestion des zones de mémoires.
- La communication avec les différents partenaires de communication.

La modification des paramètres par défaut du système d'exploitation permet d'influer sur le comportement de la CPU dans des cas précis. Par exemple, le changement des classes de priorités des blocs d'organisation dans les CPU S7 400.

6.1.2. Le programme utilisateur :

Il est créé par l'utilisateur puis chargé dans la CPU, il contient toutes les fonctions nécessaires au traitement de la tâche d'automatisation. Il doit :

- Déterminer les conditions pour le démarrage à chaud, à froid ou pour le redémarrage de la CPU.
- Traiter des données du processus (par exemple, combiner des signaux binaires, lire et exploiter des valeurs analogiques, définir des signaux binaires pour la sortie, écrire des valeurs analogiques).
- Réagir aux alarmes.
- Traiter les perturbations dans le déroulement normal du programme.

Le déroulement d'un programme commence par une mise en route qui n'est exécutée qu'une seule fois, suivie de l'exécution cyclique du programme utilisateur.

6.2. La mise en route :

- Lors de la mise en route la CPU se comporte comme suit:
- Le programme contenu dans l'OB de mise en route est exécuté, dans ce programme, on peut définir des initialisations utiles pour le programme utilisateur.
- Aucun traitement de programme déclenché par horloge n'est possible.
- Les temporisations sont mises à jour.
- Le compteur d'heure de fonctionnement est exécuté.
- Les sorties TOR des modules de signaux sont verrouillées, mais peuvent être mises à 1 par accès direct.

En plus de la mise sous tension, les causes de la mise en route peuvent être:

- Le changement de position du commutateur de mode de fonctionnement de STOP à RUN ou à RUN/P.
- A la demande d'une fonction de communication depuis la console de programmation (PG) ou par l'appel des blocs fonctionnels de communication.

Il existe 3 types de mise en route :

Démarrage à chaud, démarrage à froid et le redémarrage qui correspondent respectivement aux OB 100, OB 102, OB 101.

La seule mise en route disponible dans la CPU 314 IFM est le démarrage à chaud.

6.2.1. Démarrage à chaud :

Quand la CPU fonctionne sans pile de sauvegarde, un effacement général est automatiquement effectué à la mise sous tension ou au retour de la tension après mise hors tension, puis un démarrage à chaud est exécuté. Le programme utilisateur doit se charger de nouveau.

Au démarrage à chaud, les démarches faites par la CPU dans l'ordre sont :

- Effacer la pile des interruptions et la pile des blocs,
- Effacer les mémentos, temporisations et compteurs non rémanents,
- Effacer la mémoire image des sorties,
- Effacer les sorties des modules de signaux,
- Rejeter les alarmes de processus,
- Rejeter les alarmes temporisées,
- Rejeter les alarmes de diagnostic,
- Actualiser la liste d'état système (SZL),
- Exploiter les paramètres des modules et les transmettre aux modules ou bien leur transmettre les valeurs par défaut,
- Traiter l'OB de mise en route concerné,
- Actualiser la mémoire image des entrées,
- Valider les sorties TOR (débloquer les sorties TOR) après passage à l'état de fonctionnement "Marche".

6.2.2. Démarrage à froid et redémarrage :

Au démarrage à chaud, l'effacement des mémentos, des temporisations et des compteurs concernent uniquement les zones non rémanentes, alors que le démarrage à froid les efface tous.

Le redémarrage n'efface aucune zone de mémoire ; la pile des interruptions, la pile des blocs, les mémentos , les temporisations, les compteurs.

L'effacement de la mémoire image des sorties et les sorties des modules de signaux est paramétrable. La propriété qui distingue le redémarrage des deux autres types de mise en route est le traitement du cycle restant (partie du programme utilisateur n'ayant pas pu être exécutée en raison d'une mise hors tension).

6.3. Exécution cyclique:

A la fin de la mise en route, le programme utilisateur s'exécutera de manière cyclique dans l'OB1 comme suit :

- Ecriture de la mémoire image des sorties MIS dans les modules de sorties.
- Lecture des entrées et mise à jour de la mémoire image des entrées.
- Traitement du programme utilisateur.

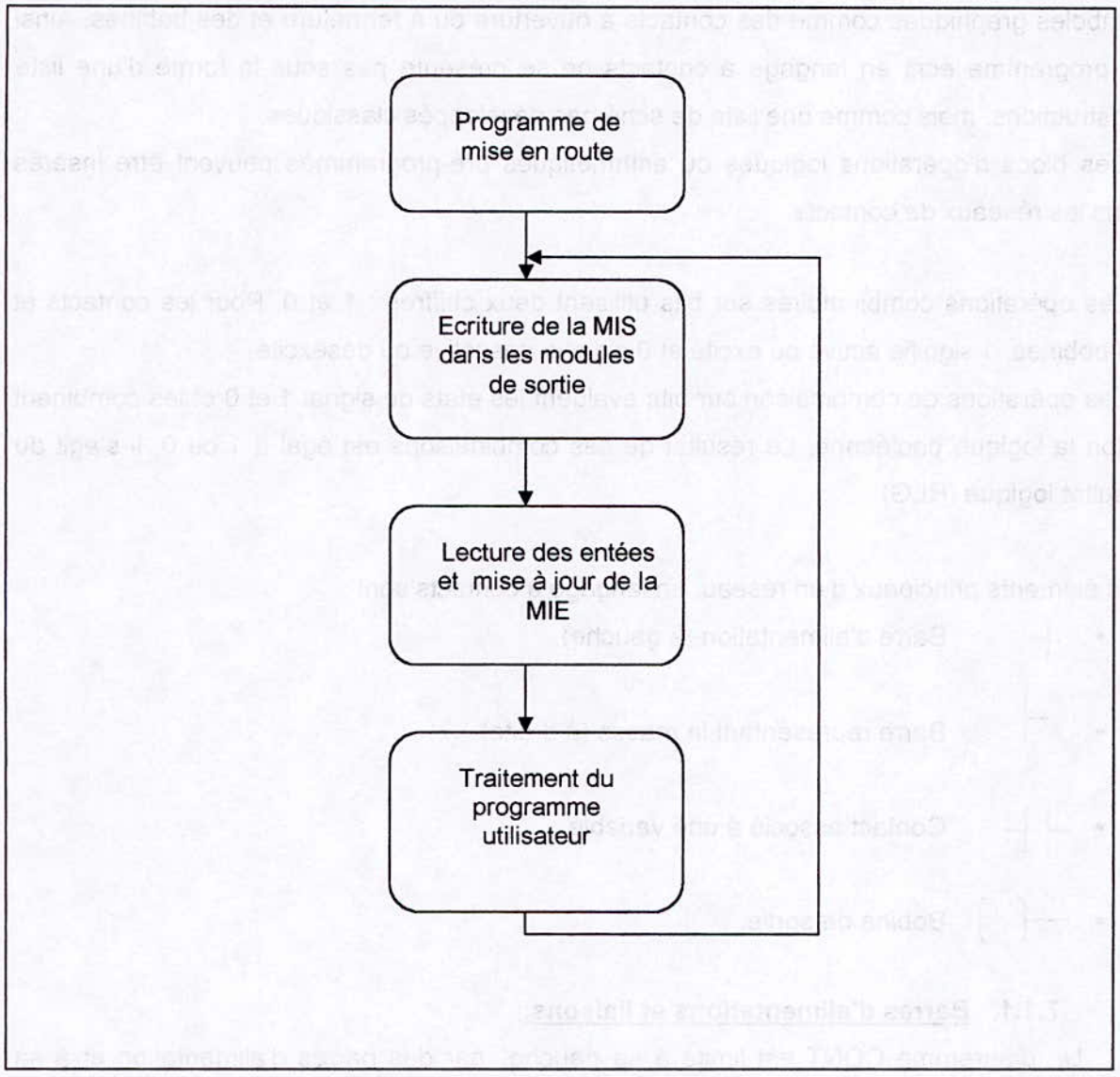


Figure2.8 Déroulement d'un programma utilisateur.

7. Les langages de programmation :

7.1. Le CONT:

C'est un langage dont la logique est inspirée des réseaux électriques, ce qui en fait un langage facile pour les habitués des montages à base de relais.

Le langage à contact, est aussi appelé CONT ou LADDER. C'est un langage entièrement graphique, bien adapté au traitement logique simple de type combinatoire. Il utilise les symboles graphiques comme des contacts à ouverture ou à fermeture et des bobines. Ainsi un programme écrit en langage à contacts ne se présente pas sous la forme d'une liste d'instructions, mais comme une liste de schémas développés classiques.

Des blocs d'opérations logiques ou arithmétiques pré-programmés peuvent être insérés dans les réseaux de contacts.

Les opérations combinatoires sur bits utilisent deux chiffres : 1 et 0. Pour les contacts et les bobines, 1 signifie activé ou excité et 0 signifie désactivé ou désexcité.

Les opérations de combinaison sur bits évaluent les états de signal 1 et 0 et les combinent selon la logique booléenne. Le résultat de ces combinaisons est égal à 1 ou 0. Il s'agit du résultat logique (RLG).

Les éléments principaux d'un réseau en langage à contacts sont :

- ├ Barre d'alimentation (à gauche).
- ┤ Barre représentant la masse (à droite).
- ┤├ Contact associé à une variable.
- —( ) Bobine de sortie.

7.1.1. Barres d'alimentations et liaisons:

Un diagramme CONT est limité à sa gauche par des barres d'alimentation et à sa droite par une barre représentant la masse.

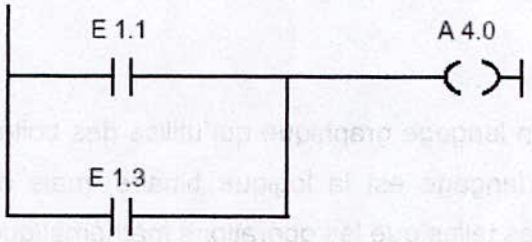
Les différents composants sont reliés entre eux par des arcs de liaisons horizontaux ou verticaux. Chaque segment de liaison peut prendre les états TRUE ou FALSE (1 ou 0).

L'état d'une liaison est propagé à sa droite.

Toute liaison horizontale connectée à une barre d'alimentation à gauche est active.

On peut avoir des liaisons multiples en combinant plusieurs arcs de liaisons horizontaux à un arc de liaison vertical.

Avec ces quelques règles on peut déjà écrire un réseau basic :



A4.0 n'est mise à 1 que si : E1.1 **ou** E1.3 sont à 1.

Figure 2.9 "OU" logique avec CONT.

7.1.2. Les contacts:

Les principaux contacts qui peuvent être utilisés dans un diagramme sont:

- Contact à fermeture 
- Contact à ouverture 

Remarques:

- Le nom de la variable associée est inscrit au-dessus du symbole graphique.
- Les contacts ne représentent pas ce qui est relié physiquement à l'automate, mais sont une interrogation à 1 ou à 0 de l'opérande associé.

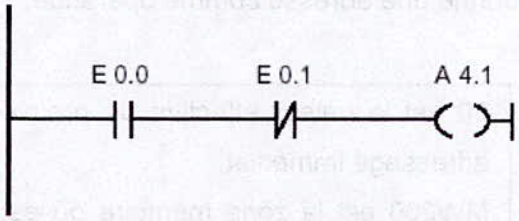
Un contact représente une liaison entre l'état d'un arc et la variable booléenne associée à ce contact.

En effet l'état de la liaison à droite du contact n'est que le résultat du ET logique entre l'état de l'arc de gauche et:

- L'état de la variable associée au contact pour un contact à fermeture.
- L'inverse de l'état de la variable associée au contact pour un contact à ouverture.

7.1.3. Les bobines:

Les bobines standard sont une affectation du résultat logique à la variable associée.



A4.1 n'est mise à 1 que si : E0.0 **et non** (E0.1) sont à 1 simultanément.

Figure 2.10 ET logique avec CONT.

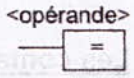
Remarque :

Les opérations CONT possibles avec STEP 7 sont citées en annexe.

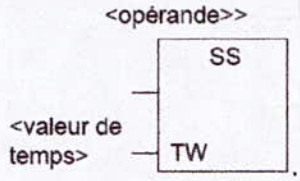
7.2. Le LOG :

Le langage LOG est un langage graphique qui utilise des boites logiques d'algèbre de BOOL. La base de ce langage est la logique binaire, mais on peut aussi faire des opérations plus complexes telles que les opérations mathématiques à l'aide de blocs.

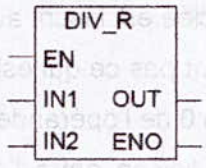
Les instructions LOG peuvent être sous quatre formes :

- Sous forme d'éléments : comme une inversion logique. 
- Sous forme de boite avec opérande : comme une affectation. 

- Sous forme de boite avec opérande et valeur : comme une temporisation.



- Sous forme de boites avec paramètres : comme diviser un nombre réel



Remarque :

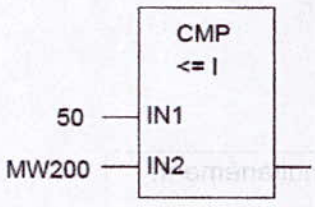
- Pour l'activation d'une boite, il faut que EN soit à l'état haut.
- Si la boite s'exécute sans erreur alors ENO est mise à 1, dans le cas contraire ENO est à 0.

Adressage :

Il n'y a que deux types d'adressage possibles avec LOG :

- Adressage immédiat : donne une constante comme opérande.
- Adressage direct : donne une adresse comme opérande.

Exemple :



50 est la valeur effective du premier paramètre, c'est un adressage immédiat.

MW200 est la zone mémoire où est stockée la valeur du second paramètre : c'est un adressage direct.

Figure 2.11 Adressages immédiat et direct avec LOG.

7.3. Le LIST :

7.3.1. Présentation :

Le langage LIST figure parmi les langages de base du logiciel STEP7, sa syntaxe est similaire à celle de l'assembleur.

C'est le langage le plus proche du langage machine MC7 des CPU S7, ce qui lui donne l'avantage d'être le langage le plus adapté pour la programmation avec optimisation d'espace mémoire et de temps d'exécution.

Il dispose d'un jeu d'instruction très important permettant la création de programmes utilisateur complets.

Tout programme écrit en CONT ou en LOG peut être réécrit en LIST.

7.3.2. Structure et éléments de LIST :

Dans le langage LIST, l'éditeur de programme divise l'espace de programmation en deux parties ; partie programme et partie commentaire, ces deux parties sont séparées par //.

7.3.3. Structure d'une instruction :

Une instruction LIST est constituée soit :

- En une opération seulement,
- En une opération + opérande.

L'opérande peut indiquer :

- une constante.
- un bit du mot d'état.
- une mnémonique.
- un bloc de données.
- un identificateur d'opérande d'une zone mémoire précise + une adresse.

Exemples :

Type d'opérande		Exemple	Explication
Constante	Entier	L 25	Charger l'entier 25
	Temporisation	L S5T#12s	Charger 12 secondes comme valeur initiale de la temporisation dans l'accumulateur 1
Adresse dans le mot d'état	Bit dans le mot d'état	U BIE	Interroger l'état du bit RB du mot d'état
Mnémonique	Nom symbolique	L vitesse	Charger l'octet, le mot ou le double mot dont le mnémonique est vitesse
Bloc de données	Donnée du bloc de données.	U DB4.DBX5.1	Interroger l'état du bit 5.1 du bloc de donnée 4

Tableau 2.2 Quelques types d'opérandes.

Chapitre

3

Les fonctions intégrées de la CPU S7-314 IFM

Les automates programmables S7-314 IFM, ont un module intégré qui comporte plusieurs fonctions intéressantes. Ces fonctions permettent de contrôler des systèmes à des coûts très appréciables. Les autres solutions possibles pour gérer une tâche d'automatisation sont :

- Programmer des blocs pour la tâche.
- Mettre en place des modules spécialisés.

Le tableau suivant montre les 3 possibilités citées ainsi que les différents critères de sélection :

Critère de sélection	**Programme utilisateur**	**Fonctions intégrées**	**Modules de fonction**
Liaison directe avec les E/S	Non	Oui	Oui
Impact sur le temps de cycle	Oui	Minime	Non
Taux de couverture des applications	Faible	Moyen (50%)	Elevé (95%)
performance au niveau du temps de réaction	Faible	Moyen	Elevé
Traitement des défauts de processus	Non	Limité	Oui

Tableau 3.1 Critères de sélections pour la résolution d'un problème d'automatisation à base d'automates programmables.

Les fonctions intégrées à la CPU S7-314 IFM sont :

- Fonction intégrée fréquencemètre.
- Fonction intégrée compteur (1 compteur pour comptage/décomptage)
- Fonction intégrée compteur A/B (2 compteurs A et B pour comptage/décomptage).
- Fonction intégrée positionnement.

Les modules spécialisés offrent un ajout sur temps de cycle nul, mais il faut noter que le prix d'une solution avec des modules spécialisés est nettement supérieur à celui d'une solution avec CPU dotées de fonctions intégrées. Il est donc préférable d'utiliser une CPU IFM si les performances d'un module spécialisé ne sont pas nécessaires.

Il existe 4 entrées spéciales sur une CPU S7-314 IFM. Les configurations possibles sont :

- 4 entrées pour les alarmes processus. (entrées TOR)
- 4 entrées TOR pour la fonction intégrée compteur.

- 4 entrées TOR pour la fonction intégrée compteurs A/B.
- 1 entrée TOR pour la fonction intégrée fréquencemètre et 3 entrées TOR standard.
- 3 entrées TOR pour la fonction intégrée positionnement et 1 entrée TOR standard.

Les entrées spéciales qui ne sont pas utilisées pour une fonction intégrée ou pour une alarme de processus peuvent être utilisées comme entrées TOR standard.

Remarques :

La fréquence maximum tolérée par ces fonctions intégrées est de 10kHz. Lorsque cette valeur est dépassée :

- Le fonctionnement correct de la CPU n'est plus garanti.
- Le temps de cycle est allongé. (Sachant que le dépassement du temps de cycle paramétré pour le chien de garde fait passer la CPU en mode STOP)
- Le temps de réaction à l'alarme processus est allongé.
- La communication peut être perturbée et même coupée.

Toutes les fonctions peuvent déclencher des alarmes processus. Ces alarmes appellent l'OB40.

Activation et paramétrage des fonctions intégrées :

L'activation des fonctions intégrées se fait par le biais de STEP 7.

Pour choisir un mode de fonctionnement, il faut ouvrir HW CONFIG de STEP 7. Après avoir effectué la configuration matérielle, il faut ouvrir les propriétés de la CPU, puis dans l'onglet *fonction intégrée* choisir la fonction à activer.

Un bouton *paramètre* permet une configuration plus approfondie des fonctions intégrées. Dans les paragraphes suivants, les configurations des fonctions intégrées seront toujours relatives aux fenêtres citées précédemment.

1. Fonction intégrée fréquencemètre :

1.1. description de la fonction intégrée fréquencemètre :

L'entrée pour la fonction fréquencemètre est à l'adresse E126.0

Cette fonction permet de mesurer une fréquence allant jusqu'à 10kHz.

La fréquence est donnée en comptant le nombre de fronts montants du signal de mesure sur une durée donnée.

Il existe 2 principes de mesure :

- Principe 1 : utilisé pour des durées de mesure de 0.1s, 1s ou 10s. La fréquence est donnée selon la formule suivante :

$$\text{Fréquence} = \frac{\text{nombre de fronts montants}}{\text{durée de mesure}}$$

- Principe 2 : utilisé pour des durées d'actualisation de 1ms, 2ms ou 4ms. La fréquence est donnée en calculant le temps écoulé entre deux fronts montants sur l'entrée de mesure.

La durée de mesure est configurable avec STEP 7.
Une nouvelle mesure est lancée dès qu'un résultat est trouvé, ainsi la fréquence est actualisée en permanence.

La fonction intégrée fréquencemètre comporte deux comparateurs qui permettent de vérifier que la plage de travail admise est bien respectée :

- Le comparateur de limite supérieure réagit dès que la fréquence dépasse un seuil U_LIMIT. Dans ce cas, le bit STATUS_U du SFB 30 se met à 1.
- Le comparateur de limite inférieure réagit dès que la fréquence est en dessous d'un seuil L_LIMIT. Dans ce cas, le bit STATUS_L du SFB 30 se met à 1.
- Le dépassement vers le haut ou vers le bas d'une limite peut, si la configuration à été faite, déclencher une alarme de processus. (la gestion des alarmes sera décrite plus tard)

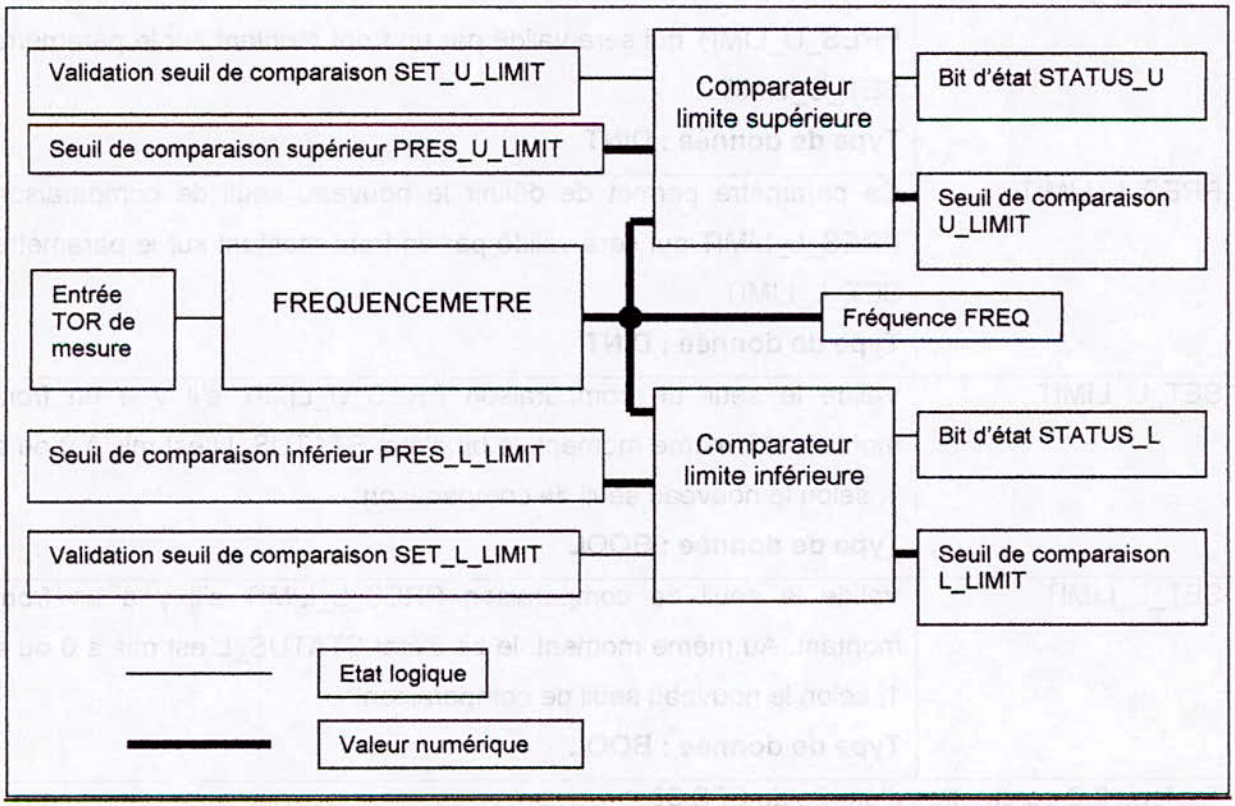


Figure 3.1 Représentation synoptique de la fonction intégrée Fréquencemètre.

1.2. Description du bloc fonctionnel système SFB 30 :

Le bloc affecté à la fonction intégrée fréquencemètre est le SFB 30.

Sa représentation en CONT est la suivante :

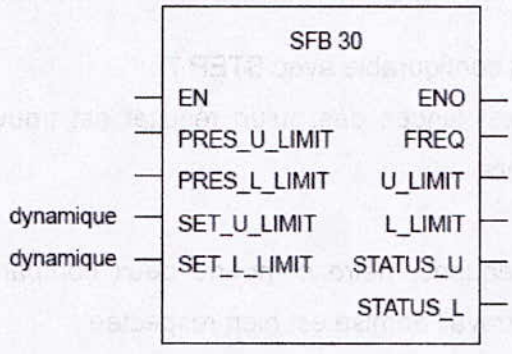


Figure 3.2 représentation du SFB 30 avec CONT.

Les tableaux suivants décrivent les paramètres d'entrée et de sortie du bloc :

Paramètre d'entrée	Description
EN	Entrée d'activation du bloc (valeur booléenne) **Type de donnée : BOOL**
PRES_U_LIMIT	Ce paramètre permet de définir le nouveau seuil de comparaison PRES_U_LIMIT qui sera validé par un front montant sur le paramètre SET_U_LIMIT. **Type de donnée : DINT**
PRES_L_LIMIT	Ce paramètre permet de définir le nouveau seuil de comparaison PRES_L_LIMIT qui sera validé par un front montant sur le paramètre SET_L_LIMIT. **Type de donnée : DINT**
SET_U_LIMIT	Valide le seuil de comparaison PRES_U_LIMIT s'il y a un front montant. Au même moment, le bit d'état STATUS_U est mis à 0 ou à 1, selon le nouveau seuil de comparaison. **Type de donnée : BOOL**
SET_L_LIMIT	Valide le seuil de comparaison PRES_L_LIMIT s'il y a un front montant. Au même moment, le bit d'état STATUS_L est mis à 0 ou à 1, selon le nouveau seuil de comparaison. **Type de donnée : BOOL**

Tableau 3.2 paramètres d'entrée du SFB 30.

Paramètre de sortie	Description
ENO	Est validée à 1 si aucun défaut de traitement n'a été détecté sur le bloc SFB30. Si ENO = 0, le SFB n'a pas été traité ou a été traité de façon insatisfaisante. **Type de donnée : BOOL**
FREQ	Donne la fréquence mesurée en mHz. **Type de donnée : DINT**
U_LIMIT	Donne le seuil de comparaison supérieur utilisé pour le cycle en cours. **Type de donnée : DINT**
L_LIMIT	Donne le seuil de comparaison inférieur utilisé pour le cycle en cours. **Type de donnée : DINT**
STATUS_U	Se met à 1 si le seuil de comparaison supérieur à été dépassé vers le haut. Dans tous les autres cas il est à 0. **Type de donnée : BOOL**
STATUS_L	Se met à 1 si le seuil de comparaison inférieur a été dépassé vers le bas. Dans tous les autres cas il est à 0. **Type de donnée : BOOL**

Tableau 3.3 Paramètres sorties du SFB 30.

1.3. description du bloc de données associé au SFB30 :

Ce bloc système doit être lié à un DB d'instance. Ce DB est par défaut le DB 62, mais il peut être configuré avec STEP 7.

La structure de ce bloc de données est la suivante :

Opérande	Mnémonique	Signification
DBD 0	PRES_U_LIMIT	Seuil de comparaison (nouveau) pour limite supérieure
DBD 4	PRES_L_LIMIT	Seuil de comparaison (nouveau) pour limite inférieure
DBX 8.0	SET_U_LIMIT	Validation du seuil de comparaison pour limite sup.
DBX 8.1	SET_L_LIMIT	Validation du seuil de comparaison pour limite inf.
DBD 10	FREQ	Fréquence
DBD 14	U_LIMIT	Seuil de comparaison (actuel) pour limite supérieure
DBD 18	L_LIMIT	Seuil de comparaison (actuel) pour limite inférieure
DBX 22.0	STATUS_U	Bit d'état Limite supérieure
DBX 22.1	STATUS_L	Bit d'état Limite inférieure

Tableau 3.4 Structure du DB 62.

Les données pour la fonction intégrée fréquencemètre occupent 24 octets dans le DB d'instance en commençant par l'adresse 0.

2. Fonction intégrée compteur :

2.1. Description de la fonction intégrée compteur :

La fonction intégrée compteur permet de faire un comptage et un décomptage à partir de signaux d'une fréquence allant jusqu'à 10kHz.

Ce comptage (décomptage) peut être soit sensible aux fronts montants soit aux fronts descendants en configurant ce paramètre à l'aide de STEP 7.

Les entrées/sorties utilisées pour la fonction intégrée compteur sont :

Adresse	Fonction
E126.0	Entrée TOR comptage
E126.1	Entrée TOR décomptage
E126.2	Entrée TOR sens
E126.3	Entrée TOR Marche/Arrêt
A124.0	Sortie TOR A
A124.1	Sortie TOR B

Tableau 3.4 Entrées/Sorties associées à la fonction intégrées compteur.

L'entrée TOR « sens » permet d'inverser le comptage et le décomptage, si l'entrée E126.2 est à 0 alors l'entrée TOR de comptage effectue une décrémentation et l'entrée TOR de décomptage une incrémentation.

Le schéma de la page suivante décrit l'architecture de la fonction intégrée compteur.

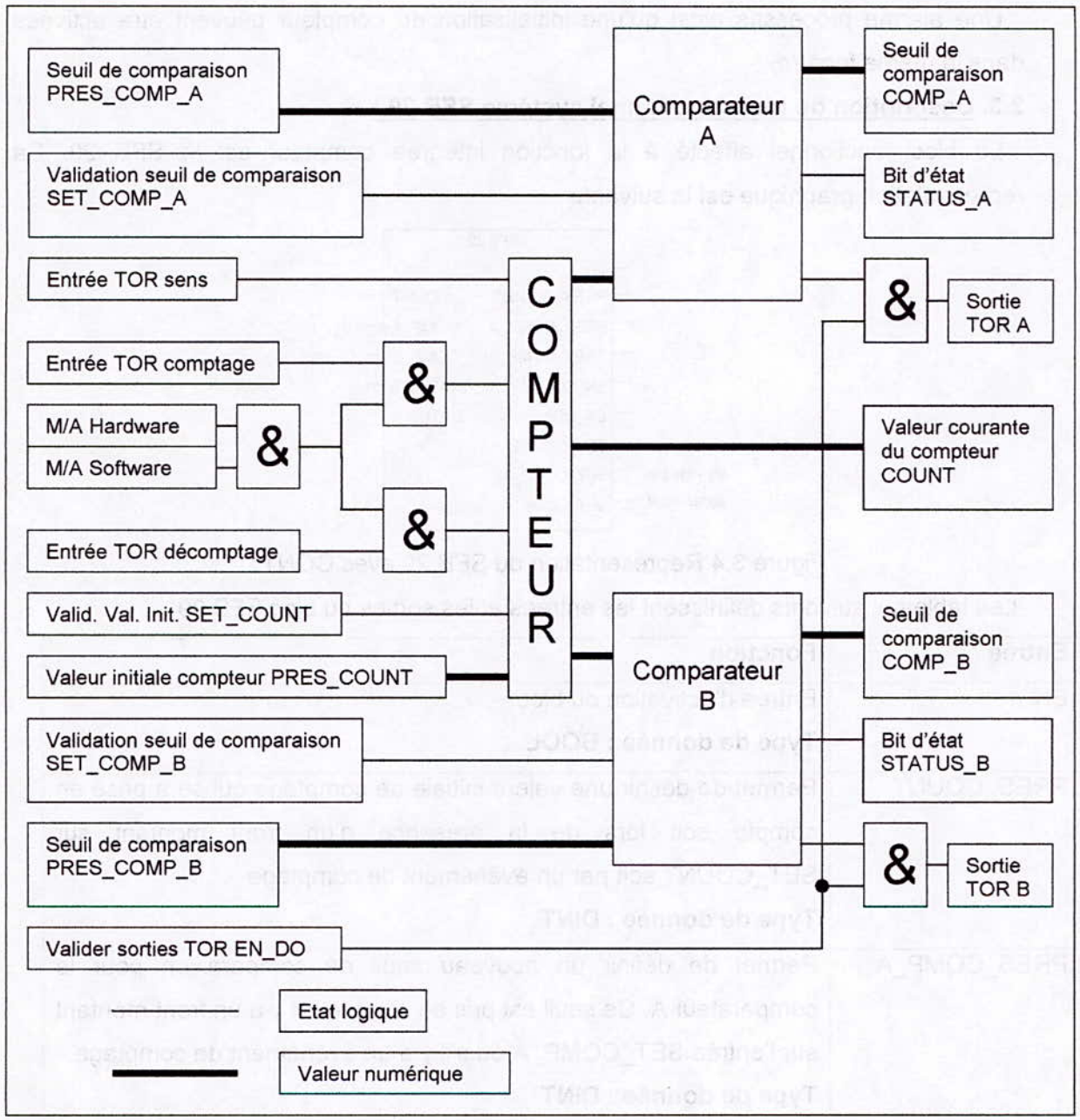


Figure 3.3 Représentation synoptique de la fonction intégrée compteur.

2.2. Fonctionnement des comparateurs A et B :

Ces deux comparateurs sont configurables. Les bits d'état se mettent à 1 si le compteur indique une valeur supérieure ou égale au seuil de comparaison.

Ainsi la réaction des sorties TOR A (respectivement TOR B) peut être configurée comme suit :

- Aucun changement quel que soit l'état du compteur.
- Mise à 1 si le compteur atteint le seuil de comparaison PRES_COMP_A (respectivement PRES_COMP_B) par le bas.
- Mise à 1 si le compteur dépasse le seuil de comparaison PRES_COMP_A (respectivement PRES_COMP_B) vers le bas.

Une alarme processus ainsi qu'une initialisation du compteur peuvent être activées dans la même fenêtre.

2.3. Description du bloc fonctionnel système SFB 29 :

Le bloc fonctionnel affecté à la fonction intégrée compteur est le SFB 29. Sa représentation graphique est la suivante :

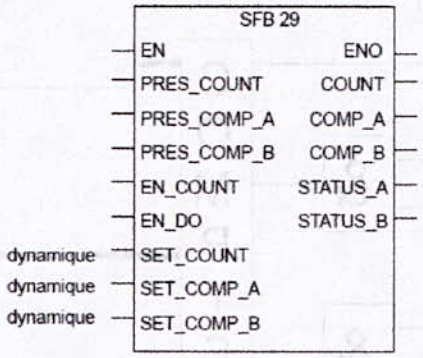


Figure 3.4 Représentation du SFB 29 avec CONT.

Les tableaux suivants définissent les entrées et les sorties du bloc SFB 29 :

Entrée	Fonction
EN	Entrée d'activation du bloc. **Type de donnée : BOOL**
PRES_COUNT	Permet de définir une valeur initiale de comptage qui sera prise en compte soit lors de la présence d'un front montant sur SET_COUNT soit par un évènement de comptage **Type de donnée : DINT**
PRES_COMP_A	Permet de définir un nouveau seuil de comparaison pour le comparateur A. Ce seuil est pris en compte s'il y a un front montant sur l'entrée SET_COMP_A ou s'il y a un évènement de comptage. **Type de donnée : DINT**
PRES_COMP_B	Permet de définir un nouveau seuil de comparaison pour le comparateur B. Ce seuil est pris en compte s'il y a un front montant sur l'entrée SET_COMP_B ou s'il y a un évènement de comptage. **Type de donnée : DINT**
EN_COUNT	Permet d'activer le compteur. Si ce paramètre est à 0 le compteur ne fonctionne pas. Il faut aussi savoir que ce bit est combiné avec l'entrée matérielle avec un ET logique, ce qui fait que le compteur fonctionne si et seulement si les deux paramètres sont à 1. **Type de donnée : BOOL**
EN_DO	Active les sorties TOR A et B. **Type de donnée : BOOL**

Entrée	Fonction
SET_COUNT	Initialise le compteur avec la valeur PRES_COUNT **Type de donnée : BOOL**
SET_COMP_A	Valide le seuil de comparaison du comparateur A s'il y a un front montant. **Type de donnée : BOOL**
SET_COMP_B	Valide le seuil de comparaison du comparateur B s'il y a un front montant. **Type de donnée : BOOL**

Tableau 3.5 Paramètres d'entrée du SFB 29.

Sortie	Fonction
ENO	Se met à 1 si aucun défaut de traitement n'a été constaté dans le bloc. **Type de donnée : BOOL**
COUNT	Fournit la valeur courante du compteur. • Un dépassement de la valeur Max. vers le haut : le compteur continu à partir de la valeur minimale admise. • Un dépassement de la valeur Min. vers le bas : le compteur continu à partir de la valeur maximale admise. **Type de donnée : DINT**
COMP_A	Fournit le seuil de comparaison actuel pour le comparateur A. **Type de donnée : DINT**
COMP_B	Fournit le seuil de comparaison actuel pour le comparateur B. **Type de donnée : DINT**
STATUS_A	Si COUNT ≥ COMP_A alors STATUS_A=1 Si COUNT< COMP_A alors STATUS_A=0 **Type de donnée : BOOL**
STATUS_B	Si COUNT ≥ COMP_B alors STATUS_B=1 Si COUNT < COMP_B alors STATUS_B=0 **Type de donnée : BOOL**

Tableau 3.6 Paramètres de sortie du SFB 29.

Remarque :

DINT est un nombre entier double, il est donc codé sur 32 bits. Les valeurs admises sont donc de -2^{31} à $+2^{31}-1$ ce qui donne une plage de –2147483648 à 2147483647.

2.4. Description du bloc de données associé au SFB 29 :

Ce bloc système doit être lié à un DB d'instance. Ce DB est par défaut le DB 63, mais il peut être configuré avec STEP 7.

La structure de ce bloc de données est la suivante :

Opérande	Mnémonique	Signification
DBD 0	PRES_COUNT	Valeur initiale du compteur
DBD 4	PRES_COMP_A	Seuil de comparaison COMP_A (nouveau)
DBD 8	PRES_COMP_B	Seuil de comparaison COMP_B (nouveau)
DBX 12.0	EN_COUNT	Marche / Arrêt hardware
DBX 12.1	EN_DO	Validation des sorties TOR
DBX 12.2	SET_COUNT	Initialisation du compteur
DBX 12.3	SET_COMP_A	Validation du seuil de comparaison COMP_A
DBX 12.4	SET_COMP_B	Validation du seuil de comparaison COMP_B
DBD 14	COUNT	Valeur courante du compteur
DBD 18	COMP_A	Seuil de comparaison COMP_A (actuel)
DBD 22	COMP_B	Seuil de comparaison COMP_B (actuel)
DBX 26.0	STATUS_A	Bit d'état A
DBX 26.1	STATUS_B	Bit d'état B

Tableau 3.7 Structure du DB 63.

3. Fonction intégrée compteur A/B :

3.1. Description de la fonction intégrée compteur A/B :

Tableau représentant les broches pour l'utilisation de la fonction intégrée compteur A/B.

Adresse	Fonction
E 126.0	Compteur A : Comptage (comptage/décomptage)
E 126.1	Compteur A : Décomptage (Sens)
E 126.2	Compteur B : Comptage (comptage/décomptage)
E 126.3	Compteur B : Décomptage (Sens)
A 124.0	Sortie TOR Compteur A
A 124.1	Sortie TOR Compteur B

Tableau 3.8 Entrées/Sorties associées à la fonction intégrée Compteur A/B.

Cette fonction comporte en fait deux compteurs A et B indépendants.

Chaque compteur comporte deux entrées et une sortie.

Il y a deux configurations possibles pour chaque compteur:

- Une entrée pour le comptage et une autre pour le décomptage.
- Une entrée pour le comptage/décomptage et une autre pour le sens.

Les entrées ne sont sensibles qu'aux fronts montants.

Contrairement à la fonction compteur précédente, celle-ci ne possède pas d'entrée d'activation matérielle. La seule façon d'activer ces compteurs est l'entrée EN_COUNT.

Le schéma suivant montre la structure de cette fonction :

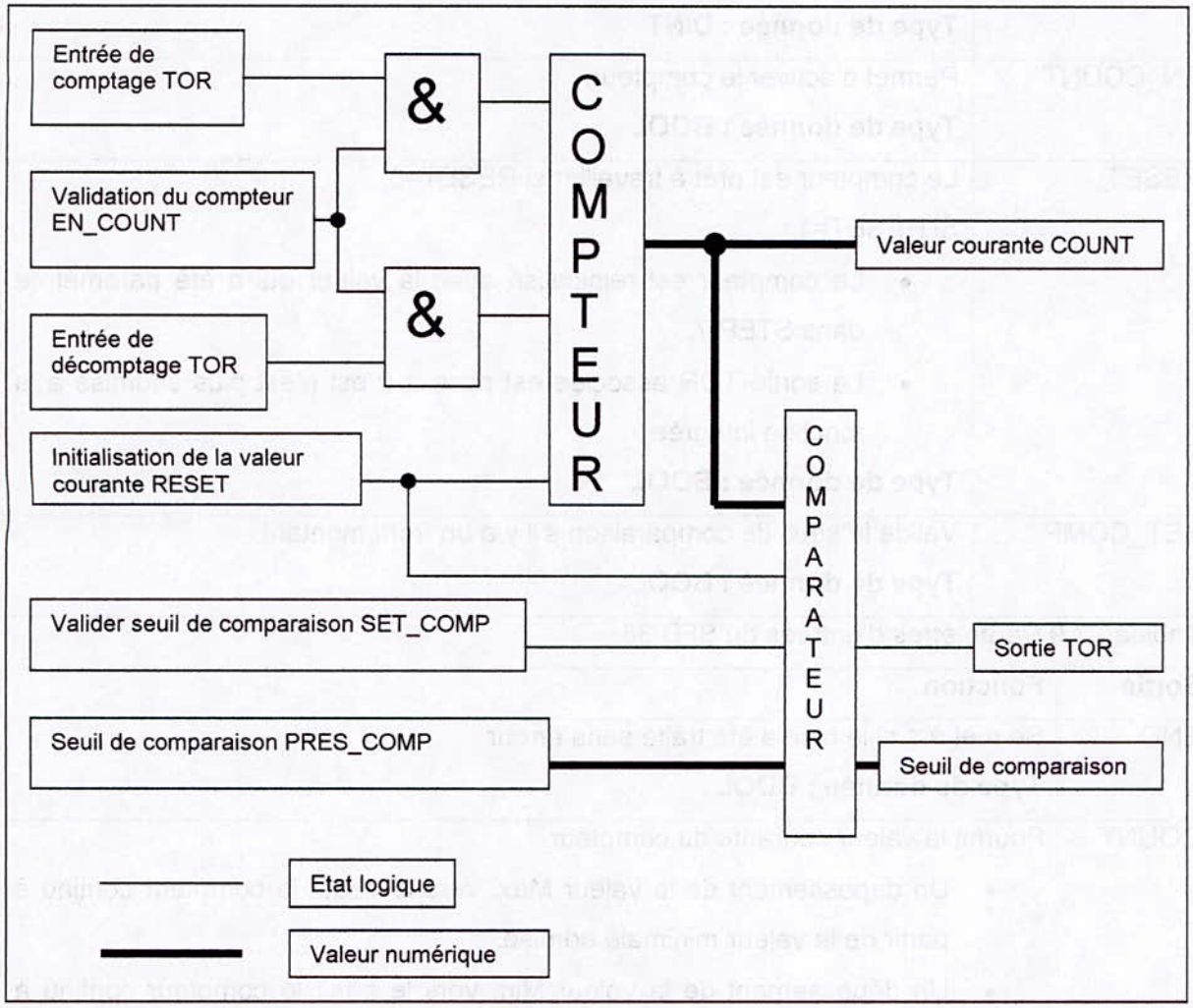


Figure 3.5 Représentation synoptique de la fonction intégrée compteur A/B (pour un seul compteur)

3.2. Le bloc fonctionnel système associé à cette fonction est le SFB 38 :

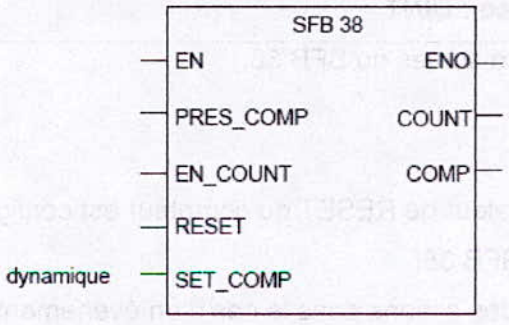


Figure 3.6 Représentation du SFB 38 avec CONT.

Les tableaux suivants décrivent les entrées et sorties de ce bloc :

Entrée	Fonction
EN	Entrée d'activation du bloc. **Type de donnée : BOOL**
PRES_COMP	Permet de définir le nouveau seuil de comparaison qui sera pris en compte s'il a un front montant sur l'entrée SET_COMP **Type de donnée : DINT**
EN_COUNT	Permet d'activer le compteur. **Type de donnée : BOOL**
RESET	Le compteur est prêt à travailler si RESET=0 Si RESET=1 : • Le compteur est réinitialisé avec la valeur qui a été paramétrée dans STEP 7. • La sortie TOR associée est mise à 0 est n'est plus soumise à la fonction intégrée. **Type de donnée : BOOL**
SET_COMP	Valide le seuil de comparaison s'il y a un front montant. **Type de donnée : BOOL**

Tableau 3.9 Paramètres d'entrées du SFB 38.

Sortie	Fonction
ENO	Se met à 1 si le bloc a été traité sans erreur **Type de donnée : BOOL**
COUNT	Fournit la valeur courante du compteur. • Un dépassement de la valeur Max. vers le haut : le compteur continu à partir de la valeur minimale admise. • Un dépassement de la valeur Min. vers le bas : le compteur continu à partir de la valeur maximale admise. **Type de donnée : DINT**
COMP	Fournit le seuil de comparaison actuel. **Type de donnée : DINT**

Tableau 3.10 Paramètres de sorties du SFB 38.

Remarque :

Il faut noter que la valeur de RESET du compteur est configurée à l'aide de STEP 7 et non pas dans le SFB 38.

On peut configurer des actions dans le cas d'un évènement compteur. Ces évènements sont :

- Valeur du comparateur atteinte par le bas.
- Valeur du comparateur quittée vers le bas.

Les actions possibles sont :

Paramètres	Etats possibles
Sortie TOR	▪ Non affectée. ▪ Activée. ▪ Désactivée. ▪ Bascule.
Alarme processus	▪ Activée. ▪ Désactivée
Initialiser compteur	▪ Activée. ▪ Désactivée
Initialiser comparateur	▪ Activée. ▪ Désactivée

Tableau 3.11 Actions possibles en cas d'un évènement compteur.

3.3. description des blocs de données associés au SFB 38 :

Les DB d'instance des compteurs sont par défaut :

- DB 60 pour le compteur A.
- DB 61 pour le compteur B.

La structure de ces DB d'instance est :

Opérande	Mnémonique	Signification
DBD 0	PRES_COMP	Seuil de comparaison (nouveau)
DBX 4.0	EN_COUNT	Validation du compteur
DBX 4.1	RESET	Initialisation du compteur
DBX 4.2	SET_COMP	Positionnement du comparateur
DBD 6	COUNT	Valeur courante du compteur
DBD 10	COMP	Seuil de comparaison (actuel)

Tableau 3.12 Structure des DB60 et DB61

4. Fonction intégrée Positionnement :

4.1. description de la fonction intégrée positionnement :

La fonction intégrée Positionnement de la CPU 314 IFM permet de commander le positionnement d'axes en liaison avec un programme utilisateur.

Les entrées utilisées pour cette fonction sont :

Entrées	Adresse	Rôle

Entrée TOR piste A Entrée TOR piste B	E 126.0 E 126.1	Raccordement d'un codeur incrémental pour la saisie du déplacement
Entrée TOR contact point de référence	E 126.2	Raccordement d'un contact du point de référence pour la synchronisation

Tableau 3.13 Entrées associées à la fonction intégrée positionnement.

La position est prélevée à l'aide d'un codeur incrémental sans signaux inversés 24V. Ce codeur délivre deux trains d'impulsions A et B déphasés de 90°, ce qui permet de connaître la vitesse et le sens de rotation.

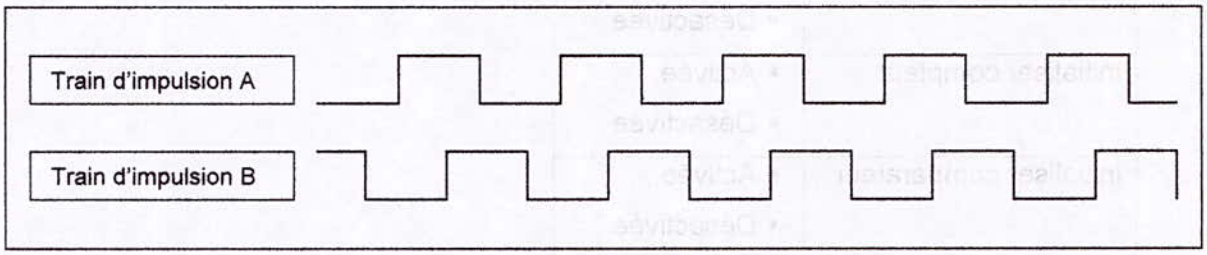


Figure 3.7 Signal délivré par un codeur sans signaux inversés

Le traitement du signal délivré par le codeur est dit en mode simple. Cela veut dire que seuls les fronts montants seront pris en considération.

La fonction intégrée positionnement donne la possibilité de brancher un signal de synchronisation de valeur initiale. Ce signal est donné par la plupart des codeurs et permet de re-calibrer la position initiale à chaque passage par un point de référence donné.

Ce point de référence est matérialisé par un détecteur de position qui délivre un signal de 24V lorsque sa position est atteinte par le moteur.

4.2. Types de connexions possibles avec un moteur :

L'automate programmable S7-314 IFM ne peut pas être raccordé directement à un moteur, il est indispensable d'utiliser une interface de puissance pour la commande en position.

Ces interfaces peuvent être de deux types :

- Montage à contacts : permet de commander un moteur à deux vitesses.
- Variateur de fréquence : permet de commander un moteur synchrone ou asynchrone.

4.2.1. Commande d'un moteur à deux vitesses :

Pour ce faire, 4 sorties TOR sont utilisées :

Sortie	Adresse	Rôle
Sortie TOR petite vitesse	A124.0	Sorties pour le choix de la vitesse du moteur
Sortie TOR grande vitesse	A124.1	
Sortie TOR marche avant	A124.2	Sorties pour le choix du sens de rotation
Sortie TOR marche arrière	A124.3	

Tableau 3.14 Sorties associées à la commande d'un moteur à deux vitesses.

La figure suivante montre le profil de vitesse que l'on peut avoir pour un moteur à deux vitesses :

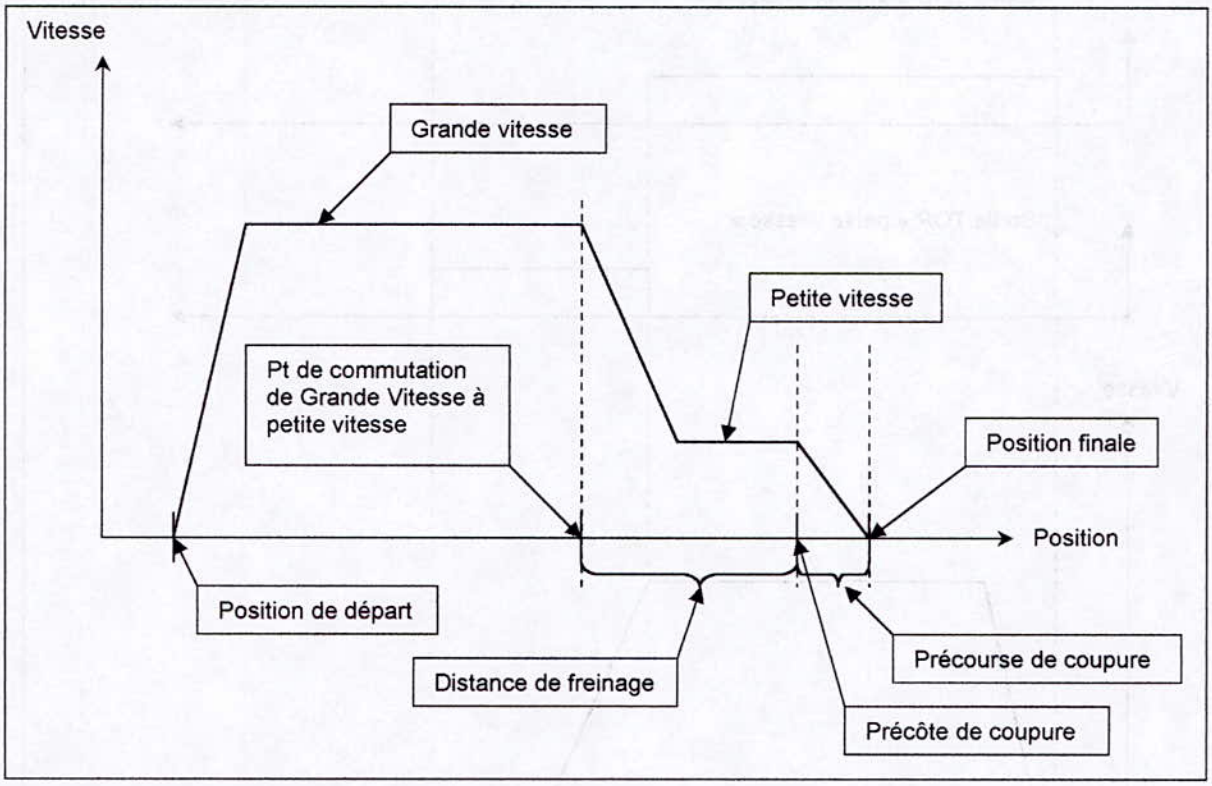


Figure 3.8 Profil de vitesse d'un moteur à deux vitesses.

Le principe est de faire démarrer le moteur avec la grande vitesse, puis lors de la phase de freinage, on commute sur la petite vitesse. Juste avant d'arriver à la position finale, on coupe la petite vitesse et on laisse le moteur s'arrêter jusqu'à atteindre la position désirée.

La figure suivante montre le comportement des sorties TOR lors de la commande :

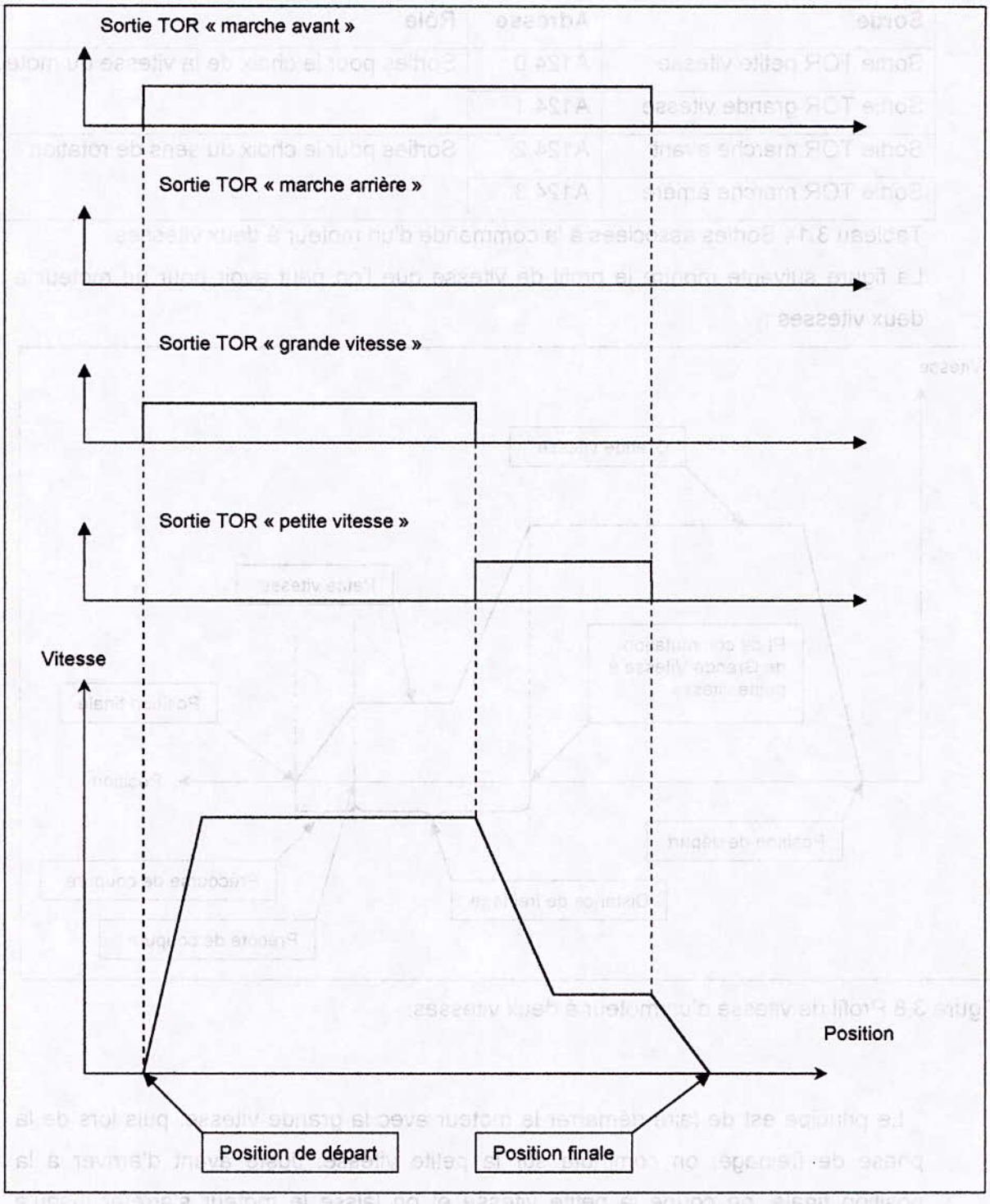


Figure 3.9 Chronogramme des sorties TOR pour la fonction intégrée positionnement. (Positionnement en marche avant)

4.2.2. Commande par convertisseur de fréquence :

Cette méthode est utilisée pour actionner des moteurs synchrones ou asynchrones.

Les sorties utilisées sont :

Sortie	Adresse	Rôle
Sortie TOR marche arrière	A124.2	Utilisé si le convertisseur ne gère que les signaux analogiques positifs.
Sortie TOR marche avant	A124.3	
Sortie analogique vitesse	PAW 128	Donne la valeur de la vitesse : ▪ Délivre une sortie de 0-20mA ou de 0-10V dans le cas d'un convertisseur qui ne gère que des signaux analogiques positifs. ▪ Délivre une sortie de ±20mA ou ±10V pour un convertisseur qui gère des signaux positifs et négatifs.

Tableau 3.15 Sorties associées à la commande d'un moteur par variateur de vitesse.

La commande par convertisseur de fréquence a comme contraintes principales :

- La vitesse maximale admise par le système.
- L'accélération maximale admise par le système.

La commande donnée fait en sorte que ces valeurs ne soient jamais dépassées tout en optimisant le temps.

Le profil de vitesse utilisé est de type trapézoïdal.

La figure qui suit montre les profils de vitesse et d'accélération utilisés par la fonction intégrée positionnement de la CPU S7-314 IFM:

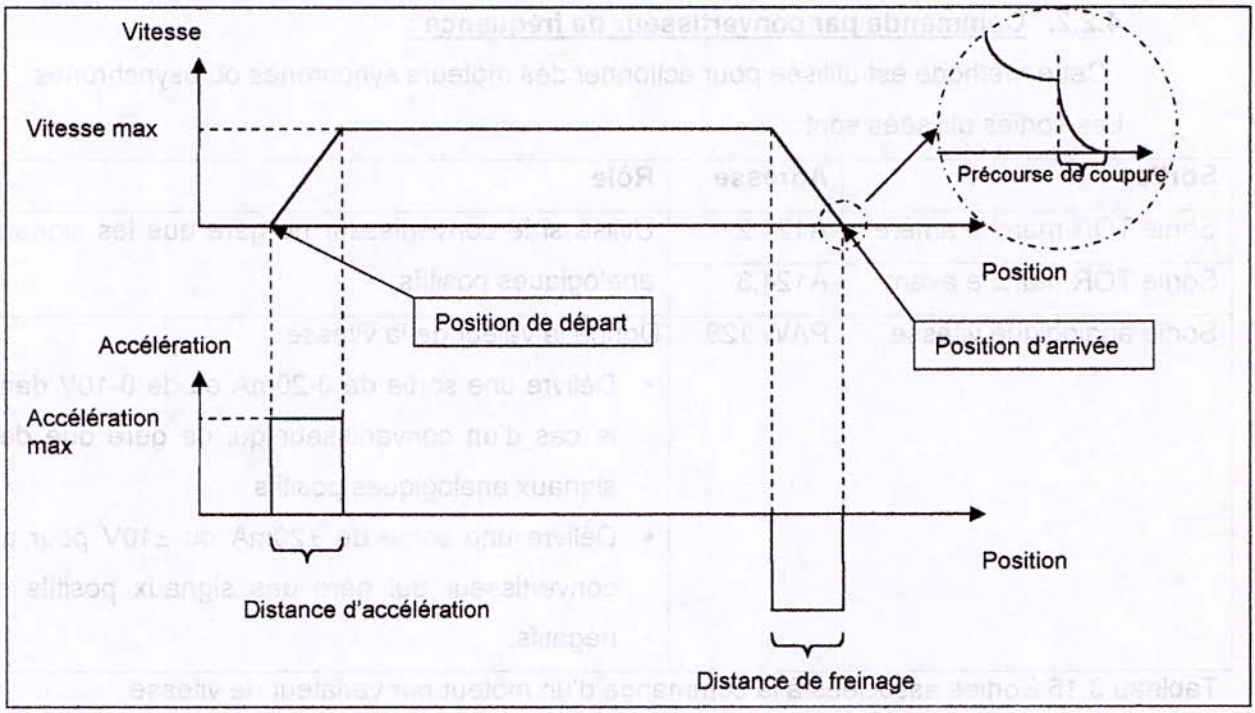


Figure 3.10 Profils de vitesses et d'accélération pour la fonction positionnement.

La vitesse maximale est définie dans le programme utilisateur.

La distance d'accélération et la distance de freinage sont à paramétrer avec *STEP 7*.

On remarque qu'il existe une précourse de coupure pour l'arrêt du moteur, ce paramètre est donné dans le programme utilisateur.

Valeurs sur les sorties :

Comme indiqué précédemment, deux configurations sont possibles pour la commande :

- 1 sortie analogique (pour la valeur de la vitesse) qui délivre 0-20mA ou 0-10V Avec deux sorties TOR (1 pour marche arrière et 1 pour marche avant).
- 1 sorties analogique (pour la valeur de la vitesse et le sens) qui délivre ±20mA ou ±10V.

Il faut savoir que la sortie analogique est donnée sous forme d'échelons. Ces échelons sont au nombre de 10 et ont la même largeur, mais possèdent des hauteurs différentes.

La hauteur des échelons est prédéfinie par la fonction intégrée positionnement.

La largeur est calculée de la manière suivante :

$$\text{Largeur de l'échelon} = \frac{\text{Distance d'accélération/freinage}}{10}$$

Cette valeur est tronquée de sa partie réelle pour être sûr que la distance d'accélération/de freinage parcourue effectivement ne soit jamais supérieure à la distance paramétrée.

La valeur analogique délivrée par la sortie se calcule de la manière suivante :

$$V_{Max} = \frac{10V}{256} \times (256 - BREAK)$$

ou :

$$I_{Max} = \frac{20mA}{256} \times (256 - BREAK)$$

La valeur de BREAK est donnée dans le SFB 39.

Chapitre 4

Application

Section

A

Circuits de simulation

Les circuits présentés ci-après ont été conçus pour pouvoir faire des simulations avec l'automate programmable S7-314 IFM sans avoir recourt au forçage de variables par l'intermédiaire d'un PC.

La CPU S7-314 IFM possède :

- 16 entrées TOR standard,
- 4 entrées TOR spéciales,
- 4 entrées analogiques compatibles avec ±10V et ±20mA,
- 16 sorties TOR standard,
- 1 sortie analogique compatible avec ±10V et ±20mA.

La simulation des entrées se fera en deux parties :

- une partie pour simuler les entrées TOR (0-24V),
- une partie pour simuler les entrées analogiques.

La manipulation de tensions étant plus facile, ce sont les entrées ±10V qui seront utilisées.

1. Simulation des entrées TOR :

Les caractéristiques des modules d'entrées montrent que les tensions à appliquer aux entrées TOR de la CPU S7-314 IFM sont de 0V et 24V en logique positive.
Le schéma de principe pour chaque entrée TOR est le suivant :

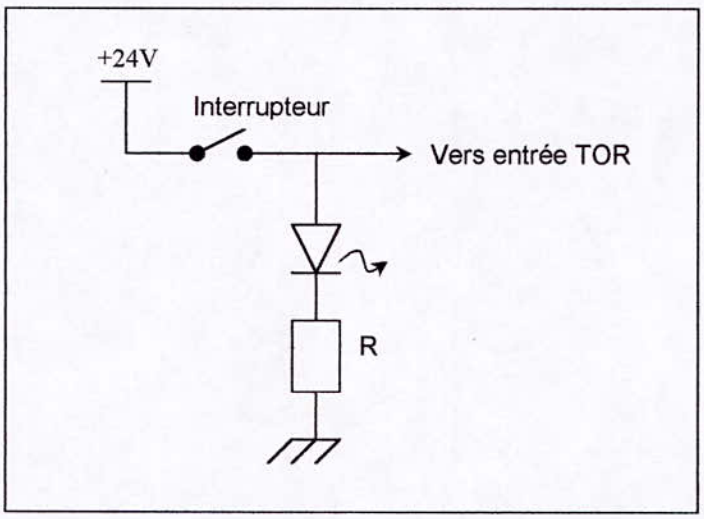


Figure 4.1 Schéma de principe pour la simulation d'une entrée TOR.

Le courant nécessaire aux LED utilisées pour s'allumer convenablement est d'environ 15mA.

Les résistances utilisées seront donc : $R = \frac{U}{I} = \frac{24}{0.015} = 1.6k\Omega$

Le courant total consommé par les éléments de la plaque est : $I_{plaque} = 15 \times 20 = 300mA$

Les tensions nécessaires au fonctionnement de la plaque de simulation des entrées TOR seront délivrées par l'alimentation de l'automate.

Au regard des caractéristiques techniques des modules du S7-314 IFM, on remarque que :

- l'automate consomme un courant nominal de 1A.
- l'alimentation du module TOR standard consomme un courant max de 40 mA.
- chaque entrée TOR standard consomme un courant nominal de 7 mA.
- chaque entrée TOR spéciale consomme un courant nominal de 6.5 mA.
- la sortie analogique délivre au maximum 20mA.

L'alimentation doit être en mesure de fournir un courant de :

$I = 1000 + 40 + 7 \times 16 + 6.5 \times 4 + 20 + 300 = 1498mA \approx 1.5A$

L'alimentation utilisée est une PS307-2A pouvant délivrer du 24V stabilisée avec un courant nominal de 2A, il est donc possible de l'utiliser pour alimenter la plaque de simulation des entrées TOR.

Les entrées spéciales servent à mettre en œuvre 4 types de fonctions intégrées :

- fréquencemètre.
- Compteur.
- 2 compteurs A et B.
- positionnement.

Le tableau suivant montre le rôle des différentes entrées spéciales pour chaque configuration de fonction intégrée:

	Fréquencemètre	Compteur	Compteur A/B	Positionnement
E126.0	Entrée du signal	Comptage	Comptage A	Codeur piste A
E126.1	Non utilisée	Décomptage	Décomptage A	Codeur piste B
E126.2	Non utilisée	Sens	Comptage B	Contact de référence
E126.3	Non utilisée	Marche/arrêt	Décomptage B	Non utilisée

Tableau 4.1 signification des entrées spéciales pour chaque fonction intégrées.

D'après ce tableau il a été retenu d'utiliser:

- 16 switchs ON/OFF pour les entrées standards
- 2 boutons poussoirs pour les entrées spéciales E126.0 et E126.1
- 2 switchs ON/OFF pour les entrées spéciales E126.2 et E126.3

2. Simulation des entrées analogiques :

Pour la simulation des entrées analogiques, ce sont les entrées en tension qui ont été choisies car les tensions sont plus faciles à manipuler que les courants.
Ces entrées fonctionnent pour une plage nominale de ±10V.

2.1. Fonctionnement des modules analogiques :

Sur le S7-314 IFM, les entrées et la sortie analogiques possèdent des convertisseurs analogique-numérique et numérique-analogique d'une résolution de 11bits+bit de signe.

Les images des valeurs analogiques sont représentées sur 16 bits, pour les modules possédant une résolution inférieure à 15bits+bit de signe, les bits de poids plus faible ne sont pas représentatifs.

Dans le cas du S7-314 IFM ce sont les 4 bits de poids plus faible qui ne sont pas représentatifs.

Le tableau qui suit représente l'étendue de mesure des entrées/sorties analogiques avec un codage sur 15 bits+bit de signe en complément à deux :

Unités décimales	Mot de données	Valeur en %	Tension de sortie (V)	Courant de sortie (mA)	Plage
32767	7FFF	118.515	Pas de valeur	Pas de valeur	Débordement en haut
32512	7F00	117.593			
32511	7EFF	117.589	11.759	23.52	Domaine de dépassement haut
27649	6C01	100.004	10.0004	20.0008	
27648	6C00	100	10	20	Etendue nominale
1	1	0.003617	361.7µV	723.4nA	
0	0	0	0	0	
-1	FFFF	-0.003617	-361.7µV	-723.4nA	
-27648	9400	-100	-10	-20	
-27649	93FF	-100.004	-10.0004	-20.0008	Domaine de dépassement bas
-32512	8100	-117.593	-11.759	-23.52	
-32513	80FF	-117.596	Pas de valeur	Pas de valeur	Débordement bas
-32768	8000	-118.519			

Tableau 4.2 Codage pour les E/S analogiques.

La CPU S7-314 IFM ne codant ses valeurs que sur 11bits+bit de signe, la première valeur significative positive est 0010H. La résolution est de :

$$\text{Res}_V = \frac{20V}{2^{12}} = 0{,}0048828125\text{V} = 4{,}88mV$$

ou

$$\text{Res}_I = \frac{40mA}{2^{12}} = 0{,}009765625\text{mA} = 9{,}77\mu A$$

2.2. Réalisations :

2.2.1. circuit de simulation analogique :

Les signaux utilisés pour les entrées analogiques ont été réalisés à partir d'un diviseur de tension, le schéma de principe est le suivant :

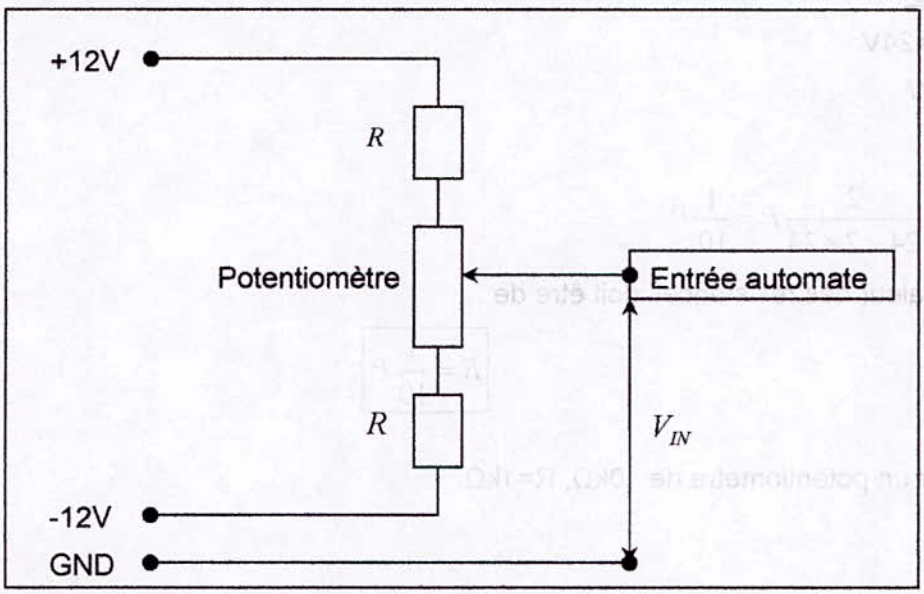


Figure 4.2 Schéma de principe des signaux du circuit de simulation des entrées analogiques. (Une entrée)

Ce montage sera répété 4 fois pour pouvoir attaquer les quatre entrées analogiques de la CPU S7-314 IFM.

Le choix d'une alimentation symétrique ±12V se justifie par la rareté des régulateurs de tension de 10V (7810 et 7910).

Pour un potentiomètre P, la valeur des résistances R doit être de $\frac{P}{10}$. En effet, la maille donne l'équation suivante :

$$V_{cc} = (2R+P)I \Rightarrow I = \frac{V_{cc}}{2R+P}$$

Pour avoir la plage désirée, les résistances doivent causer une chute de tension U telle que :

$$U = RI \Rightarrow U = R\frac{V_{cc}}{2R+P}$$

Ce qui donne :

$$R = \frac{U}{Vcc-2U}P$$

L'alimentation est de ±12V, la chute de tension sur chaque résistance pour atteindre une plage de ±10V est de 2V.

Donc :

Vcc=24V

U=2V

$$R = \frac{2}{24-2\times 24}P = \frac{1}{10}P$$

La valeur des résistances doit être de :

$$\boxed{R = \frac{1}{10}P}$$

Pour un potentiomètre de 10kΩ, R=1kΩ.

Problème :

Lors de l'achat des potentiomètres, il s'est avéré que leur précision était très faible (de l'ordre de 15%), ce qui a pour effet de changer notablement la plage de variation de la tension de sortie.

Solution :

La solution est donc de remplacer les résistances par des potentiomètres afin de pouvoir ajuster la tension de sortie selon la valeur réelle du potentiomètre principal.

Le schéma de principe devient alors :

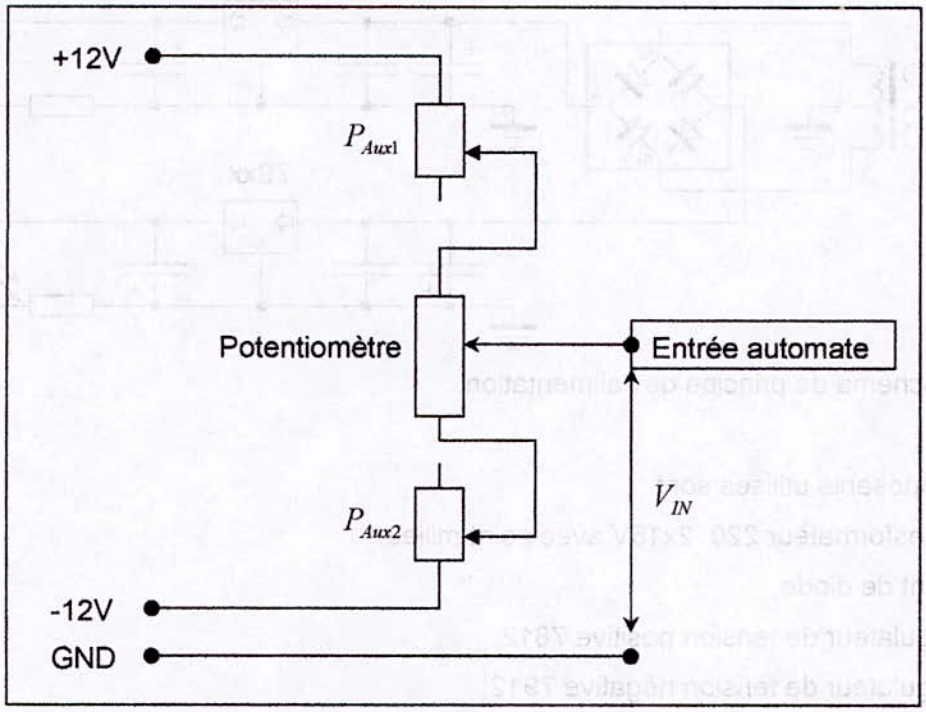


Figure 4.3 Schéma de principe des signaux de la plaque de simulation des entrées analogiques

Les potentiomètres P_{Aux1} et P_{Aux2} ont une valeur de 4.7kΩ, cela permet de faire varier la plage de sortie de ±5.8V à ±12V.

Le fait de dépasser légèrement la valeur de 10V sur les entrées analogiques de l'automate permet d'exploiter le déclenchement d'alarmes aux entrées.

Le courant total consommé pour ce montage est :

$$I_{total} = 4 \times I_{partiel} = 4 \times \frac{U}{R_{eq}} = 4 \times \frac{24}{10+4.7+4.7} = 4 \times 1.24 = 4.95mA \approx 5mA$$

$$I_{total} \approx 5mA$$

2.2.2. Alimentation :

Le circuit est alimenté par une simple alimentation symétrique de ±12V.

Le schéma de principe de l'alimentation est le suivant :

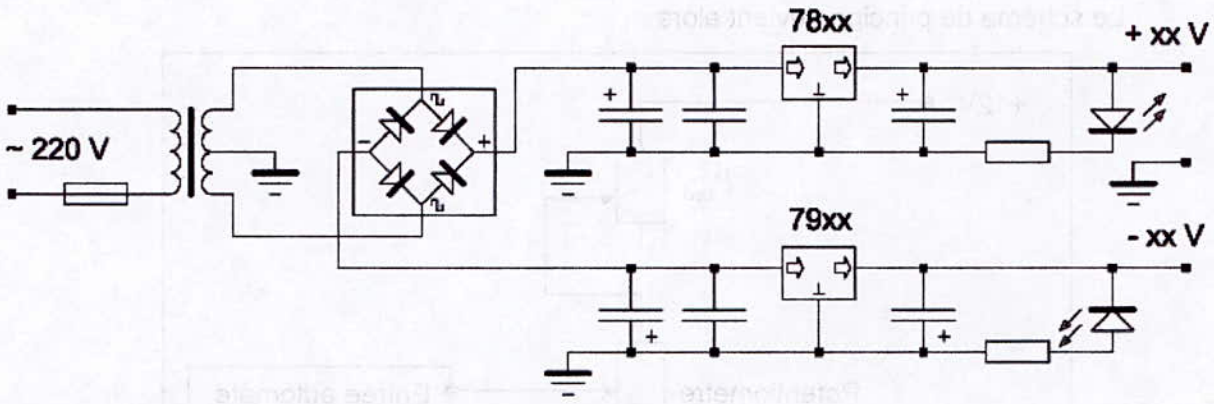


Figure 4.4 schéma de principe de l'alimentation.

Les composants utilisés sont :

- 1 transformateur 220 2x15V avec point milieu,
- 1 pont de diode,
- 1 régulateur de tension positive 7812,
- 1 régulateur de tension négative 7912,
- 2 capacités de 2200µF,
- 2 capacités de 100nF,
- 2 capacités de 100µF,
- 2 résistances de 1kΩ,
- 2 LED,
- 1 fusible de 125mA,
- 1 porte fusible,
- 1 interrupteur,
- 1 bornier 2 vis,
- 1 bornier 3 vis,
- 1 plaque d'époxy simple face d'environ 7cm x 14cm.

3. Montage de l'automate programmable S7-314 IFM :

Le montage de l'automate est des plus aisé puisque les modules d'entrées/sorties sont déjà intégrés.

Les éléments nécessaires pour effectuer un montage complet sont :

Qté	Désignation	Référence
1	Profilé support	**6ES7 390-1AE80-0AA0**
1	Alimentation PS 307 (PS) avec peigne de jonction (VK)	**6ES7 307-1EA00-0AA0**
1	CPU 314IFM	**6ES7 314-5AE03-0AB0**
1	Pile de sauvegarde	**6ES7 971-1AA00-0AB0**

Tableau 4.3 Eléments nécessaires pour le montage de la CPU S7-314 IFM

3.1. Câblage de l'alimentation et de la CPU :

Les étapes à suivre pour un bon câblage sont les suivantes :

Etape	Action
1	Ouvrir les portes frontales de l'alimentation et de la CPU
2	Desserrer le collier pour la décharge de traction de l'alimentation
3	Dénuder le câble secteur et le brancher à l'alimentation en respectant les couleurs (phase, neutre et terre)
4	Visser le collier de décharge de traction
5	Brancher le peigne de jonction entre l'alimentation et la CPU puis le visser correctement.
6	Vérifier si la sélection de tension sur l'alimentation est bien sur la bonne position (par défaut 230V). Si ce n'est pas le cas, il suffit d'enlever le cache de protection de la sélection avec un tournevis plat et de changer la position.

Tableau 4.4 Etapes pour le câblage de l'alimentation.

3.2. Câblage des modules d'entrées/sorties :

La figure suivante représente les modules intégrés de la CPU S7-314 IFM :

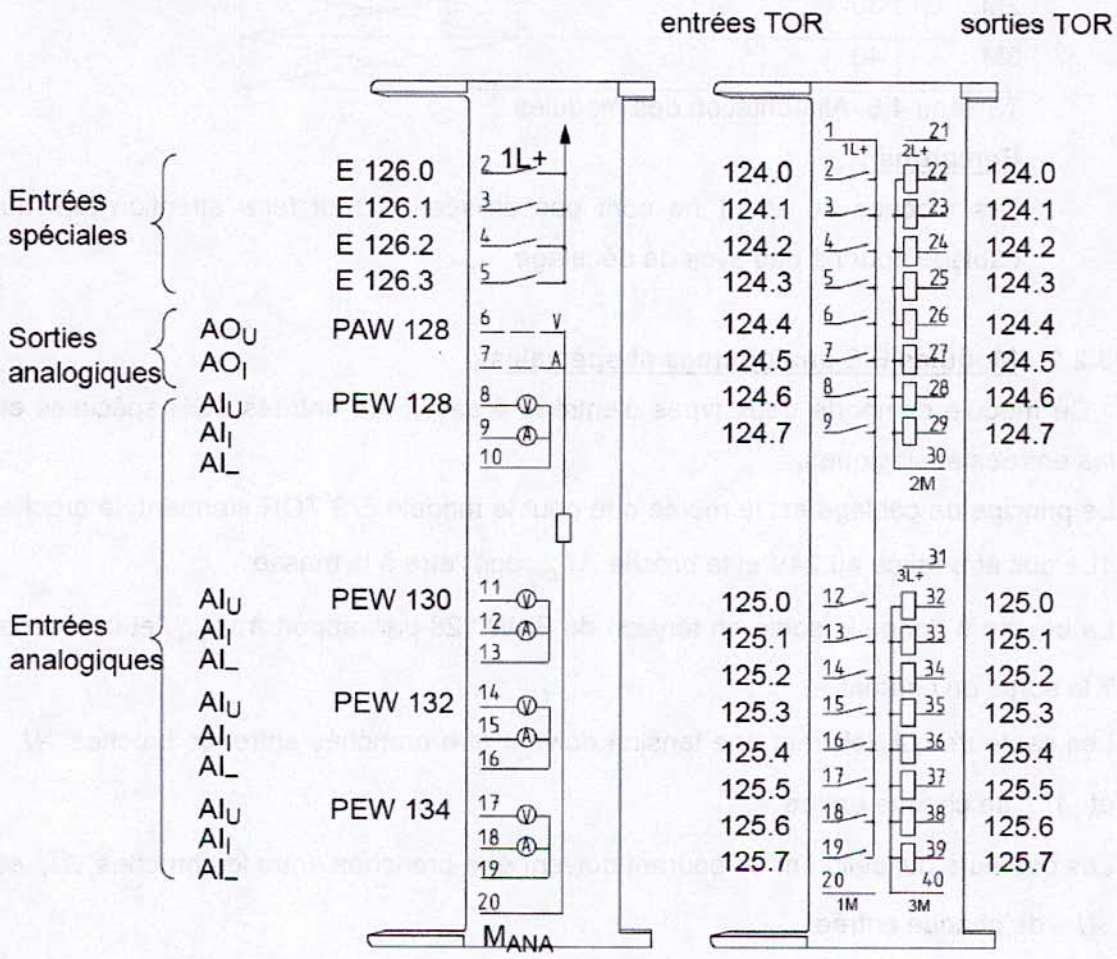


Figure 4.5. Modules d'entrées/sorties avec leurs adressages.

3.2.1. modules E/S TOR (module de droite) :

Le module d'entrées/sorties TOR est constitué d'un groupe 16 d'entrées reliées galvaniquement entre elles, et de deux groupes de 8 sorties reliées galvaniquement entre elles.

3.2.1.1. Alimentation des modules :

Le câblage consiste à relier les broches L au 24V, et les broches M à la masse. On peut ainsi avoir une isolation galvanique entre les modules et l'automate programmable.

Les entrées fonctionnent avec une logique positive 0/24V, les adresses sont indiquées sur la porte du module.

Au vu de la figure ci-dessus, le câblage de l'alimentation du module TOR sera :

Broche	Numéro	Reliée au 24V	Reliée à la masse
1L+	1		
2L+	21		
3L+	31		
1M	20		
2M	30		
3M	40		

Tableau 4.5. Alimentation des modules

Remarque :

Les broches 10 et 11 ne sont pas utilisées, il faut faire attention lors du câblage pour ne pas avoir de décalage.

3.2.2. Modules E/S analogiques et spéciales :

Ce module comporte deux types d'entrées à savoir les entrées TOR spéciales et les entrées analogiques.

Le principe de câblage est le même que pour le module E/S TOR standard, la broche 1L+ doit être reliée au 24V et la broche M_{ANA} doit l'être à la masse.

La broche 6 donne la sortie en tension de PAW 128 par rapport à M_{ANA}, et la broche 7 la sortie en courant.

Les capteurs qui délivrent une tension doivent être branchés entre les broches AI_U et $AI\_$ de chaque entrée.

Les capteurs qui délivrent un courant doivent être branchés entre les broches AI_I et $AI\_$ de chaque entrée.

Remarque :

Le constructeur conseil de relier les broches $AI\_$ à M_{ANA}.

3.3. Mise en marche du matériel :

Le tableau suivant regroupe les étapes à suivre pour la mise en service de la CPU :

Etape	Action	Résultat attendu
1	Relier la CPU au PC par l'intermédiaire du câble de liaison. Vérifier que la CPU est sur la position STOP et que la porte frontale est fermée.	
2	Brancher l'alimentation au secteur puis mettre l'interrupteur sur la position marche	Sur l'alimentation : la LED CC24V s'allume. Sur la CPU : toutes les LED s'allument brièvement, les LED SF, les LED BATF et la LED CC5V restent allumées. La LED STOP clignote pendant 3 s, puis reste allumée.
3	Insérer la pile de sauvegarde : – Branchez le connecteur de la pile de sauvegarde dans la prise prévue à cet effet, dans le compartiment de la CPU. L'encoche du connecteur doit se trouver à gauche. – Placez la pile dans le compartiment de la CPU. – Fermez la porte frontale de la CPU	La LED BATF s'éteint, suivie peu après par la LED SF.
4	Sur le PC lancer SIMATIC MANAGER	La fenêtre du gestionnaire s'ouvre.
5	Effectuer un effacement général de la CPU : • Mettre le sélecteur de mode sur la position MRES jusqu'à ce que la LED STOP s'allume pour la deuxième fois et reste allumée (opération d'environ 3 secs). • Remettre le sélecteur sur la position MRES dans un délai de 3 secondes après la première opération. La LED STOP commence à clignoter rapidement (la CPU effectue un effacement général). Lorsque la LED reste allumée, la CPU a fini son effacement.	

Tableau 4.6. Mise en marche de la CPU.

3.4. Branchement des circuits de simulation :

3.4.1. Circuit de simulation des entrées TOR :

Le circuit de simulation des entrées TOR doit être branché avant la mise sous tension de la CPU.

Le circuit se branche à l'aide de borniers répartis en trois groupes de respectivement 20, 2 et 4 sorties.

Les 20 sorties de gauche doivent être branchées sur le module d'entrées TOR standard, le bornier d'extrême gauche correspondant à l'entrée E124.0.

Les 4 sorties d'extrême droite doivent être branchées sur les entrées TOR spéciales E126.0 à E126.3.

Les borniers du centre doivent être reliés à l'alimentation de la CPU en respectant la polarité qui est indiquée.

3.4.2. Circuit de simulation des entrées analogiques :

Le circuit de simulation des entrées analogiques se compose de deux parties :

- L'alimentation,
- Et le circuit potentiométrique délivrant les 4 signaux.

Les signaux de sortie de ce montage attaquent directement les entrées analogiques de l'automate.

Il y a 5 borniers de sortie :

- 4 pour les tensions de sortie,
- 1 pour la masse.

Pour le branchement, il faut tout d'abord relier l'alimentation au circuit potentiométrique en faisant attention aux polarités.

Brancher le secteur de l'alimentation et mettre l'interrupteur en position ON.

Avec l'aide d'un voltmètre, régler les potentiomètres secondaires pour avoir une plage de sortie de ±10V.

Une fois cette opération effectuée, éteindre l'alimentation pour brancher les 4 signaux de sortie sur les broches AI_U, et la masse du montage avec la masse de l'automate.

Section

B

Exemples de programmation

1. Réalisation d'un GRAFCET

Les automatismes industriels sont, dans la plupart des cas, des systèmes séquentiels. On peut alors les traduire sous forme de GRAFCET ou de réseaux de Pétri.
Le type de programmation décrit ici reprend l'esprit des circuits avec bascules.

Chaque étape est associée à une bascule, et à des actions.
STEP 7 offre la possibilité d'utiliser des bascules RS et SR. Le choix de la bascule à utiliser dépendra du processus à automatiser. La différence entre les deux types de bascules étant respectivement la priorité de S ou de R.

Dans tous les programmes décrits, ce sont des bascules SR qui sont choisies pour avoir une priorité à la désactivation.

Avant d'écrire le programme, il est tout d'abord nécessaire de créer un projet, de configurer le matériel et de définir la table des mnémoniques.

Dans tous les exemples cités plus tard, la configuration matérielle de base est :

- CPU S7-314 IFM 6ES7 314-5AE03-0A0B version matérielle 1.2
- Alimentation PS 307 2A 6ES7 307-1BA00-0AA0

Pour mieux gérer un système, il est conseillé de le subdiviser en plusieurs sous-systèmes. Ceci permet de mieux tester et débugger les programmes.

Ce choix se traduit par la réalisation de chaque partie du système soit dans une subroutine soit dans une fonction.

Pour réaliser un GRAFCET, il est préférable d'utiliser une subroutine qui, à la différence d'une fonction, possède des données statiques contenues dans le bloc de données (DB) associé. Ceci permet de sauvegarder des variables pour une utilisation dans le cycle qui suit.

Remarque 1 :

Il est primordial de noter que pour chaque bascule doivent être associées 2 variables :

- l'une pour l'état précédent de la bascule,
- et l'autre pour l'état actuel de la bascule.

Ceci est nécessaire car l'exécution du programme est séquentielle : le changement d'état d'une bascule ne doit pas influencer les autres bascules durant le même cycle.

Ce n'est qu'à la fin de chaque cycle que sont actualisés les états précédents avec les nouvelles valeurs contenues dans les états actuels.

Remarque 2 :

Lors du démarrage de l'automate, les valeurs contenues dans la RAM peuvent être à 1 ou à 0 selon l'historique de la CPU (démarrage à chaud, etc...)

Il est donc nécessaire d'initialiser les bascules à 1 ou à 0 pour permettre un fonctionnement correct du GRAFCET.

Ceci est possible en utilisant l'OB100 qui est le bloc d'organisation exécuté lors de la mise en RUN de la CPU.

2. Exemple 1 :

L'exemple suivant montre les bases de la programmation pour la réalisation d'un GRAFCET. On se propose de réaliser le GRAFCET suivant :

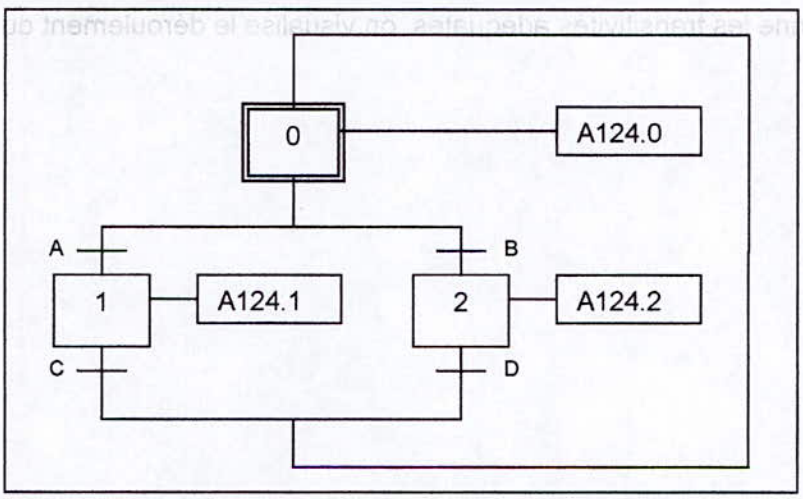


Figure 4.6 GRAFCET à réaliser.

On suppose que les signaux A, B, C et D sont respectivement les entrées : E124.0, E124.1, E124.2 et E124.3. Il faut donc les définir dans la table des mnémoniques.

Pour pouvoir réaliser ce GRAFCET, il est nécessaire d'utiliser 3 bascules dont les états sont : X0, X1 et X2.

Les signaux d'activation et de désactivation de chaque bascule sont :

$$\begin{cases} S_0 = X_1.C + X_2.D \\ R_0 = X_0.(A+B) \end{cases}$$

$$\begin{cases} S_1 = X_0.A \\ R_1 = X_1.C \end{cases}$$

$$\begin{cases} S_2 = X_0.B \\ R_2 = X_2.D \end{cases}$$

Pour ce premier exemple, nous utiliserons le bloc principal de programmation : OB1.

Puisqu'il faut deux variables par bascules, il faudra en prévoir 6 pour cet exemple. Il est préférable de définir toutes ces variables dans la table des mnémoniques pour pouvoir ainsi manipuler des variables plus parlantes.

Pour que les bascules soient initialisées au démarrage de la CPU, on utilise l'OB100
Il faut que x0 et xp0 soient à 1 et les autres à 0 au début du programme sachant que ces valeurs sont dans les bits de mémento M0.0 à M0.5.
La valeur à envoyer à l'octet de mémento MB0 est donc 2#00000011 ou 16#3

Après chargement du programme dans la CPU, on met le commutateur sur le mode RUN.
La sortie A124.0 s'allume, c'est le signe d'une bonne initialisation au niveau de l'OB100.
Lorsque l'on donne les transitivités adéquates, on visualise le déroulement du GRAFCET.

Propriétés de la table des mnémoniques

Nom :	Mnémoniques
Commentaire :	
Date de création :	28/05/04 19:46:11
Dernière modification :	19/06/04 08:09:45
Dernier filtre sélectionné :	Tous les mnémoniques
Nombre de mnémoniques :	11/11
Dernier tri :	Mnémonique ordre croissant

Etat	Mnémonique	Opérande	Type de données	Commentaire
	A	E 124.0	BOOL	
	B	E 124.1	BOOL	
	C	E 124.2	BOOL	
	COMPLETE RESTART	OB 100	OB 100	Complete Restart
	D	E 124.3	BOOL	
	x0	M 0.0	BOOL	Etat actuel de la bascule 0
	x1	M 0.2	BOOL	Etat actuel de la bascule 1
	x2	M 0.4	BOOL	Etat actuel de la bascule 2
	xp0	M 0.1	BOOL	Etat du cycle précédent de la bascule 0
	xp1	M 0.3	BOOL	Etat du cycle précédent de la bascule 1
	xp2	M 0.5	BOOL	Etat du cycle précédent de la bascule 2

OB100 - <offline>

"COMPLETE RESTART" Complete Restart
Nom : **Famille :**
Auteur : **Version** : 0.1
Version de bloc : 2
Horodatage Code : 19/06/04 07:17:22
Interface : 15/02/96 16:51:10
Longueur (bloc/code /données locales) : 00126 00014 00020

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB100_EV_CLASS	Byte	0.0		16#13, Event class 1, Entering event state, Event logged in diagnostic buffer
OB100_STRTUP	Byte	1.0		16#81/82/83/84 Method of startup
OB100_PRIORITY	Byte	2.0		Priority of OB Execution
OB100_OB_NUMBR	Byte	3.0		100 (Organization block 100, OB100)
OB100_RESERVED_1	Byte	4.0		Reserved for system
OB100_RESERVED_2	Byte	5.0		Reserved for system
OB100_STOP	Word	6.0		Event that caused CPU to stop (16#4xxx)
OB100_STRT_INFO	DWord	8.0		Information on how system started
OB100_DATE_TIME	Date_And_Time	12.0		Date and time OB100 started

Bloc : OB100 "Complete Restart"

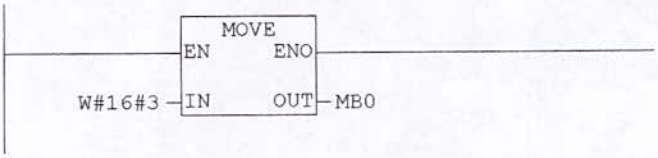
Réseau : 1 Mise à 1 d'un bit.

Ce réseau est écrit en LIST car il n'y a pas d'équivalent en CONT.
Cette mise à 1 peut servir si on a besoin d'une constante booléenne à 1 ou à 0.

```
      SET
      =     M    255.7
```

Réseau : 2 Initialisation des variables.

Les états des bascules doivent être initialiser avecdes valeurs nulles sauf
pour
celles contenant les états de X0.
Donc: le mémento MB0 doit contenir 2#000011 qui équivaut à 3.



OB1 - <offline>

""

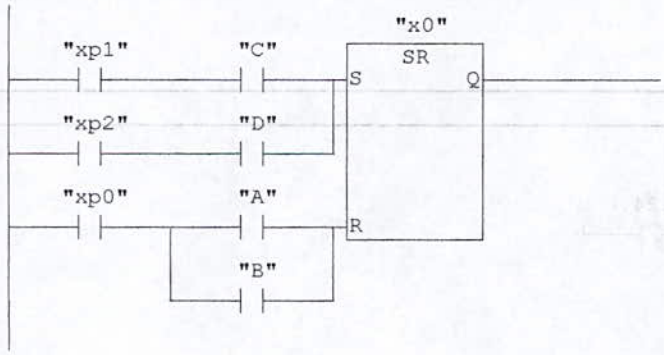
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 19/06/04 07:13:54
Interface : 15/02/96 16:51:12
Longueur (bloc/code /données locales) : 00212 00084 00020

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB1_EV_CLASS	Byte	0.0		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1	Byte	1.0		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
OB1_PRIORITY	Byte	2.0		Priority of OB Execution
OB1_OB_NUMBR	Byte	3.0		1 (Organization block 1, OB1)
OB1_RESERVED_1	Byte	4.0		Reserved for system
OB1_RESERVED_2	Byte	5.0		Reserved for system
OB1_PREV_CYCLE	Int	6.0		Cycle time of previous OB1 scan (milliseconds)
OB1_MIN_CYCLE	Int	8.0		Minimum cycle time of OB1 (milliseconds)
OB1_MAX_CYCLE	Int	10.0		Maximum cycle time of OB1 (milliseconds)
OB1_DATE_TIME	Date_And_Time	12.0		Date and time OB1 started

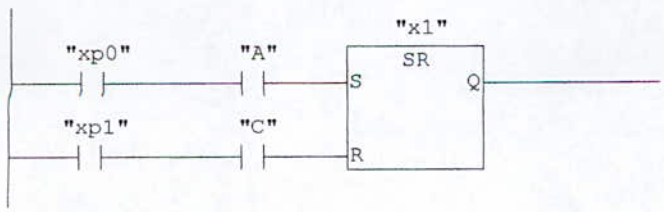
Bloc : OB1 "Main Program Sweep (Cycle)"

Réseau : 1 Etape 1.

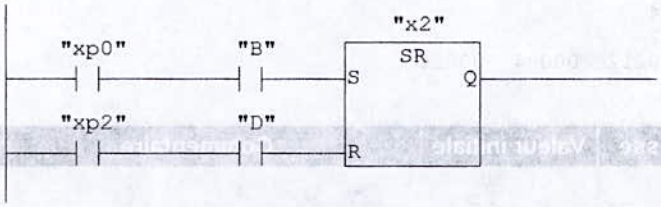
La branche du SET représente (XP1 ET C) OU (XP2 ET D)
et celle du RESET XP0 ET (A OU B)



Réseau : 2 Etape 1.

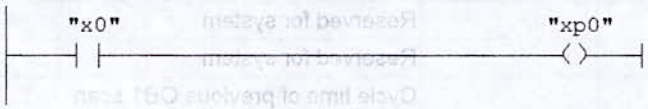


Réseau : 3 Etape 2.

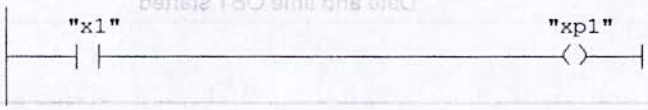


Réseau : 4 Actualisation.

Les réseaux de 4 à 6 actualisent les variables contenant les états du cycle précédent par les nouvelles valeurs.



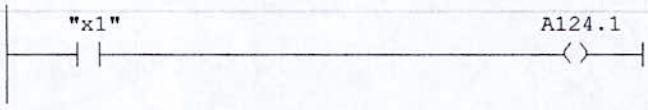
Réseau : 5



Réseau : 6

Réseau : 7 Action 1.

Réseau : 8 Action 2.



Réseau : 9 Action 3.

3. Chariot:

Cet exemple a pour but de réaliser un GRAFCET avec un bloc fonctionnel, utiliser des temporisations et des compteurs.

Un chariot se déplace sur des rails possédant 4 détecteurs de position A, B, C et D.
Ce chariot possède deux moteurs pour assurer son déplacement, et un signal d'entrée pour commander le sens de rotation des moteurs.

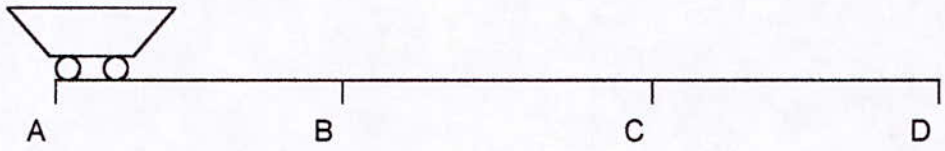


Figure 4.7 Schéma de principe pour le déplacement du chariot.

Fonctionnement :

Lorsque l'opérateur appuie sur un bouton Marche et que le chariot se trouve à la position A, celui-ci démarre le moteur M1.
Arrivé au point B, ce sont les deux moteurs qui fonctionnent.
Arrivé au point C le moteur M1 s'arrête et M2 continue à fonctionner.
Arrivé au point D plus aucun moteur ne fonctionne, et une temporisation T0 est lancée.

Lorsque la temporisation T0 est terminée, le chariot revient en sens inverse jusqu'au point A en utilisant le moteur M2, attend pendant un temps T1 puis refait le cycle précédent.
Ce cycle d'opérations est répété 2 fois.

Si au moment du démarrage, le chariot ne se trouve pas au point A, le moteur M1 démarre vers le point A (sens négatif) et attend une confirmation avec le bouton marche pour exécuter les trois cycles.

Ce cahier des charges est traduit en un GRAFCET:

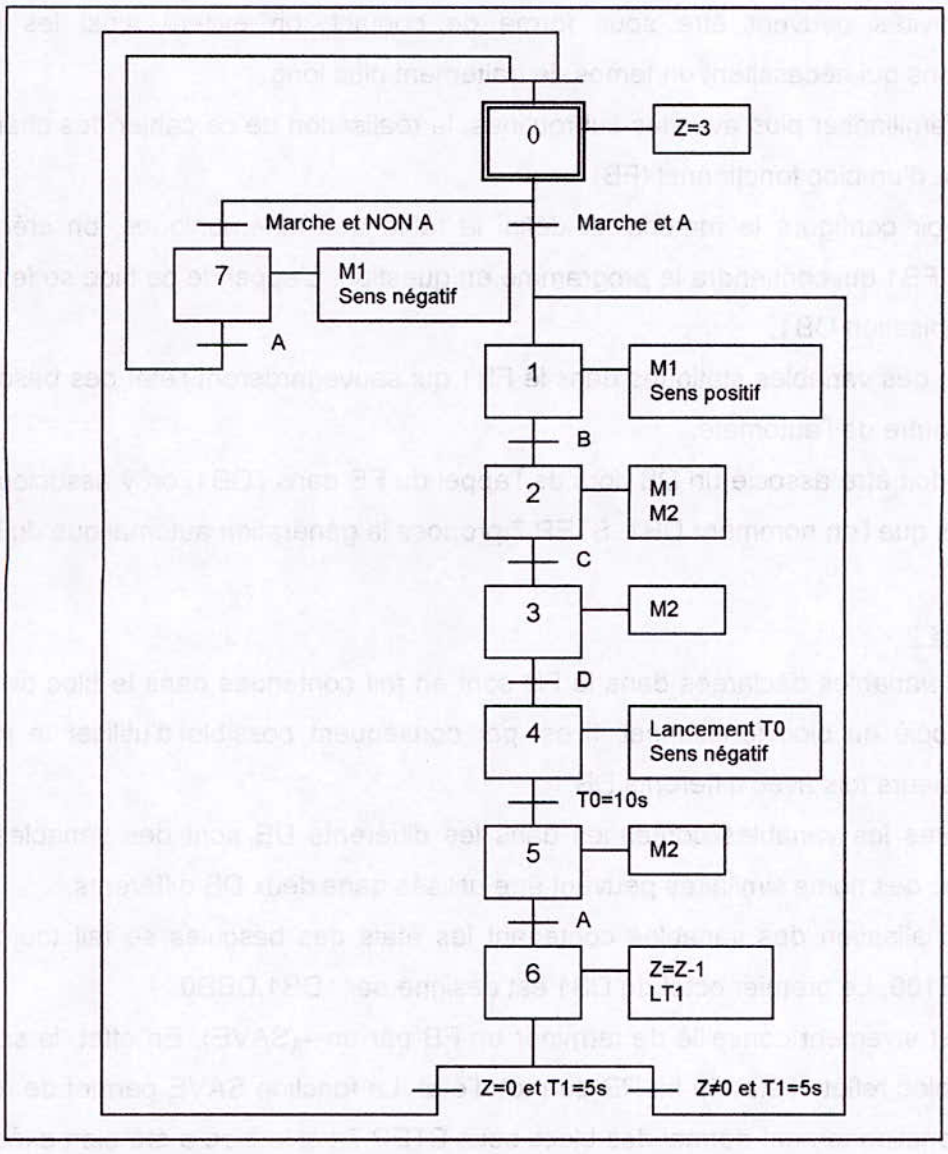


Figure 4.8. GRAFCET décrivant le fonctionnement du chariot

Le démarrage des moteurs est une combinaison en OU, il faudra deux contacts en parallèle.
La fin des temporisations doit engendrer un front montant pour désactiver l'étape en cours et activer l'étape qui suit.
Pour cela, les temporisations utilisées sont de type impulsion. Elles s'activent avec l'étape en cours et se désactivent à la fin du temps programmé ou si l'étape se désactive.
Le contact utilisé au niveau des bascules est un contact à ouverture pour créer le front montant à la fin du temps programmé.

Un contact peut donner l'état d'un compteur :

- Il est activé lorsque la valeur du compteur est différente de zéro.
- Et désactivé lorsque la valeur du compteur est nulle.

Les réceptivités peuvent être sous forme de contact, on évitera ainsi les blocs de comparaisons qui nécessitent un temps de traitement plus long.

Pour se familiariser plus avec les subroutines, la réalisation de ce cahier des charges sera faite à l'aide d'un bloc fonctionnel (FB).

Après avoir configuré le matériel et défini la table des mnémoniques, on crée un bloc fonctionnel FB1 qui contiendra le programme en question. L'appel de ce bloc se fera dans le bloc d'organisation OB1.

On définit des variables statiques dans le FB1 qui sauvegarderont l'état des bascules d'un cycle à un autre de l'automate.

A un FB doit être associé un DB, lors de l'appel du FB dans l'OB1, on y associera un bloc de données que l'on nommera DB1. STEP 7 propose la génération automatique du DB.

Remarques :

- Les variables déclarées dans le FB sont en fait contenues dans le bloc de données associé au bloc fonctionnel. Il est par conséquent possible d'utiliser le même FB plusieurs fois avec différents DB.
- Toutes les variables contenues dans les différents DB sont des variables locales, donc des noms similaires peuvent être utilisés dans deux DB différents.
- L'initialisation des variables contenant les états des bascules se fait toujours avec l'OB100. Le premier octet du DB1 est désigné par : DB1.DBB0.
- Il est vivement conseillé de terminer un FB par un –(SAVE). En effet, la sortie ENO du bloc reflète l'état du bit RB du mot d'état. La fonction SAVE permet de reproduire le fonctionnement normal des blocs sous STEP 7 : si le bloc a été bien exécuté alors ENO se met à 1.

Propriétés de la table des mnémoniques

Nom :	Mnémoniques
Commentaire :	
Date de création :	15/05/04 20:27:27
Dernière modification :	19/06/04 11:28:02
Dernier filtre sélectionné :	Tous les mnémoniques
Nombre de mnémoniques :	20/20
Dernier tri :	Opérande ordre croissant

Etat	Mnémonique	Opérande	Type de données	Commentaire
	Etape0	A 124.0	BOOL	
	Etape1	A 124.1	BOOL	
	Etape2	A 124.2	BOOL	
	Etape3	A 124.3	BOOL	
	Etape4	A 124.4	BOOL	
	Etape5	A 124.5	BOOL	
	Etape6	A 124.6	BOOL	
	Etape7	A 124.7	BOOL	
	M1	A 125.0	BOOL	Sortie de commande du moteur M1
	M2	A 125.1	BOOL	Sortie de commande du moteur M2
	TEMPO1	A 125.2	BOOL	Sortie indiquant l'état de la première temporisation
	TEMPO2	A 125.3	BOOL	Sortie indiquant l'état de la seconde temporisation
	SENS	A 125.4	BOOL	Sortie commandant le sens de rotation des moteurs
	FB1OK	A 125.5	BOOL	Sortie signalant le bon fonctionnement du bloc de fonction
	Marche	E 124.0	BOOL	Bouton de mise en marche du système
	A	E 124.1	BOOL	Détecteur A
	B	E 124.2	BOOL	Détecteur B
	C	E 124.3	BOOL	Détecteur C
	D	E 124.4	BOOL	Détecteur D
	COMPLETE RESTART	OB 100	OB 100	Complete Restart

OB100 - <offline>

"COMPLETE RESTART" Complete Restart
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 19/06/04 07:20:19
Interface : 15/02/96 16:51:10
Longueur (bloc/code /données locales) : 00130 00018 00020

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB100_EV_CLASS	Byte	0.0		16#13, Event class 1, Entering event state, Event logged in diagnostic buffer
OB100_STRTUP	Byte	1.0		16#81/82/83/84 Method of startup
OB100_PRIORITY	Byte	2.0		Priority of OB Execution
OB100_OB_NUMBR	Byte	3.0		100 (Organization block 100, OB100)
OB100_RESERVED_1	Byte	4.0		Reserved for system
OB100_RESERVED_2	Byte	5.0		Reserved for system
OB100_STOP	Word	6.0		Event that caused CPU to stop (16#4xxx)
OB100_STRT_INFO	DWord	8.0		Information on how system started
OB100_DATE_TIME	Date_And_Time	12.0		Date and time OB100 started

Bloc : OB100 "Complete Restart"

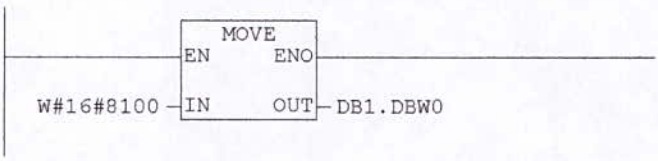
Réseau : 1

Mise à 1 d'un bit qui le restera pendant le déroulement du programme.

```
      SET
      =     M    255.7
```

Réseau : 2

Initialisation des variables statiques du bloc de données associé au bloc fonctionnel.
Le mot à initialiser est le numéro 0 donc DBW0, qui se trouve dans le bloc de données DB1, d'où DB1.DBW0
La valeur qu'il doit contenir est écrite sous format héxadécimal.



OB1 - <offline>

""

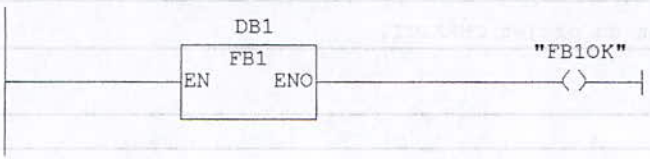
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 12/06/04 09:05:58
Interface : 15/02/96 16:51:12
Longueur (bloc/code /données locales) : 00150 00038 00026

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB1_EV_CLASS	Byte	0.0		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1	Byte	1.0		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
OB1_PRIORITY	Byte	2.0		Priority of OB Execution
OB1_OB_NUMBR	Byte	3.0		1 (Organization block 1, OB1)
OB1_RESERVED_1	Byte	4.0		Reserved for system
OB1_RESERVED_2	Byte	5.0		Reserved for system
OB1_PREV_CYCLE	Int	6.0		Cycle time of previous OB1 scan (milliseconds)
OB1_MIN_CYCLE	Int	8.0		Minimum cycle time of OB1 (milliseconds)
OB1_MAX_CYCLE	Int	10.0		Maximum cycle time of OB1 (milliseconds)
OB1_DATE_TIME	Date_And_Time	12.0		Date and time OB1 started

Bloc : OB1 "Main Program Sweep (Cycle)"

Réseau : 1 Sortie signalant le bon fonctionnement du bloc de fonction

Appel du bloc fonctionnel FB1 en utilisant le bloc de données DB1.
La bobine de fin est un test qui permet de visualiser le bon fonctionnement du bloc fonctionnel.



FB1 - <offline>

""

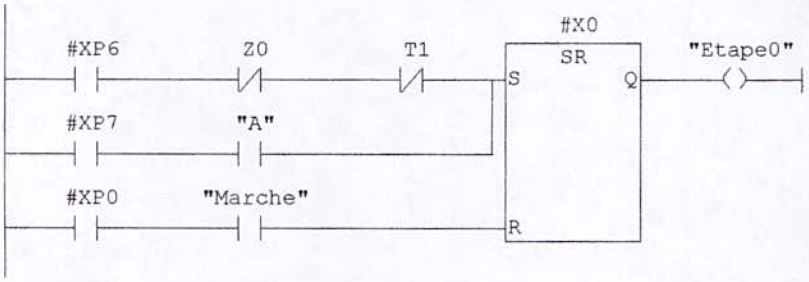
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 19/06/04 10:30:46
Interface : 12/06/04 08:44:37
Longueur (bloc/code /données locales) : 00566 00396 00000

Nom	Type de données	Adresse	Valeur initiale	Commentaire
IN		0.0		
OUT		0.0		
IN_OUT		0.0		
STAT		0.0		
X0	Bool	0.0	FALSE	Valeur actuelle de X0
X1	Bool	0.1	FALSE	Valeur actuelle de X1
X2	Bool	0.2	FALSE	Valeur actuelle de X2
X3	Bool	0.3	FALSE	Valeur actuelle de X3
X4	Bool	0.4	FALSE	Valeur actuelle de X4
X5	Bool	0.5	FALSE	Valeur actuelle de X5
X6	Bool	0.6	FALSE	Valeur actuelle de X6
XP0	Bool	0.7	FALSE	Valeur du cycle de précédent de X0
XP1	Bool	1.0	FALSE	Valeur du cycle de précédent de X1
XP2	Bool	1.1	FALSE	Valeur du cycle de précédent de X2
XP3	Bool	1.2	FALSE	Valeur du cycle de précédent de X3
XP4	Bool	1.3	FALSE	Valeur du cycle de précédent de X4
XP5	Bool	1.4	FALSE	Valeur du cycle de précédent de X5
XP6	Bool	1.5	FALSE	Valeur du cycle de précédent de X6
X7	Bool	1.6	FALSE	Valeur actuelle de X7
XP7	Bool	1.7	FALSE	Valeur du cycle de précédent de X7
TEMP		0.0		

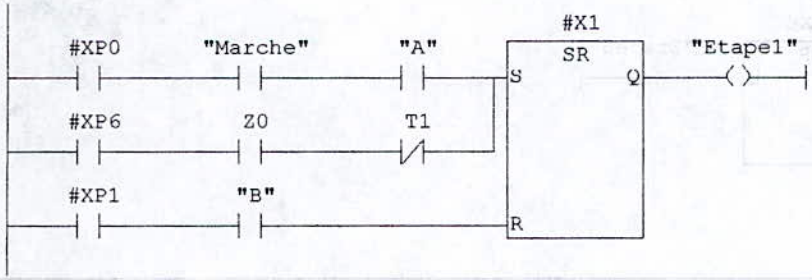
Bloc : FB1

Bloc fonctionnel permettant la réalisation du projet CHARIOT.

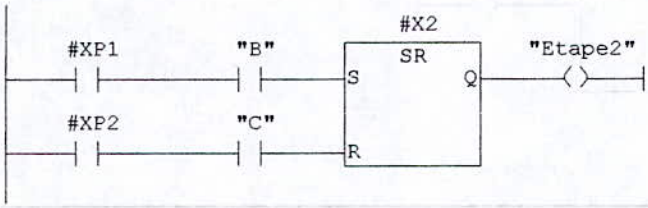
Réseau : 1 Etape 0.



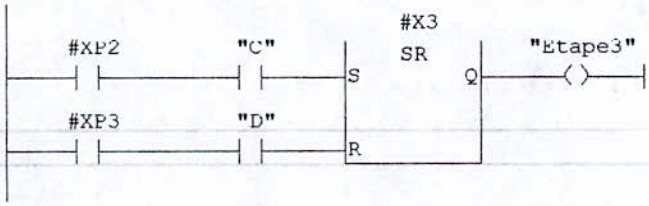
Réseau : 2 Etape 1.



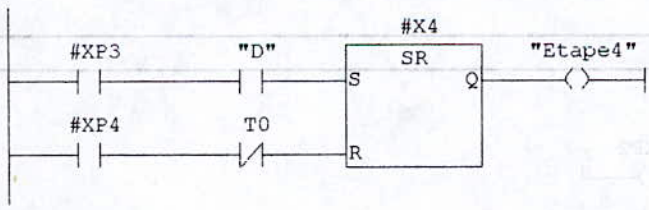
Réseau : 3 Etape 2.



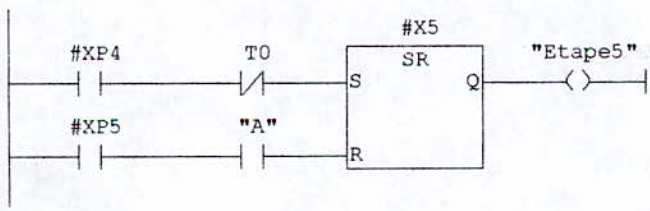
Réseau : 4 Etape 3.



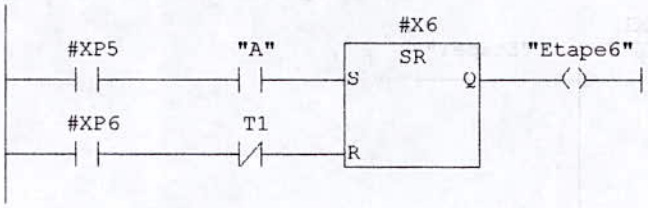
Réseau : 5 Etape 4.



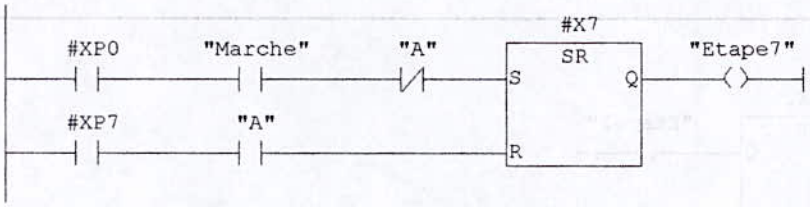
Réseau : 6 Etape 5.



Réseau : 7 Etape 6.



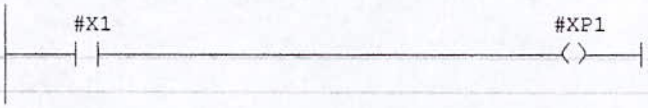
Réseau : 8



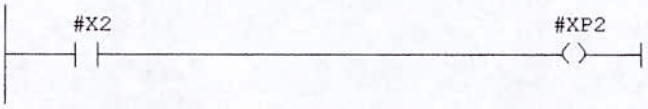
Réseau : 9

Les réseaux de 9 à 16 actualisent les valeurs des variables statiques du DB1.

Réseau : 10

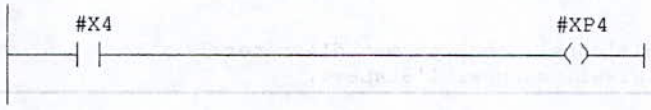


Réseau : 11



Réseau : 12

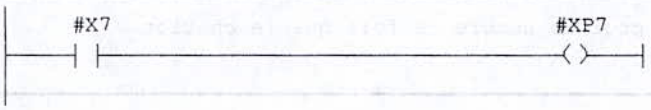
Réseau : 13



Réseau : 14

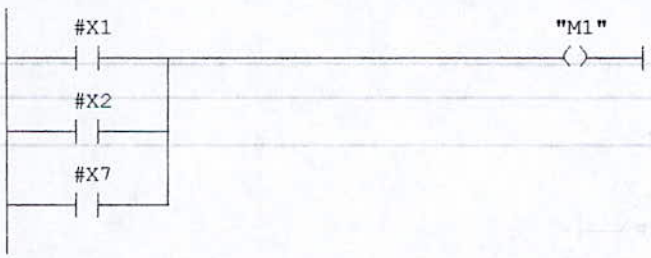
Réseau : 15

Réseau : 16



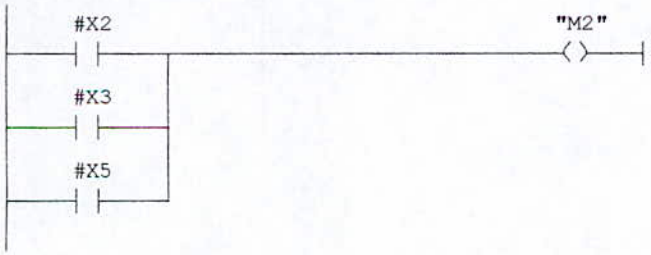
Réseau : 17 Activation du moteur M1.

Le moteur M1 est en route pendant les étapes 1 et 2.



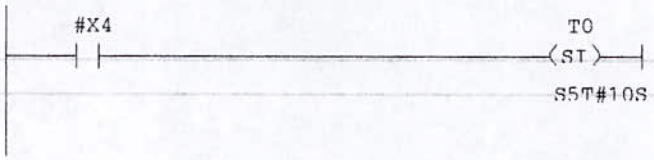
Réseau : 18 Activation du moteur M2.

Le moteur M2 est en route pendant les étapes 2, 3 et 5.



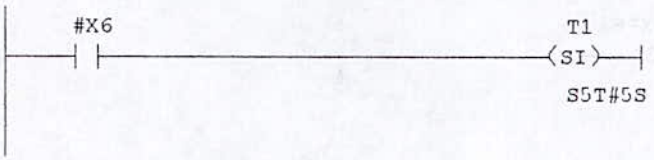
Réseau : 19

Lancement de la temporisation de 10 secondes pour permettre au chariot de rester
au point D pendant ce laps de temps.
La temporisation utilisée est de type impulsionnel, ceci permet d'arrêter la temporisation si un évenement quelconque faisait stopper l'étape 4.



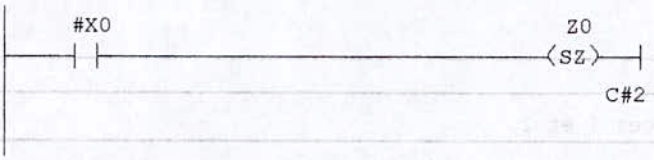
Réseau : 20

Lancement de la temporisation de 5 secondes pour permettre au chariot de rester
au point A pendant ce laps de temps.
La temporisation utilisée est de type impulsionnel, ceci permet d'arrêter la temporisation si un évenement quelconque faisait stopper l'étape 6.



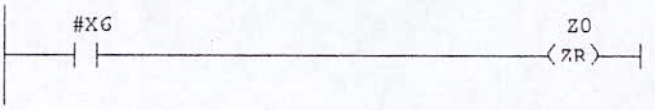
Réseau : 21

Initialisation du compteur Z0 à l'étape 0 pour le nombre de fois que le chariot doit exécuter le cycle.
La valeur est écrite sous format BCD.

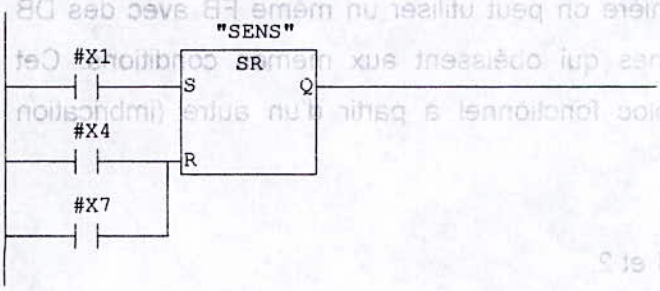


Réseau : 22

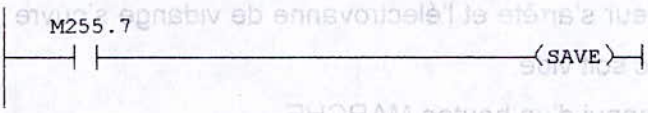
Décrémentation du compteur Z0 à l'étape 6.



Réseau : 23 Sortie commandant le sens de rotation des moteurs
Mise à 1 ou à 0 du sens de rotation. Lorsque SENS est à 1 le chariot se déplace de A vers D. Lorsque SENS est à 0 le chariot se déplace de D vers A.



Réseau : 24
Mise à 1 du bit du mot d'état RB qui sera l'image de la sortie ENO lors de l'appel du bloc fonctionnel dans le bloc d'organisation OB1.



4. Mélangeur :

L'exemple suivant montre de quelle manière on peut utiliser un même FB avec des DB différents afin de commander deux vannes qui obéissent aux mêmes conditions. Cet exemple comprend aussi un appel de bloc fonctionnel à partir d'un autre (imbrication d'appel).

Fonctionnement :

Le système doit mélanger deux produits 1 et 2.

Le bac de mélange doit se remplir avec le produit A jusqu'à un niveau intermédiaire, puis avec le produit B jusqu'au remplissage total.

Lorsque le bac est rempli, un moteur est enclenché pour effectuer le mélange pendant un temps T=10s.

Une fois la temporisation terminée, le moteur s'arrête et l'électrovanne de vidange s'ouvre pour vider le produit fini jusqu'à ce que le bac soit vide.

Toutes ces étapes sont démarrées lors de l'appui d'un bouton MARCHE.

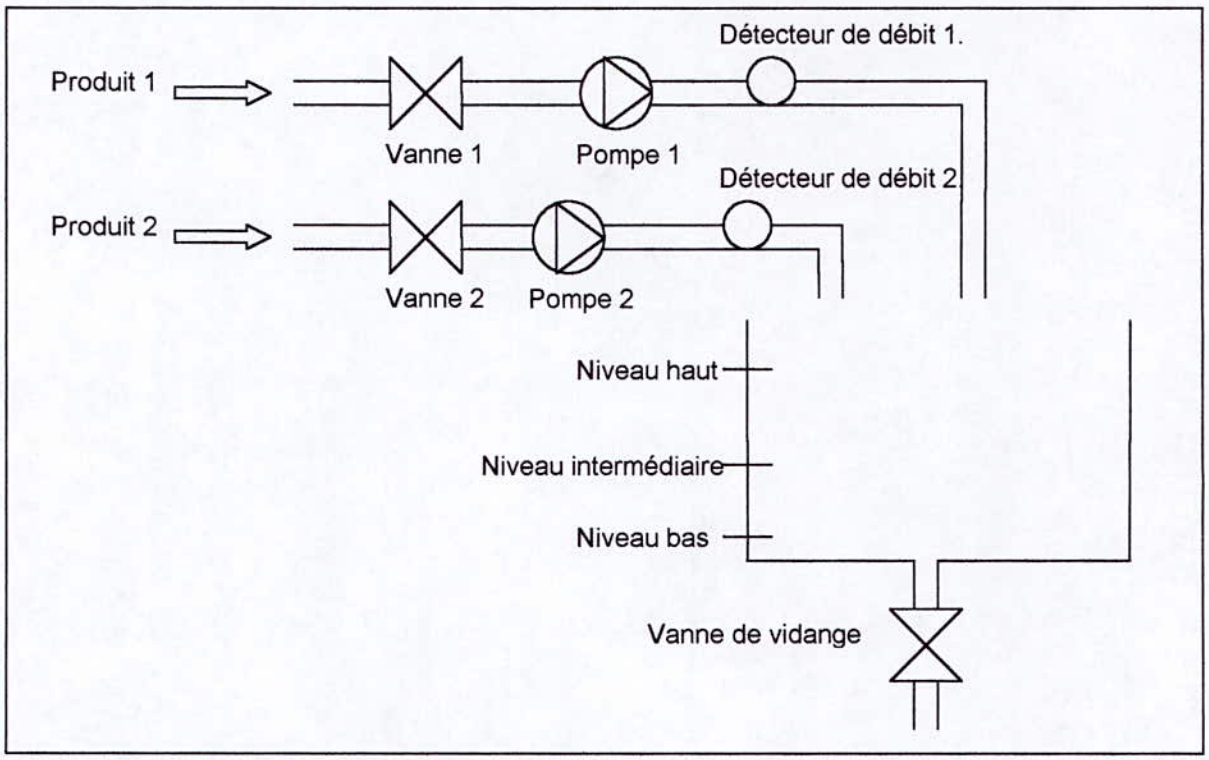


Figure 4.9 Schéma synoptique pour le mélange de deux produits.

Contraintes :

Les pompes qui acheminent les produits A et B ne doivent pas fonctionner plus de 5 secondes s'il n'y a pas de produit.

Les conduites des produits A et B doivent donc être équipées de détecteurs de débit.

Si les détecteurs ne signalent pas de débit alors que les pompes fonctionnent, celles-ci sont arrêtées au bout de 5 secondes. Et une alarme est déclenchée pour signaler la défaillance. Un bouton d'acquittement permet de relancer le système là où il s'est arrêté.

Le système est traduit en deux GRAFCET, l'un pour le fonctionnement général et l'autre pour la commande des vannes.

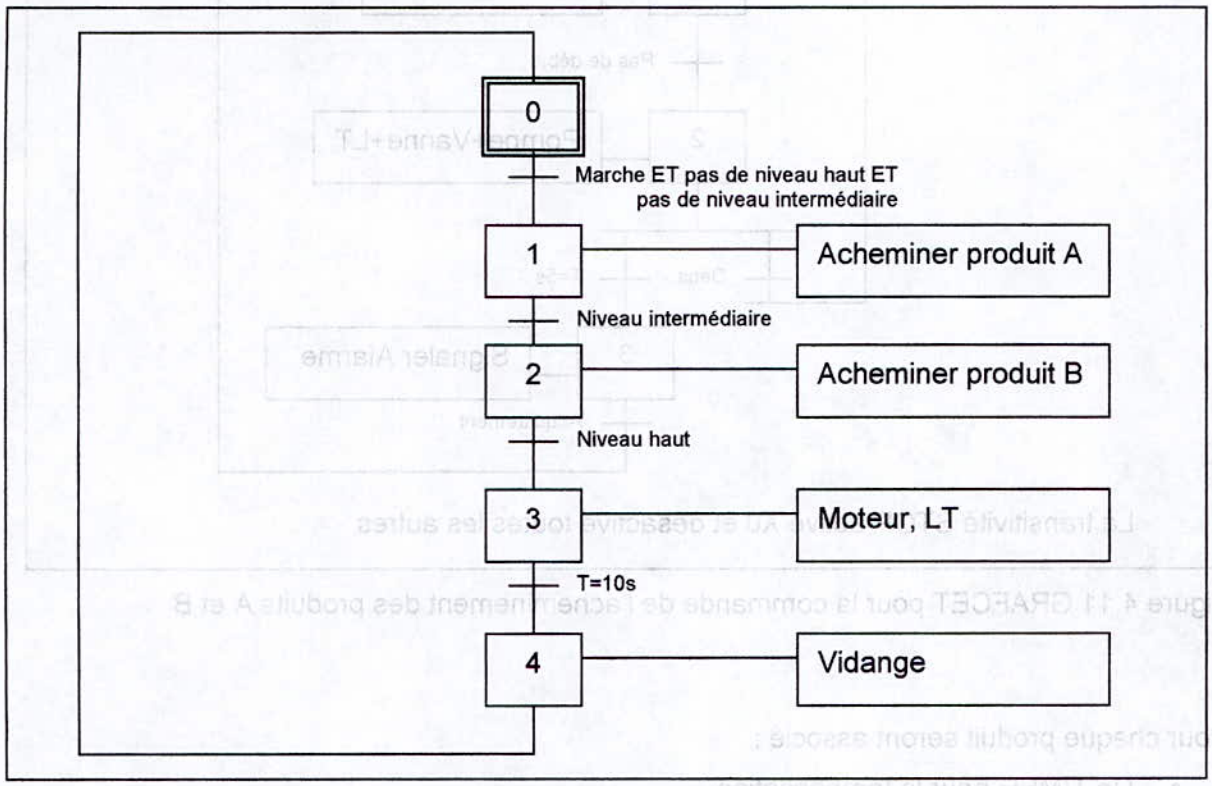


Figure 4.10 GRAFCET du système global.

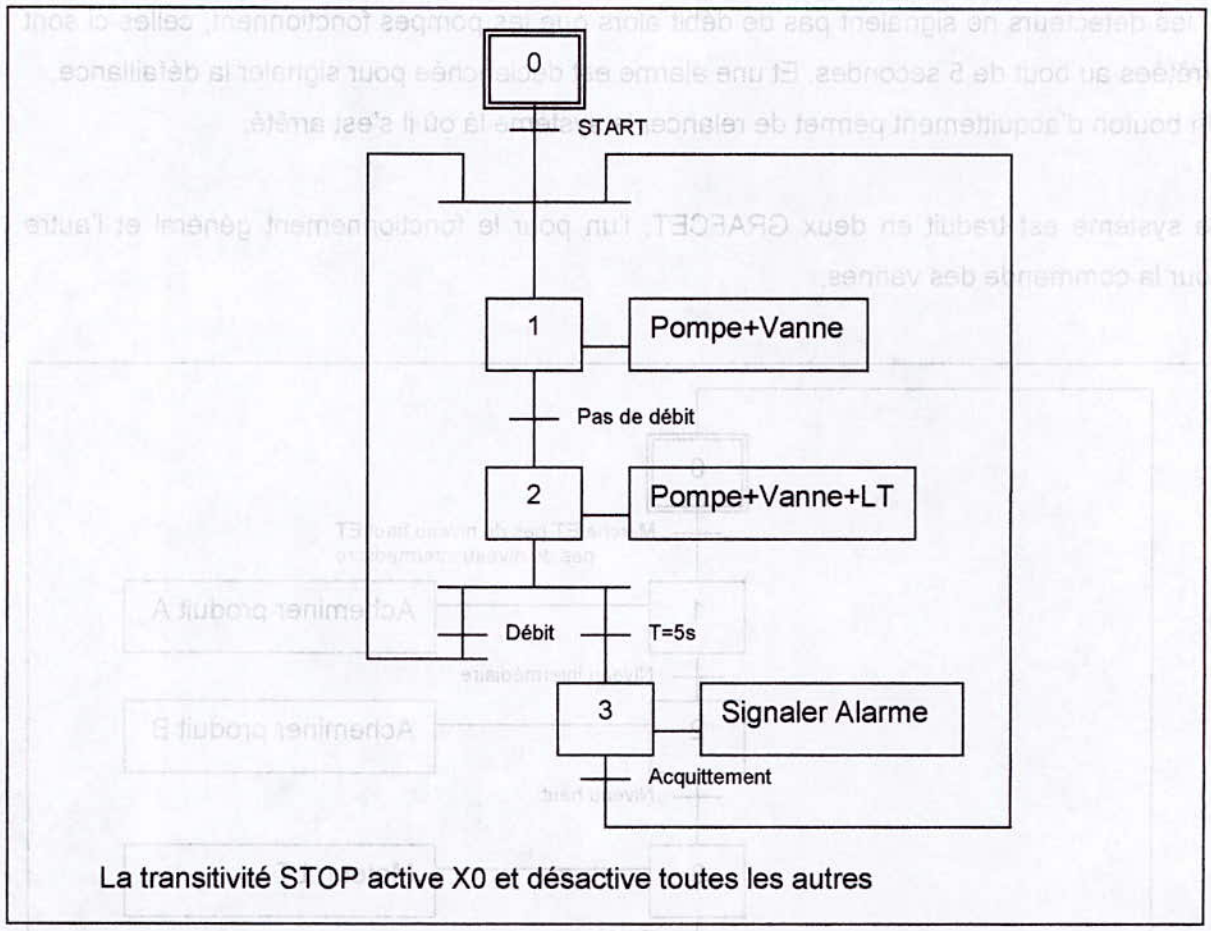


Figure 4.11 GRAFCET pour la commande de l'acheminement des produits A et B

Pour chaque produit seront associé :

- Un TIMER pour la temporisation.
- Une entrée donnant l'état de la temporisation associée.
- Une entrée START.
- Une entrée STOP.
- Une entrée pour le détecteur de débit.
- Une sortie pour commander la vanne.
- Une sortie pour commander la pompe.
- Une sortie pour signaler l'erreur.
- Et une sortie pour le lancement de la temporisation associée.

Il faut donc déclarer toutes ces entrées et sorties dans l'entête du FB de commande de l'acheminement des produits A et B.

Lors de l'appel de ce FB, à chaque produit sera associé un bloc de données.

Ces blocs de données auront la même structure.

Propriétés de la table des mnémoniques

Nom :	Mnémoniques pour mélange
Commentaire :	
Date de création :	17/05/04 11:20:48
Dernière modification :	19/06/04 11:37:54
Dernier filtre sélectionné :	Tous les mnémoniques
Nombre de mnémoniques :	20/20
Dernier tri :	Opérande ordre croissant

Etat	Mnémonique	Opérande	Type de données	Commentaire
	Etape0	A 124.0	BOOL	
	Etape1	A 124.1	BOOL	
	Etape2	A 124.2	BOOL	
	Etape3	A 124.3	BOOL	
	Etape4	A 124.4	BOOL	
	Pompe1	A 125.0	BOOL	Commande de la pompe 1.
	Pompe2	A 125.1	BOOL	Commande de la pompe 2.
	Vanne1	A 125.2	BOOL	Commande de la vanne 1.
	Vanne2	A 125.3	BOOL	Commande de la vanne 2.
	Moteur	A 125.4	BOOL	Commande du moteur qui effectuera le mélange.
	Vidange	A 125.5	BOOL	Commande de la vanne de vidange.
	Erreur debit1	A 125.6	BOOL	Sortie signalant un problème au niveau du débit de la pompe 1.
	Erreur debit2	A 125.7	BOOL	Sortie signalant un problème au niveau du débit de la pompe 2.
	Marche	E 124.0	BOOL	Bouton de mise en marche du système.
	Acquittement	E 124.1	BOOL	Bouton d'acquittement de problèmes.
	Débit1_ok	E 124.2	BOOL	Entrée indiquant la présence de débit sur la pompe 1.
	Débit2_ok	E 124.3	BOOL	Entrée indiquant la présence de débit sur la pompe 2.
	Niveau_bas	E 124.4	BOOL	Détecteur de niveau bas.
	Niveau_intermediaire	E 124.5	BOOL	Détecteur de niveau intermédiaire.
	Niveau_haut	E 124.6	BOOL	Détecteur de niveau haut.

OB100 - <offline>

""

Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 15:54:43
Interface : 15/02/96 16:51:10
Longueur (bloc/code /données locales) : 00172 00054 00020

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB100_EV_CLASS	Byte	0.0		16#13, Event class 1, Entering event state, Event logged in diagnostic buffer
OB100_STRTUP	Byte	1.0		16#81/82/83/84 Method of startup
OB100_PRIORITY	Byte	2.0		27 (Priority of 1 is lowest)
OB100_OB_NUMBR	Byte	3.0		100 (Organization block 100, OB100)
OB100_RESERVED_1	Byte	4.0		Reserved for system
OB100_RESERVED_2	Byte	5.0		Reserved for system
OB100_STOP	Word	6.0		Event that caused CPU to stop (16#4xxx)
OB100_STRT_INFO	DWord	8.0		Information on how system started
OB100_DATE_TIME	Date_And_Time	12.0		Date and time OB100 started

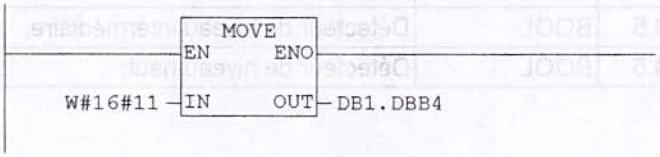
Bloc : OB100 "Complete Restart"

Réseau : 1

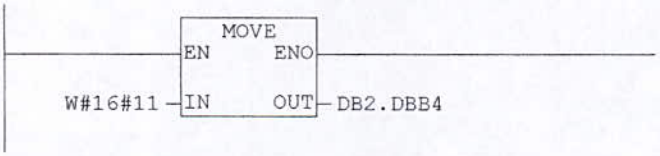
Mise à 1 d'un bit qui le restera pendant l'exécution du programme.

```
      SET
      =     M    255.7
```

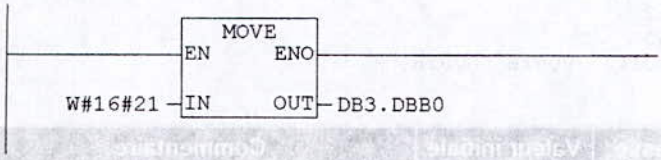
Réseau : 2 Initialisation des variables statiques du DB1.



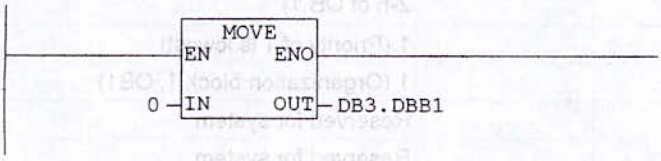
Réseau : 3 Initialisation des variables statiques du DB2.



Réseau : 4 Initialisation des variables statiques du DB3.



Réseau : 5 Initialisation des variables statiques du DB3.



OB1 - <offline>

""

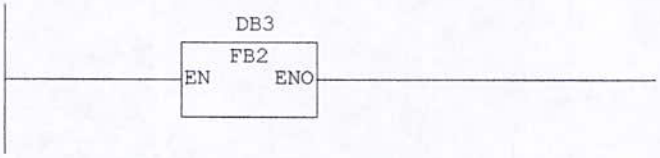
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 14:27:13
Interface : 17/05/04 11:20:50
Longueur (bloc/code /données locales) : 00148 00036 00026

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB1_EV_CLASS	Byte	0.0		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1	Byte	1.0		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
OB1_PRIORITY	Byte	2.0		1 (Priority of 1 is lowest)
OB1_OB_NUMBR	Byte	3.0		1 (Organization block 1, OB1)
OB1_RESERVED_1	Byte	4.0		Reserved for system
OB1_RESERVED_2	Byte	5.0		Reserved for system
OB1_PREV_CYCLE	Int	6.0		Cycle time of previous OB1 scan (milliseconds)
OB1_MIN_CYCLE	Int	8.0		Minimum cycle time of OB1 (milliseconds)
OB1_MAX_CYCLE	Int	10.0		Maximum cycle time of OB1 (milliseconds)
OB1_DATE_TIME	Date_And_Time	12.0		Date and time OB1 started

Bloc : OB1

Réseau : 1

Appel du FB2 associé au DB3.



FB2 - <offline>

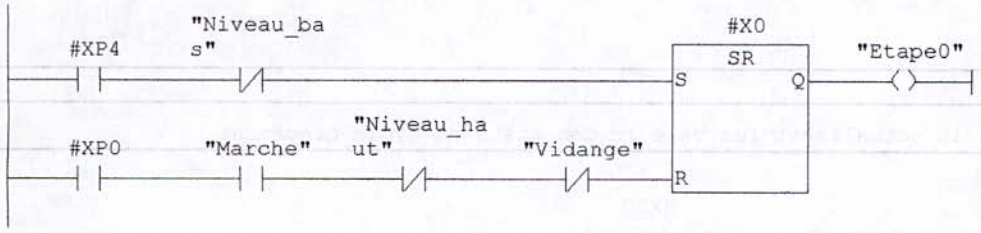
""

Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 19/06/04 10:37:44
Interface : 17/05/04 13:25:15
Longueur (bloc/code /données locales) : 00614 00470 00006

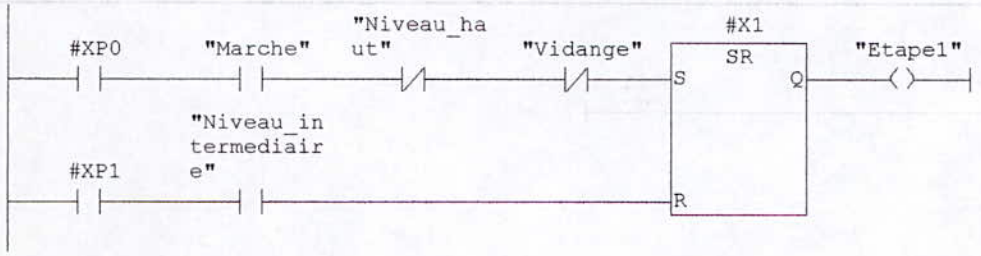
Nom	Type de données	Adresse	Valeur initiale	Commentaire
IN		0.0		
OUT		0.0		
IN_OUT		0.0		
STAT		0.0		
X0	Bool	0.0	FALSE	
X1	Bool	0.1	FALSE	
x2	Bool	0.2	FALSE	
X3	Bool	0.3	FALSE	
X4	Bool	0.4	FALSE	
XP0	Bool	0.5	FALSE	
XP1	Bool	0.6	FALSE	
XP2	Bool	0.7	FALSE	
XP3	Bool	1.0	FALSE	
XP4	Bool	1.1	FALSE	
TEMP		0.0		

Bloc : FB2 Bloc fonctionnel pour la gestion d'un mélangeur de deux produits

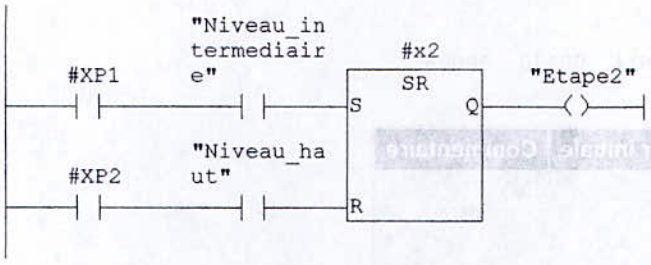
Réseau : 1 Etape de repos.



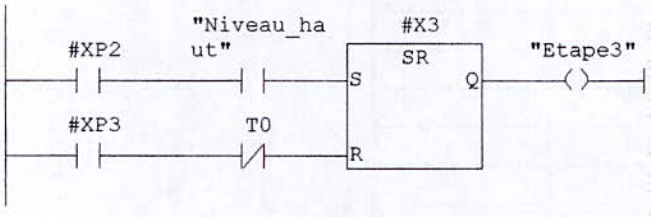
Réseau : 2 Etape de remlissage du produit 1.



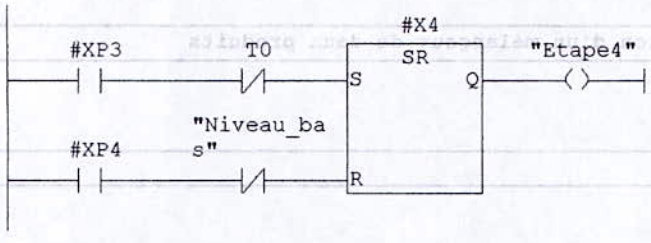
Réseau : 3 Etape de remlissage du produit 2.



Réseau : 4 Etape de mélange.



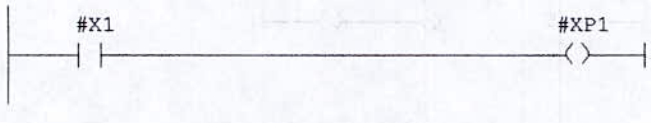
Réseau : 5 Etape de vidange.



Réseau : 6

Les réseaux de 6 à 10 actualisent les valeurs des états du cycle précédent.

Réseau : 7



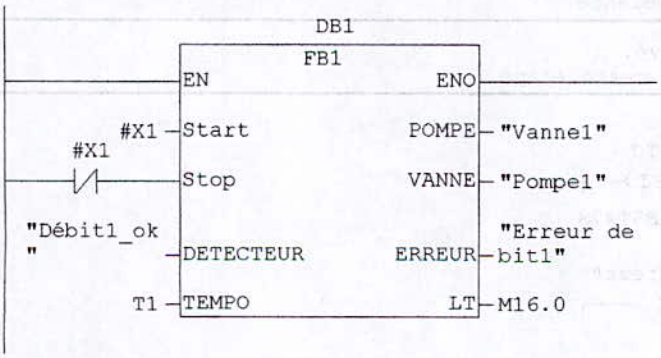
Réseau : 8

Réseau : 9

Réseau : 10

Réseau : 11 Appel du bloc de commande de la vanne et de la pompe 1.

En associant le bloc de données DB1 au FB1, c'est la pompe 1 qui est sélectionée.
La pompe 1 s'active à l'étape 1 et se désactive dès que l'étape 1 est arrêtée.
Le TIMER associé est T1 et le détecteur de débit est à l'entrée E124.2
Le bit M16.0 lance ou non la temporaisation.

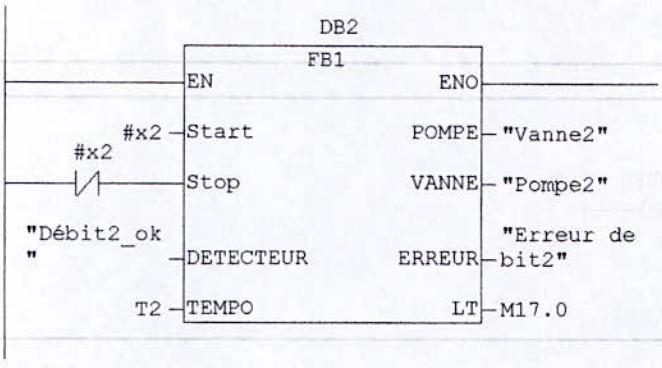


Réseau : 12 Lancement de la temporisation associée au produit 1.

La temporisation est de type impulsionnel ce qui permet de l'arrêter si M16.0 est mis à 0.

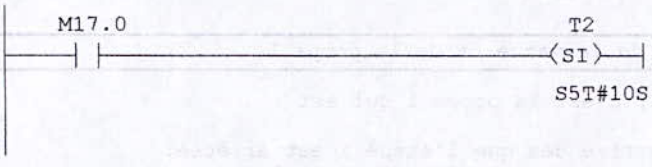
Réseau : 13	Appel du bloc de commande de la vanne et de la pompe 2.

En associant le bloc de données DB2 au FB1, c'est la pompe 2 qui est sélectionée.
La pompe 2 s'active à l'étape 2 et se désactive dès que l'étape 2 est arrêtée.
Le TIMER associé est T2 et le détecteur de débit est à l'entrée E124.3
Le bit M17.0 lance ou non la temporaisation.



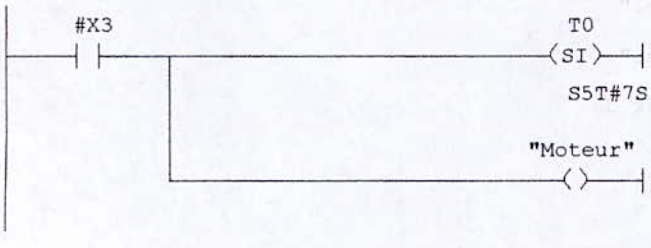
Réseau : 14	Lancement de la teporisation associée au produit 2.

La temporisation est de type impulsionnel ce qui permet de l'arrêter si M16.0 est mis à 0.



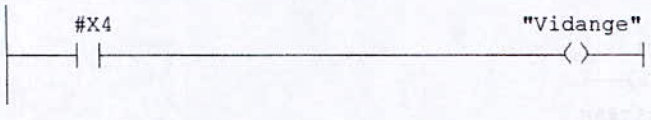
Réseau : 15	Activation du moteur de mélange.

A l'étape 3 le moteur de mélange est activé.
Une temporisation est lancée pour arrêter cette étape.

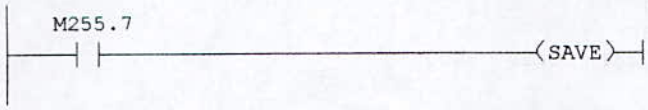


Réseau : 16	Vidange.

La vidange du bac est effectuée à l'étape 4.



Réseau : 17	Activation de RB.
Mise à 1 du bit RB du mot d'état dont la sortie ENO de FB sera l'image.	



FB1 - <offline>

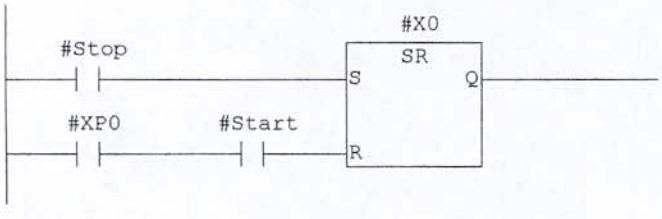
""

Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 16:04:17
Interface : 17/05/04 13:59:38
Longueur (bloc/code /données locales) : 00364 00218 00000

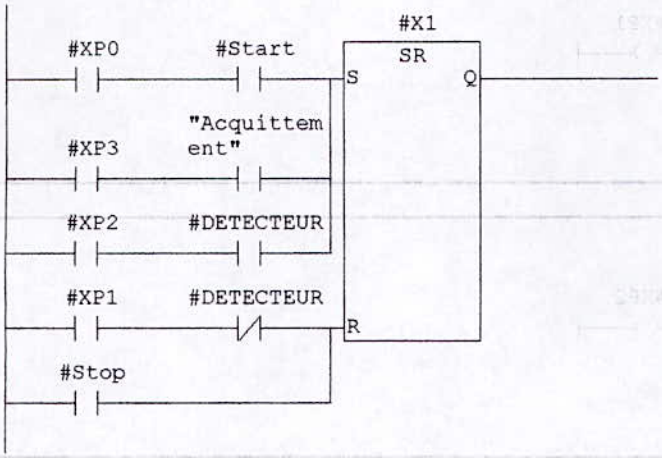
Nom	Type de données	Adresse	Valeur initiale	Commentaire
IN		0.0		
Start	Bool	0.0	FALSE	Signal d'activation de la pompe.
Stop	Bool	0.1	FALSE	Signal d'arrêet de la pompe.
DETECTEUR	Bool	0.2	FALSE	Signal donnant l'état du débit associé.
TEMPO	Bool	0.3	FALSE	Signal donnant l'état de la temporisation associée.
OUT		2.0		
POMPE	Bool	2.0	FALSE	Signal de commande de la pompe.
VANNE	Bool	2.1	FALSE	Signal de commande de la vanne.
ERREUR	Bool	2.2	FALSE	Signal d'erreur sur le débit.
LT	Bool	2.3	FALSE	Signal de lancement de temporisation.
IN_OUT		0.0		
STAT		0.0		
X0	Bool	4.0	FALSE	Etat actuel de l'étape 0.
X1	Bool	4.1	FALSE	Etat actuel de l'étape 1.
X2	Bool	4.2	FALSE	Etat actuel de l'étape 2.
X3	Bool	4.3	FALSE	Etat actuel de l'étape 3.
XP0	Bool	4.4	FALSE	Etat du cycle précédent de l'étape 0.
XP1	Bool	4.5	FALSE	Etat du cycle précédent de l'étape 1.
XP2	Bool	4.6	FALSE	Etat du cycle précédent de l'étape 2.
XP3	Bool	4.7	FALSE	Etat du cycle précédent de l'étape 3.
TEMP		0.0		

Bloc : FB1 Bloc fonctionnel pour la commande des vannes et pompes.

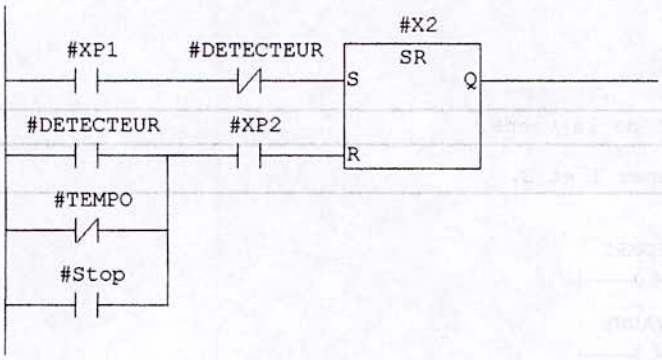
Réseau : 1 Etape 0.



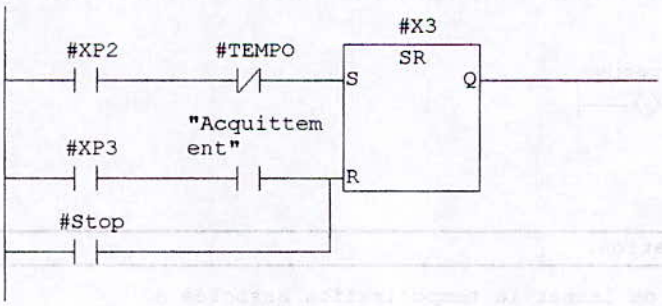
Réseau : 2 Etape 1.



Réseau : 3 Etape 2.



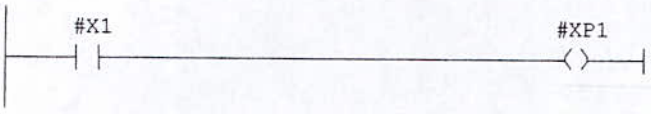
Réseau : 4 Etape 3.



Réseau : 5

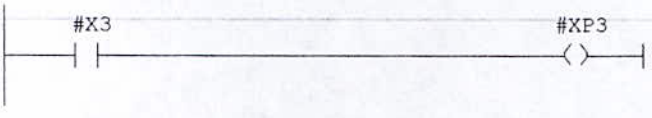
Les réseaux 5 à 8 actualisent les évariables statiques contenant les états du cycle précédent.

Réseau : 6



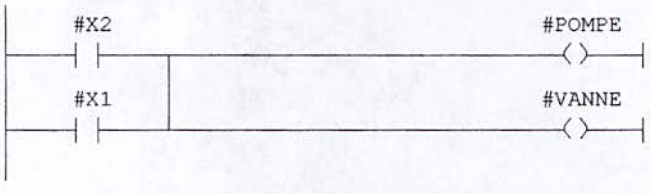
Réseau : 7

Réseau : 8

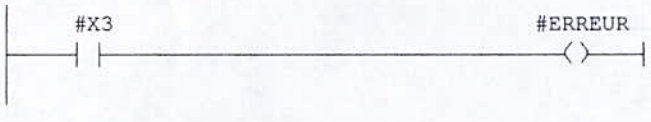


Réseau : 9 Activation de la pompe et de la vanne.

La pompe et la vanne s'activent aux l'étapes 1 et 2.



Réseau : 10 Signalement de l'erreur.



Réseau : 11 Lancement de la temporisation.

Ce réseau active la sortie qui permettra de lancer la temporisation associée à la pompe.

Réseau : 12 Activation de RB.
Mise à 1 du bit du mot d'état RB dont la sortie ENO du FB1 sera l'image.

5. Utilisation du fréquencemètre :

Cette application permet de profiter de la fonction intégrée fréquencemètre pour commander un moteur à deux vitesses.

Fonctionnement :

Un convoyeur achemine des bouteilles. Le moteur faisant tourner ce convoyeur comporte deux vitesses, une lente et une rapide.

Si le flux de bouteilles est correct, le moteur doit tourner lentement, si le flux de bouteilles diminue sous une limite inférieure pendant plus d'une seconde, le moteur passe à la vitesse rapide jusqu'à ce que le flux ré-atteigne la limite supérieure pendant plus d'une seconde.

Le moteur est démarré avec un bouton MARCHE et désactivé avec un bouton STOP.

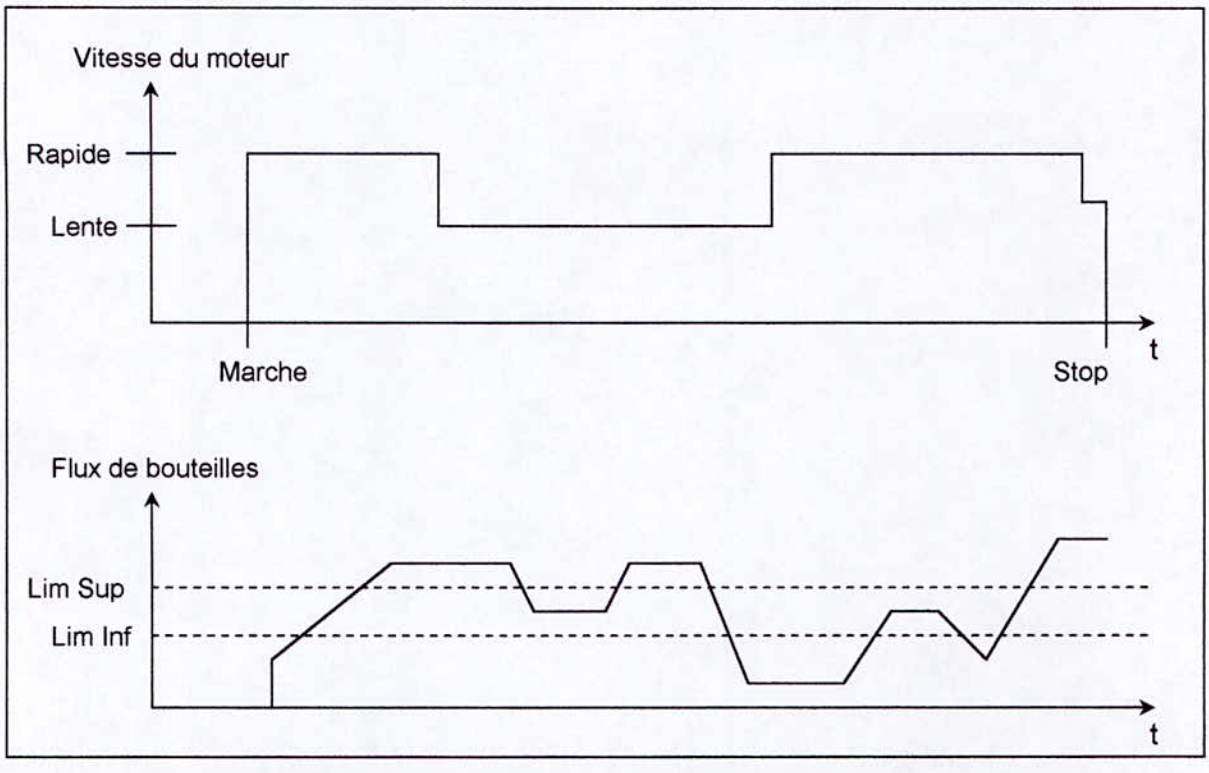


Figure 4.12. Chronogramme pour le changement de vitesse du moteur à deux vitesses.

Pour réaliser ce programme, on utilise le SFB30 qui gère la fonction fréquencemètre.

Avant d'écrire le programme, il faut configurer le matériel :

- On commence par réaliser la configuration matérielle de base citée au début de ce chapitre,
- Puis on ouvre les propriétés de la CPU et on configure la fonction intégrée sur fréquencemètre,

- Le cahier des charges demande une mesure toutes les secondes, on configure donc la durée de mesure à 1s.
- Enfin, on compile et sauvegarde.

Remarque :

L'entrée de signal pour le fréquencemètre est E126.0

Le programme est exécuté à l'aide d'un bloc fonctionnel FB1 qui sera appelé dans le bloc d'organisation OB1.

Comme dans les programmes précédents, il faut initialiser certaines variables.

Les valeurs limites du SFB 30 ne sont prises en comptes que s'il y a front montant sur les entrées :

- SET_U_LIMIT
- SET_L_LIMIT

Il faut donc lancer le bloc SFB 30 dans l'OB100 en donnant des valeurs nulles aux entrée SET_U_LIMIT et SET_L_LIMIT. Au cours du fonctionnement normal (du FB1), les entrées SET_U_LIMIT et SET_L_LIMIT recevront une valeur 1, ce qui provoquera un front montant lors du premier cycle d'exécution, et appliquera ainsi les valeurs seuils contenues dans PRES_U_LIMIT et PRES_L_LIMIT.

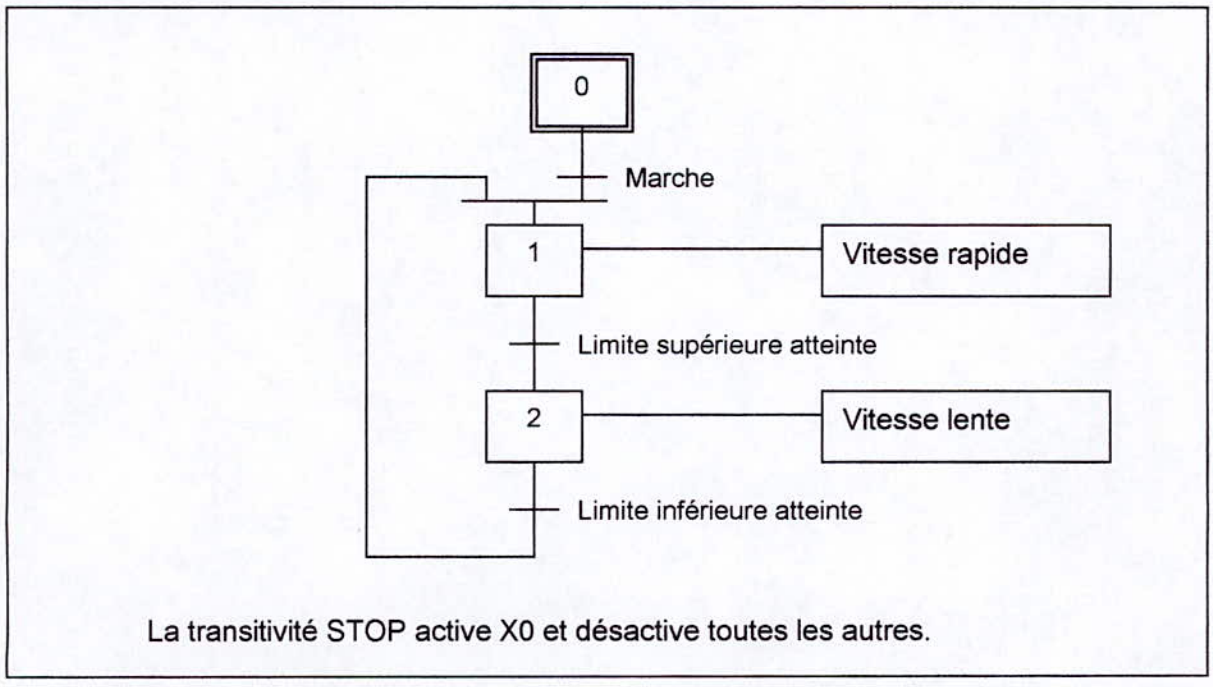


Figure 4.13 GRAFCET pour la commande d'un moteur à deux vitesses.

Propriétés de la table des mnémoniques

Nom :	Mnémoniques
Commentaire :	
Date de création :	19/05/04 12:07:31
Dernière modification :	11/06/04 15:26:05
Dernier filtre sélectionné :	Tous les mnémoniques
Nombre de mnémoniques :	11/11
Dernier tri :	Mnémonique ordre croissant

Etat	Mnémonique	Opérande	Type de données	Commentaire
	Arrêt	E 124.1	BOOL	Arrêt du système
	COMPLETE RESTART	OB 100	OB 100	Complete Restart
	Dépassement_bas	M 5.1	BOOL	
	Dépassement_haut	M 5.0	BOOL	
	FREQ_MES	SFB 30	SFB 30	Frequency Meter (Integrated Function, CPU 312 IFM, 314 IFM)
	Fréquence_actuelle	MD 0	DINT	Fréquence actuelle mesurée par la fonction intégrée fréquencemètre.
	Marche	E 124.0	BOOL	Démarrage du système
	Moteur_ON_OFF	A 124.0	BOOL	Mise en marche ou à l'arrêt du moteur
	SCALE	FC 105	FC 105	Scaling Values
	Vitesse_lente	A 124.1	BOOL	Commande de la vitesse lente du moteur
	Vitesse_rapide	A 124.2	BOOL	Commande de la vitesse rapide du moteur

OB100 - <offline>

"COMPLETE RESTART" Complete Restart
Nom : Famille :
Auteur : Version : 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 14:08:30
Interface : 15/02/96 16:51:10
Longueur (bloc/code /données locales) : 00200 00084 00026

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB100_EV_CLASS	Byte	0.0		16#13, Event class 1, Entering event state, Event logged in diagnostic buffer
OB100_STRTUP	Byte	1.0		16#81/82/83/84 Method of startup
OB100_PRIORITY	Byte	2.0		Priority of OB Execution
OB100_OB_NUMBR	Byte	3.0		100 (Organization block 100, OB100)
OB100_RESERVED_1	Byte	4.0		Reserved for system
OB100_RESERVED_2	Byte	5.0		Reserved for system
OB100_STOP	Word	6.0		Event that caused CPU to stop (16#4xxx)
OB100_STRT_INFO	DWord	8.0		Information on how system started
OB100_DATE_TIME	Date_And_Time	12.0		Date and time OB100 started

Bloc : OB100 "Complete Restart"

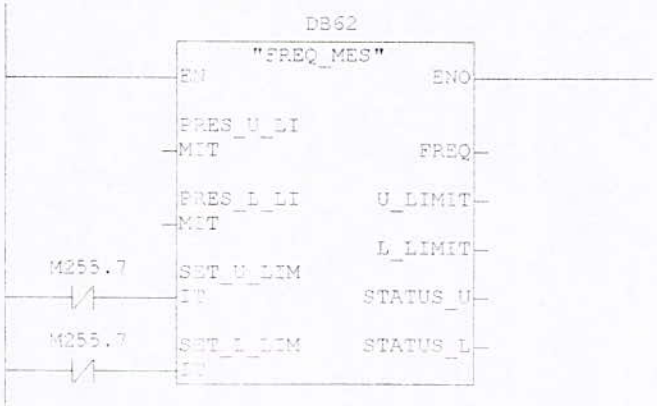
Réseau : 1

Mise à 1 d'un bit qui le restera pendant tout le déroulement du programme.

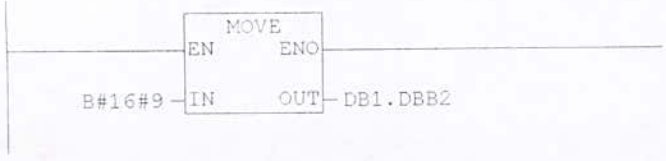
```
      SET
      =     M    255.7
```

Réseau : 2 Appel du bloc fonctionnel système SFB30 associé au bloc de donné

Les entrées SET_U_LIMIT et SET_L_LIMIT sont mise à 0 pour avoir un front
montant
lors du premier cycle de l'OB1.



Réseau : 3 Initialisation des variables statiques du DB1.



OB1 - <offline>

""

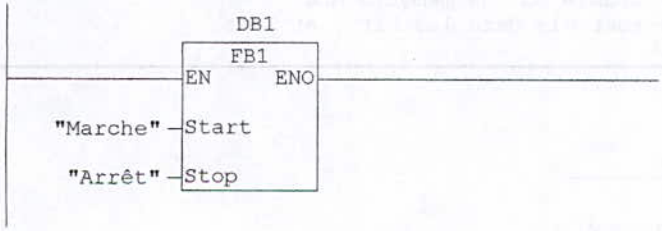
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 14:05:38
Interface : 15/02/96 16:51:12
Longueur (bloc/code /données locales) : 00180 00068 00026

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB1_EV_CLASS	Byte	0.0		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1	Byte	1.0		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
OB1_PRIORITY	Byte	2.0		Priority of OB Execution
OB1_OB_NUMBR	Byte	3.0		1 (Organization block 1, OB1)
OB1_RESERVED_1	Byte	4.0		Reserved for system
OB1_RESERVED_2	Byte	5.0		Reserved for system
OB1_PREV_CYCLE	Int	6.0		Cycle time of previous OB1 scan (milliseconds)
OB1_MIN_CYCLE	Int	8.0		Minimum cycle time of OB1 (milliseconds)
OB1_MAX_CYCLE	Int	10.0		Maximum cycle time of OB1 (milliseconds)
OB1_DATE_TIME	Date_And_Time	12.0		Date and time OB1 started

Bloc : OB1 "Main Program Sweep (Cycle)"

Réseau : 1

Appel du bloc fonctionnel FB1 en lui associant le bloc de données DB1.



FB1 - <offline>

""

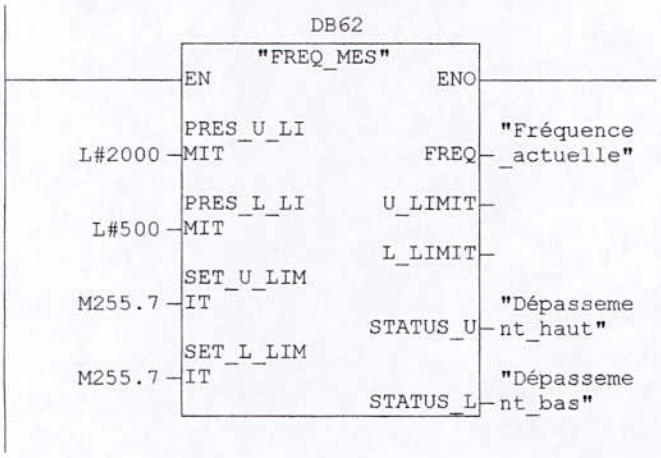
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 14:18:51
Interface : 19/05/04 11:53:44
Longueur (bloc/code /données locales) : 00384 00256 00006

Nom	Type de données	Adresse	Valeur initiale	Commentaire
IN		0.0		
Start	Bool	0.0	FALSE	Entrée permettant le démarrage du système.
Stop	Bool	0.1	FALSE	Entrée permettant l'arrêt du système.
OUT		0.0		
IN_OUT		0.0		
STAT		0.0		
X0	Bool	2.0	FALSE	Etat actuel de l'étape 0.
X1	Bool	2.1	FALSE	Etat actuel de l'étape 1.
X2	Bool	2.2	FALSE	Etat actuel de l'étape 2.
XP0	Bool	2.3	FALSE	Etat du cycle précédent de l'étape 0.
XP1	Bool	2.4	FALSE	Etat du cycle précédent de l'étape 1.
XP2	Bool	2.5	FALSE	Etat du cycle précédent de l'étape 2.
TEMP		0.0		

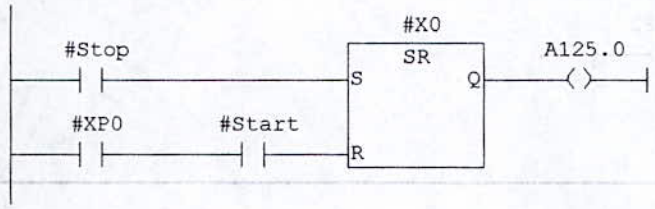
Bloc : FB1 Bloc foncionnel permettant le réalisation du système.

Réseau : 1 Appel du bloc fonctionnel système SFB30.

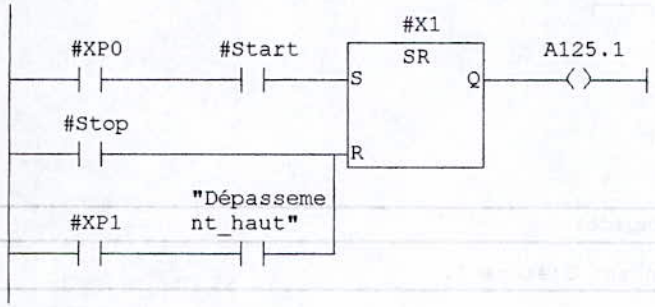
Ce bloc est associé au bloc de données DB62 qui a été configuré dans HW Config.
Les bits SET_U_LIMIT et SET_L_LIMIT sont mis à 1 pour avoir un front montant
lors de l'exécution du premier cycle de l'OB1 et prendre ainsi en considération
les valeurs limites des comparateurs A et B.
La fréquence en cours est stockée dans le double mot de mémento MD0.
Les états de dépassement des comparateurs sont mis dans les bit 0 et 1 de
l'octet de mémento 5.



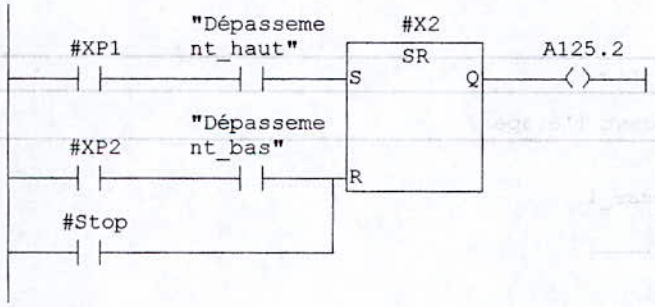
Réseau : 2 Etape 0.



Réseau : 3 Etape 1.



Réseau : 4 Etape 2.



Réseau : 5 Actualisation des variables statiques

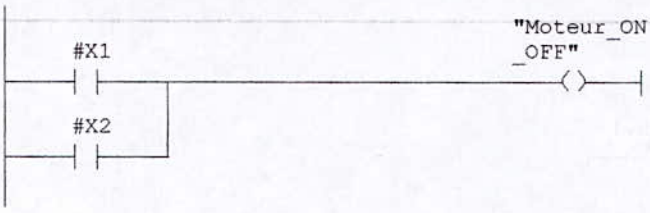
Les réseaux de 5 à 7 actualisent les états de cycle des étapes 0 à 2.

Réseau : 6

Réseau : 7

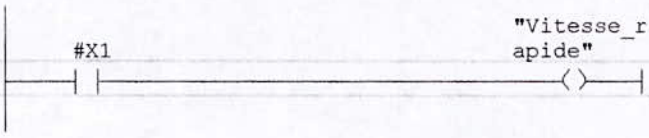
Réseau : 8 Mise en marche du moteur.

Le moteur fonctionne pendant les étapes 1 et 2.



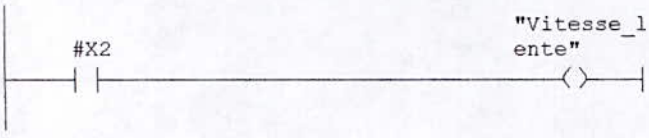
Réseau : 9 Activation de la vitesse rapide.

Le moteur tourne avec la vitesse rapide pendant l'étape 1.



Réseau : 10 Activation de la vitesse lente.

Le moteur tourne avec la vitesse lente pendant l'étape 2.



Réseau : 11 Activation de RB.

Mise à 1 du bit RB du mot d'état dont la sortie ENO du bloc fonctionnel sera l'image.

6. Distributeur de carburant :

Cet exemple industriel a pour but :

- L'utilisation des entrées analogiques.
- La conversion des valeurs lues sur les entrées analogiques selon la plage de mesure des capteurs.
- L'utilisation des blocs de comparaison.
- L'introduction de la notion de connecteur de sauvegarde.
- L'introduction du mémento de cadence.

Fonctionnement :

Un même pipeline approvisionne la station de Chiffa, Willaya de Blida avec de l'essence normale, de l'essence super et du gasoil.

Le stockage se fait dans trois bacs différents, selon la nature du produit, un quatrième bac fait office de bac de sécurité.

On se propose de trouver une solution automatisée à base d'automates programmables pour la distribution du carburant dans les différents bacs.

Les trois carburants possèdent des densités différentes qui sont :

Produit	Plage de densité
Essence normale	0.730 à 0.770
Essence super	0.780 à 0.810
Gasoil	0.830 à 0.860

Tableau 4.7. Densités des carburants.

Un débitmètre signale la présence de carburant dans le pipeline et un densimètre indique son type.

Avant l'arrivée du produit, c'est la vanne du bac de sécurité qui est ouverte, les trois autres bacs étant fermés.

Dés qu'un carburant arrive, une temporisation est lancée pour détecter son type.
La vanne du bac du produit correspondant s'ouvre à la fin cette temporisation si le niveau haut du bac n'est pas atteint.

La vanne de sécurité s'ouvre si aucune des trois plages de densité n'est présente, ou s'il y a niveau haut dans l'un des trois bacs.

Si le niveau haut du bac de sécurité est atteint, une alarme est lancée, attendant un acquittement de l'opérateur.

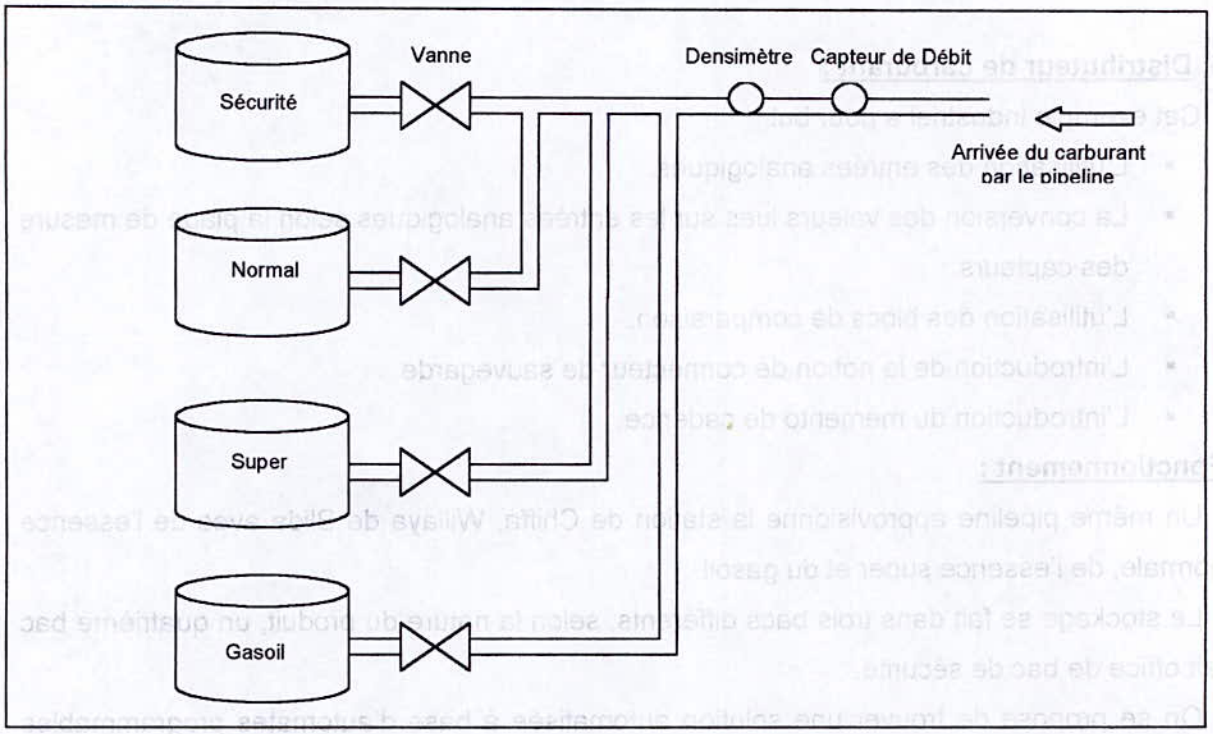


Figure 4.14 Schéma synoptique de la distribution de carburant.

La GRAFCET de la page suivante décrit les étapes de mise en œuvre de la distribution.

Remarques :

Le bloc FC105.effectue la conversion de l'entrée analogique en nombre réel suivant la plage de mesure du capteur utilisé. On peut choisir un fonctionnement en bipolaire ou non.

Le mémento de cadence doit être configuré dans le matériel.

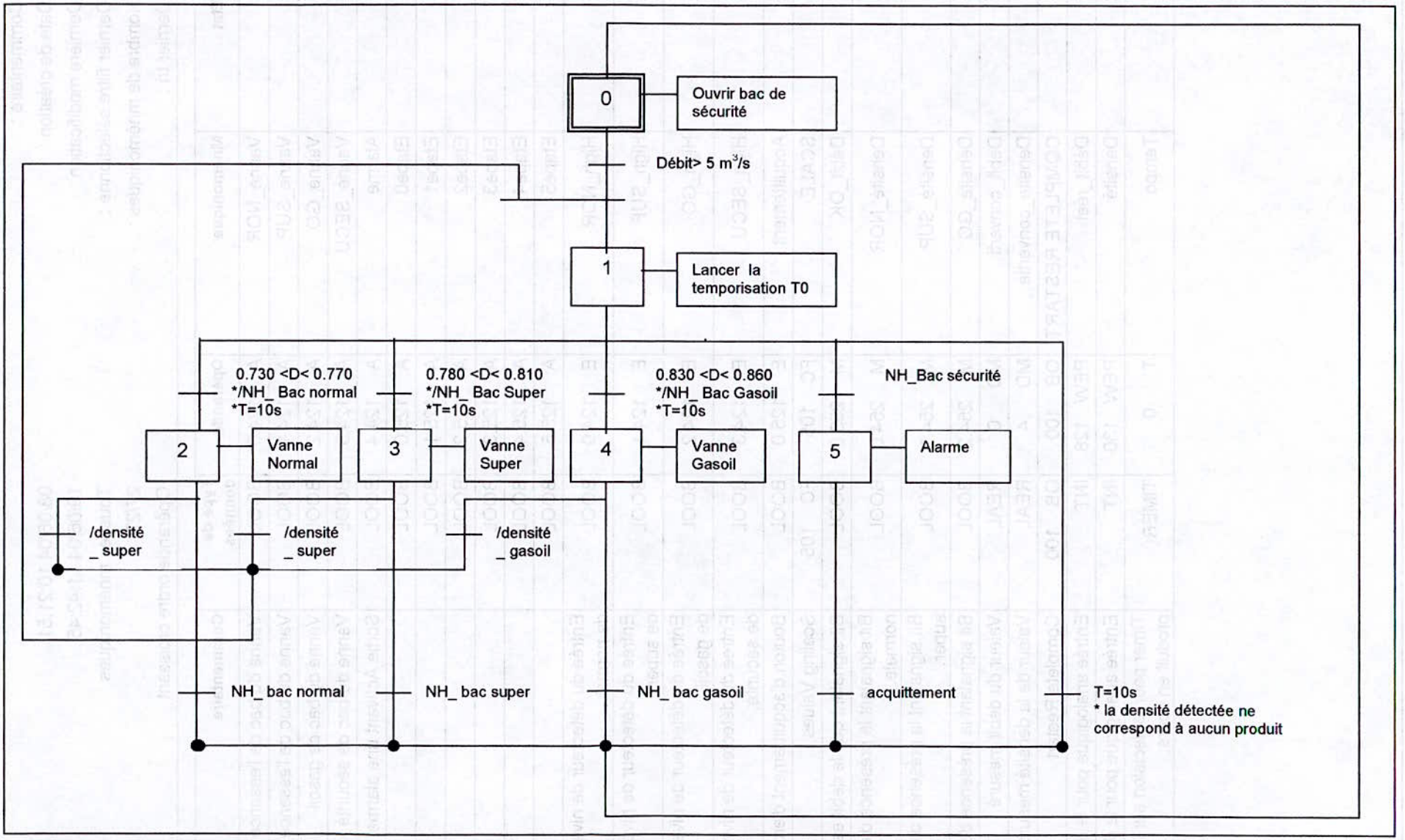


Figure 4.15 GRAFCET de la commande de la distribution de carburant.

Propriétés de la table des mnémoniques

Nom :	Mnémoniques
Commentaire :	
Date de création :	08/06/04 10:21:51
Dernière modification :	19/06/04 11:42:45
Dernier filtre sélectionné :	Tous les mnémoniques
Nombre de mnémoniques :	27/27
Dernier tri :	Opérande ordre croissant

Etat	Mnémonique	Opérande	Type de données	Commentaire
	Vanne_NOR	A 124.0	BOOL	Vanne du bac de l'essance normale
	Vanne_SUP	A 124.1	BOOL	Vanne du bac de l'essence super
	Vanne_GO	A 124.2	BOOL	Vanne du bac de gasoil
	Vanne_SECU	A 124.3	BOOL	Vanne du bac de sécurité
	Alarme	A 124.4	BOOL	Sortie Activant une alarme
	Etape0	A 125.0	BOOL	
	Etape1	A 125.1	BOOL	
	Etape2	A 125.2	BOOL	
	Etape3	A 125.3	BOOL	
	Etape4	A 125.4	BOOL	
	Etape5	A 125.5	BOOL	
	High_NOR	E 124.0	BOOL	Entrée du détecteur de niveau pour le bac de normal.
	High_SUP	E 124.1	BOOL	Entrée du détecteur de niveau pour le bac de super.
	High_GO	E 124.2	BOOL	Entrée du détecteur de niveau pour le bac de gasoil.
	High_SECU	E 124.3	BOOL	Entrée du détecteur de niveau pour le bac de sécurité.
	Acquittement	E 125.0	BOOL	Bouton d'acquittement d'erreur.
	SCALE	FC 105	FC 105	Scaling Values
	Débit_OK	M 253.0	BOOL	Bit signalant que le débit est bon.
	Densité_NOR	M 254.0	BOOL	Bit signalant la présence d'essence normale.
	Densité_SUP	M 254.1	BOOL	Bit signalant la présence d'essence super.
	Densité_GO	M 254.2	BOOL	Bit signalant la présence de gasoil.
	Débit_converti	MD 0	REAL	Valeur du débit mesuré.
	Densité_convertie	MD 4	REAL	Valeur de la densité mesurée.
	COMPLETE RESTART	OB 100	OB 100	Complete Restart
	Débit_réel	PEW 128	INT	Entrée analogique pour le débitmètre.
	Densité	PEW 130	INT	Entrée analogique pour le densimètre.
	Tempo	T 0	TIMER	Timer pour la décision sur le type de produit en cours.

OB100 - <offline>

"COMPLETE RESTART" Complete Restart
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 16:35:29
Interface : 15/02/96 16:51:10
Longueur (bloc/code /données locales) : 00132 00018 00020

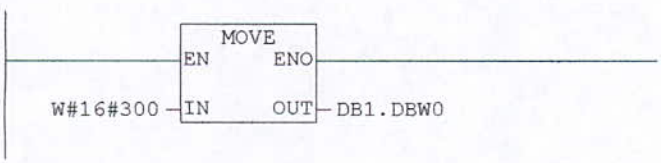
Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB100_EV_CLASS	Byte	0.0		16#13, Event class 1, Entering event state, Event logged in diagnostic buffer
OB100_STRTUP	Byte	1.0		16#81/82/83/84 Method of startup
OB100_PRIORITY	Byte	2.0		Priority of OB Execution
OB100_OB_NUMBR	Byte	3.0		100 (Organization block 100, OB100)
OB100_RESERVED_1	Byte	4.0		Reserved for system
OB100_RESERVED_2	Byte	5.0		Reserved for system
OB100_STOP	Word	6.0		Event that caused CPU to stop (16#4xxx)
OB100_STRT_INFO	DWord	8.0		Information on how system started
OB100_DATE_TIME	Date_And_Time	12.0		Date and time OB100 started

Bloc : OB100 "Complete Restart"

Réseau : 1 Mise à 1 du bit M255.7

```
      SET
      =     M    255.7
```

Réseau : 2 Initialisation des variables statiques du DB1.



OB1 - <offline>

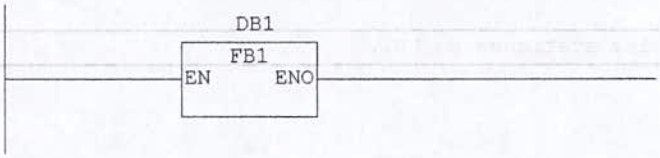
""

Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 16:28:41
Interface : 31/05/04 11:13:01
Longueur (bloc/code /données locales) : 00148 00036 00026

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB1_EV_CLASS	Byte	0.0		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1	Byte	1.0		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
OB1_PRIORITY	Byte	2.0		1 (Priority of 1 is lowest)
OB1_OB_NUMBR	Byte	3.0		1 (Organization block 1, OB1)
OB1_RESERVED_1	Byte	4.0		Reserved for system
OB1_RESERVED_2	Byte	5.0		Reserved for system
OB1_PREV_CYCLE	Int	6.0		Cycle time of previous OB1 scan (milliseconds)
OB1_MIN_CYCLE	Int	8.0		Minimum cycle time of OB1 (milliseconds)
OB1_MAX_CYCLE	Int	10.0		Maximum cycle time of OB1 (milliseconds)
OB1_DATE_TIME	Date_And_Time	12.0		Date and time OB1 started

Bloc : OB1

Réseau : 1 Appel du FB1.



FB1 - <offline>

""

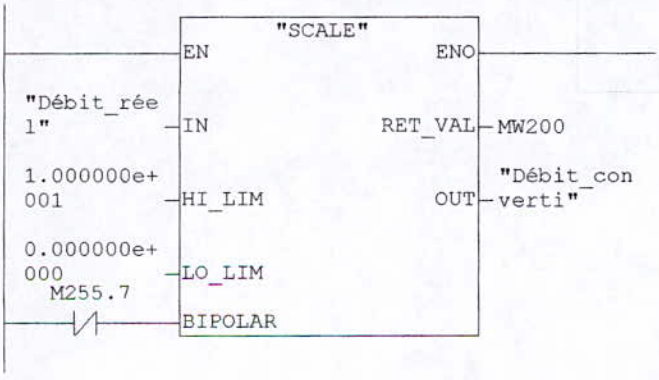
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 19/06/04 10:45:50
Interface : 31/05/04 11:53:54
Longueur (bloc/code /données locales) : 00832 00654 00014

Nom	Type de données	Adresse	Valeur initiale	Commentaire
IN		0.0		
OUT		0.0		
IN_OUT		0.0		
STAT		0.0		
X0	Bool	0.0	FALSE	Etat actuel de l'étape 0.
XP0	Bool	0.1	FALSE	Etat du cyc le précédent de l'étape 0.
X1	Bool	0.2	FALSE	Etat actuel de l'étape 1.
XP1	Bool	0.3	FALSE	Etat du cyc le précédent de l'étape 1.
X2	Bool	0.4	FALSE	Etat actuel de l'étape 2.
XP2	Bool	0.5	FALSE	Etat du cyc le précédent de l'étape 2.
X3	Bool	0.6	FALSE	Etat actuel de l'étape 3.
XP3	Bool	0.7	FALSE	Etat du cyc le précédent de l'étape 3.
X4	Bool	1.0	FALSE	Etat actuel de l'étape 4.
XP4	Bool	1.1	FALSE	Etat du cyc le précédent de l'étape 4.
X5	Bool	1.2	FALSE	Etat actuel de l'étape 5.
XP5	Bool	1.3	FALSE	Etat du cyc le précédent de l'étape 5.
X6	Bool	1.4	FALSE	Etat actuel de l'étape 6.
XP6	Bool	1.5	FALSE	Etat du cyc le précédent de l'étape 6.
X7	Bool	1.6	FALSE	Etat actuel de l'étape 7.
XP7	Bool	1.7	FALSE	Etat du cyc le précédent de l'étape 7.
TEMP		0.0		

Bloc : FB1 Commande de distribution dans une station d'hydrocarbure.

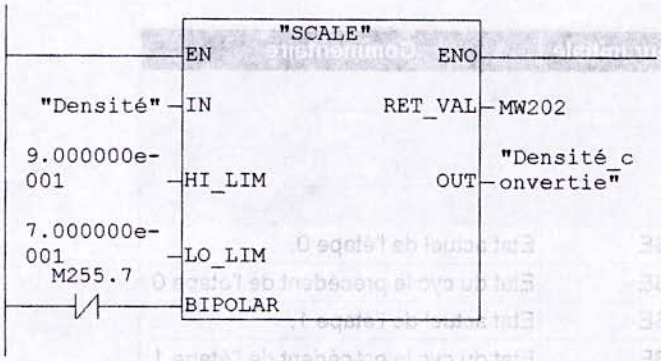
Réseau : 1 Conversion du débit.

La fonction FC105 est utilisée pour convertir l'entrée analogique qui est
un nombre entier en valeur réelle d'après la nature du capteur.
Les valeurs prisent sont arbitraires.
La valeur réelle est stockée dans le mémento MD0.



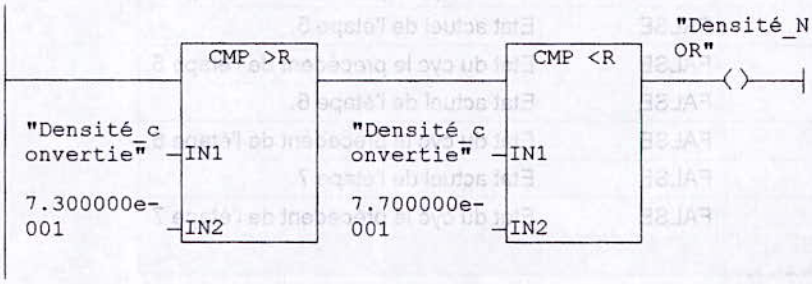
Réseau : 2	Conversion de la densité.

La fonction FC105 est utilisée pour convertir l'entrée analogique qui est un nombre entier en valeur réelle d'après la nature du capteur.
Les valeurs prisent sont arbitraires.
La valeur réelle est stockée dans le mémento MD4.



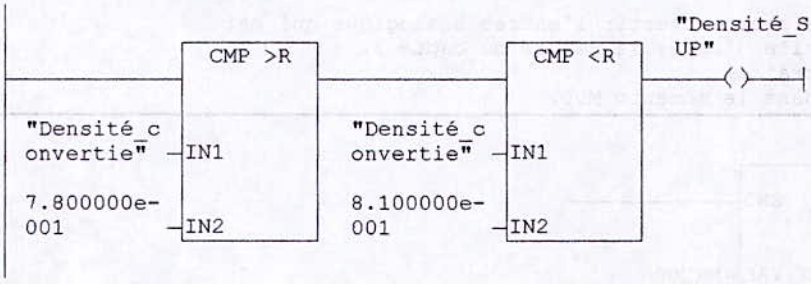
Réseau : 3	Détection de l'essence normale.

Deux comparateurs sont utilisés pour détecter la présence d'essence normale sur le pipeline.
La densité de l'essence normale est entre 0.730 et 0.770



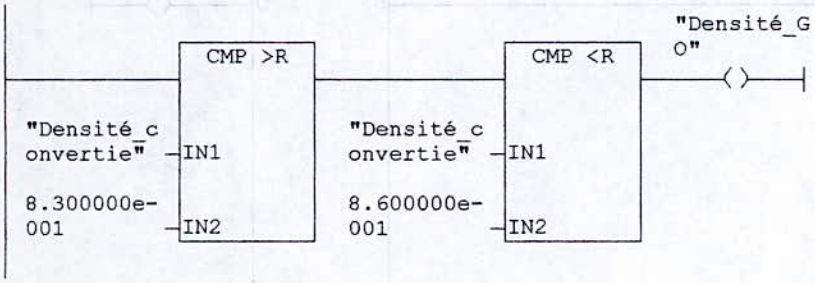
Réseau : 4	Détection de l'essence super.

Deux comparateurs sont utilisés pour détecter la présence d'essence super sur le pipeline.
La densité de l'essence super est entre 0.780 et 0.810



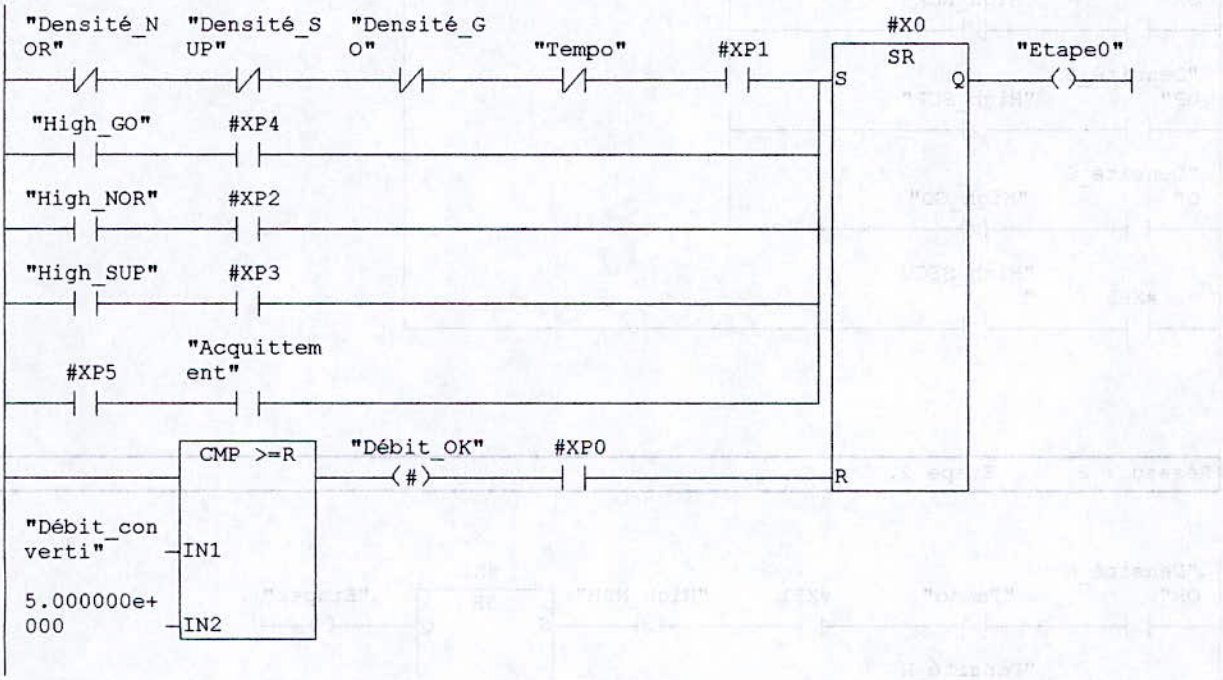
Réseau : 5	Détection de gasoil.

Deux comparateurs sont utilisés pour détecter la présence d'essence normale sur le pipeline.
La densité du gasoil est entre 0.830 et 0.860

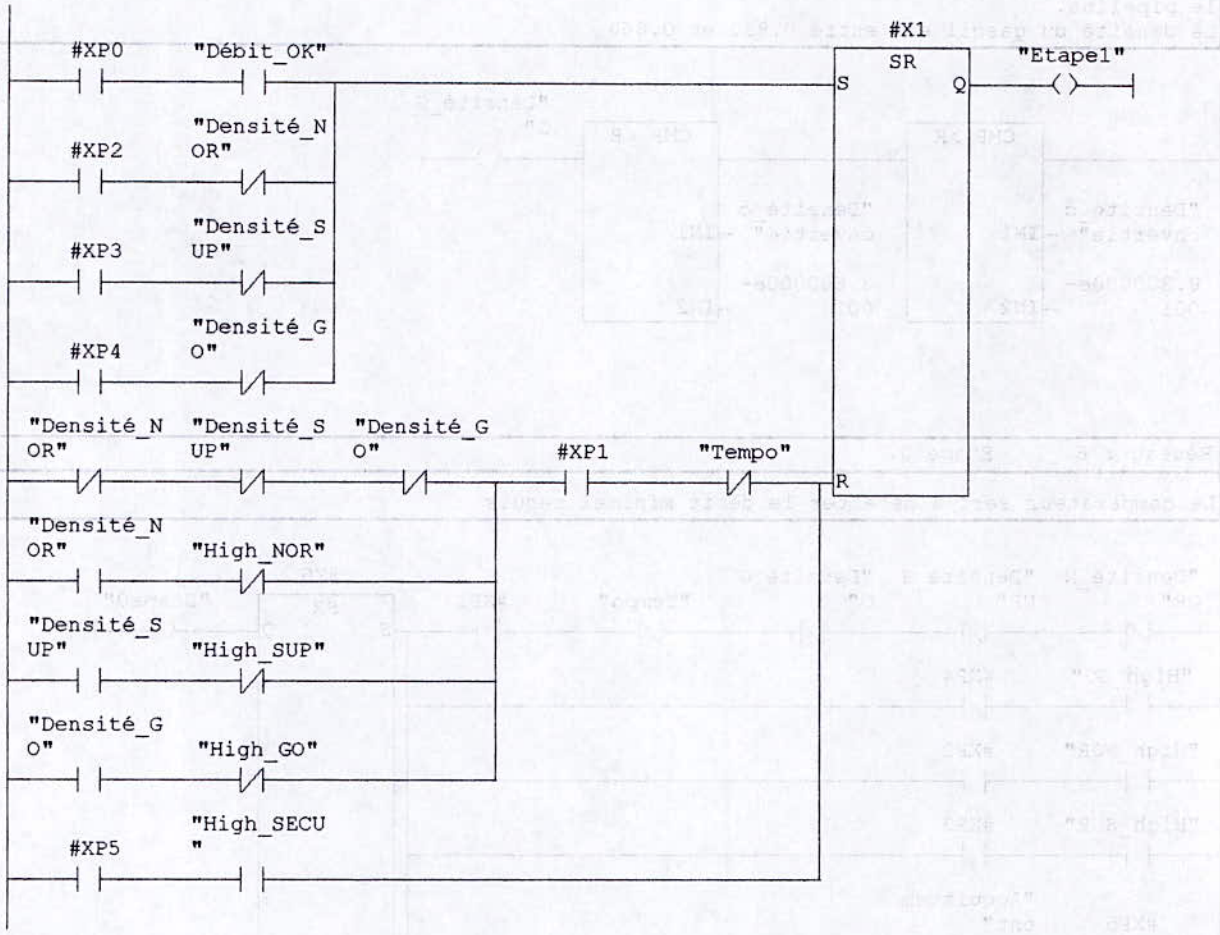


Réseau : 6	Etape 0.

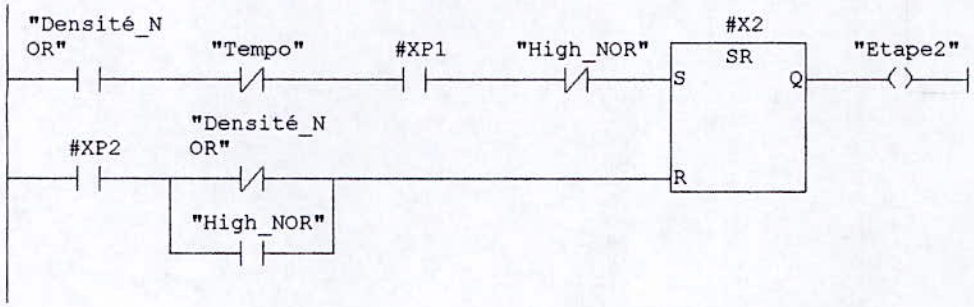
Le comparateur sert à détecter le débit minimal requis



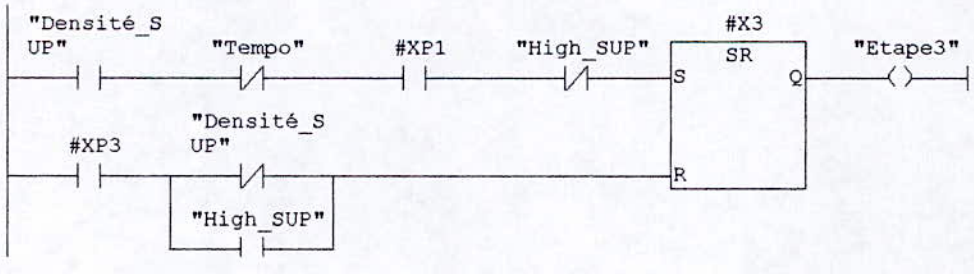
Réseau : 7 Etape 1.



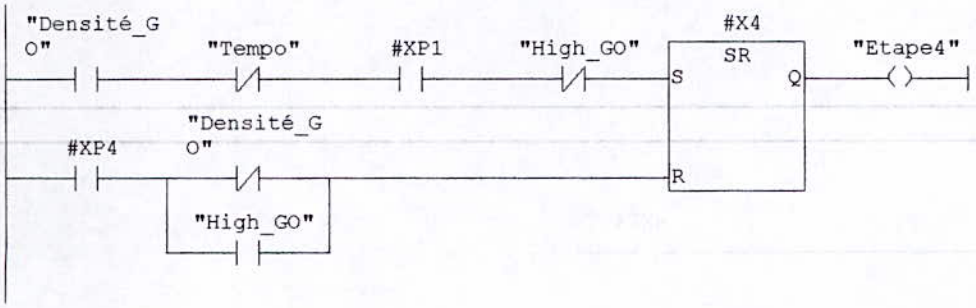
Réseau : 8 Etape 2.



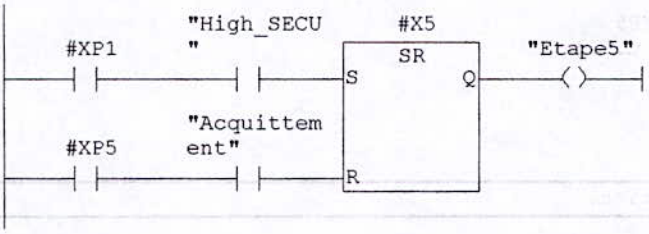
Réseau : 9 Etape 3.



Réseau : 10 Etape 4.

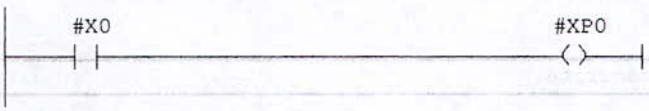


Réseau : 11 Etape 5.

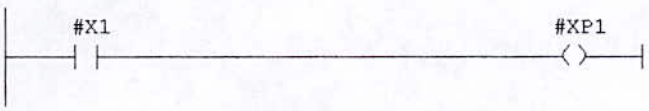


Réseau : 12

Les réseaux 12 à 17 actualisent les valeurs des états du cycle précedent des étapes.

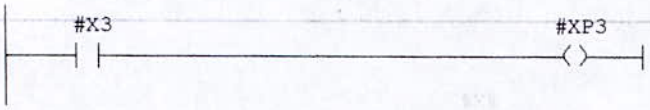


Réseau : 13



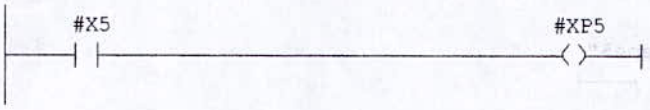
Réseau : 14

Réseau : 15

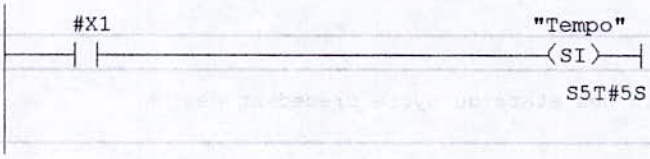


Réseau : 16

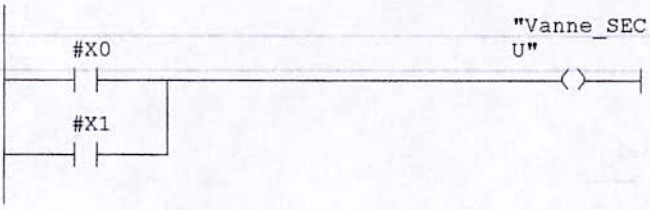
Réseau : 17



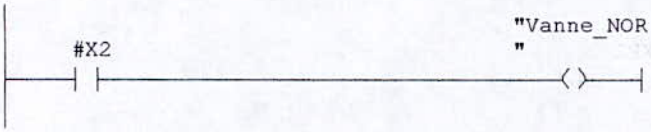
Réseau : 18 Lancement de la temporisation.



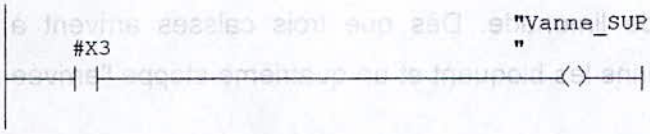
Réseau : 19 Ouverture de la vanne de sécurité.



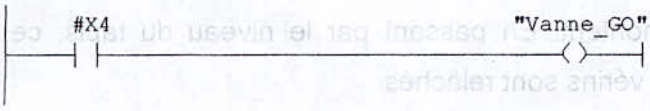
Réseau : 20 Ouverture de la vanne de l'essence normale.



Réseau : 21 Ouverture de la vanne de l'essence super.

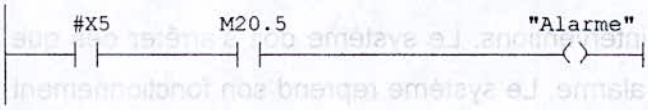


Réseau : 22 Ouverture de la vanne de gasoil.



Réseau : 23 Déclanchement de l'alarme.

L'alarme se déclanche à l'étape 5.
Le bit M20.5 est le bit du mémento de cadence qui offre une période de 1 seconde.



Réseau : 24 Mise à 1 de RB.

Activation du bit RB du mot d'état dont la sortie ENO sera l'image.

7. Machine à Dévisser les bouchons:

Un convoyeur transporte des caisses de limonade. Dès que trois caisses arrivent à l'intérieur de la machine à dévisser, trois vérins les bloquent et un quatrième stoppe l'arrivée d'autres caisses.

Au même moment, les têtes de dévissage descendent. Arrivées en bas, elles retiennent les bouchons et font tourner le moteur durant un certain temps permettant ainsi le dévissage total de tous les bouchons.

Après cette temporisation, les têtes remontent. En passant par le niveau du tapis, ce dernier s'ouvre jusqu'au fin de course et les vérins sont relâchés.

Une fois les têtes arrivées en haut et le tapis sorti complètement, les bouchons sont relâchés et une temporisation est lancée pour permettre à tous les bouchons de tomber sur le tapis.

Lorsque la temporisation est finie, le tapis se referme et fait tomber les bouchons dans un sac.

La machine possède une porte pour les interventions. Le système doit s'arrêter dès que cette porte est ouverte en déclanchant une alarme. Le système reprend son fonctionnement normal une fois la porte refermée.

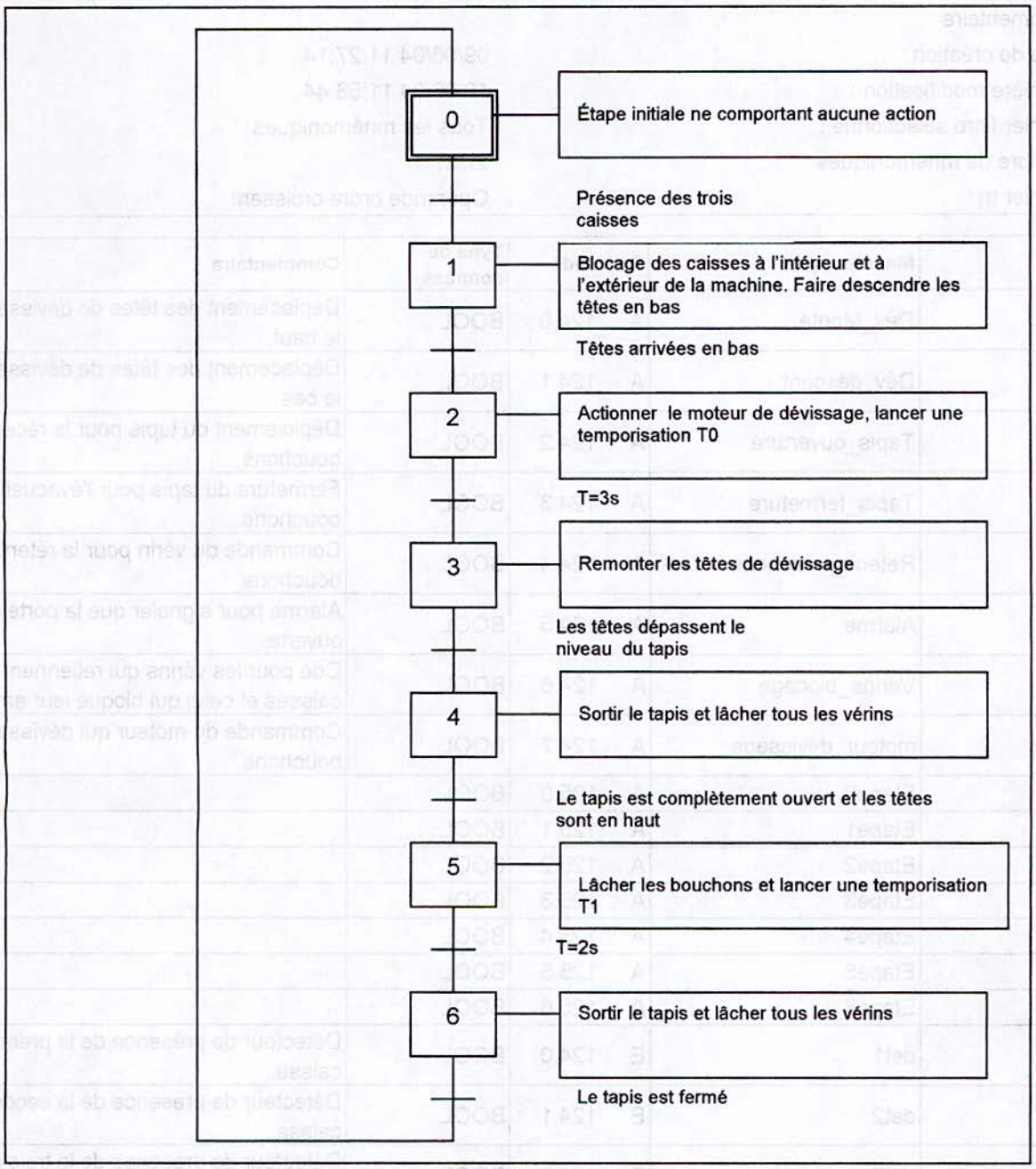


Figure 4.16 GRAFCET de la commande de la machine à dévisser les bouchons

Propriétés de la table des mnémoniques

Nom :	Mnémoniques
Commentaire :	
Date de création :	09/06/04 11:27:14
Dernière modification :	19/06/04 11:58:44
Dernier filtre sélectionné :	Tous les mnémoniques
Nombre de mnémoniques :	27/27
Dernier tri :	Opérande ordre croissant

Etat	Mnémonique	Opérande	Type de données	Commentaire
	Dév_Monte	A 124.0	BOOL	Déplacement des têtes de dévissage vers le haut.
	Dév_déscent	A 124.1	BOOL	Déplacement des têtes de dévissage vers le bas.
	Tapis_ouverture	A 124.2	BOOL	Déploiement du tapis pour la réception des bouchons.
	Tapis_fermeture	A 124.3	BOOL	Fermeture du tapis pour l'évacuation des bouchons.
	Retenir_bouchons	A 124.4	BOOL	Commande du vérin pour la rétention des bouchons.
	Alarme	A 124.5	BOOL	Alarme pour signaler que la porte est ouverte.
	Verins_blocage	A 124.6	BOOL	Cde pour les vérins qui retiennent les caisses et celui qui bloque leur arrivée.
	moteur_dévissage	A 124.7	BOOL	Commande du moteur qui dévisse les bouchons.
	Etape0	A 125.0	BOOL	
	Etape1	A 125.1	BOOL	
	Etape2	A 125.2	BOOL	
	Etape3	A 125.3	BOOL	
	Etape4	A 125.4	BOOL	
	Etape5	A 125.5	BOOL	
	Etape6	A 125.6	BOOL	
	det1	E 124.0	BOOL	Détecteur de présence de la première caisse.
	det2	E 124.1	BOOL	Détecteur de présence de la seconde caisse.
	det3	E 124.2	BOOL	Détecteur de présence de la troisième caisse.
	FC_haut	E 124.3	BOOL	Fin de course vers le haut du vérin qui déplace les têtes de dévissage.
	FC_bas	E 124.4	BOOL	Fin de course vers le bas du vérin qui déplace les têtes de dévissage.
	Det_Niveau_Tapis	E 124.5	BOOL	Détecteur de position au niveau du tapis.
	FC_Tapis_Sorti	E 124.6	BOOL	Fin de course pour l'ouverture du tapis.
	FC_Tapis_Rentré	E 124.7	BOOL	Fin de course pour la fermeture du tapis.
	Porte_fermée	E 125.0	BOOL	Détecteur d'ouverture de la porte.
	COMPLETE RESTART	OB 100	OB 100	Complete Restart
	tempo_dévissage	T 0	TIMER	Timer pour la temporisation de dévissage.
	tempo_Tapis	T 1	TIMER	Temporisation entre le lâchée des bouchons et la fermeture du tapis

OB100 - <offline>

"COMPLETE RESTART" Complete Restart
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 10:39:52
Interface : 15/02/96 16:51:10
Longueur (bloc/code /données locales) : 00142 00026 00020

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB100_EV_CLASS	Byte	0.0		16#13, Event class 1, Entering event state, Event logged in diagnostic buffer
OB100_STRTUP	Byte	1.0		16#81/82/83/84 Method of startup
OB100_PRIORITY	Byte	2.0		Priority of OB Execution
OB100_OB_NUMBR	Byte	3.0		100 (Organization block 100, OB100)
OB100_RESERVED_1	Byte	4.0		Reserved for system
OB100_RESERVED_2	Byte	5.0		Reserved for system
OB100_STOP	Word	6.0		Event that caused CPU to stop (16#4xxx)
OB100_STRT_INFO	DWord	8.0		Information on how system started
OB100_DATE_TIME	Date_And_Time	12.0		Date and time OB100 started

Bloc : OB100 "Complete Restart"

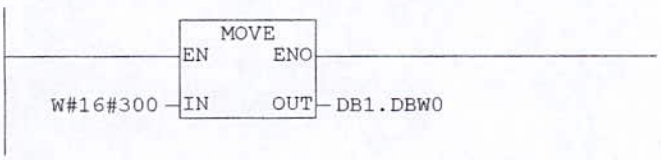
Réseau : 1

Mise à 1 d'un bit qui le restera pendant le déroulement du programme.

```
      SET
      =     M    255.7
```

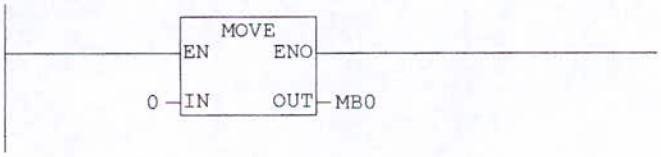
Réseau : 2

Initialisation des variables statiques du DB1.



Réseau : 3

Initialisation des bits des bascules pour les vérins.



OB1 - <offline>

""

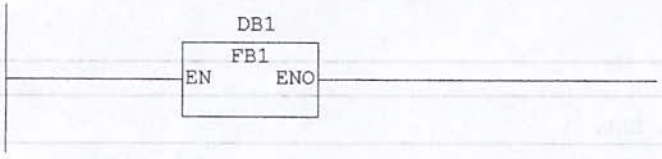
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 14:02:04
Interface : 15/02/96 16:51:12
Longueur (bloc/code /données locales) : 00148 00036 00026

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB1_EV_CLASS	Byte	0.0		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1	Byte	1.0		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
OB1_PRIORITY	Byte	2.0		Priority of OB Execution
OB1_OB_NUMBR	Byte	3.0		1 (Organization block 1, OB1)
OB1_RESERVED_1	Byte	4.0		Reserved for system
OB1_RESERVED_2	Byte	5.0		Reserved for system
OB1_PREV_CYCLE	Int	6.0		Cycle time of previous OB1 scan (milliseconds)
OB1_MIN_CYCLE	Int	8.0		Minimum cycle time of OB1 (milliseconds)
OB1_MAX_CYCLE	Int	10.0		Maximum cycle time of OB1 (milliseconds)
OB1_DATE_TIME	Date_And_Time	12.0		Date and time OB1 started

Bloc : OB1 "Main Program Sweep (Cycle)"

Réseau : 1

Appel du bloc fonctionnel FB1 en lui associant le bloc de données DB1.



FB1 - <offline>

""

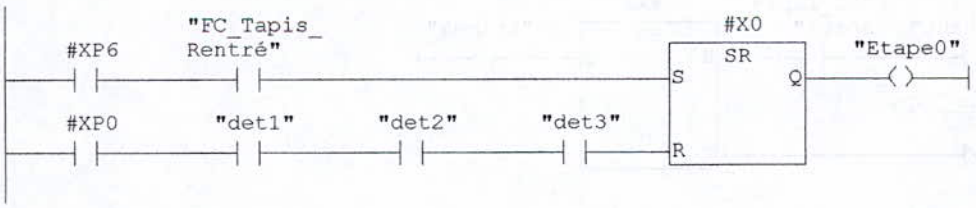
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 11/06/04 13:58:55
Interface : 09/06/04 10:34:37
Longueur (bloc/code /données locales) : 00564 00396 00000

Nom	Type de données	Adresse	Valeur initiale	Commentaire
IN		0.0		
OUT		0.0		
IN_OUT		0.0		
STAT		0.0		
X0	Bool	0.0	FALSE	Etat actuel de l'étape 0.
XP0	Bool	0.1	FALSE	Etat précédent de l'étape 0.
X1	Bool	0.2	FALSE	Etat actuel de l'étape 1.
XP1	Bool	0.3	FALSE	Etat précédent de l'étape 1.
X2	Bool	0.4	FALSE	Etat actuel de l'étape 2.
XP2	Bool	0.5	FALSE	Etat précédent de l'étape 2.
X3	Bool	0.6	FALSE	Etat actuel de l'étape 3.
XP3	Bool	0.7	FALSE	Etat précédent de l'étape 3.
X4	Bool	1.0	FALSE	Etat actuel de l'étape 4.
XP4	Bool	1.1	FALSE	Etat précédent de l'étape 4.
X5	Bool	1.2	FALSE	Etat actuel de l'étape 5.
XP5	Bool	1.3	FALSE	Etat précédent de l'étape 5.
X6	Bool	1.4	FALSE	Etat actuel de l'étape 6.
XP6	Bool	1.5	FALSE	Etat précédent de l'étape 6.
TEMP		0.0		

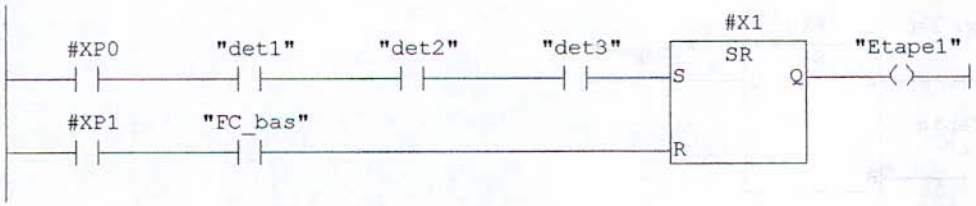
Bloc : FB1 Dévisseuse

Bloc fonctionnel gérant la dévisseuse.

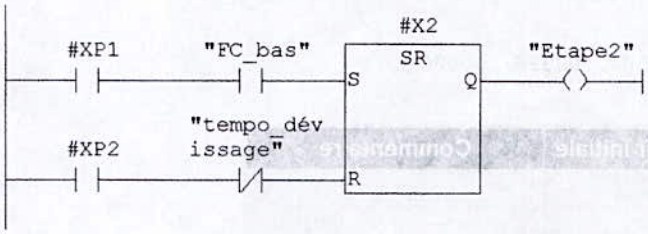
Réseau : 1 Etape 0.



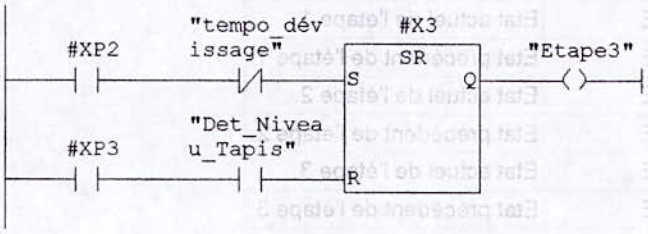
Réseau : 2 Etape 1.



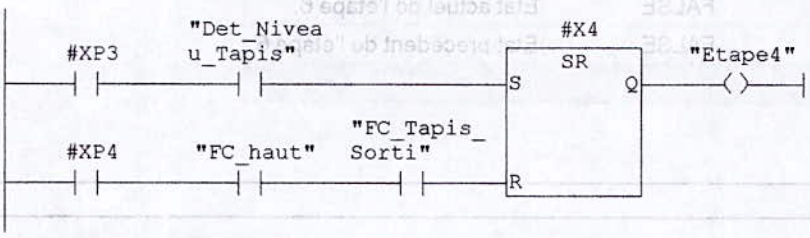
Réseau : 3 Etape 2.



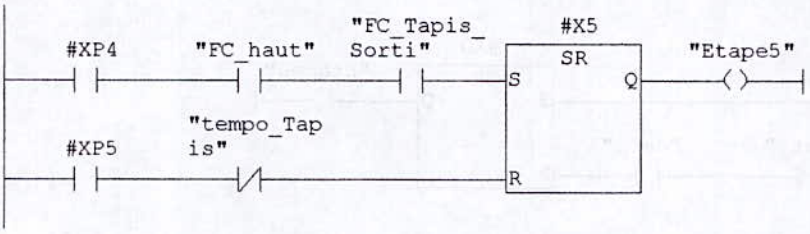
Réseau : 4 Etape 3.



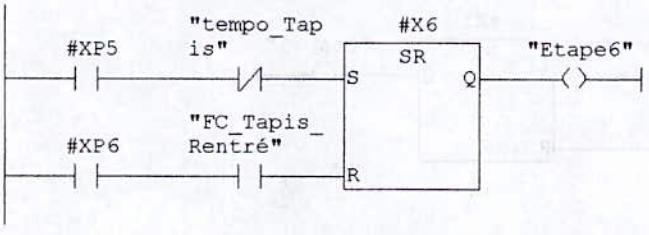
Réseau : 5 Etape 4.



Réseau : 6 Etape 5.

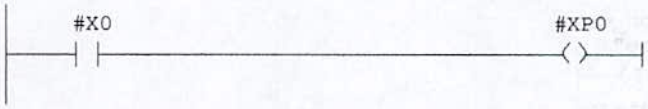


Réseau : 7 Etape 6.

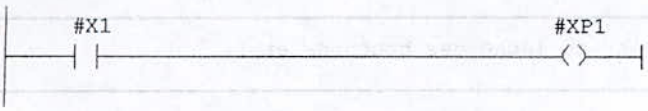


Réseau : 8

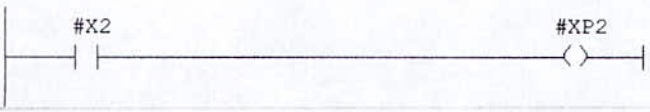
Les réseaux de 8 à 14 actualisent les états des variables statiques.



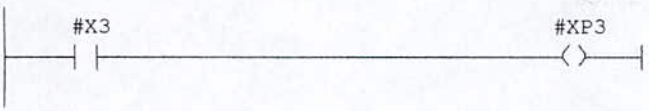
Réseau : 9



Réseau : 10

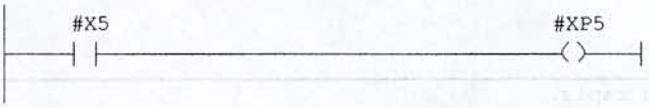


Réseau : 11



Réseau : 12

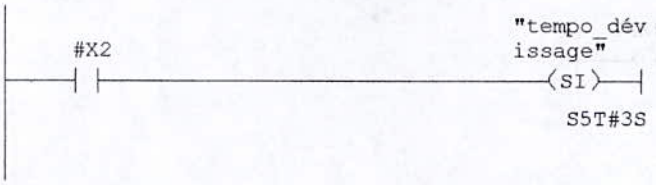
Réseau : 13



Réseau : 14

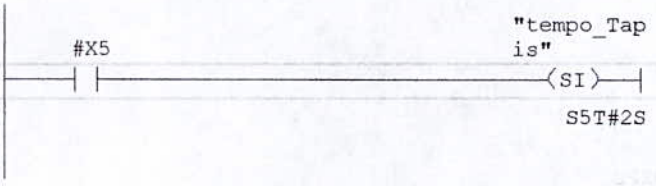
Réseau : 15

Lancement de la temporisation de dévissage à l'étape 2.



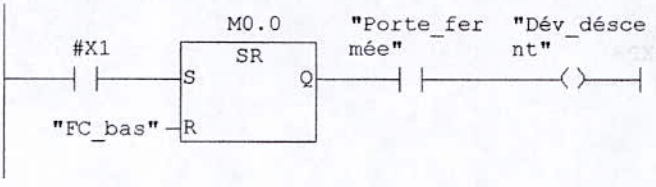
Réseau : 16

Lancement de la temporisation entre le moment du lâché des bouchons et la fermeture du tapis à l'étape 5.



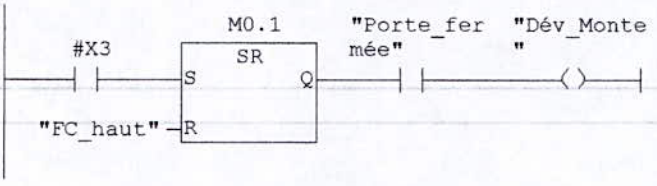
Réseau : 17 Commande du vérins déplaçant les têtes de dévissages.

Le vérin monte à l'étape 1 et s'arrête au niveau du fin de course.
Si la porte est ouverte le verin est stoppé.



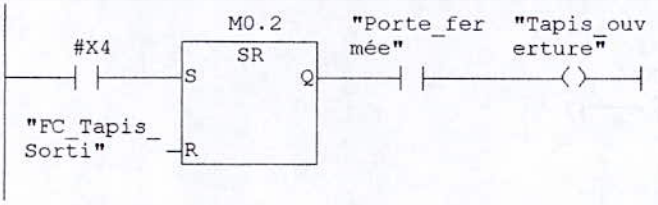
Réseau : 18 Commande du vérins déplaçant les têtes de dévissages.

Le vérin descend à l'étape 3 et s'arrête au niveau du fin de course.
Si la porte est ouverte le verin est stoppé.



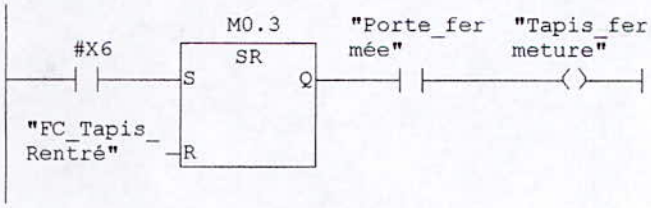
Réseau : 19 Commande de l'ouverture du tapis.

Le tapis s'ouvre à l'étape 4 et s'arrête au niveau du fin de course.
Si la porte est ouverte le tapis s'arrête



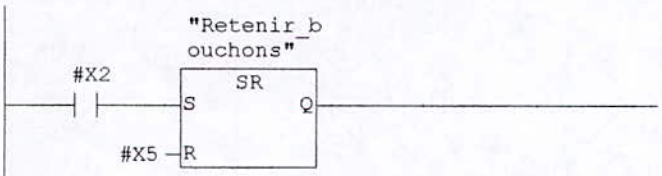
Réseau : 20	Commande de la fermeture du tapis.

Le tapis est fermé à l'étape 6 jusqu'au fin de course.
Si la porte est ouverte le tapis s'arrête.



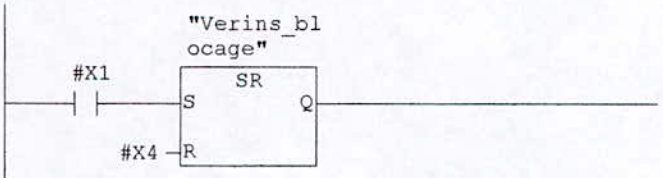
Réseau : 21	Rétention des bouchons.

Les bouchons sont retenus à l'étape 2 et sont lâchés à l'étape 5.



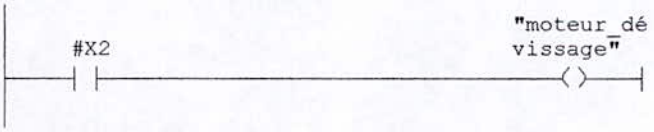
Réseau : 22	Blocage des caisses

Les trois vérins qui retiennent les caisses et celui qui arrête leur arrivée
obéïssent aux mêmes conditions:
Les vérins sortent à l'étape 1 et rentrent à l'étape 4.



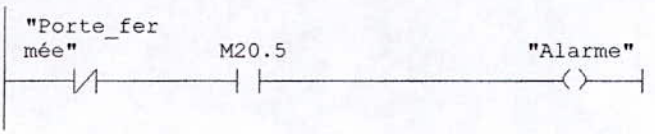
Réseau : 23	Moteur de dévissage

Les bouchons sont dévissés à l'étape 2.

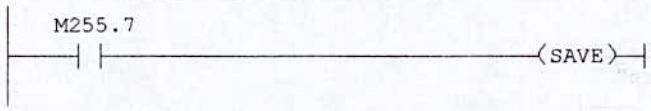


Réseau : 24	Alarme.

Une alarme est déclanchéesi la porte est ouverte.
Le bits M20.5 est un bit du mémento de cadence qui a été configuré
dans le matériel, il offre un signal d'une période de 1 seconde.



Réseau : 25 Activation de ENO
Mise à 1 du bit RB du mot d'état dont la sortie ENO sera l'image.



8. Gestion parking :

L'exemple suivant exploite la fonction intégrée compteur A/B et ses sorties spéciales pour un automatisme qui n'est pas industriel.

Fonctionnement :

Un parking est constitué de deux niveaux dans lesquels il y a un nombre de places limité.
Le but est de compter le nombre de voitures entrantes et sortantes dans chaque niveau pour connaître le nombre de places occupées.
Si un niveau est complet, une sortie est activée pour le signaler.

Remarques :

- La fonction intégrée compteur A/B doit être activée depuis la configuration matérielle.
- Les DB utilisés lors de l'appel du bloc de comptage doivent correspondre aux DB de la configuration matérielle.
- Un front montant doit être appliqué à l'entrée de validation du comparateur.

Propriétés de la table des mnémoniques

Nom :	Mnémoniques
Commentaire :	
Date de création :	19/06/04 12:10:00
Dernière modification :	19/06/04 11:14:21
Dernier filtre sélectionné :	Tous les mnémoniques
Nombre de mnémoniques :	18/18
Dernier tri :	Opérande ordre croissant

Etat	Mnémonique	Opérande	Type de données	Commentaire
	Floor1_Is_Full	A 124.0	BOOL	Signale que le permier niveau est plein.
	Floor2_Is_Full	A 124.1	BOOL	Signale que le second niveau est plein.
	Enable_Count1	E 124.0	BOOL	Active le compteur du permier niveau.
	Enable_Count2	E 124.1	BOOL	Active le compteur du second niveau.
	Reset1	E 124.2	BOOL	Réinitialise le compteur du premier niveau à 0.
	Reset2	E 124.3	BOOL	Réinitialise le compteur du second niveau à 0.
	Floor1_Car_IN	E 126.0	BOOL	Détecteur d'entrée d'une voiture au niveau 1.
	Floor1_Car_OUT	E 126.1	BOOL	Détecteur de sortie d'une voiture au niveau 1.
	Floor2_Car_IN	E 126.2	BOOL	Détecteur d'entrée d'une voiture au niveau 2.
	Floor2_Car_OUT	E 126.3	BOOL	Détecteur de sortie d'une voiture au niveau 2.
	Count1	MD 0	DINT	Nombre de voitures au niveau 1.
	Comp1	MD 4	DINT	Indique la valeur limite actuelle au niveau 1.
	Count2	MD 8	DINT	Nombre de voitures au niveau 2.
	Comp2	MD 12	DINT	Indique la valeur limite actuelle au niveau 2.
	Limit_1	MD 16	DINT	Nombre maximum de voitures au niveau 1.
	Limit_2	MD 20	DINT	Nombre maximum de voitures au niveau 2.
	COMPLETE RESTART	OB 100	OB 100	Complete Restart
	HSC_A_B	SFB 38	SFB 38	Counter A/B (Integrated Function, CPU 314 IFM)

OB100 - <offline>

"COMPLETE RESTART" Complete Restart
Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 19/06/04 10:17:25
Interface : 15/02/96 16:51:10
Longueur (bloc/code /données locales) : 00248 00130 00026

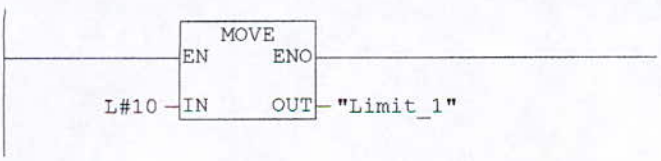
Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB100_EV_CLASS	Byte	0.0		16#13, Event class 1, Entering event state, Event logged in diagnostic buffer
OB100_STRTUP	Byte	1.0		16#81/82/83/84 Method of startup
OB100_PRIORITY	Byte	2.0		Priority of OB Execution
OB100_OB_NUMBR	Byte	3.0		100 (Organization block 100, OB100)
OB100_RESERVED_1	Byte	4.0		Reserved for system
OB100_RESERVED_2	Byte	5.0		Reserved for system
OB100_STOP	Word	6.0		Event that caused CPU to stop (16#4xxx)
OB100_STRT_INFO	DWord	8.0		Information on how system started
OB100_DATE_TIME	Date_And_Time	12.0		Date and time OB100 started

Bloc : OB100 "Complete Restart"

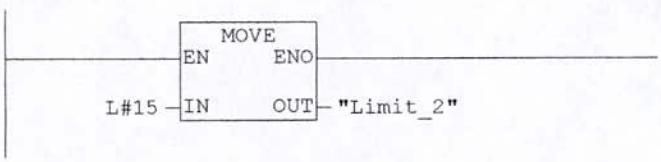
Réseau : 1

```
SET
=     M    255.7
```

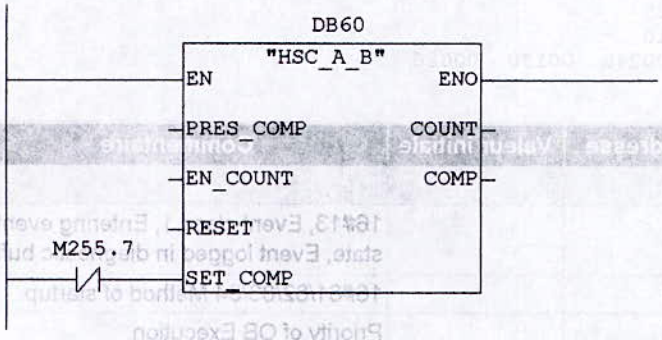
Réseau : 2 Affectation du nombre de voitures max au niveau 1.



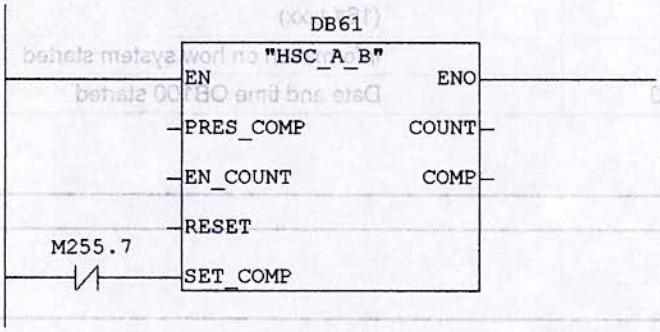
Réseau : 3 Affectation du nombre de voitures max au niveau 2.



Réseau : 4 Appliquer un signal nul sur l'entrée SET_COMP du compteur B.



Réseau : 5 Appliquer un signal nul sur l'entrée SET_COMP du compteur B.



OB1 - <offline>

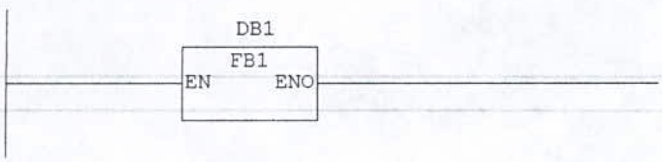
""

Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 19/06/04 08:17:23
Interface : 15/02/96 16:51:12
Longueur (bloc/code /données locales) : 00148 00036 00026

Nom	Type de données	Adresse	Valeur initiale	Commentaire
TEMP		0.0		
OB1_EV_CLASS	Byte	0.0		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
OB1_SCAN_1	Byte	1.0		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
OB1_PRIORITY	Byte	2.0		Priority of OB Execution
OB1_OB_NUMBR	Byte	3.0		1 (Organization block 1, OB1)
OB1_RESERVED_1	Byte	4.0		Reserved for system
OB1_RESERVED_2	Byte	5.0		Reserved for system
OB1_PREV_CYCLE	Int	6.0		Cycle time of previous OB1 scan (milliseconds)
OB1_MIN_CYCLE	Int	8.0		Minimum cycle time of OB1 (milliseconds)
OB1_MAX_CYCLE	Int	10.0		Maximum cycle time of OB1 (milliseconds)
OB1_DATE_TIME	Date_And_Time	12.0		Date and time OB1 started

Bloc : OB1 "Main Program Sweep (Cycle)"

Réseau : 1



FB1 - <offline>

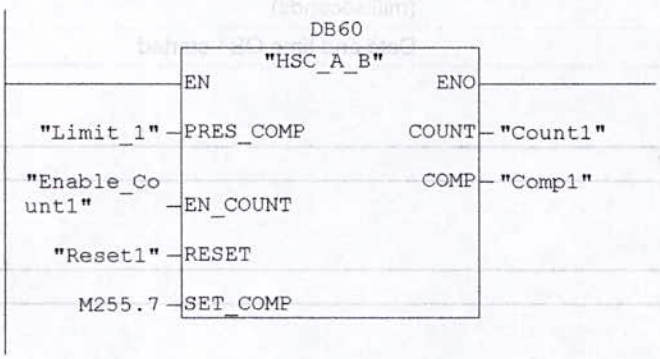
""

Nom : **Famille :**
Auteur : **Version :** 0.1
Version de bloc : 2
Horodatage Code : 19/06/04 10:14:41
Interface : 19/06/04 07:48:14
Longueur (bloc/code /données locales) : 00306 00210 00006

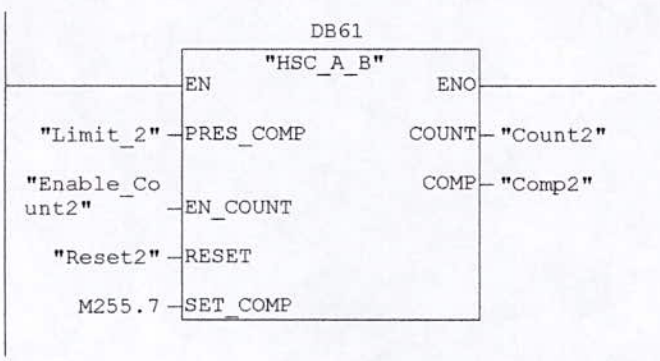
Nom	Type de données	Adresse	Valeur initiale	Commentaire
IN		0.0		
OUT		0.0		
IN_OUT		0.0		
STAT		0.0		
TEMP		0.0		

Bloc : FB1

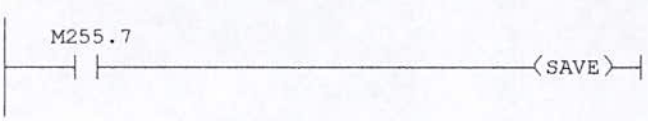
Réseau : 1



Réseau : 2



Réseau : 3



Conclusion :

Les automates programmables de la série S7-300 sont de machines dotées d'une grande flexibilité, ce qui permet leur utilisation dans de nombreux systèmes automatisés.

SIEMENS offre un logiciel très puissant pour la mise en œuvre de solutions à base d'API. En effet, STEP 7 est un logiciel qui permet d'exploiter de manière optimale les fonctionnalités des différentes CPU de la gamme.

Ce projet nous a permis de nous familiariser avec les automates programmables ainsi que leurs langages de programmation qui obéissent à des règles spécifiques.

Les circuits réalisés permettent de simuler les capteurs et les détecteurs qui pourraient être reliés à l'automate. Le test des programmes est ainsi grandement facilité.

La supervision est un élément inséparable pour les systèmes automatisés. Le pack de logiciels fourni avec la CPU ne comporte pas de logiciel de supervision. Un outil comme WinCC aurait permis de créer des systèmes plus souples et plus parlants.

Un autre point qu'il serait judicieux de traiter est la mise en réseau d'API, cela permettrait d'avoir des performances accrues tout en diminuant le câblage et donc le temps de diagnostic et de dépannage.

En se familiarisant avec cette CPU, nous nous sommes rendus compte que sa puissante était limitée, ce qui ne nous a pas permis d'implémenter des commandes plus complexes.

Les questions à se poser sont :

Quelle est la puissance nécessaire pour l'implémentation de ces commandes ?et jusqu'à quelle précision peut-on arriver avec la technologie actuelle des automates programmables ?

Bibliographie

[1] G. MICHEL, « Les A.P.I. Architecture et application des automates programmables industriels », Edition DUNOD ,1987.

[2] A. SIMON, « Automate Programmable Industriel, Niveau 1 », Edition L'ELAN, 1991.

[3] M. BERTRAND, «Automates programmables industriels », Techniques de l'ingénieur , Vol. S 8 015.

[4] P. JARGOT, « Langages de programmation pour API. Norme IEC 1131-3 », Techniques de l'ingénieur , Vol. S 8 030.

[5] D. DUPONT, D. DUBOIS, « Réalisation technologique du GRAFCET », Techniques de l'ingénieur , Vol. S 8 032.

Manuels :

[6] SCHNEIDER ELECTRIC, « Automates Nano et Plate-forme d'automatisme Micro »,1999.

[7] SIEMENS, « Mise en route STEP7 V5.2.Getting Started », Réf. 6ES7810-4CA06-8CA0,SIMATIC, 2002.

[8] SIEMENS, « Programmer avec STEP7 V5.2 », Réf. 6ES7810-4CA06-8CA0, SIMATIC, 2002.

[9] SIEMENS , «Langage CONT pour SIMATIC S7-300/400 », Réf. 6ES7810-4CA06-8CR0, SIMATIC,2002.

[10] SIEMENS , «Langage LOG pour SIMATIC S7-300/400 », Réf. 6ES7810- 4CA06-8CR0, SIMATIC,2002.

[11] SIEMENS, «Langage LIST pour SIMATIC S7-300/400 », Réf. 6ES7810-4CA06-8CR0, SIMATIC, 2002.

[12] SIEMENS, «Automate programmable S7-300, Fonctions intégrées, CPU 312 IFM/314 IFM », Réf. EWA 4NEB 710 6058-03a Edition 2, SIMATIC, 2000.

[13] SIEMENS, « Liste des opérations S7-300, CPU 312 IFM, 314 IFM, 313, 314, 315, 315-2 DP, 316-2 DP, 318-2 », Réf. 6ES7 398-8AA03-8CN0 Edition 2, SIMATIC,2000.

[14] SIEMENS, « Automate programmable S7-300, Installation et configuration - caractéristiques techniques des CPU», Réf. 6ES7 398-8AA03-8CA0 Edition 2, SIMATIC,1999.

[15] SIEMENS, « Automate programmable S7-300 Caractéristiques des CPU, CPU 312 IFM–318-2 DP », Réf. 6ES7398-8FA10-8CA0, SIMATIC, 2001.

[16] SIEMENS, «Automate programmable S7-300, Getting Started, Premières étapes pour montage et mise en service », Réf. A5E00069341 01, SIMATIC, 2000.

Annexe

Sommaire :

Le langage CONT

Le langage LOG

Le langage LIST

Sommaire des tableaux :

Sommaire des figures :

Le langage CONT

1. opération possible avec CONT :

1.1. Opération sur BIT :

1.1.1. Contact à fermeture :

Décrit précédemment, il reflète l'état de la variable associée.

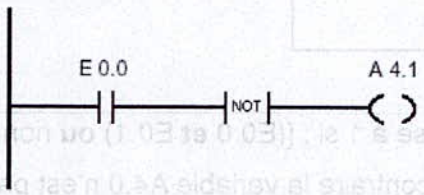
1.1.2. Contact à ouverture :

Reflète l'inverse de l'état de la variable associée.

1.1.3. Inverseur de résultat logique :

Symbole : -|NOT|-

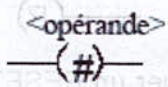
Inverse le résultat logique.



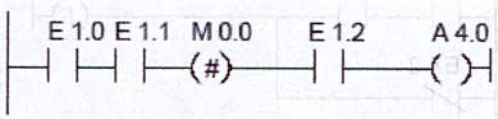
L'inverseur fait que A4.1 n'est à 1 que si E0.0 est à 0.

Figure A.1.1 Inverseur logique avec CONT.

1.1.4. Connecteur :



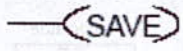
Cet élément sert à sauvegarder le résultat logique dans la variable qui lui est associée (opérande). Sa mise en série avec d'autres contacts n'influe pas sur la logique du résultat.



Le résultat de E1.0 **et** E1.1 est sauvegardé dans M0.0
A4.0 est égale à : E1.0 **et** E1.1 **et** E1.2

Figure A.1.2 Connecteur

1.1.5. Sauvegarder le résultat logique dans le bit RB :

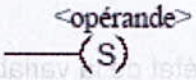


Cet élément permet de sauvegarder le résultat logique dans le bit RB du mot d'état, le bit /PI (première interrogation) n'est pas affecté.

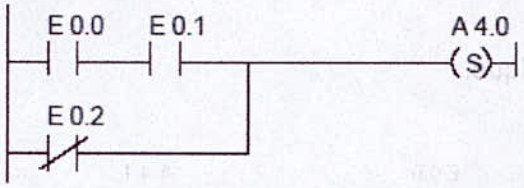
Ainsi une opération ET dans le réseau suivant prendra en considération l'état du bit RB.

1.2. Utilisation de SET et RESET :

1.2.1. Mise à 1 :



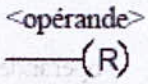
Cet élément permet d'effectuer un SET sur l'opérande, celui-ci est mis à 1 si le résultat logique est 1, mais contrairement à une bobine classique, l'opérande n'est pas changé si le RLG tombe à 0.



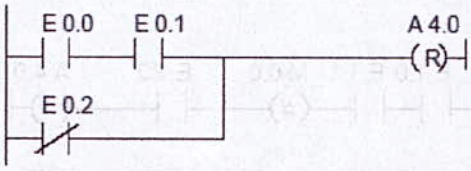
A4.0 est mise à 1 si : [(E0.0 **et** E0.1) **ou** non(E0.2)]=1
Dans le cas contraire la variable A4.0 n'est pas affectée.

Figure A.1.3 SET avec CONT.

1.2.2. Mise à 0 :



Cet élément permet d'effectuer un RESET sur l'opérande, celui-ci est mis à 0 si le résultat logique est 1, mais contrairement à une bobine classique, l'opérande n'est pas changé si le RLG tombe à 0.



A4.0 est mise à 0 si : [(E0.0 **et** E0.1) **ou** non(E0.2)]=1
Dans le cas contraire la variable A4.0 n'est pas affectée.

Figure A.1.4 RESET avec CONT

1.2.3. Bascule mise à 1, mise à 0 (SR) :

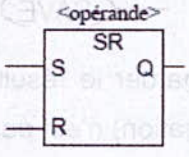


Table de vérité de la bascule SR :

S	R	Q
1	0	1
0	1	0
0	0	Statu quo
1	1	0 et <opérande> est mise à 0

Tableau A.1.1 table de vérité d'une bascule SR.

- <opérande> représente la variable qui doit être mise à 1 ou à 0.
- Q donne l'état de signal de <opérande>

1.2.4. Bascule mise à 0, mise à 1 (RS) :

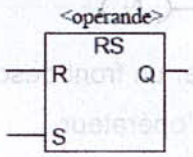


Table de vérité de la bascule RS:

S	R	Q
1	0	1
0	1	0
0	0	Statu quo
1	1	1 et <opérande> est mise à 1

Tableau A.1.2 table de vérité d'une bascule RS.

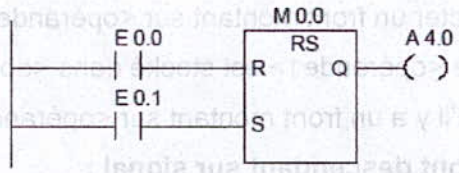
- <opérande> représente la variable qui doit être mise à 1 ou à 0.
- Q donne l'état de signal de <opérande>.

Remarque :

Ces deux types de bascules sont en fait des bascules RS à priorité :

- La bascule SR donne la priorité à R.
- La bascule RS donne la priorité à S.

Exemple :



Si E0.0 = 1 et E0.1 = 0 alors M0.0 est mis à 0. Comme Q donne l'état de M0.0 alors A4.0 sera mise à 0 aussi.

Si E0.0 = 0 et E0.1 = 1 alors M0.0 est mis à 1. Si les deux entrées sont à 1 simultanément alors M0.0 est mis à 1. Comme Q donne l'état de M0.0 alors A4.0 sera mise à 1 aussi.

Figure A1.5 Bascule RS.

1.3. Détection de front :

1.3.1. Détection d'un front montant sur RLG:

<opérande1>
—(P)

Cette opération permet de détecter un front montant sur le RLG, ce passage de 0 à 1 génère un 1 sur la sortie de l'opérateur.

L'opérateur compare le résultat logique (RLG) actuel à celui stocké dans le mémento de front (<opérande 1>).

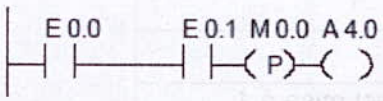
1.3.2. Détection d'un front descendant sur RLG:

<opérande1>
—(N)

Cette opération permet de détecter un front descendant sur le RLG, ce passage de 1à 0 génère un 1 sur la sortie de l'opérateur.

L'opérateur compare le résultat logique (RLG) actuel à celui stocké dans le mémento de front (<opérande 1>).

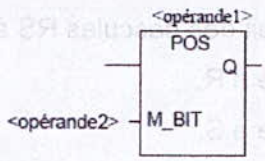
Exemple :



La sortie A4.0 se met à 1 s'il y a un front montant sur le RLG qui est égal à E0.0 **ET** E0.1.
L'état précédent est mémorisé dans M0.0

Figure A1.6 Détection d'un front montant avec CONT.

1.3.3. Détection de front montant sur signal :

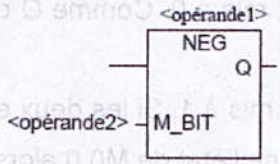


Ce bloc permet de détecter un front montant sur <opérande1>.

Le résultat précédent de <opérande1> est stocké dans <opérande2>.

La sortie Q se met à 1 s'il y a un front montant sur <opérande1>.

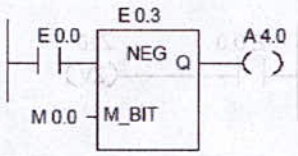
1.3.4. Détection de front descendant sur signal :



Ce bloc permet de détecter un front descendant sur <opérande1>.

Le résultat précédent de <opérande1> est stocké dans <opérande2>.

La sortie Q se met à 1 s'il y a un front descendant sur <opérande1>.

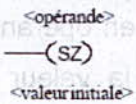


La sortie A4.0 se met à un si : E0.0 est à 1 (activation du bloc) et s'il y a un front descendant sur le signal d'entrée E0.3. L'état précédent est mémorisé dans M0.0

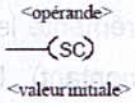
Figure A1.7 détection de front montant sur signal.

1.4. Opération sur compteurs :

1.4.1. Initialiser un compteur :



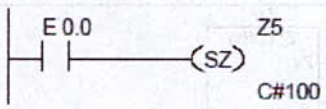
ou



Paramètre	Type de données	Zone de mémoire	description
Opérande	COUNTER	Z	Indique le numéro de compteur qui doit être initialisé
Valeur initiale	WORD	E,A,M,D,L	Valeur entre 0 et 999. Le format d'écriture doit être C#<valeur>. Ex :C#100

Tableau A1.2 paramètres d'initialisation d'un compteur.

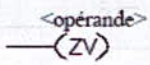
Exemple :



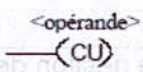
Le compteur Z5 est initialisé avec la valeur 100 si E0.0 passe de 0 à 1 (front montant), dans le cas contraire la valeur du compteur n'est pas affectée.
L'écriture sous format BCD est indiquée par C#.

Figure A1.8 Initialisation d'un compteur.

1.4.2. Incrémenter un compteur :

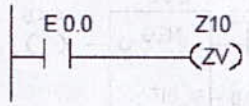


ou



Cet élément incrémente le compteur qui est mis en opérande si le RLG passe de 0 à 1 (front montant). Dans le cas contraire la valeur du compteur reste inchangée.

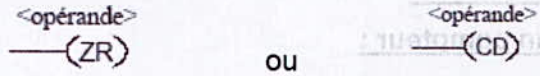
Si la valeur du compteur est de 999 et qu'il y a un front montant sur le RLG, la valeur du compteur ne bouge pas.



Le compteur Z10 est incrémenté de 1 s'il y a un front montant sur E0.0 et si sa valeur est différente de 999. Dans tous les autres cas sa valeur n'est pas changée.

Figure A1.9 Incrémenter compteur.

1.4.3. Décrémenter un compteur :



Cet élément décrémente le compteur qui est mis en opérande si le RLG passe de 0 à 1 (front montant). Dans le cas contraire la valeur du compteur reste inchangée.

Si la valeur du compteur est de 0 et qu'il y a un front montant sur le RLG, la valeur du compteur ne bouge pas.

1.4.4. Blocs de gestion de compteurs :

Il existe aussi des blocs qui gèrent les compteurs :

Type de compteur	Notation SIMATIC	Notation internationale
Bloc compteur incrémental/décrémental	Z n° ZAEHLER ZV Q ZR S ZW DUAL DEZ R	C n° S_CUD CU Q CD S PV CV CV_BCD R
Bloc compteur incrémental	Z n° Z_VORW ZV Q S ZW DUAL DEZ R	C n° S_CU CU Q S PV CV CV_BCD R
Bloc compteur décrémental	Z n° Z_RUECK ZR Q S ZW DUAL DEZ R	C n° S_CD CD Q S PV CV CV_BCD R

Tableau A1.3 Blocs de gestion de compteurs.

Les paramètres de ces blocs sont :

Paramètre	Type	Description
n°	COUNTER	Numéro d'identification du compteur
ZV	BOOL	Entrée d'incrémentation
ZR	BOOL	Entrée de décrémentation
S	BOOL	Entrée d'initialisation du compteur
ZW	WORD	Valeur initiale du compteur qui doit être comprise entre 0 et 999 et écrite sous format BCD (C#xxx)
R	BOOL	Entrée de remise à zéro
Q	BOOL	Etat du compteur
DUAL	WORD	Valeur de comptage en cours (format binaire)
DEZ	WORD	Valeur de comptage en cours (format DCB)

Tableau A1.4 paramètres des blocs de gestion de compteurs.

Remarque :

Les entrées/sorties sur le bloc en notation internationale sont équivalentes à celles en notation germanique.

1.5. Opération sur timers :

Les opérations de temporisation sont très importantes lors de la programmation d'un automate programmable. Nous décrirons ici les différents types de temporisation existants dans STEP7.

STEP 7 peut gérer jusqu'à 256 temporisations, mais le nombre de temporisations que l'on peut utiliser concrètement dépend de la CPU.

Pour chaque temporisation, un registre de 16 bits est alloué.

Il faut savoir que pour démarrer une temporisation il faut toujours qu'il y ait un changement d'état sur l'entrée d'activation (ce n'est pas l'état de l'entrée d'activation qui lance la temporisation mais un front montant sur celle-ci).

1.5.1. Types de temporisations possibles :

Le tableau suivant représente les temporisations possibles sous STEP7 :

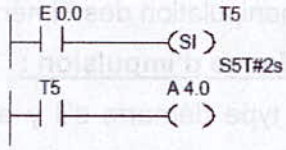
Type de temporisation	Lancement avec une bobine	Blocs sous forme SIMATIC	Blocs sous formes internationales
Temporisation sous forme d'impulsion	<opérande> (SI) (SP) <valeur de temps>	T n° S_IMPULS S TW Q DUAL DEZ R	T n° S_PULSE S TV Q BI BCD R
Temporisation sous forme d'impulsion prolongée	<opérande> (SV) (SE) valeur de temps	T n° S_VIMP S TW Q DUAL DEZ R	T n° S_PEXT S TV Q BI BCD R
Temporisation sous forme de retard à la montée	<opérande> (SE) (SD) valeur de temps	T n° S_EVERZ S TW Q DUAL DEZ R	T n° S_ODT S TV Q BI BCD R
Temporisation sous forme de retard à la montée mémorisé	<opérande> (SS) valeur de temps	T n° S_SEVERZ S TW Q DUAL DEZ R	T n° S_ODTS S TV Q BI BCD R
Temporisation sous forme de retard à la retombée	<opérande> (SA) (SF) valeur de temps	T n° S_AVERZ S TW Q DUAL DEZ R	T n° S_OFFDT S TV Q BI BCD R

Tableau A1.5 Différents types de temporisations avec CONT.

Remarques :

- Pour chaque temporisation, deux formes sont données. La première correspond au format allemand et la seconde au format international.
- Les chronogrammes des différentes temporisations sont donnés juste après l'exemple.
- Les bobines peuvent lancer une temporisation de la même manière qu'un bloc, mais ne permettent pas le RESET ou encore la surveillance du TIMER.

Exemple :



S'il y a un front montant sur l'entrée E0.0, alors la temporisation T5 est lancée pour 2 secondes. La sortie A4.0 suit alors l'état de la temporisation T5.

Figure A1.9 Initialisation d'une temporisation sous forme d'impulsion.

La figure suivante montre les chronogrammes des différentes temporisations offertes par STEP 7.

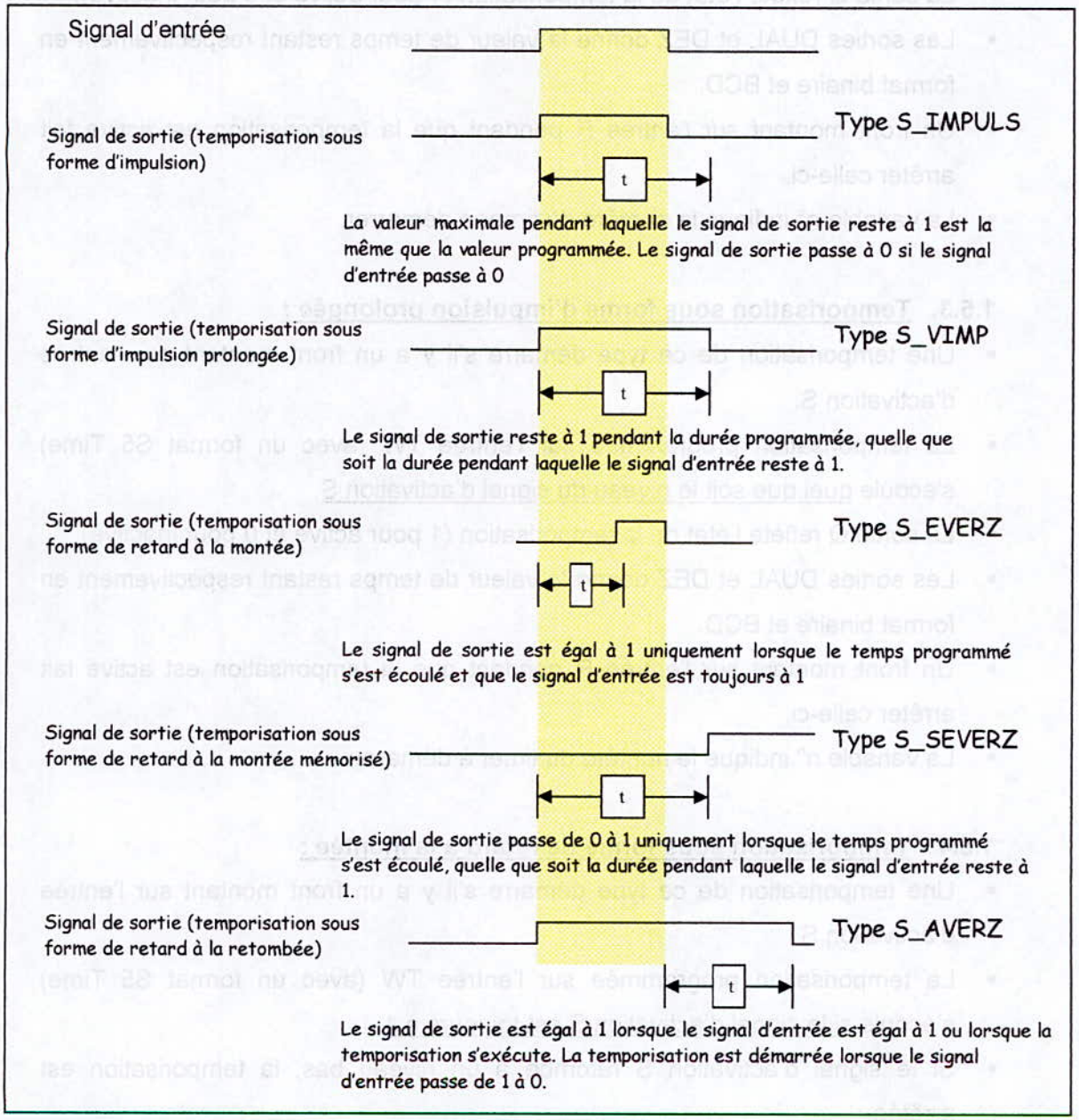


Figure A1.10 Chronogrammes des différentes temporisations possibles avec STEP 7.

Les blocs de lancement de temporisations sont proposés par STEP 7 dans le but d'avoir davantage de possibilités lors de la manipulation des Timers.

1.5.2. Temporisation sous forme d'impulsion :

- Une temporisation de ce type démarre s'il y a un front montant sur l'entrée d'activation S.
- La temporisation programmée sur l'entrée TW (avec un format S5 Time) s'écoule si le signal d'activation S est toujours à 1.
- Si le signal d'activation S retombe à un niveau bas, la temporisation est arrêtée.
- La sortie Q reflète l'état de la temporisation (1 pour active et 0 pour inactive).
- Les sorties DUAL et DEZ donne la valeur de temps restant respectivement en format binaire et BCD.
- Un front montant sur l'entrée R pendant que la temporisation est active fait arrêter celle-ci.
- La variable n° indique le numéro du timer à démarrer.

1.5.3. Temporisation sous forme d'impulsion prolongée :

- Une temporisation de ce type démarre s'il y a un front montant sur l'entrée d'activation S.
- La temporisation programmée sur l'entrée TW (avec un format S5 Time) s'écoule quel que soit le niveau du signal d'activation S.
- La sortie Q reflète l'état de la temporisation (1 pour active et 0 pour inactive).
- Les sorties DUAL et DEZ donne la valeur de temps restant respectivement en format binaire et BCD.
- Un front montant sur l'entrée R pendant que la temporisation est active fait arrêter celle-ci.
- La variable n° indique le numéro du timer à démarrer.

1.5.4. Temporisation sous forme de retard à la montée :

- Une temporisation de ce type démarre s'il y a un front montant sur l'entrée d'activation S.
- La temporisation programmée sur l'entrée TW (avec un format S5 Time) s'écoule si le signal d'activation S est toujours à 1.
- Si le signal d'activation S retombe à un niveau bas, la temporisation est arrêtée.
- La sortie Q se met à 1 lorsque la temporisation s'est exécutée sans erreur et que l'entrée S est toujours à 1.

- Les sorties DUAL et DEZ donne la valeur de temps restant respectivement en format binaire et BCD.
- Un front montant sur l'entrée R pendant que la temporisation est active fait arrêter celle-ci.
- La variable n° indique le numéro du timer à démarrer.

1.5.5. Temporisation sous forme de retard à la montée mémorisé :

- Une temporisation de ce type démarre s'il y a un front montant sur l'entrée d'activation S.
- La temporisation programmée sur l'entrée TW (avec un format S5 Time) s'écoule quel que soit l'état du signal S.
- S'il y a un front montant sur le signal d'activation S pendant que la temporisation s'exécute, celle-ci est redémarrée.
- La sortie Q se met à 1 lorsque la temporisation s'est exécutée sans erreur et que l'entrée S est toujours à 1.
- Les sorties DUAL et DEZ donne la valeur de temps restant respectivement en format binaire et BCD.
- Un front montant sur l'entrée R pendant que la temporisation est active fait arrêter celle-ci.
- La variable n° indique le numéro du timer à démarrer.

1.5.6. Temporisation sous forme de retard à la retombée :

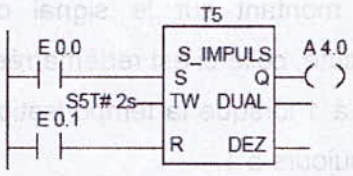
- Une temporisation de ce type démarre s'il y a un front descendant sur l'entrée d'activation S.
- S'il y à un front montant sur le signal d'activation S pendant que la temporisation s'exécute, celle-ci est redémarrée.
- La sortie Q se met à 1 si S est à 1 ou si la temporisation s'exécute.
- Les sorties DUAL et DEZ donne la valeur de temps restant respectivement en format binaire et BCD.
- Un front montant sur l'entrée R pendant que la temporisation est active fait arrêter celle-ci.
- La variable n° indique le numéro du timer à démarrer.

1.5.7. Type des paramètres utilisés pour les temporisations :

Paramètre	Type de donnée	Zone mémoire
n°	Timer	T
S	BOOL	E, A, M, D, L, T, Z
TW	S5Time	E, A, M, D, L
R	BOOL	E, A, M, D, L, T, Z
Q	BOOL	E, A, M, D, L
DUAL	WORD	E, A, M, D, L
DEZ	WORD	E, A, M, D, L

Tableau A1.6 paramètres des blocs de temporisation.

Exemple :



La temporisation est lancée s'il y a un front montant sur l'entrée E0.0
La durée de la temporisation est de 2 secondes.
La temporisation est arrêtée s'il y a un front montant sur l'entrée E0.1 (RESET) ou si l'entrée E0.0 revenait à 0 (voir temporisation sous forme d'impulsion).

Figure A1.11 Lancement d'une temporisation sous forme d'impulsion avec un bloc.

1.6. Opérations arithmétiques :

Les automates programmables S7 permettent de faire des opérations arithmétiques.

1.6.1. addition et soustraction :

Les blocs suivants permettent de faire des additions sur des entiers ou des réels.

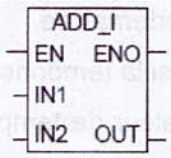
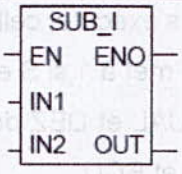
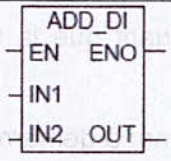
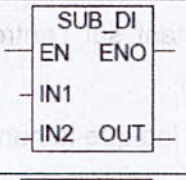
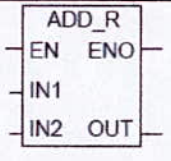
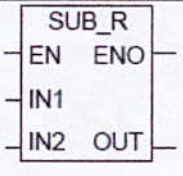
Opérations	Addition	Soustraction
Entiers sur 16 bits		
Entiers sur 32 bits		
Réels		

Tableau A1.7 Blocs d'addition et de soustraction.

1.6.2. Multiplication et division :

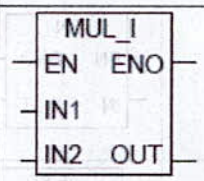
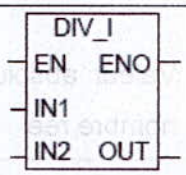
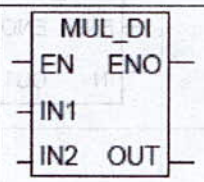
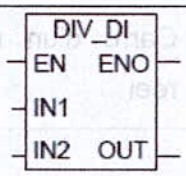
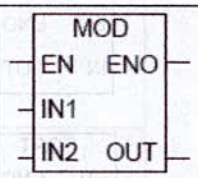
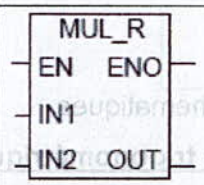
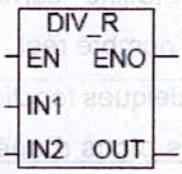
Opérations	Multiplication	Division	Reste de division
Entiers sur 16 bits			N'existe pas
Entiers sur 32 bits			
Réels			

Tableau A1.8 blocs de multiplication et de division.

Les paramètres sont :

Paramètres	Description
EN	Entrée de validation du bloc
IN1	Première valeur pour l'opération
IN2	Seconde valeur pour l'opération
ENO	Sortie de validation
OUT	Résultat de l'opération

Tableau A1.9 Paramètres des blocs de multiplication et de division.

Les blocs sont activés si l'état de l'entrée de validation est à 1. Le bloc effectue l'opération voulue entre les variables IN1 et IN2 puis donne le résultat sur la sortie OUT (dans le cas d'une division d'entier c'est le quotient ou résultat tronqué qui est donné comme sortie). Si le résultat est en dehors de la plage autorisée, les bits de débordement et de débordement mémorisé sont mis à 1 et la sortie ENO est mise à zéro.

1.6.3. Autres blocs disponibles avec CONT :

D'autres blocs sont disponibles pour faire des opérations mathématiques :

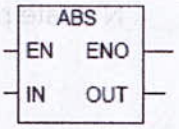
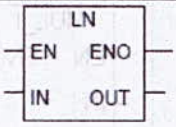
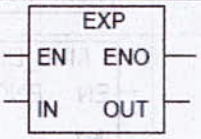
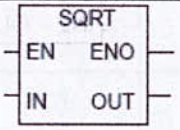
Bloc	Fonction	Bloc	Fonction
	Valeur absolue d'un nombre réel		Logarithme naturel d'un nombre réel
	Carré d'un nombre réel		Exponentiel d'un nombre réel
	Racine carrée d'un nombre réel		

Tableau A1.10 Quelques fonctions mathématiques.

1.6.4. Des blocs d'opérations trigonométriques sont aussi présents :

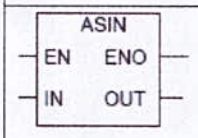
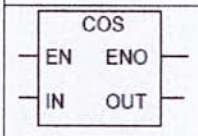
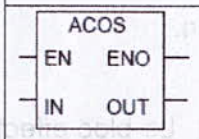
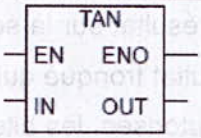
Bloc	Fonction
	Sinus d'un nombre réel exprimé en radians
	Arc sinus d'un nombre réel. Le résultat de l'opération est un angle exprimé en radians.
	Cosinus d'un nombre réel exprimé en radians
	Arc cosinus d'un nombre réel. Le résultat de l'opération est un angle exprimé en radians.
	Tangente d'un nombre réel exprimé en radians
	Arctangente d'un nombre réel. Le résultat de l'opération est un angle exprimé en radians.

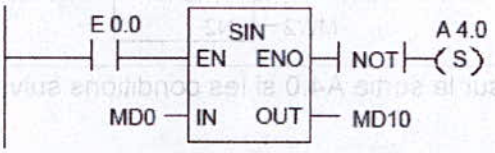
Tableau A1.11 Fonction trigonométriques.

Les entrées/sorties sont :

Paramètre	Type	Rôle
EN	BOOL	Entrée de validation
IN	REAL	Nombre réel
OUT	REAL	Valeur après traitement de la fonction
ENO	BOOL	Sortie de validation

Tableau A1.12 Paramètres des blocs de fonctions trigonométriques.

Exemple :



Le bloc de calcul s'active si l'entréeE0.0 est à 1, il calcule le sinus de l'angle exprimé en degré contenu dans le double mot MD10 puis stocke le résultat dans le double mot MD10.

S'il y a une erreur de traitement du bloc ou si E0.0 est à 0, la sortie ENO sera à 0 et donc il y aura un SET sur la sortie A4.0

Figure A1.12 Fonction Sinus avec CONT.

1.7. Opérations de comparaison :

On peut comparer des entiers sur 16 bits, des entiers sur 32 bits ou encore des réels. Ces blocs donnent une sortie ENO=1 si la relation est vraie, dans le cas contraire ENO=0.

L'inscription sur chaque bloc représente le type de comparaison suivie du type d'opérande :

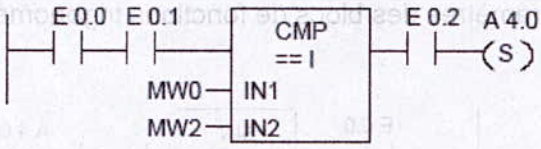
Type de comparaison	Symbole figurant dans le titre de la boîte
IN1 est égal à IN2.	==
IN1 est différent de IN2.	<>
IN1 est supérieur à IN2.	>
IN1 est inférieur à IN2.	<
IN1 est supérieur ou égal à IN2.	>=
IN1 est inférieur ou égal à IN2.	<=

Tableau A1.13 Types de comparaison existants avec CONT.

Symbole du type d'opérande	Type d'opérande
I	Entier sur 16 bits
D	Entier sur 32 bits
R	Réel

Tableau A1.14 Types d'opérandes possibles.

Exemple :



Un SET est effectué sur la sortie A4.0 si les conditions suivantes sont réunies:

- E0.0 est à 1
- E0.1 est à 1
- La valeur contenue dans le mémento MW0 est égale à la valeur contenue dans le mémento MW2
- E0.2 est à 1

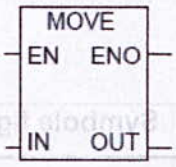
Figure A1.13 Test d'égalité su entier avec CONT.

1.8. Opération de transfert et de conversion :

1.8.1. Bloc d'affectation de valeur :

Permet d'affecter à une variable une valeur donnée à partir d'une autre variable ou d'une constante.

La sortie de validation ENO prend la même valeur que l'entrée de validation EN.



Paramètre	Type	Rôle
EN	BOOL	Entrée d'activation
ENO	BOOL	Sortie d'activation (prend la valeur de EN)
IN	Tout type de données 8, 16 ou 32bits	Valeur source
OUT	Tout type de données 8, 16 ou 32bits	Adresse de destination

Tableau A1.15 Paramètres du bloc de transfert.

1.8.2. Blocs de conversion :

Voici maintenant des blocs de conversion de type ou encore d'arrondissement, de troncature etc...

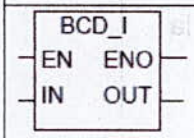
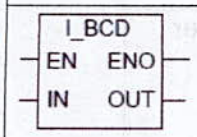
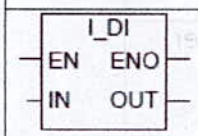
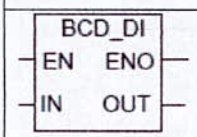
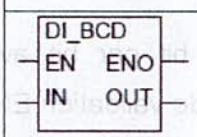
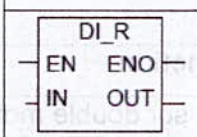
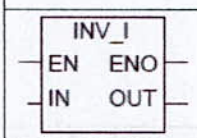
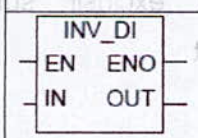
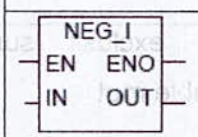
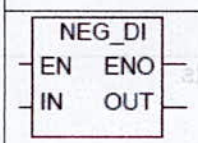
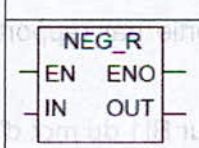
Bloc	Description
	Convertir un nombre codé en BCD en un nombre entier sur 16 bits
	Convertir un nombre entier sur 16 bits en un nombre codé en BCD
	Convertir un entier sur 16 bits en un entier sur 32 bits
	Convertir un nombre codé en BCD en un nombre entier sur 32 bits
	Convertir un nombre entier sur 32 bits en un nombre codé en BCD
	Convertir un nombre entier sur 32 bits en un nombre réel
	Complément à 1 d'un entier sur 16 bits
	Complément à 1 d'un entier sur 32 bits
	Complément à 2 d'un entier sur 16 bits
	Complément à 2 d'un entier sur 32 bits
	Inverser le signe d'un nombre réel

Tableau A1.16 Blocs de conversion.

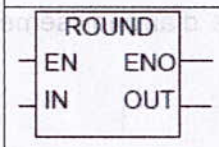
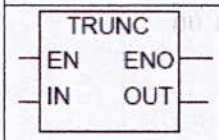
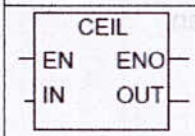
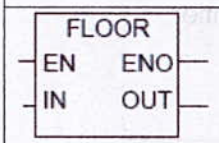
Bloc	Description
	Arrondir un réel en un entier de 32 bits
	Prendre la valeur entière d'un réel et la mettre sous forme d'un entier de 32 bits
	Convertir un nombre réel en un l'entier supérieur le plus proche (sur 32 bits)
	Convertir un nombre réel en un l'entier inférieur le plus proche (sur 32 bits)

Tableau A1.17 Blocs d'arrondissement et de troncature.

1.9. Opérations combinatoires sur mots et doubles mots :

Les blocs suivants combinent les deux entrées EN1 et EN2 bit par bit avec l'opération indiquée et donne le résultat sur la sortie OUT. La sortie de validation ENO prend la valeur de l'entrée de validation EN.

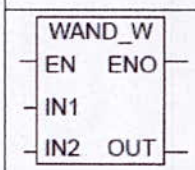
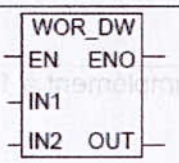
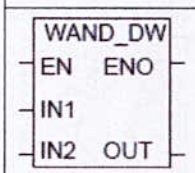
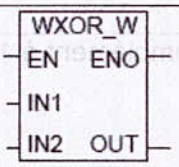
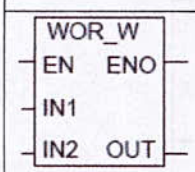
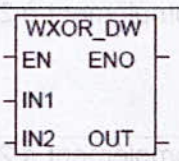
Bloc	Fonction	Bloc	Fonction
	ET sur mot		OU sur double mot
	ET sur double mot		OU exclusif sur mot
	OU sur mot		OU exclusif sur double mot

Tableau A1.18 Opérations combinatoires sur mots et doubles mots.

Remarque :

Le bit BI1 du mot d'état réagit en fonction de la valeur de la sortie par rapport à zéro :

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

- Si, à la sortie OUT, le résultat est égal à 0, le bit indicateur BI1 du mot d'état est égal à 0.

1.10. Opération de décalage :

1.10.1. Décalage vers la gauche de mot et de double mot :

L'opération de décalage à gauche, permet de décaler de N fois vers la gauche de l'entrée IN, les bits à droite sont remplacés par des zéros, les N-1 bits décalés sont perdus, on retrouve le dernier bit décalé dans le bit BI1 du mot d'état.

Remarque :

Si les bits perdus ne sont que des zéros, alors l'opération aura effectué une multiplication de IN par 2^N et l'aura rangée dans OUT.

La sortie ENO prend la valeur de EN.

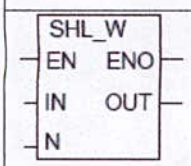
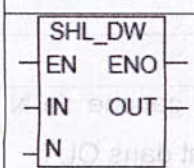
Bloc	Rôle
	Décalage vers la gauche de mot
	Décalage vers la gauche de double mot

Tableau A1.19 Blocs de décalage vers la gauche sur mots et doubles mots.

1.10.2. Décalage vers la droite de mot et de double mot :

L'opération de décalage à droite, permet de décaler de N fois vers la droite de l'entrée IN, les bits à gauche sont remplacés par des zéros, les N-1 bits décalés sont perdus, on retrouve le dernier bit décalé dans le bit BI1 du mot d'état.

La sortie ENO prend la valeur de EN

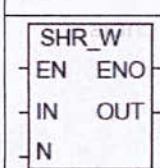
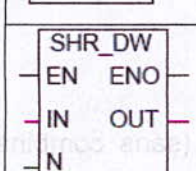
Bloc	Rôle
	Décalage vers la droite de mot
	Décalage vers la droite de double mot

Tableau A1.20 Blocs de décalage vers la droite sur mots et doubles mots.

1.10.3. Décalage vers la droite d'entiers sur 16 bits ou 32 bits :

Ce bloc fait un décalage vers la droite de l'entier IN.

Ce bloc complète les positions de gauche par l'état du bit de signe :

- Par des 0 si le nombre est positif.
- Par des 1 si le nombre est négatif.

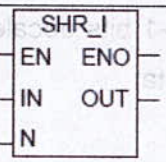
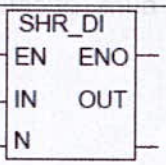
Bloc	Rôle
	Décalage vers la droite d'entier 16 bits
	Décalage vers la droite d'entier 32 bits

Tableau A1.21 Blocs de décalage vers la gauche sur entiers.

1.10.4. Opération de rotation :

Il existe deux blocs pour effectuer une rotation.

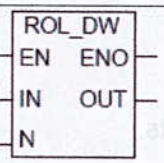
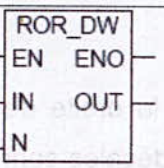
Bloc	Rôle	Explication
	Rotation vers la gauche de double mot	Effectue une rotation vers la gauche de N bits de l'opérande IN et met le résultat dans OUT. Les bits décalés vers la gauche sont remis à droite
	Rotation vers la droite de double mot	Effectue une rotation vers la droite de N bits de l'opérande IN et met le résultat dans OUT. Les bits décalés vers la droite sont remis à gauche

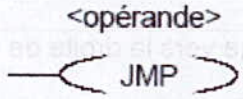
Tableau A1.22 Blocs de rotation sur doubles mots.

1.11. Opérations de saut :

STEP 7 permet de faire des opérations de sauts conditionnels ou inconditionnels. Ces opérations réagissent à une interrogation à 1 ou à 0 selon la commande choisie.

STEP 7 permet aussi de définir des étiquettes.

1.11.1. Saut inconditionnel :



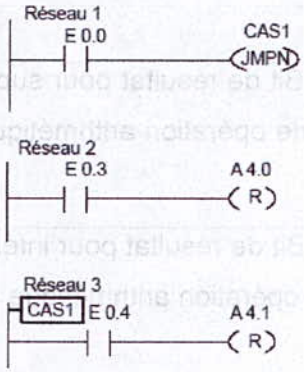
Cet élément lorsqu'il est utilisé directement dans un réseau (sans combinaison logique qui le précède) effectue un saut inconditionnel vers l'étiquette définit dans opérande.

1.11.2. Saut conditionnel :

Deux sortes de sauts conditionnels sont possibles, ces éléments doivent être précédés de combinaisons logiques qui représenteront la condition :

Elément	Description
<opérande> JMP	Effectue un saut si le résultat logique est à 1
<opérande> JMP N	Effectue un saut si le résultat logique est à 0

Tableau A1.23 Sauts conditionnels.



Si l'état de E0.0 est à 0 alors un saut est effectué à l'étiquette CAS1, sinon le réseau 2 puis le 3 sont exécutés.

Figure A1.14 Opération de saut conditionnel.

1.11.3. Etiquettes :

Le nom de l'étiquette doit contenir au maximum 4 caractères alphanumériques dont le premier est une lettre.

1.12. Opérations sur les bits du mot d'état :

Le résultat de l'interrogation des éléments cités plus tard sera :

- Une combinaison en ET s'il est utilisé en série avec d'autres éléments.
- Une combinaison en OU s'il est utilisé en parallèle avec d'autres éléments.

1.12.1. Bit d'anomalie « Registre RB » :

Le tableau suivant comporte les représentations SIMATIC et internationale :

Opération	Forme SIMATIC	Forme internationale
Interrogation à 1	BIE	BR
Interrogation à 0	BIE	BR

Tableau A1.24 Bit d'anomalie avec CONT.

1.12.2. Bit de résultat :

Les éléments suivants interrogent les bits BI0 et BI1 du mot d'état pour comparer le résultat d'une opération arithmétique par rapport à 0.

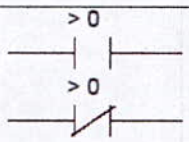
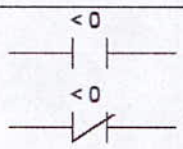
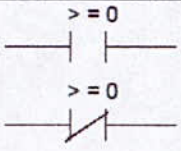
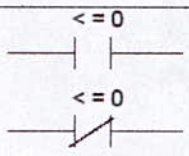
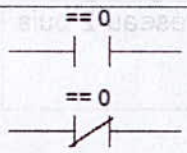
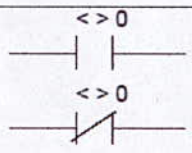
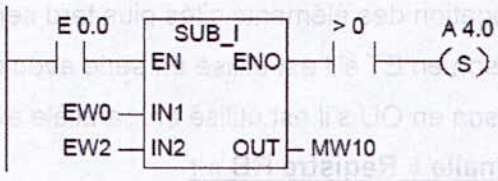
Elément	Description
	L'opération « Bit de résultat pour supérieur à 0 » détermine si le résultat d'une opération arithmétique est ou non supérieur à 0.
	L'opération « Bit de résultat pour inférieur à 0 » détermine si le résultat d'une opération arithmétique est ou non inférieur à 0.
	L'opération « Bit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

Tableau A1.25 Test sur bits de résultats.

Exemple :



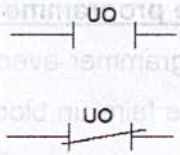
Le bloc de soustraction s'active lorsque E0.0 est à 1. Il soustrait IN2 à IN1 et sauvegarde le résultat dans MW10.
Si le résultat de l'opération s'exécute sans erreur et que son résultat est strictement supérieur à 0, un SET est effectué sur la sortie A4.0

Figure A1.15 Test sur soustraction.

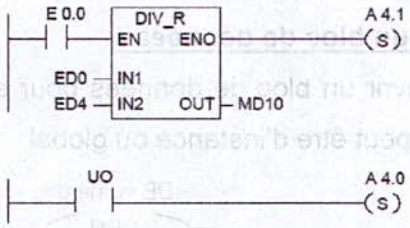
1.12.3. Bit d'anomalie « opération illicite » :

Ce test permet de déterminer si une opération à virgule flottante est illicite, c'est-à-dire que tous les opérandes sont des nombres à virgule flottante valide. Ce sont les bits BI0 et BI1 qui sont testés pour obtenir le résultat.

La représentation de l'élément est :



Exemple :



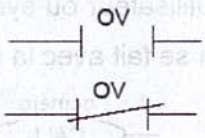
Un SET est effectué sur la sortie A4.0 si l'opération est exécutée que l'un des doubles mots ED0 et ED4 n'est pas un nombre à virgule flottante valide.

Figure A1.16 Bit d'anomalie.

1.12.4. Bit de débordement :

Cet élément permet de détecter s'il y a eu débordement lors de la dernière opération arithmétique (c'est-à-dire que le résultat est en dehors de la plage autorisée). Le bit interrogé et le bit DEB du mot d'état, qui se met à 1 si un débordement est détecté. Ce test n'est utile que s'il y a plusieurs réseaux. En effet, pour un bloc donné, si le résultat est en dehors de la plage autorisée, la sortie ENO est mise à zéro.

Les représentations de l'interrogation à 1 et à 0 sont respectivement :



Remarque :

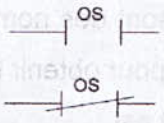
Le bit DEB est remis à 0 lorsque l'erreur est corrigée.

1.12.5. Bit de débordement mémorisé :

Cet élément permet de détecter s'il y a eu débordement lors de la dernière opération arithmétique (c'est-à-dire que le résultat est en dehors de la plage autorisée).

Le bit interrogé et le bit DM du mot d'état, qui se met à 1 si un débordement est détecté.

Mais contrairement au bit DEB, le bit DM reste à 1 même après que l'erreur soit corrigée.

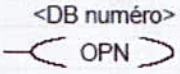


1.13. Opérations pour la gestion de programme :

La façon la plus organisée de programmer avec STEP 7 est de diviser le problème à résoudre en plusieurs parties, puis de faire un bloc distinct pour chacune des parties. Le programme principal sera donc une succession d'appels de fonctions ou de procédure. La partie qui suit traite des différentes commandes permettant de gérer un programme.

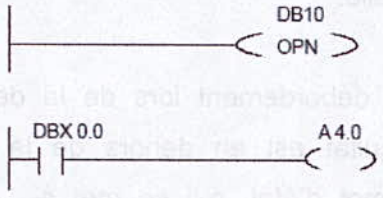
1.13.1. Ouverture d'un bloc de données :

Il est possible d'ouvrir un bloc de données pour en lire ou en modifier le contenu. Ce bloc de données peut être d'instance ou global.



Il suffit de spécifier le bloc de données à ouvrir.

Exemple :

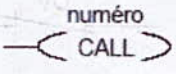


Le premier réseau ouvre le bloc de données DB10, puis dans le second A4.0 prend l'état du bit DBX0.0 qui se trouve dans le DB ouvert.

Figure A1.17 Ouverture d'un bloc de données.

1.13.2. Appel d'un FC ou SFC sans paramètre :

Il se peut qu'une fonction utilisateur ou système ne soit pas pourvue de paramètre, l'appel de ce type de fonction se fait avec la bobine suivante:



ou *numéro* est le FC ou SFC désiré (ex : FC20 ou SFC38).

Le fait d'appeler un FC ou un SFC entraîne :

- La sauvegarde de l'adresse de retour au bloc appelant.
- La sauvegarde des sélecteurs des blocs de données (global et d'instance).
- Le changement de la zone de données locales en cours en zone de données locales précédentes.
- L'empilement du bit MA dans la pile des blocs.

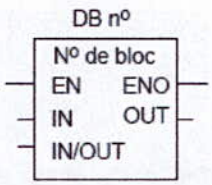
- La création de la nouvelle zone de données locales pour le FC ou le SFC appelé.

Ce n'est qu'après ces étapes que le bloc appelé est traité.

1.13.3. Appel d'un FC, SFC, FB, SFB :

Il est possible d'appeler ces blocs en les sélectionnant dans la boite de dialogue « éléments de programme », puis dans les menus :

- Blocs FB.
- Blocs FC.
- Blocs SFB.
- Blocs SFC.



Les blocs insérés le sont avec tous les paramètres d'entrée et de sortie.

L'appel d'un bloc entraîne :

- La sauvegarde de l'adresse de retour au bloc appelant.
- La sauvegarde des sélecteurs des blocs de données (global et d'instance).
- Le changement de la zone de données locales en cours en zone de données locales précédentes.
- L'empilement du bit MA dans la pile des blocs.
- La création de la nouvelle zone de données locales pour le FC ou le SFC appelé.

Le bloc appelé est traité après avoir effectué ces étapes.

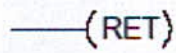
Remarque :

La sortie d'activation ENO est l'image du bit RB du mot d'état. Lors de l'ouverture d'un bloc, il se peut que ENO n'est pas la valeur attendue si le RLG n'a pas été sauvegardé dans RB.

Il faut donc sauvegarder 1 dans RB si l'exécution du bloc s'est faite sans erreur et sauvegarder un 0 s'il y a une erreur de traitement.

Cette sauvegarde doit être le dernier réseau pour être sûr qu'il s'exécute en dernier.

1.13.4. Retour :



Cet élément permet de quitter les blocs en utilisant une condition et de sauvegarder le RLG dans le bit RB du mot d'état.

Le langage LOG

2. Fonctions possibles avec LOG :

2.1. Combinaison sur bits :

Les éléments LOG cités dans cette partie sont utilisés pour faire des opérations combinatoires avec des bits. On les trouve dans le dossier *« opérations sur bits »* du menu *« éléments de programme »*.

2.1.1. Eléments ET, OU et XOR :

Les éléments de base pour les combinaisons logiques sont ET, OU et XOR.

ET	<opérande> <opérande> — &
OU	<opérande> <opérande> — >=1
XOR	<opérande> <opérande> — XOR

Tableau A2.1 Opérateur logique de base avec LOG.

Les opérandes peuvent être de type BOOL, TIMER ou encore COUNTER.

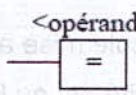
2.1.2. Insertion d'entrée binaire :

Les blocs précédents étaient présentés avec deux entrées seulement. Il est possible d'augmenter ce nombre en cliquant sur le bouton *« Entrée Binaire »* ou encore en appuyant sur le raccourci F8.

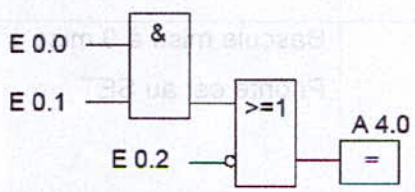
2.1.3. Inversion d'entrée binaire :

L'opérateur NOT est facilement utilisable en cliquant sur la variable à inverser puis sur le bouton *« inverser l'entrée binaire »* en encore avec le raccourci F9.

2.1.4. Affectation :



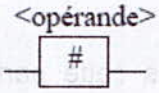
Cette opération permet d'associer à un opérande booléen le résultat logique (RLG) de la combinaison qui précède.



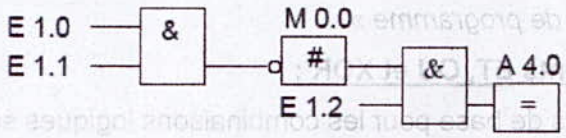
A4.0 est égale au résultat logique de (E0.0 ET E0.1) OU NOT(E0.2)

Figure A2.1 Exemple d'opération logique avec LOG.

2.1.5. Connecteur :



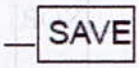
Permet de sauvegarder un résultat logique intermédiaire dans l'opérande associé.



M0.0 enregistre le résultat inversé de E1.0 ET E1.1
Le résultat de A4.0 n'est pas affecté par l'opération d'affectation, A4.0 est égale à : NOT(E1.0 ET E1.1) ET E1.2

Figure A2.2 Connecteur avec LOG.

2.1.6. Sauvegarde de 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.

2.1.7. SET, RESET et bascule:

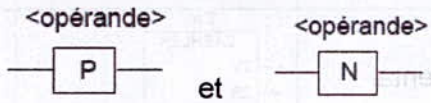
Il est possible de mettre à 1 ou à 0 une variable en utilisant l'une des opérations suivantes :

Opération	Rôle
<opérande> S	Mise à 1 de l'opérande. Seul un RESET remet l'opérande à 0.
<opérande> R	Mise à 0 de l'opérande. Seul un SET remet l'opérande à 1.
<opérande> SR S R Q	Bascule mise à 1 mise à 0. Priorité est au RESET
<opérande> RS R S Q	Bascule mise à 0 mise à 1. Priorité est au SET

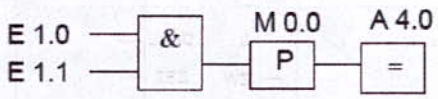
Tableau A2.2 SET et RESET avec LOG.

2.2. Détection de front :

2.2.1. Détection de front montant et descendant sur RLG:



Ces opérations donnent des sorties égales à 1 si des fronts sur le RLG qui précède les blocs sont détectés. Le RLG du cycle précédent est sauvegardé dans <opérande>.



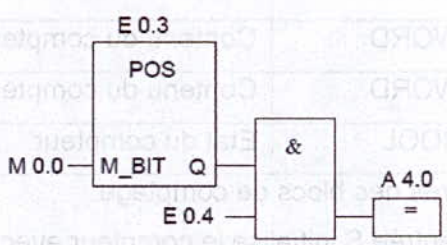
A4.0 ne se met à 1 que s'il y a front montant sur le résultat logique de E1.0 ET E1.1
Le RLG du cycle précédent est sauvegardé dans le mémento M0.0

Figure A2.3 Détection de front montant su RLG.

2.2.2. Détection de front montant ou descendant sur des signaux :

Opération	Fonctionnement
<opérande1> POS M_BIT Q	Détecte un front montant sur <opérande1>, le RLG précédent est sauvegardé dans M_BIT Q est mise à 1 s'il y a détection de front.
<opérande1> NEG M_BIT Q	Détecte un front descendant sur <opérande1>, le RLG précédent est sauvegardé dans M_BIT Q est mise à 1 s'il y a détection de front.

Tableau A2.3 Détection de font sur signaux avec LOG.



A4.0 ne se met à 1 que si E0.4 est à 1 et qu'il y a un front montant sur l'entrée E0.3

Figure A2.4 Détection de front avec LOG.

2.3. Opérations sur compteur :

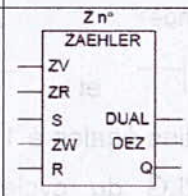
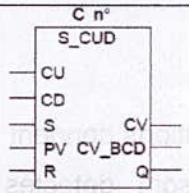
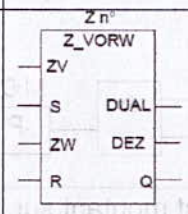
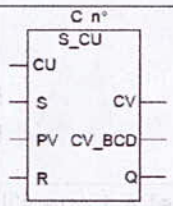
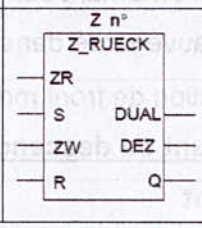
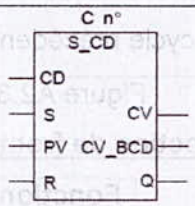
Type de compteur	Notation SIMATIC	Notation internationale
Compteur incrémental/décrémental		
Compteur incrémental		
Compteur décrémental		

Tableau A2.4 Opérations sur compteur.

Les paramètres sont :

Paramètre	Type	Signification
n°	COUNTER	Numéro d'identification du compteur (la plage dépend de la CPU)
ZV ou CU	BOOL	Entrée d'incrémentation
ZR ou CD	BOOL	Entrée de décrémentation
S	BOOL	Entrée d'initialisation du compteur
ZW ou PV	WORD	Valeur d'initialisation
R	BOOL	Entrée de remise à zéro
DUAL ou CV	WORD	Contenu du compteur en format BINAIRE
DEZ ou CV_BCD	WORD	Contenu du compteur en format BCD
Q	BOOL	Etat du compteur

Tableau A2.5 Paramètres des blocs de comptage.

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 sauf s'il contient 999 : dans ce cas le compteur repasse à 0.

Un front montant sur l'entrée ZR décrémente le compteur de 1 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 de 0, sinon elle donne 0.

Remarque :

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

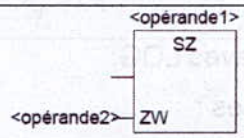
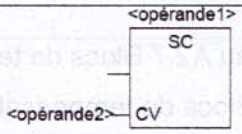
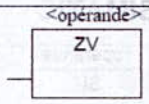
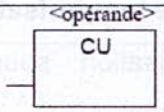
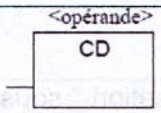
Fonction	Format SIMATIC	Format international
Initialiser compteur		
Incrémenter compteur		
Décrémenter compteur		

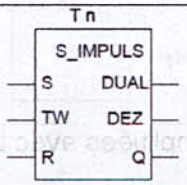
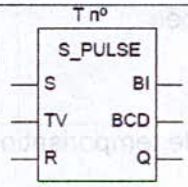
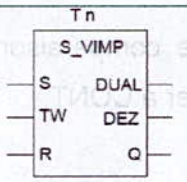
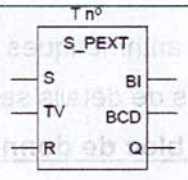
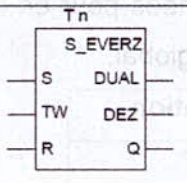
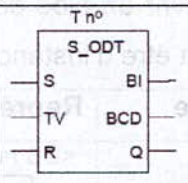
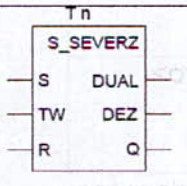
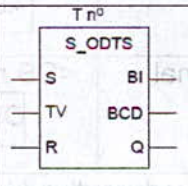
Tableau A2.6 Bocs de comptage simplifiés.

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.

2.4. Opérations de temporisations :

Les opérations de temporisations possibles sous LOG sont les mêmes que celles sous CONT. Le tableau représente les différents blocs de gestion de temporisations. Pour les détails et diagramme, au langage CONT.

Type de temporisation	Format SIMATIC	Format international
Temporisation sous forme d'impulsion		
Temporisation sous forme d'impulsion prolongée		
Temporisation sous forme de retard à la montée		
Temporisation sous forme de retard à la montée mémorisé		

Temporisation sous forme de retard à la retombée	Tn S_AVERZ S DUAL TW DEZ R Q	Tn° S_OFFDT S BI TV BCD R Q

Tableau A2.7 Blocs de temporisation avec LOG.

Blocs de temporisations simplifiées :

Type de temporisation	Format SIMATIC	Format international
Temporisation sous forme d'impulsion	<opérande> SI <valeur de temps> TW	<opérande> SP <valeur de temps> TV
Temporisation sous forme d'impulsion prolongée	<opérande> SV <valeur de temps> TW	<opérande> SE <valeur de temps> TV
Temporisation sous forme de retard à la montée	<opérande> SE <valeur de temps> TW	<opérande> SD <valeur de temps> TV
Temporisation sous forme de retard à la montée mémorisé	<opérande> SS <valeur de temps> TW	<opérande> SS <valeur de temps> TV
Temporisation sous forme de retard à la retombée	<opérande> SA <valeur de temps> TW	<opérande> SF <valeur de temps> TV

Tableau A2.8 blocs de temporisations simplifiées avec LOG.

Remarque :

Les blocs d'opérations arithmétiques et de comparaisons sont les mêmes que pour le langage CONT. Pour plus de détails se référer à CONT.

2.5. Ouverture d'un bloc de donnée :

Il est possible d'ouvrir un bloc de données pour en lire ou en modifier le contenu. Ce bloc de données peut être d'instance ou global.

Format d'écriture	Représentation
Format SIMATIC	<DB numéro> AUF
Format international	<DB numéro> OPN

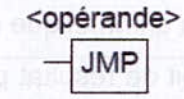
Tableau A2.9 Ouverture d'un bloc de données.

2.6. Opérations de saut :

STEP 7 permet de faire des opérations de sauts conditionnels ou inconditionnels. Ces opérations réagissent à une interrogation à 1 ou à 0 selon la commande choisie.

STEP 7 permet aussi de définir des étiquettes.

2.6.1. 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 opérande.

2.6.2. Saut conditionnel :

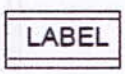
Deux sortes de sauts conditionnels sont possibles, ces éléments doivent être précédés de combinaisons logiques qui représenteront la condition :

Elément	Description
<opérande> JMP	Effectue un saut si le résultat logique est à 1
<opérande> JMPN	Effectue un saut si le résultat logique est à 0

Tableau A2.10 Opérations de saut conditionnel.

2.6.3. Etiquette :

Les étiquettes sont définies par :



Elles doivent contenir au maximum 4 caractères alphanumériques dont le premier est une lettre.

2.7. Opérations sur les bits du mot d'état :

2.7.1. Bit d'anomalie « Registre RB » :

Le tableau suivant comporte les représentations SIMATIC et Internationale :

Opération	Représentation SIMATIC	Représentation internationale
Interrogation de BIE	BIE	BR

2.7.2. Bits de résultat :

Les éléments suivants interrogent les bits BI0 et BI1 du mot d'état pour comparer le résultat d'une opération arithmétique par rapport à 0.

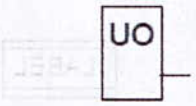
Elément	Description
> 0	L'opération « Bit de résultat pour supérieur à 0 » détermine si le résultat d'une opération arithmétique est ou non supérieur à 0.
< 0	L'opération « Bit de résultat pour inférieur à 0 » détermine si le résultat d'une opération arithmétique est ou non inférieur à 0.
> = 0	L'opération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

Tableau A2.11 bits résultats avec LOG.

2.7.3. Bit d'anomalie « opération illicite » :

Ce test permet de déterminer si une opération à virgule flottante est illicite, c'est-à-dire que tous les opérandes sont des nombres à virgule flottante valide. Ce sont les bits BI0 et BI1 qui sont testés pour obtenir le résultat.

La représentation de l'élément est :



2.7.4. Bit de débordement :

Cet élément permet de détecter s'il y a eu débordement lors de la dernière opération arithmétique (c'est-à-dire que le résultat est en dehors de la plage autorisée). Le bit interrogé et le bit DEB du mot d'état, qui se met à 1 si un débordement est détecté.

Ce test n'est utile que s'il y a plusieurs réseaux. En effet, pour un bloc donné, si le résultat est en dehors de la plage autorisée, la sortie ENO est mise à zéro.

Les représentations de l'interrogation à 1 et à 0 sont respectivement :

Remarque :

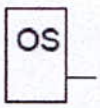
Le bit DEB est remis à 0 lorsque l'erreur est corrigée.

2.7.5. Bit de débordement mémorisé :

Cet élément permet de détecter s'il y a eu débordement lors de la dernière opération arithmétique (c'est-à-dire que le résultat est en dehors de la plage autorisée).

Le bit interrogé et le bit DM du mot d'état, qui se met à 1 si un débordement est détecté.

Mais contrairement au bit DEB, le bit DM reste à 1 même après que l'erreur soit corrigée.



2.7.6. Opérations de gestion de programme :

Les opérations de gestion de programme sont équivalentes celles existantes sous CONT. Pour voir le déroulement des opérations et les actions effectuées se référer à CONT.

Le tableau répertorie les fonctions principales de gestion de programme :

Fonction	Représentation
Appeler un FC ou un SFC sans paramètres	<numéro> CALL
Appeler un FC, SFC, FB, SFB avec paramètres	DB n° Bloc n° EN IN OUT IN/OUT ENO
Retour	RET

Tableau A2.12 Blocs de gestions de programme.

Le langage LIST

3. Types d'adressage :

3.1. Adressage immédiat :

L'opérande est une constante, l'opération agit directement sur cette valeur.

D'autres opérations fournissent automatiquement leurs opérandes.

Exemples

SET mettre le RLG à 1

L W#16# 542 charger le mot hexadécimal 542 dans l'accumulateur 1.

3.2. Adressage direct :

L'opérande est l'adresse de la valeur que l'opération doit traiter, elle est divisée en deux parties :

- l'identificateur d'opérande : il spécifie la zone de mémoire de l'adresse,
- l'adresse exacte de l'opérande.

Exemples :

U E 0.0 //interrogation du bit d'entrée d'adresse 0.0

= M 115.4 //Affecter le RLG au bit de mémento M 115.4

3.3. Adressage indirect en mémoire :

L'opérande n'est pas l'adresse de la valeur que l'opération doit traiter mais l'indique par un pointeur.

Elle est constituée de deux parties :

- Identificateur d'opérande, il indique la zone de mémoire utilisée,
- l'un des pointeurs suivants
  - Un mot contenant le numéro d'une temporisation T, d'un compteur Z, d'un DB, d'une FC, ou d'un FB.
  - Un double mot contenant l'adresse exacte située à l'intérieur de la zone de mémoire indiquée par l'identificateur d'opérande.

Ces mots ou double mots peuvent être situés dans les zones de mémoires suivantes :

- mémento (M),
- bloc de données (DB),
- données locales (L).

Exemples :

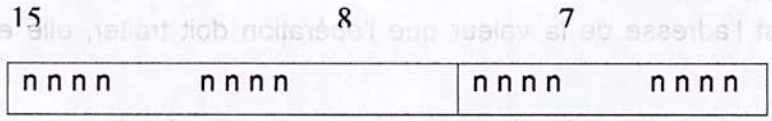
= DBX [DBD2] //Affecter le RLG au bit de données dont l'adresse exacte se trouve dans le double mot de données DBD2.

O A [LD3] //effectuer un OU logique sur le bit de sortie dont l'adresse se trouve dans le double mot de données locales LD3.

O A [voiture] //cette fois l'adresse est désignée par « voiture », mnémonique de LD3.

3.4. Utilisation des pointeurs en format mot et double mot :

Un pointeur en format mot est utilisé si on veut pointer un NUMERO (entre 0 et 65 535) d'une temporisation T, d'un compteur Z, d'un bloc de données DB, d'une fonction FC ou d'un bloc fonctionnel FB.



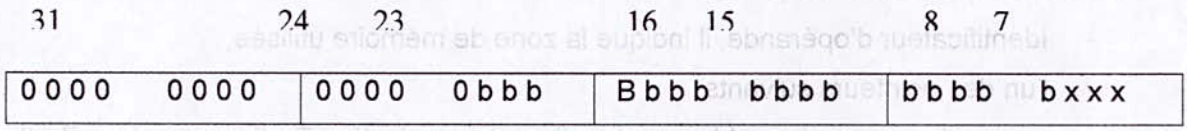
Exemple :

L +5 //Charger l'entier + 5 dans l'accumulateur 1.

T MW2 //transférer le contenu de l'accumulateur 1 dans le mot de mémento MW2.

AUF DB[MW2] //Ouvrir le bloc de données DB5.

Alors que le pointeur en format double mot est utilisé pour pointer des adresses exactes :



Les 3 premiers bits pour désigner les bits adressés, les bit 3 au 18 sont pour désigner le numéro de l'octet adressé.

Exemple :

L P#8.7 //Charger l'adresse 8.7 écrite sous la forme (2#0000 0000 0000 0000 0000 0000 0100 0111) dans l'accumulateur 1.

T MD2 //Ranger l'adresse 8.7 dans le double mot de mémento MD2.

U E [MD2] //L'automate interroge le bit d'entrée E 8.7 et affecte son état de
= A [MD2] signal au bit de sortie A 8.7.

Remarque :

Si on veut pointer une adresse dans un DB, il faut d'abord l'ouvrir :

Exemple :

AUF DB 15 //ouvrir le DB 15

U E [DBD 10] //interrogation du bit d'entrée dont l'adresse se trouve dans le double mot de données 10 du BD15.

Avantage de l'adressage indirect en mémoire :

La possibilité de modifier l'opérande de l'instruction dynamiquement pendant l'exécution du programme.

3.5. adressage indirect intrazone par registre :

C'est un adressage indirect utilisant un registre d'adressage et un déplacement. La valeur exacte de l'opérande est la somme de la valeur dans le registre d'adresse et le déplacement indiqué par le pointeur intrazone.

Exemple :

= A [AR1, P#1.1] affecter le contenu du bit RLG à la sortie, dont l'adresse est calculée comme suit :

Contenu du registre d'adresse AR1 : par exemple 8.7

\+ Déplacement p# 1.2

10.1

d'où l'adresse d'opérande est le bit de sortie A 10.0.

Format du pointeur :

31 24	23 16	15 8	7
0 0 0 0 0 0 0 0	0 0 0 0 0 b b b	B b b b b b b b	b b b b b x x x

Le pointeur est organisé comme suit :

- x x x : désignent le numéro du bit adressé,
- b b b b b b b b b b b b b b b b : numéro de l'octet adressé,
- le bit **31** est mis à **zéro** pour montrer que c'est un pointeur intra zone.

Exemple :

L P#8.7 //Charger le pointeur double mot désignant l'adresse du bit 8.7 dans l'accumulateur 1.

LAR1 //transférer le contenu de l'accumulateur 1 dans le registre d'adresse 1.

U E [AR1, P#0.0] //interrogation du bit d'entrée dont l'adresse est indiqué par l'adressage indirect intrazone par registre.

= A [AR1, P#1.1] //Le résultat de l'interrogation est affecté à la sortie A 10.0.

3.6. Adressage indirect interzone par registre :

Ce pointeur contient un identificateur de zone.

Format du pointeur :

31 24	23 16	15 8	7
1000 0 rrr	0000 0bbb	Bbbb bbbb	bbbb bxxx

il est organisé comme suit :

- x x x : désignent le numéro du bit adressé,
- b b b b b b b b b b b b b b b b : numéro de l'octet adressé,
- r r r : indique la zone de mémoire à pointer,
- le bit **31** est mis à1, pour monter que c'est un pointeur interzone.

Exemple :

L P#E 8.7 //Charger le pointeur double mot désignant l'adresse du bit E 8.7 dans l'accumulateur 1.

LAR1 //transférer ce pointeur au registre d'adresse 1.

U [AR1, P#0.5] //interroger l'entrée E 9.4

4. Liste des opérations

4.1. Opérations sur accumulateurs et sur registres d'adresses :

Appellation des accumulateurs :

Accumulateur	Bits utilisés
ACCUx	Bits de 0 à 31
ACCUx-L	Bits de 0 à 15
ACCUx-H	Bits de 16 à 31
ACCUx-LL	Bits de 0 à 7
ACCUx-LH	Bits de 8 à 15
ACCUx-HL	Bits de 16 à 23
ACCUx-HH	Bits de 24 à 31

Tableau A3.1 Appellation des accumulateurs

x représentant le numéro de l'accumulateur (1 ou 2).

- **TAK :** permuter le contenu des deux accumulateurs 1 et 2.
- **PUSH :** copier le contenu de l'accumulateur 1, sans le modifier, dans l'accumulateur 2.
- **POP :** copier le contenu de l'accumulateur 2, sans le modifier, dans l'accumulateur 1.
- **INC :** ajouter à la valeur de l'accumulateur 1-L-L l'entier sur 8 bits indiqué par cette instruction. Le résultat sera sauvegardé dans l'accumulateur 1-L-L.
- **DEC :** soustraire de la valeur de l'accumulateur 1-L-L l'entier sur 8 bits indiqué par l'instruction. Le résultat sera sauvegardé dans l'accumulateur 1-L-L.

INC et **DEC** ne sont pas utilisés pour les opérations arithmétiques, car elles n'effectuent pas de report à l'octet fort du mot de poids faible de l'accumulateur 1-L-H. En effet, les octets 1-L-H, 1-H-L et 1-H-H restent inchangés à l'utilisation de ces opérations.

- **+AR1 (+AR2) :** ajouter au contenu du registre d'adresse AR1 (AR2), le décalage précisé soit :
  - dans l'instruction : **+AR1<P#octet.bit> (AR2 <P#octet.bit>)**,
  - dans l'accumulateur 1-L : **+AR1(+AR2)**.

Exemple1 :

```
L  +257          //charger l'entier +257 dans l'accumulateur 1-L,
+AR1                 additionner le contenu de 1-L au contenu du registre d'adresse
                     AR1.
```

Exemple2 :

```
+AR2    p#300.0    //ajouter au registre AR2, le décalage 300 octets.
```

4.2. Opérations n'exécutant aucune fonction :

opération	Description
NOP0	Opérations n'effectuant aucune fonction, leur code contient un profil de 16 zéros.
NOP1	

Tableau A3.2 opérations n'exécutant aucune fonction.

4.3. Opérations combinatoires sur bit :

- U (ET) et sa forme inverse UN (ET NON),
- (OU) et sa forme inverse ON (OU NON),
- X (OU exclusif) et sa forme inverse XN (OU NON exclusif).

Ces opérations exécutent les fonctions de base suivantes :

- interrogation de l'état d'un bit s'il est à « 1 » (activé, en fonction) ou à « 0 » (Inactivé, hors fonction),
- interrogation de l'état d'une cellule de temporisation ou d'un compteur « 0 » (Valeur de la cellule = 0) ou « 1 » (valeur de la cellule > 0).

Le RLG varie en fonction du bit /PI :

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

4.3.1. Tables de vérité à l'intérieur d'une séquence combinatoire :

Abréviation	Opération	RLG avant l'exécution	Etat de l'opérande	RLG après l'exécution
U	ET	0	0	0
		0	1	0
		1	0	0
		1	1	1
UN	ET NON	0	0	0
		0	1	0
		1	0	1
		1	1	0
O	OU	0	0	0
		0	1	1
		1	0	1
		1	1	1
ON	OU NON	0	0	1
		0	1	0
		1	0	1
		1	1	1
X	OU EXCLUSIF	0	0	0
		0	1	1
		1	0	1
		1	1	0
XN	OU NON EXCLUSIF	0	0	1
		0	1	0
		1	0	0
		1	1	1

Tableau A3.3 Table de vérité pour une opération combinatoire avec LIST.

4.3.2. Exemples :

a) Effectuer un ET logique sur les deux entrées E 1.0 et E 1.1 et affecter le résultat à la sortie A 4.0 :

U E 1.0 //interrogation de l'état du bit E 1.0 et le ranger dans le RLG(première interrogation),

U E 1.1 //combiner le RLG avec l'état du bit E 1.1 selon la table de vérité ET, le ranger dans le RLG,

= A 4.0 //affecter le RLG à la sortie A 4.0.

b) Opération OU logique sur deux entrées E 1.0 et E 1.1 et affecter le résultat à la sortie A 4.0 :

O E 1.0 //interrogation de l'état du bit E 1.0 et le ranger dans le RLG(première interrogation),

O E 1.1 //combiner le RLG avec l'état du bit E 1.1 selon la table de vérité OU, le ranger de nouveau dans le RLG,

= A 4.0 //affecter le RLG à la sortie A 4.0.

c) Opération OU exclusif sur deux entrées E 1.0 et E 1.1 et affecter le résultat à la sortie A 4.0 :

X E 1.0 //interrogation de l'état du bit E 1.0 et le ranger dans le RLG(première interrogation),

X E 1.1 //combiner le RLG avec l'état du bit E 1.1 selon la table de vérité OU EXCLUSIF, le ranger de nouveau dans le RLG,

= A 4.0 //affecter le RLG à la sortie A 4.0.

4.4. Interrogation du mot d'état :

Les opérations U, UN, O, ON, X, XN permettent de tester les bits suivants du mot d'état: **RB**, **BI1** avec **BI0**, **DEB** et enfin **DM**.

Ces bits sont nécessaires pour connaître le résultat des opérations arithmétiques, les opérations de comparaison et de conversion.

Liste des opérations :

Opérande	Interrogation du mot d'état
BIE	Tester RB = 1
>0	Tester ((BI1 = 1) ET (BI0 = 0))
<0	Tester ((BI1 = 0) ET (BI0 = 1))
<>0	Tester (((BI1 = 1) ET (BI0 = 0)) OU ((BI1 = 0) ET (BI0 = 1)))
>=0	Tester (((BI1 = 1) ET (BI0 = 0)) OU ((BI1 = 0) ET (BI0 = 0)))
<=0	Tester (((BI1 = 0) ET (BI0 = 1)) OU ((BI1 = 0) ET (BI0 = 0)))
==0	Tester ((BI1 = 0) ET (BI0 = 0))
UO	Tester ((BI1 = 1) ET (BI0 = 1))
OV	Tester DEB = 1
OS	Tester DM = 1

Tableau A3.4 Interrogation du mot d'état.

4.5. Opérations combinatoires d'expressions entre parenthèses :

Une opération entre parenthèses est un calcul intermédiaire représentant une interruption de la séquence en cours. En effet, le dernier RLG avant les parenthèses est transféré vers la pile de parenthèses.

Cela signifie qu'une nouvelle séquence est commencée avant que l'ancienne séquence ne soit achevée. Cette dernière se poursuivra après la fin des instructions effectuées entre parenthèses, le bit /PI, sauvegardé lors de l'ouverture d'une parenthèse, est restauré.

Exemple :

U (//la première séquence entre parenthèse constitue une combinaison normale,

O E 124.0 / interrogation de l'état du bit, le résultat est sauvegardé dans le RLG,

O M 10.0 //combinaison de l'état du bit M 1.0 avec le RLG, selon la table de vérité OU, le résultat est sauvegardé dans RLG,

)

U (//copie le bit RLG dans la pile de parenthèses, et met fin à la séquence en remettant le /PI à 0,

O E 124.3 // interrogation du bit et sauvegardé le résultat dans le RLG,

O M 10.1 interrogation du bit et combinaison selon OU avec le RLG, le résultat est dans RLG,

) //combine le RLG final avec celui de la pile des parenthèses, selon la table de vérité ET, et le sauvegarde dans le RLG,

U E 124.7 //combine l'interrogation de ce bit, avec le RLG selon la table de vérité ET,

= A 124.7 // le RLG est affecté à la sortie A 124.7.

L'exemple précédent est l'équivalent du schéma à contact :

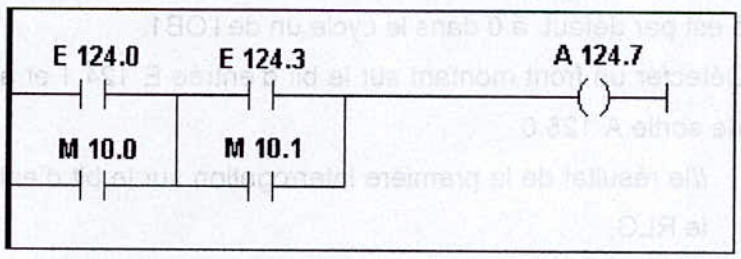


Figure A3.1 Equivalence entre CONT et LIST.

4.6. ET avant OU

Il n'est pas nécessaire d'utiliser des parenthèses dans le cas d'une séquence utilisant uniquement ET avant OU.

Exemple :

U E 124.3 //première interrogation sauvegardée dans le RLG,

U E 124.7 //combinaison du RLG avec la nouvelle interrogation, selon la table de vérité ET, le résultat remplace l'ancien RLG,

O //copie le RLG dans le bit OU, et met fin à la séquence,

U M 10.1 //première interrogation sauvegardée dans le RLG,

U M 10.2 //combinaison du RLG avec la nouvelle interrogation, le résultat remplace l'ancien RLG, en plus de cette opération, le RLG en cours est combiné avec celui dans le bit OU, le résultat se trouve dans le RLG,

= A 125.0 //affecte le RLG à la bobine de sortie A 125.0

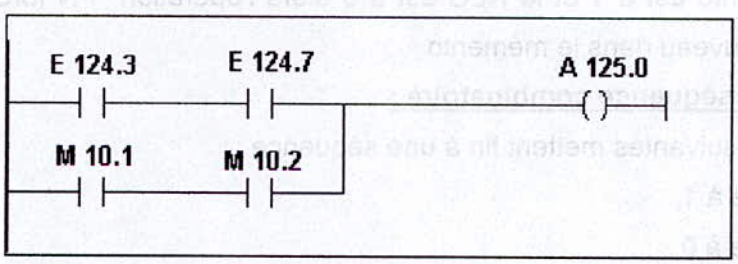


Figure A3.2 ET avant OU.

Avec cette opération, on peut anticiper le résultat final, car la bobine s'excite même avant la fin de la séquence si la première partie ET donne un RLG à 1.

4.7. Opérations de front : FP et FN :

4.7.1. Détection d'un front montant : FP < bit >

Pour détecter un front montant, le RLG en cours est comparé avec le mémento qui contient le RLG du cycle précédent.

Si le RLG de l'interrogation est à 1et le RLG du cycle précédent sauvegardé dans le mémento est à 0 alors l'opération FP met le RLG à 1, dans les autres cas il reste à zéro. Et puis, elle le copie dans le mémento de front montant.

Le mémento est par défaut à 0 dans le cycle un de l'OB1.

Exemple : Détecter un front montant sur le bit d'entrée E 124.1 et affecter le résultat à la bobine de sortie A 125.0

```
U E 124.1      //le résultat de la première interrogation sur le bit d'entrée est mis dans
               le RLG,
FP M 10.0      //cette opération compare le RLG avec le RLG du cycle précédent qui
               se trouve dans le mémento M 10.0, le résultat est mis de nouveau
               dans le RLG et dans le mémento M 10.0,
= A 125.0      //affectation du RLG à la sortie A 125.0
```

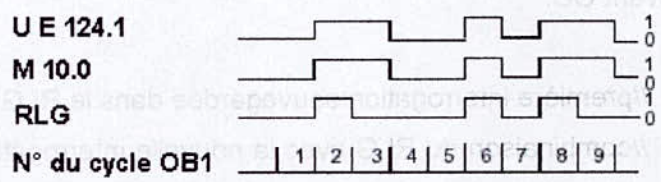


Figure A3.3 détection d'un front montant.

4.7.2. Détection d'un front descendant : FN < bit >

Cette opération compare le RLG fourni par l'interrogation d'un bit ou de toute une séquence de combinaisons, avec le RLG du cycle précédent sauvegardé dans un mémento de front .

Si le mémento est à 1 et le RLG est à 0 alors l'opération FN force le RLG à 1, le copie de nouveau dans le mémento.

4.8. Fin d'une séquence combinatoire :

Les opérations suivantes mettent fin à une séquence :

- S mettre à 1,
- R mettre à 0,
- = affectation.

Chacune des trois opérations remet le bit de la première interrogation /PI à zéro, pour marquer la fin de la séquence.

4.8.1. Opération S (mettre à 1) :

Si le RLG est à 1, l'opération S met à 1 l'état de signal du contact ou de la bobine à laquelle elle accède.

4.8.2. Opération R (mettre à 0) :

Si le résultat logique RLG a été mis à 1 dans l'instruction précédente, l'opération R met à 0 l'état de signal du contact ou de la bobine à laquelle elle accède.

4.8.3. Opération = (affectation) :

Quel que soit l'état du résultat logique RLG, la valeur du RLG est affectée à l'opérande auquel elle accède.

Remarques :

- Ces trois opérations dépendent du MCR (Master Control Relay),
- L'état de l'opérande affectée par l'opération S ou R est statique contrairement à celui affecté par = , qui est dynamique,
- L'opérande auquel accède l'opération S peut être un bit ; celui auquel accède l'opération R peut être un bit, un numéro de temporisation ou un numéro de compteur.

4.9. Opérations affectant le RLG :

Ces opérations permettent de modifier le contenu du bit RLG du mot d'état.

4.9.1. Opérations NOT (Négation) :

Elle permet d'inverser le RLG en cours.

4.9.2. Opération SET (mettre à 1) :

Elle permet de mettre le RLG à un inconditionnellement.

4.9.3. Opération CLR (mettre à 0) :

Elle permet de mettre le RLG inconditionnellement à 0. Elle remet également les bits /PI, OU et ETAT à 0.

4.9.4. Opération SAVE (sauvegarde du RLG) :

Elle permet d'enregistrer le RLG dans le bit RB (résultat binaire) du mot d'état, pour un usage ultérieur.

Opération	Effet sur les bits du mot d'état								
	RB	BI1	BI0	DEB	DM	OU	ETAT	RLG	/PI
NOT						-	1	x	-
SET						0	1	1	0
CLR						0	0	0	0
SAVE	x								

Tableau A3.5 Sauvegarde du RLG.

4.10. Opération de temporisation :

Une temporisation, utilisée dans le programme, permet :

- D'autoriser des temps d'attente : l'action n'est effectuée qu'après l'écoulement du temps de la temporisation,
- D'autoriser des temps de contrôle : Le programme contrôle, par exemple, la vitesse d'un moteur pendant 30 secondes une fois qu'on appuie sur le bouton de démarrage.
- De générer des impulsions : Le programme fournit, par exemple, des impulsions qui font clignoter une lampe.
- De mesurer le temps : Le programme peut, par exemple, déterminer le temps qu'il faut pour remplir un récipient.

4.10.1. Zone de mémoire :

La zone de mémoire utilisée pour les temporisations réserve un mot de 16 bits pour chaque temporisation.

Le nombre de temporisations prises en charge par LIST est 256, mais il est réduit de moitié par la CPU 314 IFM.

4.10.2. Chargement d'une temporisation :

Pour charger une valeur de temps prédéfini, on peut utiliser de deux méthodes différentes :

- **Première méthode :**

On charge la temporisation sous forme d'une base de temps, et une valeur de temps: **L W#16#txyz**

- **t** représente la base de temps choisie (résolution),
- **xyz** représentent la valeur de temps en format DCB.

Valeur de temps :

Les bits 0 à 9 du mot de temporisation contiennent la valeur de temps en DCB, cette plage va de 0s à 9990s maximum, l'équivalent de 2 heures 46 minutes et 30 secondes.

La valeur de temps indique un nombre d'unités.

Un décrément d'une unité se fait à chaque fois que l'intervalle précisé par la base de temps s'écoule.

Base de temps :

La base de temps indique les valeurs que les unités peuvent représenter.

Le choix de la base se fait en utilisant les bits 12 et 13 du mot de temporisation.

Le tableau suivant précise les bases de temps :

Base de temps	code binaire pour la base de temps	Plage des valeurs maximales
10ms	00	10MS à 9S_990MS
100ms	01	100MS à 1M_39S_900MS
1s	10	1S à 16M_39S
10s	11	10S à 2HR_46M_30S

Tableau A3.6 Bases de temps pour les temporisations.

- **Deuxième méthode :**

On charge la temporisation sous forme S5TIME ; **L S5T#aH_bbM_ccS_dddMS**

- a = heures, bb = minutes, cc = secondes et ddd = millisecondes,
- la base de temps est sélectionnée automatiquement, et la valeur de la temporisation est arrondie au nombre inférieur le plus proche de cette base de temps.

4.10.3. Les opérations de temporisation fournies par le LIST :

Pour démarrer une temporisation, il faut

- interroger l'état du signal d'activation,
- charger la valeur temps et la base de temps correspondante dans l'accumulateur,
- utiliser l'une des cinq temporisations disponibles dans LIST :
  - 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.

Exemples :

```
U E 124.0          //un front montant sur cette entrée déclenche la temporisation,
L S5T#2M_12S       //charger la valeur de la temporisation dans l'accumulateur 1,
SV T1              // démarrer la temporisation T1 sous forme d'impulsion prolongée
```

4.10.4. Redémarrage d'une temporisation FR :

Opération non nécessaire, utilisée pour redémarrer la temporisation.

Elle fournit un front montant pour redémarrer la temporisation si l'entrée d'activation est à 1.

Dans le cas où le signal d'activation est à 0, cette instruction n'opère pas.

4.10.5. Charger la valeur de temps d'une temporisation L et LC :

Elles permettent de charger la valeur de temps se trouvant dans le mot de temporisation comme nombre entier dans l'accumulateur 1-L.

Le nombre est sous forme binaire avec L et DCB avec LC.

4.10.6. Remettre à zéro une temporisation R :

Opération permettant de remettre une temporisation à zéro et d'effacer la valeur de temps et la base de temps du mot de temporisation indiqué s'il y a front montant sur son entrée d'activation.

Exemple :

```
U    E 124.0
FR   T1          //Valider la temporisation T1,
U    E 124.1
L    S5T#12S     //charger dans l'accumulateur 1 la valeur de la temporisation,
SV   T1          //Démarrer la temporisation T1 sous forme d'impulsion
                 prolongée,
U    E 124.2
RT1              //Remettre la temporisation T1 à 0 s'il y a front montant sur
                 E 124.2,
U    T1          //Interroger l'état de signal de la temporisation T1,
=    A 125.2     //et l'affecter à la sortie A 125.2
```

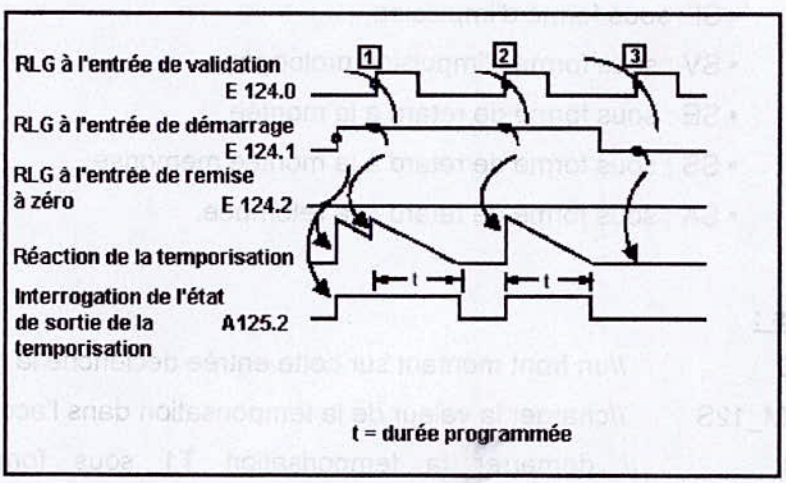


Figure A3.4 Chronogramme de temporisation.

4.11.Opérations de comptage :

4.11.1. Zone de mémoire :

Elle réserve un mot de 16 bits à chaque compteur (Z).

Le nombre de compteurs pris en charge par LIST est 256, mais il est réduit à 64 par la CPU 314 IFM. La plage de comptage est de 0 à 999.

4.11.2. Les fonctions de comptage disponibles sont :

Fonction		Description
FR	Valider compteur	Un front montant la valide, elle permet de ré initialiser le compteur malgré l'état 1 de leurs RLG.
L	Charger valeur de comptage en cours comme entier dans l'accumulateur 1	charge la valeur de comptage comme nombre entier dans l'accumulateur 1-L après sauvegarde du contenu de l'accumulateur 1dans l'accumulateur 2.
LC	Charger valeur de comptage	charge la valeur de comptage comme nombre DCB dans l'accumulateur 1-L après sauvegarde du contenu de l'accumulateur 1dans l'accumulateur 2.
R	Remettre compteur à zéro	charge la valeur de comptage 0 dans le compteur en accès si le RLG passe à 1.
S	Initialiser compteur	charge la valeur de comptage figurant dans l'accumulateur 1-L dans le compteur concerné si le RLG passe à 1.
ZV	Incrémenter	Un incrément est ajouté à la valeur de comptage si le RLG passe de 0 à 1 Si la valeur de comptage atteint sa limite supérieure de 999, l'incrémentation s'arrête. Une modification suivante du RLG n'a aucun effet sur le bit de débordement.
ZR	Décrémenter	Un décrément est enlevé de la valeur de comptage si le RLG passe de 0 à 1. Si le compteur atteint sa limite inférieure de 0, la décrémentation s'arrête. Une modification suivante du RLG n'a aucun effet, car le compteur n'opère pas avec des valeurs négatives.

Tableau A3.7 Fonctions de comptage disponibles avec LIST.

Exemple :

```
L  C#14     //charger la valeur d'initialisation du compteur en DCB dans
            l'accumulateur 1-L,
U E 0.1     //Compteur initialisé après détection du front montant à l'entrée E 0.1,
S Z1        //Charger la valeur pré définie dans le compteur Z1 si celui-ci est
            validé,
U E 0.0     //interroger l'état de l'entrée E 0.0,
```

ZR Z1 //enlever un incrément à chaque front montant de l'entrée E 0.0,

UN Z1 //interrogation de l'état inverse du compteur,

= A 0.0 //cette sortie donne 1 si le compteur atteint la valeur 0.

4.12.Opérations sur mot :

Ces opérations combinent les deux mots ou doubles mots, qui se trouvent dans les deux accumulateurs, cela se fait bit par bit selon les combinaisons booléennes, le résultat est mis dans l'accumulateur 1, après l'opération.

Le bit **BI1** est affecté par ce type d'opération ; il est mis à 1si le résultat de l'opération est différent de 0. Le bit **BI0** et **DEB** sont mis à zéro.

Liste de opérations :

Opération	Description
UW	ET logique sur mot
OW	OU logique sur mot
XOW	OU EXCLUSIF sur mot
UD	ET logique sur double mot
OD	OU logique sur double mot
XOD	OU EXCLUSIF sur double mot

Tableau A3.8 liste des opérations sur mots.

Exemples : opération ET mot

L MW 10 //charger la valeur du mot de mémento 10 dans l'accumulateur 1-L,

L MW 12 // charger la valeur du mot de mémento 12 dans l'accumulateur 2-L,

UW //Combiner, bit par bit, le contenu de l'accumulateur 1-L au contenu de l'accumulateur 2-L selon la table de vérité ET ; ranger le résultat dans l'accumulateur 1-L,

T MW 9 //transférer le résultat dans le mot de mémento MW9.

4.13.Opération de comparaison :

Elles permettent de comparer le contenu de l'accumulateur 2 à celui de l'accumulateur 1. Le RLG est mis à un si la comparaison est vraie.

Opération	symbole
égal à	==x
Différent de	<>x
Supérieur à	>x
Inférieur à	<x
Supérieur ou égal	>=x
Inférieur ou égal	<=x

Tableau A3.9 Opération de comparaison avec LIST.

Les valeurs utilisées sont :

x= I : entier de 16 bits,

x= D :entier de 32 bits,

x= R : réel à virgule flottante de 32 bits.

Exemple :

L MW10 //Charger le contenu du mot de mémento MW10 dans l'accumulateur 1,

L EW0 //Charger le contenu du mot d'entrée EW0 dans l'accumulateur 1, l'ancien contenu de l'accumulateur 1 est transféré dans l'accumulateur2.

==I //comparer les contenus des mots du poids le plus faible des deux accumulateurs,

= A 4.0 //la sortie A 4.0 est mise à1 si MW10 et EW0 sont égaux,

>I //Comparer la valeur figurant dans 2-L à 1-L,

= A 4.1 //La sortie A 4.1 mise à 1 si MW10 est supérieur à EW0,

<I // Comparer la valeur figurant dans 2-L à 1-L ,

= A 4.2 //La sortie A 4.2 est mise à 1 si MW10 est inférieur à EW0.

4.14.Opérations de conversion :

Toutes les opérations et le résultat se font dans l'accumulateur 1.

Liste de opérations :

4.14.1. Conversion des nombres BCD, des nombres entiers et des nombres réels :

Opération	Fonction
BTI	Convertir DCB en entier 16 bits.
ITB	Convertir entier 16 bits en DCB.
BTD	Convertir DCB en entier 32 bits.
DTB	Convertir entier 32 bits en DCB.
ITD	Convertir entier 16 bits en entier 32 bits
DTR	Convertir entier 32 bits en réel 32 bits

Tableau A3.10 fonctions de conversion.

Remarques :

- Dans les opérations :**BTI** et **BTD**, si l'on fait rentrer un nombre DCB dans une plage incorrecte (entre 10 et 15), une erreur BCDF est signalée lors de la conversion, l'automate passe automatiquement à l'arrêt. Cependant, l'**OB121** permet de programmer une autre réaction à cette erreur synchrone.
- Si la conversion donne un nombre BCD incorrecte (entre 10 et 15), les deux bits du mot d'état **DEB** et **DM** se mettent à 1.

Exemple :

```
L MW10    //Charger le nombre DCB dans l'accumulateur 1-L,
BTI       //Convertir le nombre DCB en un nombre entier et ranger le résultat
          dans l'accumulateur 1-L,
T MW20    //Transférer le résultat (nombre entier de 16 bits) dans le mot de
          mémento MW20.
```

4.14.2. Former le complément de nombres entiers ou réaliser l'inversion de nombres à virgule flottante :

Opération	Fonction
INVI	Complément à 1 d'un entier de 16 bits
INVD	Complément à 1 d'un entier de 32 bits
NEGI	Complément à 2 d'un entier de 16 bits
NEGD	Complément à 2 d'un entier de 32 bits
NEGR	Inverser un nombre réel de 32 bits à virgule flottante

Tableau A3.11 Complément à deux et inversion avec LIST.

Remarques :

Les opérations **NEGI** et **NEGD** mettent le résultat de la conversion dans les bits du mot d'état **BI1**, **BI0**, **DEB** et **DM** :

4.14.3. Effet des opérations sur les bits du mot d'état :

Résultats des opérations		BI1	BI0	DEB	DM
NEGI	NEGD				
Résultat=0	Résultat=0	0	0	0	-
-32768<= Résultat <=-1	-2147483648<= Résultat<=-1	0	1	0	-
32767>= Résultat>=1	2147483647>= Résultat>=1	1	0	0	-
Résultat = 32768	Résultat =2147483648	0	1	1	1

Tableau A3.12 Effet des opérations sur les bits du mot d'état

Exemple :

```
L PEW 128    //Charger la valeur dans l'accumulateur 1-L,
INVI         //Former le complément à 1 (16 bits),
T MW10       //Transférer le résultat dans le mot de mémento MW10.
```

4.14.4. Modification de l'ordre des octets dans le mot de poids faible de l'accumulateur 1 ou dans l'accumulateur 1 entier :

Opération	Fonction
TAW	Modifier l'ordre dans l'accumulateur 1-L (16 bits)
TAD	Modifier l'ordre dans l'accumulateur 1 entier (32 bits)

Tableau A3.13 modification dans l'accumulateur 1.

Exemples :

```
L MW10       //Charger la valeur du mot de mémento MW10 dans l'accumulateur 1,
TAW          //Inverser l'ordre des octets dans l'accumulateur 1-L,
T MW20       //Transférer le résultat dans le mot de mémento MW20.
```

Effet des opérations sur les bits du mot d'état :

Contenu	ACCU 1-H-H	ACCU 1-H-L	ACCU 1-L-H	ACCU 1-L-L
Avant l'exécution de **TAW**	Valeur A	Valeur B	Valeur C	Valeur D
Après l'exécution de **TAW**	Valeur A	Valeur B	Valeur D	Valeur C

```
L MD10       //Charger la valeur du double mot de mémento MD10 dans
             l'accumulateur 1,
TAD          //Inverser l'ordre des octets dans l'accumulateur 1,
T MD20       //Transférer le résultat dans le double mot de mémento MD20.
```

Contenus des accumulateurs :

Contenu	ACCU 1-H-H	ACCU 1-H-L	ACCU 1-L-H	ACCU 1-L-L
Avant l'exécution de **TAD**	Valeur A	Valeur B	Valeur C	Valeur D
Après l'exécution de **TAD**	Valeur D	Valeur C	Valeur B	Valeur A

4.14.5. Conversion de nombres réels à virgule flottante en nombres entier 32 bits :

Opération	Fonction
RND	Arrondir à l'entier le plus proche, Si la partie fractionnaire du nombre converti se situe exactement entre un résultat pair et un résultat impair, l'opération choisit le résultat pair.
TRUNC	Arrondir par troncature, le résultat correspond à la partie entière du nombre à virgule flottante converti.
RND+	Arrondir au plus petit nombre entier supérieur ou égal au nombre à virgule flottante converti.
RND-	arrondit le résultat au plus grand nombre entier inférieur ou égal au nombre à virgule flottante converti.

Tableau A3.14 Conversion de réels en entiers.

Remarque :

Ces quatre dernières opérations, les bits d'état DEB et DM se mettent à 1 si le nombre à convertir est hors de la plage correcte.

4.15. Opérations de chargement et de transfert :

Opérations permettant l'échange d'information inconditionnel (sans tenir compte du RLG) entre les différentes zones de mémoires, et les zones de périphérie(MIE,MIS).

4.15.1. L : charger

Charger dans l'accumulateur 1, l'octet, le mot ou le double mot indiqué, une fois que l'ancien contenu de l'accumulateur 1 a été sauvegardé dans l'accumulateur 2 et que l'accumulateur 1 a été mis à 0.

4.15.2. T : transférer

Copier le contenu de l'accumulateur 1, dans l'adresse de destination dont la taille contrôle le nombre d'octets transférés.

L'activation du relais de masquage (MCR), inhibe le transfert s'il est à 0.

L	Charger
L STW	Charger mot d'état dans l'accumulateur 1, pour les CPU S7-300, uniquement le bits suivants sont chargés dans les emplacements correspondants ;RLG, DM, DEB, BI0, BI1 et RB.
LAR1	Charger contenu de l'accumulateur 1 dans registre d'adresse 1
LAR1 <D>	Charger pointeur de 32 bits dans registre d'adresse 1
LAR1 AR2	Charger contenu du registre d'adresse 2 dans registre d'adresse 1
LAR2	Charger contenu de l'accumulateur 1 dans registre d'adresse 2
LAR2 <D>	Charger pointeur de 32 bits dans registre d'adresse 2

Tableau 3.15 Opérations de chargement.

T	Transférer
T STW	Transférer le contenu les bits 0 à 8 de l'accumulateur 1 dans mot d'état. L'opération s'exécute sans tenir compte des bits du mot d'état
TAR	Permuter registre d'adresse 1 avec registre d'adresse 2
TAR1	Transférer registre d'adresse 1 dans l'accumulateur 1
TAR1 <D>	Transférer registre d'adresse 1 à l'adresse de destination, qui peut être un MD, LD, DBD ou DID .
TAR1 AR2	Transférer registre d'adresse 1 dans registre d'adresse 2
TAR2	Transférer registre d'adresse 2 dans l'accumulateur 1
TAR2 <D>	Transférer registre d'adresse 2 à l'adresse de destination, qui peut être un MD, LD, DBD ou DID.

Tableau 3.16 Opérations de transfert.

4.16.Opérations arithmétiques sur nombres entiers :

Elles combinent le contenu des accumulateurs 1 et 2, le résultat se trouve dans l'accumulateur 1.

- **Liste des opérations** :

Opération	Taille en bit	Fonction
+I	16	Additionner 1-L et 2-L
-I	16	Soustraire 1-L de 2-L
***I**	16	Multiplier 1-L et 2-L, le résultat est sur 32 bits

/I	16	Diviser 2-L par 1-L, le quotient se trouve dans 1-L et le reste de la division est dans 1-H.
+D	32	Additionner les deux accumulateurs 1et 2,
-D	32	Soustraire le contenu de l'accumulateur 1 et l'accumulateur 2
***D**	32	Multiplier les deux accumulateurs 1 et 2
/D	32	Diviser l'accumulateur 2 par l'accumulateur 1
MOD	32	Donne le reste de la division

Tableau A3.17 Opérations arithmétiques avec LIST.

Les bits BI1, BI0, DEM et DM du mot d'état sont affectés par les opérations arithmétiques :

Exemple : Soustraire la plus petite valeur de la plus grande valeur :

```
      L MW 15        //charger le contenu du mémento dans l'accumulateur 1-L,
      L MW 14        //charger le contenu du mémento dans l'accumulateur 1-L,
                     après avoir transférer MW 15 dans l'accumulateur 2-L,
        >I           // si le contenu de 2-L est supérieur à 1-L, le RLG se
                      met à 1,
      SPB saut       //saut au repère « saut » si RLG est à 1,
      TAK            //permuter les contenus des deux accumulateurs,
saut      - I        //soustraire 1-L de 2-L.
```

Etat des bits du mot d'état affectés par les opérations arithmétiques sur entiers :

1- Dans la plage autorisée

Plage autorisée pour un résultat entier de 16 bits et de 32 bits	**Bits du mot d'état**			
	BI1	**BI0**	**DEB**	**DM**
Zéro	**0**	**0**	**0**	-
Résultat négatif 16 bits : $-32768 \leq résultat < 0$ 32 bits : $-2147483648 \leq résultat < 0$	**0**	**1**	**0**	-
Résultat positif 16 bits : $32767 \geq résultat > 0$ 32 bits : $2147483647 \geq résultat > 0$	**1**	**0**	**0**	-

2- Dans la plage non autorisée

Plage non autorisée	BI1	BI0	DEB	DM
Dépassement négatif de la plage pour une addition **+I et +D** 16 bits : résultat = –65536 32 bits : résultat = –4 294 967 296	0	0	1	1
Dépassement négatif de la plage pour une multiplication ***I et *D** 16 bits : résultat < –32 768 32 bits : résultat < –2 147 483 648	0	1	1	1
Dépassement positif de la plage pour addition, soustraction **+I, -I, +D, -D, NEGI et NEGD** 16 bits : résultat > 32 767 32 bits : résultat > 2 147 483	0	1	1	1
Dépassement positif de la plage pour multiplication, division ***I, *D, /I et /D** 16 bits : résultat > 32 767 32 bits : résultat > 2 147 483	1	0	1	1
Dépassement négatif de la plage pour addition, soustraction **+I, -I, +D et -D** 16 bits : résultat < –32 768 32 bits : résultat < –2 147 483	1	0	1	1
Division par zéro **/I, /D et MOD**	1	1	1	1

- **Addition d'un nombre entier à l'accumulateur 1 :**

cette opération, permet l'addition d'une constante entière au contenu de l'accumulateur 1 :

+ Addition d'entier sur 16 bits,

+ L# addition d'entier sur 32 bits.

4.17.Opérations arithmétiques sur nombres à virgule flottante IEEE de 32 bits:

- **Opérations arithmétiques de base :**

Opérande	Fonction
+R	Additionner les deux IEEE contenus dans les accumulateurs 1 et 2
-R	Soustraire la valeur IEEE de l'accumulateur 1 de cette de l'accumulateur 2
*R	Multiplier le contenu des deux accumulateurs 1 et 2
/R	Diviser le contenu de accumulateur 2 par accumulateur 1

Ces opérations s'effectuent dans l'accumulateur 1, le résultat sera indiqué par bits du mot d'état BI1, BI0, DEB et DM.

Etat des bits du mot d'état affectés par les opérations arithmétiques sur entiers

1- Dans la plage autorisée

Plage autorisée pour un résultat à virgule flottante de 32 bits	Bits du mot d'état			
	BI1	BI0	DEB	DM
+0,-0	0	0	0	-
$-3,402823^{E}+38<$résultat$<-1,175494^{E}-38$	0	1	0	-
$1,175494^{E}-38<$résultat$<-3,402823^{E}+38$	1	0	0	-

2- Dans la plage non autorisée

Plage non autorisée pour le résultat à virgule flottante	Bits du mot d'état			
	BI1	BI0	DEB	DM
$-1,175494^{E}-38<$résultat$<-1,401298^{E}-45$	0	0	1	1
$+1,401298^{E}-45<$résultat$<+1,175494^{E}-38$	0	0	1	1
Résultat$<-3,402823^{E}+38$	0	1	1	1
Résultat $>3,402823^{E}+38$	1	0	1	1

4.17.1. ABS : valeur absolue d'un réel

L'opération ABS permet de donner la valeur absolue d'un nombre IEEE, dans l'accumulateur1.

Exemple :

L $-2,15^{E}+3$ charger la valeur -2150 dans l'accumulateur 1,

ABS calcul de la valeur absolue,

T MD10 transférer le résultat 2150 dans le double mot du mémento 10.

4.18.Opérations arithmétiques étendues :

L'opérande à manipuler doit se charger dans l'accumulateur 1, le résultat de l'opération écrase l'opérande.

La liste des opérations :

Opération	Fonction
SQRT	Racine carrée du nombre IEEE
SQR	Le carré du nombre IEEE
LN	Le logarithme naturel ? du nombre IEEE
EXP	Valeur exponentielle de base e du nombre IEEE
SIN	Le sinus de l'angle indiqué en radiant
COS	Le cosinus de l'angle indiqué en radiant
TAN	La tangente de l'angle indiqué en radiant
ASIN	L'arc sinus du nombre IEEE compris entre $-1 \leq IEEE \leq 1$
ACOS	L'arc cosinus du nombre IEEE compris entre $-1 \leq IEEE \leq 1$
ATAN	L'arctangente du nombre IEEE

Tableau A3.18 opérations mathématiques et trigonométriques avec LIST.

Effet sur les bits d'états :

- chaque opération met le bit BI1 à 1 si le résultat diffère de 0.
- le bit DEB est mis à 1 comme indication d'erreur, si la valeur de l'opérande ne correspond pas à la plage admise par cette opération (exemple : débordement).

4.19.Opérations de saut :

Elles interrompent le déroulement linéaire du programme, pour le poursuivre à l'endroit indiqué par le repère de saut, ce repère ne doit pas dépasser 4 caractères.

En ce qui concerne l'utilisation de ces opérations dans les CPU S7-300, la destination de saut ne doit jamais se trouver à l'intérieur d'une séquence d'instructions combinatoires, mais doit se mettre au début.

la portée de saut maximale est de -32768 ou +32767 mots du code de programme.

4.19.1. opérations de saut inconditionnel :

SPA : permet un saut vers le haut ou vers le bas du programme.

la destination doit être unique et se trouver dans le même bloc que l'instruction du saut correspondante.

SPL : permet de programmer une liste de sauts SPA qui ne doit pas dépasser 255 destinations.

Le nombre qui se trouve dans l'accumulateur 1-L-L, correspond au numéro de l'opération SPA à exécuter.

Si ce nombre est supérieur à la longueur de la liste, le saut se fait vers le repère désigné par SPL.

Exemple :

```
L   x              //charger une valeur entre 0 et 255
    SPL  fin       //saut vers fin si x>2,
    SPA  sot0//saut vers sot0 si x=0,     SPA N°0
    SPA  sot1//saut vers sot1 si x=1,
    SPA  sot2      //saut vers sot2 si x=2,

fin: SET
    =    A   125.5
    SPA  out
sot0: SET
    =    A   124.0
    SPA  out
sot1: SET
    =    A   124.1
    SPA  out

sot2: SET
    =    A   124.2
    SPA  out
out : SET
    =    A   125.7
```

Valeur entière de x	Résultat donné par le programme
0	Sorties A 124.0 et A 125.7 sont à 1
1	Sorties A 124.1 et A 125.7 sont à 1
2	Sorties A 124.2 et A 125.7 sont à 1
2< x< 255	Sorties A 125.5 et A 125.7 sont à 1

4.19.2. Opérations de saut selon le RLG :

SPB : saut si RLG est à 1.

SPBN : saut si RLG est à 0.

SPBB : copier le RLG dans le RB, et faire un saut si RLG est à 1.

SPBNB : copier le RLG dans le RB et faire un saut si RLG est à 0.

4.19.3. Opérations de saut selon les bits du mot d'état :

SPBI : saut si RB est à 1.

SPBIN : saut si RB est à 0.

SPO : saut si DEB est à 1.

SPS : saut si DM est à 1.

4.19.4. Opérations de saut selon le résultat du calcul :

SPZ : saut si le résultat est égal à 0.

SPN : saut si le résultat est différent de 0.

SPP : saut si le résultat est supérieur à 0.

SPM : saut si le résultat est inférieur à 0.

SPPZ : saut si le résultat est supérieur ou égal à 0.

SPMZ : saut si le résultat est inférieur ou égal à 0.

SPU : saut si le résultat est illicite c'est-à-dire :

- une division par 0,
- utilisation d'opérations illicites
- résultat illicite d'une comparaison de nombres à virgule flottante (utilisation d'un format illicite).

LOOP : permet la programmation de boucles ; le compteur de boucles est chargé dans l'accumulateur 1-L.

Le bouclage se maintient tant que le compteur n'indique pas à 0.

4.20. Opérations de décalage et de rotation :

4.20.1. Opérations de décalage :

Syntaxe :

opération ou opération<nombre>

Ces opérations permettent de décaler bit par bit le contenu du mot de poids faible de l'accumulateur 1 (1-L) ou de l'accumulateur entier 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 (0 signifie positif et 1 négatif). Le bit décalé en dernier est chargé dans le bit BI1 du mot d'état. Le nombre de bits de décalage est précisé soit par l'opérande <nombre>, soit par une valeur figurant dans l'accumulateur 2 L-L.

SSI : décalage vers la droite d'un entier signé sur 16 bits.

SSD : décalage vers la droite d'un entier signé sur 32 bits.
les bits libérés par ce décalage sont remplis par le signe du mot (0 si positif et 1 si négatif).

Exemple :

Contenu	**ACCU-H**				**ACCU-L**			
Bit	31...	.....	.....	...16	15...	.....	....	0
avant l'exécution de **SSI 6**	0101	1111	0110	0100	**1001**	**1101**	**00**11	1011
après l'exécution de **SSI 6**	0101	1111	0110	0100	1111	11**10**	**0111**	**0100**

Les positions binaires libérées par le décalage dans les opérations qui suivent sont complétées par des zéros :

SRW : décalage vers la droite d'un mot
SLW : décalage vers la gauche d'un mot.
SLD : décalage vers la gauche d'un double mot.
SRD : décalage vers la droite d'un double mot

4.20.2. Opérations de rotation :

syntaxe :
opération
ou opération<nombre>

Ces opérations permettent la rotation bit par bit, vers la droite ou vers la gauche du contenu complet de l'accumulateur 1. les bits libérés par cette rotation remplacent ceux décalés hors accumulateur.
le bit décalé en dernier est mis dans le BI1. Le nombre de positions binaires objet de la rotation est précisé soit par l'opérande <nombre>, soit par une valeur figurant dans l'accumulateur 2-L-L..

RLD : rotation vers la gauche d'un double mot.

RRD : rotation vers la droite d'un double mot.

exemple :

Contenu	**ACCU1-H**				**ACCU1-L**			
Bit	31...	....	....	...16	15...	....	....	0
Avant l'exécution de RLD 4	**0101**	1111	0110	0100	0101	1101	0011	1011
Après l'exécution de RLD 4	1111	0110	0100	0101	1101	0011	1011	**0101**

RLDA : rotation vers la gauche du double mot via BI1 :

décalage vers la gauche d'un seul bit, le bit libéré contiendra l'état du bit BI1, qui se mettra à 0 par la suite.

RRDA : rotation vers la droite du double mot via BI1 :

décalage vers la droite d'un seul bit, le bit libéré contiendra l'état du bit BI1. qui se mettra à 1 par la suite.

Contenu	**BI1**	**ACCU1-H**				**ACCU1-L**			
Bit		31...	....	....	...16	15...	....	0	31...
Avant l'exécution de RLDA	**X**	0101	1111	0110	0100	0101	1101	0011	1011
Après l'exécution de RLDA	**0**	1011	1110	1100	1000	1011	1010	0111	011**x**

4.21.Opérations sur blocs de données :

AUF<bloc de donnée> :

la manipulation de données ne peut se faire que si l'on ouvre le bloc de données correspondant

Exemple :

```
AUF DB10    //ouvrir le bloc de données DB10 comme bloc de données global,
L DBW35     //Charger dans l'accumulateur 1-L le mot de données DBW35 du bloc de
            données ouvert,
T MW22      //Transférer le contenu de l'accumulateur 1-L dans le mot de mémentos
            MW22,
AUF DB20    //Ouvrir le bloc de données DB20 comme bloc de données d'instance,
L DBB12     //Charger dans l'accumulateur 1-L-L. L'octet de données DIB12 du bloc de
            données d'instance ouvert
T DBB37     //Transférer le contenu de l'accumulateur 1-L-L dans l'octet de données
            DBB37 du bloc de données global ouvert.
```

L DBLG : charger longueur de DB global

Cette opération charge la longueur du bloc de données global dans l'accumulateur 1 une fois que l'ancien contenu de l'accumulateur 1 a été sauvegardé dans l'accumulateur 2.

L DBNO Charger le numéro de DB global

Cette opération charge dans l'accumulateur 1 le numéro du bloc de données global ouvert une fois que l'ancien contenu de l'accumulateur 1 a été sauvegardé dans l'accumulateur 2.

4.22.Opérations de gestion d'exécution de programme :

BE et BEA: interrompent inconditionnellement le déroulement normal du programme dans le bloc en cours, pour le poursuivre dans le bloc appelant.

Cette opération s'avère nécessaire si l'on veut exécuter une partie bien précise du programme.

BEB : interruption conditionnelle du déroulement normal du programme dans le bloc en cours si le RLG est à1.

CALL : appelle le bloc indiqué en opérande. Ce bloc peut être une fonction (FC, SFC), ou un bloc fonctionnel (FB, SFB) et dans ce cas, on doit préciser le numéro du DB utilisé.

CALL # nom-variable

CC : Appel de bloc conditionnel

Appelle un bloc FC ou SFC si le RLG est à1. à l'inverse de CALL, cette opération ne permet pas le transfert de paramètres.

UC: Appel de bloc inconditionnel

Appelle un bloc FC ou SFC inconditionnellement. A l'inverse de CALL, cette opération ne permet pas le transfert de paramètres

Relais de masquage : MCR

le Master Control Relay est utilisé pour conditionner le passage de flux d'énergie .

les opérations relatives au MCR s'exécutent par paires.

- MCRA et MCRD : activation et désactivation du MCR

Les instructions entre MCRA et MCRD dépendent de l'état de signal du bit MCR. En revanche, les instructions en dehors de la séquence MCRA-MCRD ne dépendent pas de l'état de signal de ce bit.

- MCR(et)MCR : ouvrir et fermer une zone MCR

à l'intérieur de la zone MCR, l'état du MCR dépend du dernier RLG, et toutes les opérations influencées par son état le sont aussi.

Le MCR est commandé par une pile de un bit de large et de huit bits de profondeur (nombre d'imbrications).

Opérations influencées par le MCR et leur comportement :

- **= : Affectation,**
- **S : Mettre à 1,**
- **R : Mettre à 0,**
- **T : Transférer un octet, mot ou double mot.**

Etat de MCR	**=**	**S,R**	**T**
Hors fonction (0)	Écrit 0	Q Reste inchangé	Écrit 0
En fonction (1)	Exécution normale	Exécution normale	Exécution normale

Résumé :

Ce travail permet de se familiariser avec le logiciel STEP7 afin d'implémenter quelques tâches d'automatisation sur l'automate S7-314 IFM qui a la particularité d'intégrer un module d'entrées/sorties TOR, un module d'entrées/sorties analogiques et un module fonctionnel. La simulation des entrées est assurée par un circuit de simulation.
Le logiciel STEP7 se présente sous forme d'interface simple à utiliser en subdivisant le problème d'automatisation en plusieurs blocs indépendants, ceci structure le programme et le rend plus facile à tester et à exécuter.
Les exemples étudiés dans ce document permettent de voir qu'avec l'automate S7-314 IFM, on a la possibilité d'automatiser une large plage de procédés industriels ou dans le domaine des services.

Mots clef : STEP7, automate programmable S7-314 IFM, circuit de simulation, exemples de programmation.

Abstract :

This work explain how to use the software STEP 7 and gives some examples of automation systems on the PLC S7-314 IFM. This PLC has a logic I/O module, analogics I/O and integrated function module. The input simulation is assumed by a circuit. STEP 7 is a software who divide the problem into several blocs, the program is simpler to run and debug.
The various examples of this document shows that the PLC S7-314 IFM is able to be used on many kinds of automation systems which can be industrial or services systems.

Key words: STEP 7, PLC, S7-314 IFM, simulation circuit, programming examples

ملخص:

هذا العمل يمكننا من استعمال STEP7 لبرمجة بعض الأمثلة عن الأنظمة الآلية على الرجل الآلي القابل للبرمجة S7-314 IFM، الذي يتميّز باحتوائه على مداخل/مخارج TOR، مداخل/مخارج Analogiques ، و عنصر وظيفي.
التصنع الوظيفي للمداخل لازم لعدم استعمالنا لأنظمة آلية حقيقية . قد تم انحاز صفيحة إلكترونية لهذا الغرض.
STEP7 يمكننا من تقسيم البرنامج إلى عدّة عناصر منفصلة عن بعضها البعض، هذا يسهّل علينا البرمجة و تصحيح الأخطاء.
الأمثلة التي درست من خلال هذا العمل تثبت لنا إمكانية برمجة مجال واسع من الأنظمة، سواء في ميادين الصّناعة أو الخدمات و ذلك باستعمال S7-314 IFM.

مفتاح: STEP7،الرجل الآلي القابل للبرمجة S7-314 IFM،التصنع الوظيفي،أمثلة عن الأنظمة الآلية.