République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

ECOLE NATIONALE POLYTECHNIQUE

Département de Génie Electrique

PROJET DE FIN D'ETUDES

En vue de l'obtention du diplôme

D'ingénieur d'état en Automatique

Thème

CONCEPTION ET MISE EN ŒUVRE D'AUTOMATISATION BASEE SUR UN PC

Proposé et Dirigé par :

Dr M.BOURI Pr D.BOUKHETALA Pr F.BOUDJEMA

Etudié par :

BENALIA Samir MAHOUCHE Mahfoud

Promotion 2008

E.N.P. 10,Avenue Hassen Badi, 16200 EL-Harrach, Alger.

DEDICACES

Je dédie ce travail à mes parents, qui ont employés tous leurs moyens et efforts, pour m'élever, pour me former, et pour me faire arriver à ce niveau. A tous les professeurs qui ont contribués à ma formation du primaire a l'université.

Je dédie aussi ce travail à mes chers amis, Amine, Rym, Mohamed, Abderrahmen, Younes, et tant d'autres. A mon ami d'enfance et Binôme MAHFOUD, Ainsi qu'a notre belle promotion 2008 d'Automatique.

A ma chère amie Nesrine.

Samir

A ma tendre mère et a mon cher père.

A tous ceux que j'aime et qui m'aiment.

A ma famille et à mes amis

Mahfoud

REMERCIMENTS

Nous tenons à remercier nos promoteurs M. D.BOUKHTALA et M. F.BOUDJEMA pour la confiance qu'ils nous ont faite, et surtout un vif remercîment à M M.BOURI qui nous a aidé tout au long de notre travail, soutenu, et permit à mener a terme notre projet.

Nous remercions les membres du jury qui nous ont fait l'honneur de bien vouloir examiner notre travail.

Nous tenons à remercier toute personne ayant contribué de près ou de loin a l'aboutissement de nos efforts.

Merci a nos professeur et formateurs du primaire à l'université.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

Liste des abréviations

PC : **P**ersonal **C**omputer une marque déposée de International business Machines Corp.

POO : **P**rogrammation **O**rientée **O**bjet.

MMS :**M**anufacturing **M**essage **S**pecification.

CEI : **C**ommission **E**lectrotechnique **I**nternationale.

ISO : Organisation Internationale de Normalisation.

API : **A**utomate **P**rogrammable **I**ndustriel.

E/S : **E**ntrées/**S**orties.

ST : **S**tructured **T**ext (langage Texte Structuré).

LD : **L**adder **D**iagram (Diagramme à Echelle).

IL : **I**nstruction **L**ist (langage Liste d'Instructions).

FBD : **F**unction **B**lock **D**iagram (Diagramme de Blocs Fonctionnels).

SFC : **S**equential **F**unction **C**hart (Diagramme de Fonctions Séquentielles).

FC : **F**low **C**hart (Diagramme de Flux).

GRAFCET : **GRA**phe **F**onctionnel de **C**ommande **E**tapes/**T**ransitions.

HMI : Human Machine Interface (Interface homme/machine).

PLC : Programmable Logic Contoller (Automate programmable industriel).

DCS : Distributed Control System (Système de contrôle distribué).

PCI : **P**eripheral **C**omponent **I**nterconnect

DLL : **D**ynamic **L**ink **L**ibrary.

TTL : **T**ransistor-**T**ransistor **L**ogic

PIB : **P**eripheral **I**nterface **B**us (Bus interface de périphériques).

INTRODUCTION GENERALE

INTRODUCTION GENERALE

L'automatisation est considérée comme l'étape d'un progrès technique où apparaissent des dispositifs techniques susceptibles de seconder l'homme, non seulement dans ses efforts musculaires, mais également dans son travail intellectuel de monitoring, de supervision, et de contrôle.

Sur le marché du contrôle automatisé de procédés industriels les motivations économiques se traduisent par une recherche de solutions plus intégrées mais apportant plus d'ouverture et plus de modularité. Les récentes évolutions technologiques en matières de microprocesseurs, de systèmes d'exploitation multitâches, d'entrées/sorties industrielles distribuées sur les bus de terrain, tendent à rapprocher fortement l'automate programmable industriel, et le micro-ordinateur (PC) (Personal Computer une marque déposée de International business Machines Corp.).

Les solutions pour l'automatisation des systèmes sont diverses, ainsi que les sociétés qui les proposent, on trouve avec une large utilisation les API, mais aussi les solutions PC qui sont de plus en plus utilisées, les deux approches ont des avantages et des inconvénients, que nous argumenterons dans ce rapport. Pour améliorer la compatibilité, et la portabilité des aspects de l'automatisation, la CEI a mis en place des normes traitant aussi bien des langages de programmation que les architectures utilisés dans le domaine de l'automatisme.

Le travail présenté dans ce rapport est de concevoir une solution d'automatisation basée sur un PC, ayant au point de départ, un logiciel d'édition et de simulation ISAGRAF qui utilise les langages graphiques et syntaxique de la norme CEI 1131-3, une carte d'entrées/sorties PCI 8255 V3 de ETT, et un outil de programmation Visual C++. Notre but est de relier un éditeur de langages conforme à la norme CEI suscitée et un hardware d'ont on dispose, pour aboutir à une solution PC prête à être utilisée.

PLAN DU PROJET

Le problème abordé au cours de ce travail est celui de la conception, et de la mise en œuvre d'une solution d'automatisation basée sur un PC.

Le chapitre I est consacré à l'étude des avantages et inconvénients des solutions d'automatisation basées sur PC et API, nous justifierons les aspects des deux solutions à travers un comparatif. On traitera aussi et des normes en vigueur dans le domaine de l'automatisation.

Une fois les avantages et les inconvénients mis en évidence il reste à spécifier les langages de programmation des systèmes d'automatisation de la norme CEI 1131-3, le chapitre II traite ces langages et nous prendrons comme exemple de logiciel de programmation conforme à cette norme ISAGRAF.

Le chapitre III quant à lui, traite du développement de l'application que nous avons appelé FlexPLC qui interprète et exécute les programmes générés par ISAGRAF, sur la carte PCI8255 v3 qui nous a été fournie par Moveit Automation. Nous mettrons en relief le travail effectué par une démonstration, sur un modèle réduit d'une unité de tri.

CHAPITRE I

SOLUTIONS D'AUTOMATISATION

ET NORMES EN VIGUEUR

CHAPITRE I

SOLUTIONS D'AUTOMATISATION ET NORMES EN VIGUEUR

I.1 INTRODUCTION

Les solutions basées PC des systèmes d'automatisation sont actuellement, disponibles et de grandes marques telles que SIEMENS ou BECKHOFF proposent une large gamme de produit dans ce segment, ou proposent des solutions combinées entre PC et API. BECKHOFF est l'un des pionniers de l'Automatisation basée PC. Le premier système de contrôle PC a été livré dès 1986.

Dans le but de mettre en œuvre une solution d'automatisation basée PC nous devons avoir des points de repères, tant sur le niveau fiabilité de cette solution que sur les normes utilisées dans le domaine. Pour concevoir une solution PC fiable et offrant une compatibilité large, nous avons donc, à travers la bibliographie, cherchés à mettre en évidence les avantages et inconvénients des solutions PC et API. Et de connaître les grandes lignes des normes en vigueur dans le domaine des systèmes automatisés.

Dans le présent chapitre nous allons traiter des avantages et inconvenants de chaque solution, et connaître quelles sont les normes internationales en ce qui concernent l'automatisation des systèmes, pour faire écouler nos efforts vers un système compatible [7].

I.2 AVANTAGES ET INCONVENIENTS DES SOLUTIONS PC ET API

I.2.1 AVANTAGES DES SOLUTIONS BASEES PC

Le PC est souvent utilisé pour commander directement des automatismes, car, dans ce type d'applications, il apporte une puissance de calcul énorme à un prix non onéreux, en comparaison avec une solution automate embarquée classique. L'immense diffusion du PC permet de trouver une incroyable quantité de périphériques de qualité professionnelle à des prix corrects. De plus, le PC dispose d'une grande variété de logiciels et de nombreux environnements de développement. Il a permis d'atteindre la totale portabilité des applications. En effet, une application développée sur un matériel de type PC, d'une marque donnée, peut être exécutée sans aucune modification sur un PC d'une autre marque, à l'unique condition de disposer du même système d'exploitation. Ainsi, un utilisateur du PC bénéficie d'une grande indépendance vis-à-vis du fournisseur du matériel et spécifiquement de la carte mère.

De même, des fabricants de cartes ou éditeurs logiciels respectant les standards PC, peuvent équiper des PC de différentes marques. Aujourd'hui, le standard PC tant matériel que logiciel est bien une réalité.

L'utilisation du PC s'impose dans différents cas. Par exemple :

-l'automatisme doit obéir à une logique très complexe,

- on veut réaliser une interface utilisateur sophistiquée (écran couleur, navigation à l'aide d'une souris)

- l'application est en liaison avec des informations situées sur un PC (bases de données)

- le développeur a une culture des environnements PC et non pas des machines dédiées à l'automatisme (API)

- l'application doit incorporer des éléments tels que le multimédia (synthèse vocale ou capture vidéo par exemple)
- l'application est intégrée dans un réseau de PCs.

En général, le PC le plus simple aura une puissance de calcul et des capacités mémoire plus que suffisantes pour toute application en relation avec le monde extérieur. Les temps de réponse mécaniques et les phénomènes physiques sont généralement longs par rapport à la vitesse de calcul du PC, et sauf dans le cas d'algorithmes très complexes, on peut même considérer qu'ils sont négligeables.

I.2.2 LIMITATIONS DES SOLUTIONS PC

I.2.2.1 SUR LE PLAN MATERIEL

La vitesse de renouvellement de la technologie du PC est extrêmement rapide au regard de l'échelle de temps d'un procédé de fabrication (10-15 ans). La durée de vie d'un procédé industriel apparaît difficilement compatible avec les évolutions très rapides du PC qui pose immanquablement le problème de la pérennité de la solution du contrôle du procédé. 18 mois après la commercialisation d'un modèle de PC, il n'est plus disponible sur le marché, par voie de conséquence, la gestion du stock de pièces de rechange n'est pas chose aisée.

La sensibilité aux contraintes des environnements industriels (vibrations, chocs, températures, corrosions...) se traduit par un accroissement très important du prix des PC pour obtenir un « durcissement » de ceux-ci ou requiert une mise en armoire de protection. Aussi Les contraintes industrielles (rayonnements) sont souvent incompatibles avec l'utilisation intensive de supports d'informations magnétiques (disque dur…). Des unités de stockage statiques de données sont certainement mieux adaptées aux contraintes industrielles.

I.2.2.2 SUR LE PLAN LOGICIEL

La cible initiale du PC, le marché de la bureautique et le marché grand public, n'a pas originellement imposé de doter le PC de capacité de traitement temps réel. Un système d'exploitation temps réel efficace est une condition nécessaire pour le contrôle de procédés industriels.

Lors du redémarrage du PC, le procédé industriel se trouve momentanément sans aucun contrôle, suite à un cas d'exception non géré par le système d'exploitation, et/ou l'application écarte actuellement le PC de bon nombre d'applications de contrôle commande.

Les virus, véritable maladie du PC, constituent une infection présentant un caractère périlleux à l'utilisation d'un PC pour assurer le contrôle d'une machine ou procédé industriel. Un virus peut en effet rendre indisponible une ressource (donnée, fichier...) indispensable à la tâche de contrôle et provoquer ainsi des perturbations, voire l'arrêt de l'outil de production.

Sous les systèmes d'exploitations tels que Windows, il n'est pas possible de garantir le déterminisme temporel des événements et des tâches en exécution. Ce déterminisme temporel que nous appelons caractéristique temps réelle d'un système d'exploitation n'est pas garanti par Windows. Il est par contre possible de lui adjoindre une extension temps réelle tels que RTX, HYPERKERNEL, INTIME ou d'autres. Windows devient alors déterministe à 10µs près.

I.2.3 AVANTAGES DES SOLUTIONS BASEES API

Les principales forces de l'API résident certainement dans ces valeurs d'usage :

-Les modes de marche (manuel/semi-automatique/automatique), d'arrêt, de remise en cycle, de reprise à froid, de reprise à chaud.

-La sauvegarde permanente de son contexte (valeur des différentes variables du système d'exploitation) qui est indispensable pour autoriser la reprise à chaud.

-La capacité de modifier en ligne le programme de contrôle en cours d'exécution évitant ainsi la perturbation de l'exploitation de l'outil de production. L'arrêt du procédé industriel est impossible ou à coût prohibitif pour les procédés continus ou semi continus.

-La branchement/débranchement sous tension des modules.

-Sa non-sensibilité aux perturbations et variations de la source électrique.

-Une chaîne ininterrompue de détection et de gestion des défauts.

-Tous ces services/fonctionnalités s'avèrent indispensables pour assurer la continuité de service requise par le contrôle industriel.

I.2.4 LIMITATIONS DES SOLUTIONS BASEES API

I.2.4.1 SUR LE PLAN LOGICIEL

La principale faiblesse à souligner réside dans la non-indépendance de l'application et du matériel, car les systèmes sont construits sur des architectures spécifiques à chaque base matérielle. En effet, jusqu'à présent, l'absence de standards forts et reconnus au niveau du logiciel (système d'exploitation compris) ne permettait pas la portabilité des applications développées pour un API d'un constructeur donné vers un matériel d'un autre constructeur. Le développement de la norme CEI1131 contribue essaye de gommer progressivement cette limitation.

Pour chaque plate-forme matérielle cible, il existe un nombre fort limité de chaînes complètes et intégrées (environnement de développement, éditeurs langages, outils de mise au point, compilateurs, chargeurs, outils de diagnostic...) permettant de gérer l'application pour la durée de vie du procédé à contrôler.

La capacité d'ouverture de l'API est très faible, voire inexistante par le passé. En effet, accéder à la base de données temps réel de l'API impose un partenariat étroit avec le constructeur pour avoir accès aux formats d'échanges, aux requêtes systèmes et aux formats internes des données. Le jeu d'instructions exclusivement centré sur le traitement booléen et les opérateurs arithmétiques de base se révèlent insuffisant pour effectuer du traitement d'informations plus sophistiqué. De même, les structures de données souvent limitées au tableau de données de même type et l'inexistence de notion de fichiers de données peut être considérée comme un maillon faible de l'API (pour assurer des calculs statistiques, des tendances, des courbes d'expérience...).

I.2.4.2 SUR LE PLAN MATERIEL

De même que pour le logiciel, l'absence de standards contraint l'utilisateur d'API à être verrouillé sur une seule source. Chaque pièce constituant la solution basée API (bacs ou platines, alimentations électriques, processeurs, modules d'entrées/sorties, coupleurs, coupleurs de communication...) pour une plate-forme donnée ne peut être qu'exclusivement approvisionnée chez un seul constructeur. Les contraintes de l'environnement (rayonnements, chocs) conduisent à écarter certaines technologies ou bien à en restreindre leur usage (mémoire magnétique...). Ceci se traduit par une très faible capacité de stockage d'informations. Par exemple, tous les attributs des variables utilisées dans une application ne sont pas stockés dans l'API pour des raisons économiques : les mémoires statiques utilisées possèdent un rapport capacité/coût prohibitif pour des stockages à grande échelle. L'API ne dispose pas naturellement d'organe de dialogue homme/machine évolués (limité à des leds affichant la valeur d'une voie tout-ou-rien ou la présence d'un défaut).

En conclusion à ce comparatif, nous ferons remarquer que pour automatiser un système industriel, nous devons décider quel aspect de cet automatisme privilégier, selon la complexité du système, les contraintes d'environnement, le besoin de stockage et de calcul, et la façon de visualiser le processus sachant que de plus en plus de machines requièrent un pupitre de conduite doté d'un écran pour des affichages textuels et/ou graphiques. L'utilisateur pourra par la suite décider de la stratégie de commande, et de la solution à adopter.

I.3 LES NORMES EN VIGUEUR POUR LES SYSTEMES AUTOMATISES

Les normes permettent de remplacer aisément un produit par un équivalent quand on rencontre une difficulté d'approvisionnement quelconque, ou d'échange. De plus, elles permettent une interopérabilité des systèmes et produits industriels entre eux. Quoique volontaires par nature, elles sont donc devenues indispensables.

On peut rencontrer la norme CEI 61131 avec l'appellation CEI 1131, le chiffre 6 avant 1131 est apparu, ceci pour harmoniser les notations locales des pays, et les réunir dans une numérotation plus large a caractère international.

I.3.1 PRESENTATION DU STANDARD CEI 61131

La diversité des aspects intervenant dans la conception des systèmes de contrôle se reflète dans celle des langages utilises pour leur programmation. Elle comporte des aspects historiques, comme la diversité de cultures techniques (automatique, électromécanique, informatique...), l'héritage de volumineuses spécifications, le caractère éprouve vis-à-vis de la sûreté.

Ces langages sont liés à l'utilisation de plate formes spécifiques pour l'exécution des contrôleurs typiquement les automates programmables industriels (API). Ils présentent des traits directement lies à un schéma d'exécution, et qui repose explicitement sur des fonctionnements cycliques, et sur des architectures matérielles représentées explicitement, parfois proches des circuits électroniques. Bien que la programmation évolue vers une séparation du matériel, par des couches logicielles de plus en plus élaborées, ces langages continuent d'être pratiqués, au moins pour les motivations historiques évoquées ci-dessus, et même dans les cas ou l'architecture matérielle utilisée est à base de PC.

Des efforts de normalisation tendent à promouvoir l'unification de ces langages et de ces architectures, et de la communication entre elles. En particulier, la norme CEI 61131 (la première édition date de 1993) définit la programmation des automates, et propose un cadre qui s'étend de la spécification aux architectures. Cette norme comporte à l'origine cinq parties (Introduction, Matériel, Langages de programmation (CEI-61131-3), Manuel et conseils à l'utilisateur, Communications.), et huit dans sa dernière version [6].

PLCOpen, est une organisation fondée en 1992 dédiée au développement de la norme CEI 61131 en particulier à la troisième partie 61131-3 [15]

I.3.2 LES PARTIES DE LA NORME CEI 61131

La norme CEI 61131 est maintenant divisée en 8 parties

CEI 61131-1 General information

Etablie les définitions et identifie les caractéristiques principales relevant de la sélection et de l'application des contrôleurs programmables et leurs périphériques associés.

CEI 61131-2 Equipment requirements and tests

Spécifie besoins des équipements et les testes liés pour les API et leurs périphériques associés.

CEI 61131-3 Programming Languages - providing the basis

définit comme un ensemble minimal, les éléments de programmation de base, les règles syntaxique et sémantique pour les langages de programmation le plus communément utilisés, y compris les langues graphiques comme Ladder Diagram Diagramme bloc fonctionnel, et les langages textuels Instruction List et de Structured Text, ainsi que les domaines d'application les plus important, les tests et les moyens par lesquels les fabricants peuvent étendre ou d'adapter ces ensembles de base à leurs propres implémentations.

CEI 61131-4 User Guidelines

Un rapport technique qui fournit une vue d'ensemble informations et applications des grandes lignes du standard, pour l'utilisateur final de la solution automatisée.

CEI 61131-5 Messaging service specification

Définit les données de communication entre les automates et les autres systèmes électroniques utilisant la spécification de la messagerie industrielle (Manufacturing Message Specification MMS) selon la norme ISO/CEI 9506.

CEI 61131-7 Fuzzy control programming

Définit les éléments de base pour le contrôle par logique floue, comme utilisé dans l'API

CEI 61131-8 Grandes lignes pour l'application et l'implémentation des langages de programmation offrant aux développeurs un guide de langages de programmation définies dans la partie 3

La section 6 est réservée pour un usage ultérieur. Et la troisième partie, CEI 1131-3, est celle qui nous intéresse particulièrement ici, elle décrit les langages de programmation et

leur sémantique. Elle définie les standards des langages et se présente sous la forme de 4 sections et d'une série de 62 tables de conformité. Lorsque l'on décrit la compatibilité d'un progiciel à ce standard, on compare généralement son jeu d'instructions avec ces tableaux de conformité.

Parmi les produits qui ont été approuvés par l'organisation PLCopen, nous pouvons mentionner : Altersys ISAGRAF, de C.J. International [16], AnchorPAW, Performance Software Associates, Inc [17], Siemens Step 7 [18], et bien d'autres.

CHAPITRE II

LES LANGAGES DE LA NORME CEI 61131-3

ET ELEMENTS D'ISAGRAF

CHAPITRE II

LES LANGAGES DE LA NORME CEI 61131-3

ET ELEMENTS D'ISAGRAF

II.1 INTRODUCTION

Comme nous l'avons vu dans le chapitre précédent la norme CEI 61131 traite dans sa troisième partie des langages des langages de programmation des automates, et l'organisation PLCopen peut valider des logiciels de programmation pour les systèmes automatisés, si ce dernier est conforme aux standards internationaux. Dans ce chapitre nous allons décrire les langages de programmation de la norme CEI 1131-3 [6], prendre comme exemple de logiciel « ISAGRAF ». Il est totalement compatible avec a norme et il est largement utilisé, aussi bien dans les applications orientées API que celles orientées PC, « ISAGRAF » fera par la suite partie intégrante de notre solution PC.

II.2 LES LANGAGES DE LA NORME CEI 1131-3

II.2.1 LANGAGE ST

Le « Structured Text » (ST), est un langage de programmation structuré de la norme CEI 1131-3. C'est un langage adapté aux systèmes d'automatisation. La structure générale d'une instruction est la suivante [6]:

<Variable> := <opérande> <opérateur> <opérande> .... ;

Chaque instruction se termine par le caractère « ; »

II.2.1.1 LES OPERATEURS

Opérateur d'affectation

En « Structured Text » l'opérateur d'affectation est le symbole « := ».

Opérateurs logiques

Opérateur	Description	Opérateur	Description
NOT	Négation logique	AND	ET logique
OR	OU logique	XOR	OU logique exclusif

Tableau II.1 *Opérateur logique du langage ST*

Opérateurs arithmétique

Opérateur	Description	Opérateur	Description
+	Addition	-	Soustraction
*	Multiplication	/	Division
mod	Reste de la division		

Tableau II.2 *Opérateur arithmétique du langage ST*

Opérateurs de comparaison

Opérateur	Description	Opérateur	Description
=	Egal à	<>	Différent de
>	Supérieur à	>=	Supérieur ou égal à
<	Inférieur à	<=	Inférieur ou égal à

Tableau II.3 *Opérateur de Comparaison du langage ST*

II.2.1.2 LES STRUCTURES DE CONTROLE

Instructions conditionnelles

pseudo langage	Langage ST	Langage C

Si (<expression logique>)
Alors
Bloc action 1
 	
Sinon Si (expression logique)
Alors
Bloc action 2
 	
.....
Sinon
 	
Bloc action 3
Fin de Si
	IF <expression logique>
	THEN
	Bloc action 1 ;
	
	ELSIF <expression logique>
	THEN
	Bloc action 2 ;
	
	.....
	ELSE
	
	Bloc action 3 ;
	END_IF
		if (<expression logique>)
		{
		Bloc action 1 ;
		}
		else if (<expression logique>)
		{
		Bloc action 2 ;
		}
		.....
		else
		{
		Bloc action 3 ;
		}

Tableau II.4 *Instructions conditionnelles du langage ST*

Instruction de choix

Pseudo langage	Langage ST	Langage C

Décider sur (<expression>)
Entre
Val_1 :
Bloc action
.....
Val_2, Val_3, ... :
Bloc action
 	
.....
Val_i à Val_j :
Bloc action
.....
default :
Bloc action
Fin de Décider
	CASE <expression>
	OF
	Val_1 :
	Bloc action
	.....
	Val_2, Val_3, ... :
	Bloc action
	
	.....
	Val_i..Val_j :
	Bloc action
	.....
	ELSE
	Bloc action
	END_CASE ;
		switch (<expression>)
		{
		case Val_1 :
		Bloc action ; break ;
		.....
		case Val_2 :
		case Val_3 :
		case ....
		Bloc action ; return(n) ;
		.....
		Pas de correspondance en C
		.....
		default :
		Bloc action ;
		}

Tableau II.5 *Instruction de choix du langage ST*

Instructions répétitives

Pseudo langage	Langage ST	Langage C

Tant que (<expression logique>)
 	
Bloc action
Répéter
	WHILE <expression logique>
	DO
	Bloc action ;
	END_WHILE ;
		while (<expression logique>)
		{
		Bloc action ;
		}

Répéter
 	
Bloc action
 	
Tant que (<expression logique>)
	REPEAT
	
	Bloc action ;
	
	UNTIL <expression logique>
	END_REPEAT ;
		do
		{
		Bloc action ;
		}
		while (<expression logique>)
		;

Pour variable = valeur initiale
Jusqu'à valeur final
Par incrément
Faire
Bloc action
Fin de Pour
	FOR variable := valeur initiale
	TO valeur final
	BY incrément
	DO
	Bloc action ;
	END_FOR ;
		for(variable = valeur initiale;
		variable <> valeur final;
		variable += incrément)
		{
		Bloc action ;
		}

Tableau II.6 *Instructions répétitives du langage ST*

ATTENTION : L'utilisation de structures répétitives peut compromettre des tâches cycliques ou périodiques.

Instruction « EXIT »

L'instruction « EXIT » permet d'interrompre une boucle répétitive.

Langage ST	Langage C
Début de boucle *Bloc action ;* **EXIT ;** *Bloc action ;* *Fin de boucle ;*	*Début de boucle* *Bloc action ;* **break ;** *Bloc action ;* *Fin de boucle ;*

Tableau II.7 *Instruction EXIT du langage ST*

Instruction « RETURN »

L'instruction « RETURN » permet d'interrompre une unité d'organisation de programme comme une fonction, un bloc fonctionnel ou un programme. Contrairement au langage C l'instruction « RETURN » ne permet pas de retourner une valeur.

II.2.2 LANGAGE LD

Le langage LD est une représentation graphique d'équations booléennes combinant des contacts (en entrée) et des relais (en sortie) [11]. Le langage LD permet la manipulation de données booléennes, à l'aide de symboles graphiques organisés dans un diagramme. Les symboles du diagramme LD sont organisés comme les éléments d'un schéma électrique à contacts. Les diagrammes LD sont limités à gauche et à droite par des barres d'alimentation.

II.2.2.1 BASES DU LANGAGE LD

Un programme LD est une suite d'équations appelées *échelles* où les contacts et les bobines de relais sont arrangés. Voici les composants de base d'un diagramme LD:

Début d'échelle (barre d'alimentation à gauche)

Chaque échelle commence par une barre d'alimentation à gauche, qui représente l'état VRAI. L'éditeur LD crée automatiquement la barre d'alimentation à gauche quand le premier

contact de l'échelle est placé par l'utilisateur. Chaque échelle peut avoir un nom logique qui peut être utilisé comme une étiquette dans les instructions de saut.

Fin d'échelle (barre d'alimentation à droite)

Une échelle se termine par une barre d'alimentation à droite, l'éditeur LD ajoute automatiquement la barre d'alimentation à droite de chaque bobine de relais posée par l'utilisateur.

Les contacts

Un contact modifie le flux de données booléen, selon la valeur de la variable booléenne qui lui est associée. Le nom de la variable est affiché en dessus du dessin du contact. Les types de contacts suivants sont supportés par l'éditeur LD:

Symbole	Signification	Symbole	Signification
	contact direct		contact inversé
	contact avec détection de front montant		contact avec détection de front descendant

Tableau II.8 *Les contacts du langage LD [6]*

Les relais

Une bobine de relais représente une action. La variable associée au relais est forcée selon l'état de l'échelle à gauche du relais. Le nom de la variable est affiché en dessus du dessin du relais. Les types de relais suivants sont supportés par l'éditeur LD:

Symbole	Signification	Symbole	Signification
	sortie directe		sortie inversée
	sortie à action "set"		sortie à action "reset"
	sortie avec détection de front montant		sortie avec détection de front descendant

Tableau II.9 *Les relais du langage LD [6]*

Autres symboles

Symbole	Signification
	Saut
	Return

Tableau II.10 *Autres symboles du langage LD*

Blocs fonctionnels

Un bloc dans un diagramme LD peut représenter un opérateur, une fonction, un sous-programme ou un bloc fonctionnel. Sa première entrée et sa première sortie sont toujours connectées sur l'échelle. Les autres paramètres en entrée et en sortie sont écrits en dehors du rectangle du bloc.

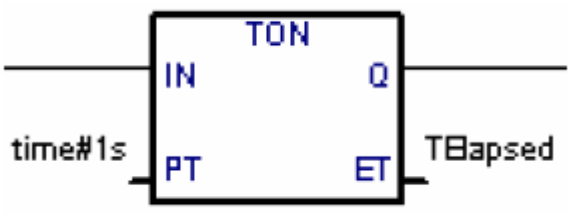

Figure II.1 *Bloc fonctionnel du langage LD*

Symbole de saut

Un symbole de saut se réfère toujours à une étiquette (nom d'échelle) définie ailleurs dans le même diagramme LD. Le symbole de saut est posé à la fin d'une échelle. Quand l'état de la liaison à gauche du symbole est TRUE, l'exécution du diagramme est déroutée vers l'échelle identifiée par l'étiquette associée au saut. Notez que les sauts en arrière sont dangereux car ils peuvent bloquer le cycle automate.

Symbole "Return"

Un symbole "Return" est placé à la fin d'une échelle. Il indique que l'exécution du programme doit être abandonnée si l'état de la liaison est TRUE. Un symbole "Return" est équivalent à un saut après la dernière échelle du diagramme.

II.2.3 LANGAGE IL

Le langage IL (Instruction List), est un langage textuel de bas niveau proche de l'assembleur. Il est particulièrement adapté aux applications de petite taille, ou à l'optimisation de portions réduites d'une application. Les instructions travaillent toujours sur un *résultat courant* (ou registre IL).L'opérateur indique le type d'opération à effectuer entre le résultat courant et l'opérande. Le résultat de l'opération est stocké à son tour dans le résultat courant.

Label	Opérateur	Opérande	Commentaires
<Etiquette>:	<Opérateur>(<Modifieur>)	<Opérande>	(* Accumulateur = Accumulateur <Opérateur>(<Modifieur>) <Opérande> *)
TOTO :	ADD	25	(* Accumulateur = Accumulateur + 25 *)

Tableau II.11 *Structure du langage IL*

Le label ou étiquette se termine toujours par le caractère « : ». Il est optionnel et Il précède une ligne d'instruction. Le label est utilisé lors de sauts conditionnels ou non.

L'opérateur correspond à l'opération à réaliser entre l'accumulateur et l'opérande. Le résultat est stocké dans l'accumulateur. Un modificateur peut être associé à l'opérateur. L'opérande est une constante ou une variable. Il est possible d'associer à chaque ligne d'instruction un commentaire. Le début de commentaire se compose des deux caractères « (* » et la fin de commentaire se compose des deux caractères « *) ».

II.2.3.1 LES OPERATEURS

Opérateurs d'affectation

Opérateur	Description	Opérateur + Modificateur	Description
LD	Accu=Opérande(Op)	LDB	Accu= not Op
ST	Op=Accu	STN	Op= not Accu
S	Si Accu=1 alors Set Op		
R	Si Accu=1 alors Reset Op		

Tableau II.12 *Opérateurs d'affectations du langage IL*

Opérateurs logiques

Opérateur	Description	Opérateur+ Modificateur	Description
AND	Accu=Accu AND Op	ANDN	Accu=Accu AND not Op
OR	Accu=Accu OR Op	ORN	Accu=Accu OR not Op
XOR	Accu=Accu XOR Op	XORN	Accu=Accu XOR not Op
NOT	Accu=not Accu		

Tableau II.13 *Opérateurs logique du langage IL*

Opérateurs arithmétiques

Opérateur	Description	Opérateur	Description
ADD	Accu=Accu + Op	SUB	Accu=Accu –Op
MUL	Accu=Accu *Op	DIV	Accu=Accu /Op
MOD	Accu=Rest de Accu/Op		

Tableau II.14 *Opérateurs arithmétiques du langage IL*

Opérateurs de comparaison

Opérateur	Description	Opérateur	Description
GT	Accu=Vrai si Accu>Op	GE	Accu=Vrai si Accu>=Op
LT	Accu=Vrai si Accu<Op	LE	Accu=Vrai si Accu<=Op
EQ	Accu=Vrai si Accu=Op	NE	Accu=Vrai si Accu<>Op

Tableau II.15 *Opérateur de comparaison du langage IL*

Branchements

Opérateur	Description	Opérateur	Description
JMPC	Saut à Label si Accu <>0	JMPCN	Saut à Label si Accu =0
JMP	Saut à Label	CAL	Appel d'un bloc de fonction
RET	Instruction « return »		
RETC	« return » si Accu <>0	RETCN	« return » si Accu=0

Tableau II.16 *Branchement dans le langage IL*

Modificateur « (» et Opérateur «) »

Le modificateur « (» peut être associé aux opérateurs logiques, arithmétiques et de comparaison (par exemple « ADD(»), Il provoque l'ouverture d'une parenthèse par la création d'un deuxième accumulateur. L'opérateur «) » provoque la fermeture de parenthèse.

Exemple :

Res = (A + B) * (C + D)

LD A (* Accu = A *)

ADD B (* Accu = Accu + B = A + B *)

MUL(C (* Ouverture de parenthèse, Accu2 = C *)

ADD D (* Accu2 = Accu2 + D = C + D *)

) (* Fermeture de parenthèse Accu = Accu * Accu2 = (A + B) * (C + D) *)

ST Res (* Res = Accu *)

II.2.4 LANGAGE FBD

Le FBD (Function Block Diagram) est un langage graphique. Il permet la construction d'équations complexes à partir des fonctions élémentaires.

Le diagramme FBD décrit une fonction entre des *variables d'entrée* et des *variables de sortie*. Une fonction est décrite comme un réseau de *fonctions élémentaires*. Les variables d'entrée et de sortie sont connectées aux boîtes fonctions par des *arcs de liaison*. Une sortie d'une boîte peut être connectée sur une entrée d'une autre boîte [13].

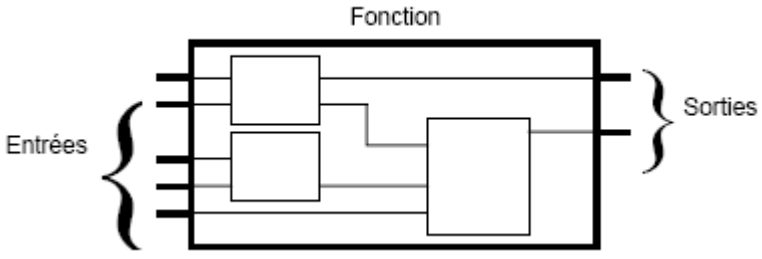

Figure II.2 *Structure du langage FBD*

Chaque boîte fonction élémentaire a un nombre prédéfini de *points de connexion* en entrée et en sortie. Une boîte fonction est représentée par un *rectangle*. Ses entrées sont connectées sur le bord *gauche* du rectangle, et Ses sorties bord *droit*.

Une boîte fonction élémentaire réalise une fonction élémentaire entre ses entrées et ses sorties. Le nom de la fonction réalisée est inscrit dans le rectangle de la boîte. Chaque entrée ou sortie de la boîte est caractérisée par un *type*.

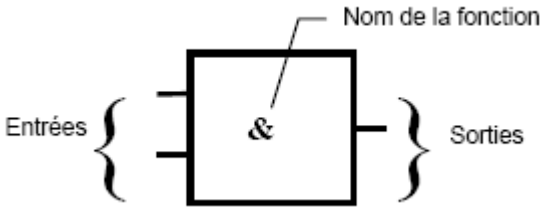

Figure II.3 *Fonction élémentaire du langage FBD*

Les variables d'entrée du diagramme FBD doivent être connectées aux entrées des boîtes fonctions. Le type de chaque variable doit être cohérent avec le type de l'entrée de la boîte correspondante. Une entrée pour le diagramme FBD peut être une expression *constante*, toute variable *interne*, d'*entrée* ou de *sortie.*

Les variables de sortie du diagramme FBD doivent être connectées aux sorties des boîtes fonctions. Le type de chaque variable doit être cohérent avec le type de la sortie de la boîte correspondante. Une sortie pour le diagramme FBD peut être une variable *interne* ou de *sortie*. Des lignes sont utilisées pour représenter les liaisons entre deux points du diagramme:

- Une variable d'entrée et une entrée de boîte

- Une sortie d'une boîte et une entrée d'une autre boîte

- Une sortie de boîte et une variable de sortie

Les liaisons sont *orientées*: une liaison transporte une information depuis son extrémité gauche vers son extrémité droite. Les extrémités gauche et droite doivent être connectées à des éléments de même type. Des liaisons multiples à droite sont utilisées pour transmettre une information depuis une extrémité à gauche vers plusieurs points à droite. Toutes les extrémités de la liaison doivent avoir le même type.

II.2.5 LANGAGE SFC

II.2.5.1 INTRODUCTION

Le « Sequential Function Chart » (SFC) est un outil de programmation. Ce langage est inspiré du GRAFCET (norme CEI 60848), mais il n'y a aucune identité entre les deux représentations, graphiques et sémantique des deux langages.

Le langage SFC est un langage graphique, et donc visuel. Il permet de bien représenter les différents états d'un système séquentiel. Associé aux autres langages (IL, ST, LD), il permet d'obtenir une programmation claire et lisible.

Le SFC propose un moyen de structuration, et d'organisation interne, d'une unité d'organisation de programme, à travers un ensemble de « steps », de « transitions », et de liaisons orientées. Un ensemble d'actions est associé aux étapes, et des conditions sont associées aux transitions.

II.2.5.2 BASES DU LANGAGE SFC

Le langage SFC décrit des comportements séquentiels. Il divise le procédé en un nombre d'étapes connues (situations stables), séparées par des transitions.

Les symboles d'un graphe SFC sont reliés par des *liaisons orientées*. L'orientation par défaut est *du haut vers le bas*. Voici les composants graphiques utilisés pour la construction d'un graphe SFC:

Symbole	Signification	Symbole	Signification
	Etape initiale		Etape
	Transition		Renvoi à une étape
	Macro-étape		Etape de début d'une macro
	Etape de fin d'une macro		

Tableau II.17 *Composants graphique du langage SFC[13]*

La programmation SFC se décompose en deux niveaux. Le *niveau 1* montre le graphe, les numéros de référence et les commentaires attachés aux étapes et aux transitions. Le *niveau 2* est la programmation en ST ou IL des actions dans les étapes et des conditions attachées aux transitions. Les actions et conditions peuvent appeler des *sous-programmes* écrits dans d'autres langages (FBD, LD, ST ou IL). Voici un exemple de programmation de niveau 1 et de niveau 2.

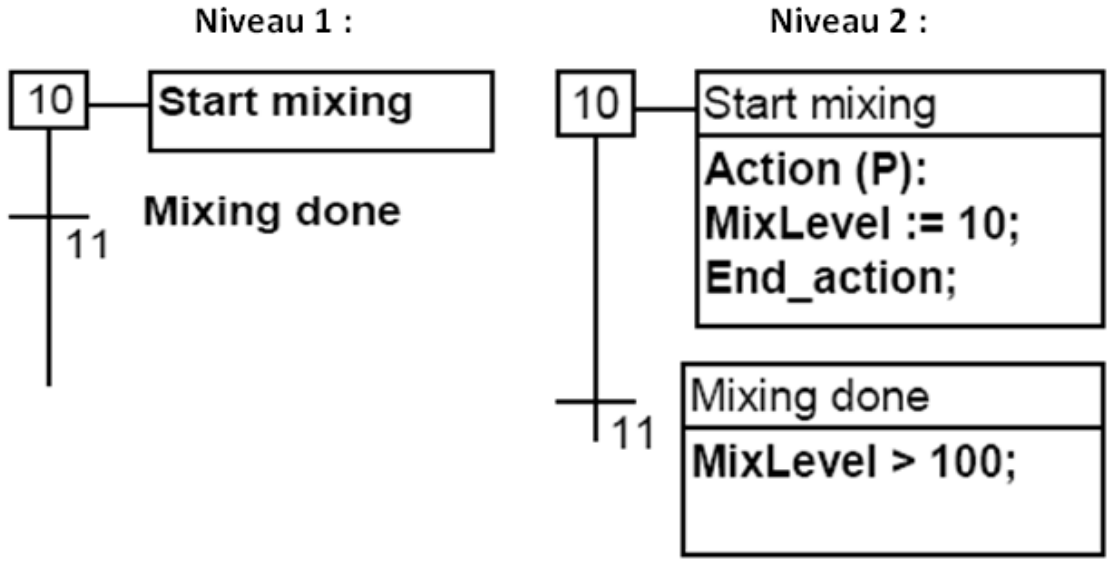

Figure II.4 *les éléments constituant d'un programme SFC*

Le niveau 2 d'une étape est entré à l'aide d'un éditeur de textes. Il contient des blocs d'action écrits en ST ou IL. Le niveau 2 d'une transition peut être saisi soit en langage texte IL ou ST, soit avec l'éditeur LD (schéma à contacts).

Divergences et convergences

Les divergences et les convergences sont utilisées pour représenter des liaisons multiples entre les étapes et les transitions. Les divergences et convergences simples représentent plusieurs possibilités (inclusives) dans l'exécution du procédé.

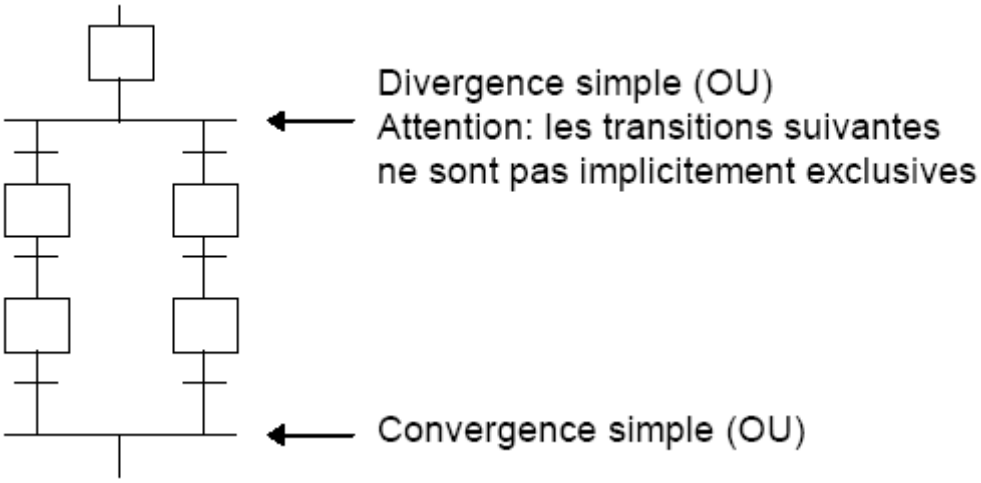

Figure II.5 *Divergences et convergences simple[13]*

Les divergences doubles représentent des procédés **parallèles**.

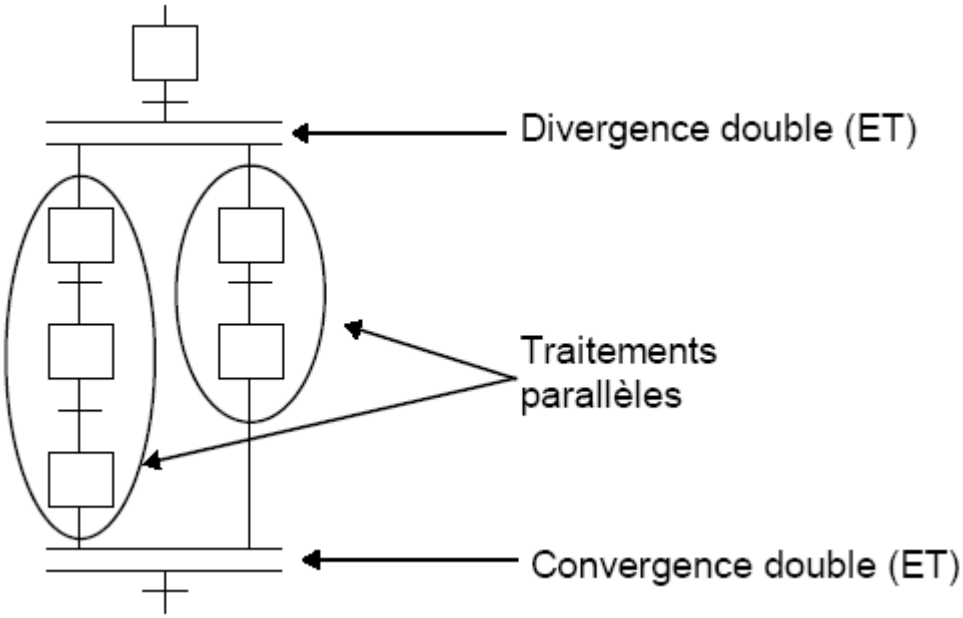

Figure II.6 *Divergences et convergences double*

Saut à une étape

L'éditeur SFC ne visualise que les liaisons tracées du haut vers le bas. Un *renvoi* à une étape peut être utilisé pour représenter un renvoi vers une partie antérieure du graphe. Par exemple:

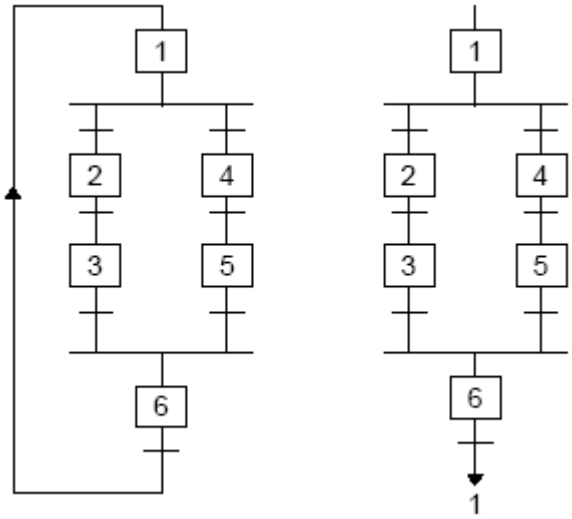

Figure II.7 *Saut a une étape dans un programme SFC [13]*

Un renvoi à une transition est interdit, et doit être explicitement représenté comme une convergence double.

Macro-étape

Une macro-étape est la représentation sous la forme d'un symbole unique d'un groupe *unique* et *connexe* d'étapes et de transitions. Le corps d'une macro-étape commence par une *étape de début* et se termine par une *étape de fin*.

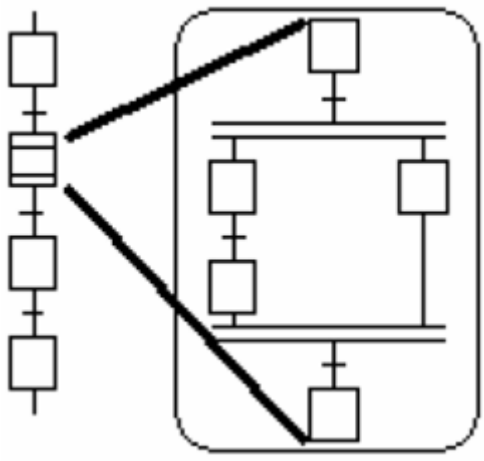

Figure II.8 *Macros-étape dans un programme SFC[13]*

La représentation de la macro-étape doit être décrite dans le même programme SFC. Le symbole de la macro-étape et son étape de début doivent avoir le même *numéro de référence*. Le corps d'une macro-étape peut contenir le symbole d'une autre macro-étape.

II.3 ELEMENTS D'ISAGRAF

II.3.1 INTRODUCTION

ISAGRAF est une solution logicielle sous licence, utilisée pour développer des applications d'automatisme, conformément à la norme CEI 1131-3 [13]. Avec l'utilitaire "Librairies", l'utilisateur dispose de fonctions permettant de gérer son projet.

Il permet de contrôler en local ou à distance une plateforme matérielle (programmation mono automate), ou plusieurs plateformes matérielles à la fois (programmation multi automates distribuée), avec échange de données entre Automates. ISAGRAF est aussi un package complet permettant d'utiliser les fonctions avancées d'alarmes, Web HMI, drivers, pouvant être utilisées comme un DCS, PLC, etc.

Dans ce chapitre nous allons nous intéresser à la manière de créer un projet écrit en SFC, ce qui est un point de départ de notre solution PC.

II.3.2 ISAGRAF ET LES LANGAGES CEI 1131-3

ISAGRAF gère une librairie de fonctions et de blocs fonctionnels écrits en langages CEI 1131-3. Les langages disponibles pour décrire une fonction ou un bloc sont le FBD (Function Block Diagram ou logigramme), le LD (Ladder Diagramme ou schéma à contact), le ST (Structured Text) et le IL (Instruction list). Notez que le LD et le FBD peuvent être

mêlés dans le même diagramme. Le langage SFC (Séquential Function Chart) peut être utilisé avec des morceaux de programmes en ST en LD.

II.3.3 EMPLOI D'ISAGRAF

II.3.3.1 FENETRE GESTIONNAIRE DE PROJETS

Lancer le gestionnaire de projets d'ISAGRAF sous Windows aura comme résultat l'affichage de la fenêtre figure II.9. La fenêtre supérieure montre la liste des projets existants. Le descripteur du projet sélectionné est affiché dans la fenêtre inférieure.

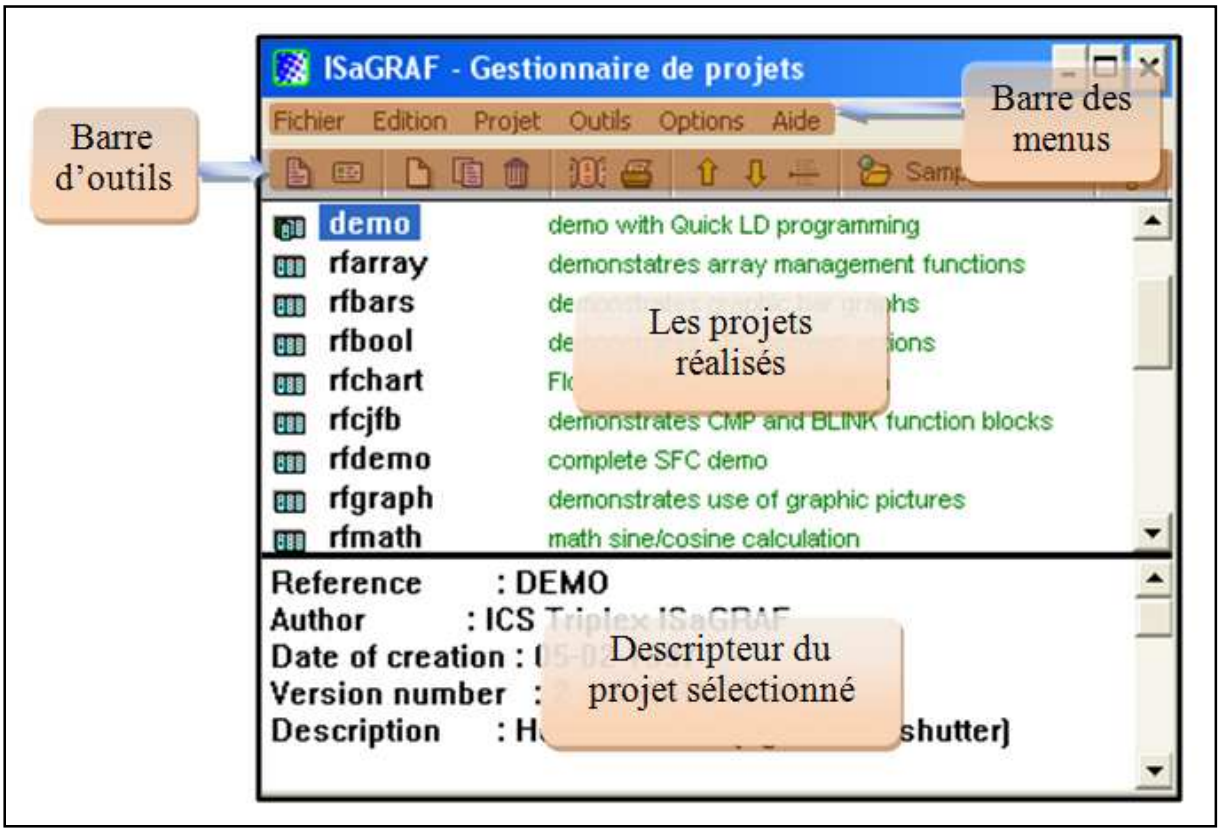

Figure II.9 *Fenêtre gestionnaire de projets*

Barre d'outils

Bouton	Signification
	Redimensionner les fenêtres
	Insérer des séparateurs
	Déplacer un projet dans la liste
	Créer un nouveau projet
	Editer le descripteur d'un projet
	Editer un projet
	L'historique des modifications
	Imprimer le dossier
	Protection par mot de passe

Tableau II.18 *Barre d'outils du gestionnaire de projet d'ISAGRAF [13]*

Plusieurs manipulations sur les projets sont possibles tableau, pour ordonner, créer, éditer les projets réalisés avec ISAGRAF, ces manipulations facilitent le travail de l'utilisateur. Néanmoins presque chaque manipulation d'ouverture ou d'édition fait appel a une autre fenêtre ce qui peut être gênant.

Création et manipulation de projets

Les commandes du menu du Gestionnaire de Projets permettent de créer de nouveaux projets, d'éditer (ouvrir) ou de manipuler les projets existants.

Créer un nouveau projet

Pour créer un nouveau projet, il faut tout d'abord entrer son nom. Un projet vide est alors créé. Une configuration d'E/S peut également être sélectionnée lors de la création du projet. Cette configuration doit être définie en librairie. Si une configuration est choisie, ISAGRAF crée automatiquement les variables correspondantes et leur câblage dans le nouveau projet. Quand on crée ou on renomme un projet, on doit respecter les règles suivantes:

• le nom ne peut pas excéder 8 caractères

• le premier caractère doit être une lettre

• les caractères suivants doivent être des lettres, chiffres ou le caractère de soulignement

• les majuscules et minuscules ne sont pas différenciées

Quand un projet est créé, utilisez la commande "Edition / Commentaire du projet" pour entrer le texte qui sera affiché avec le nom du projet dans la liste des projets.

Editer le descripteur d'un projet

Utilisez la commande "Projet / Editer le descripteur" pour entrer le descripteur du projet sélectionné. Ce document identifie de façon détaillée le projet.

Editer un projet

La commande "Fichier / Ouvrir" ouvre le Gestionnaire de Programmes pour le projet sélectionné. De cette fenêtre seront appelés les différents outils d'édition, de compilation et test. Vous pouvez également cliquer deux fois sur le nom du projet dans la liste pour l'éditer.

L'historique des modifications

L'atelier ISAGRAF enregistre toutes les modifications concernant un composant d'un projet dans un fichier historique. Chaque modification est identifiée dans l'historique par un titre, une date et une heure. Le fichier historique regroupe les 500 dernières modifications. Il y a un fichier historique pour chaque projet. La commande "Projet /Historique des modifications" permet à l'utilisateur de visualiser ou d'imprimer le contenu du fichier historique pour le projet sélectionné.

Imprimer le dossier

Cette fonction permet de construire et d'imprimer une documentation complète pour le projet sélectionné. Ce document peut grouper tous les composants (programme, variable, paramètres...) du projet. Pour construire un document spécifique (incomplet), il suffit de définir sa table des matières.

Protection par mot de passe

La fonction "Mot de passe" du menu "Fichiers" permet de protéger l'accès aux fonctions et aux données du projet à l'aide de mots de passe.

II.3.3.2 GESTION DES PROGRAMMES D'ISAGRAF

La fenêtre du Gestionnaire de Programmes montre les programmes du projet, et regroupe dans ses menus les commandes disponibles, pour mettre en place l'architecture du projet, lancer les éditeurs, les compilateurs et les outils de mise au point. Cette fenêtre est le noyau interactif de l'atelier ISAGRAF pendant le développement d'un projet. La fenêtre du Gestionnaire de Programmes est ouverte par la commande "Ouvrir" de la fenêtre du Gestionnaire de Projets.

Figure II.10 *Gestion des programmes d'ISAGRAF*

II.3.3.3 LES LANGAGES DE PROGRAMMATION

Les différents programmes qu'on peut utiliser dans ISAGRAF sont les langages de la norme CEI1131-3 (présentés précédemment), avec la possibilité d'édition de blocs de fonctions pour enrichir sa librairie. Nous avons en plus les diagrammes de flux (Flow Chart)

Symbole	Signification
	Programmé avec le langage SFC
	Programmé avec le langage FC
	Programmé avec le langage FBD
	Programmé avec le langage LD
	Programmé avec le langage ST
	Programmé avec le langage IL

Tableau II.19 *Symboles des langages de programmation [13]*

Remarque : Nous allons juste utilisés le langage SFC pour la réalisation de nos programmes.

Barre d'outils

Bouton	Signification
	Créer un nouveau programme
	Editer le contenu d'un programme
	Copier des programmes
	Supprimer des programmes
	Le dictionnaire de variables
	Générer le code de l'application
	Vérifier le programme sélectionné
	options d'exécution
	Simulation
	Câbler les entrées / sorties

Tableau II.20 *Barre d'outils du Gestionnaire des programmes d'ISAGRAF [13]*

Le menu "Fichier" groupe les commandes de création et de mise à jour des programmes, ainsi que les commandes de lancement des éditeurs de programmes.

Manipuler les programmes

Le menu "Fichier" groupe les commandes de création et de mise à jour des programmes, ainsi que les commandes de lancement des éditeurs de programmes.

-Créer un nouveau programme

-Editer le contenu d'un programme

-Copier des programmes

-Supprimer des programmes

Le dictionnaire de variables

La commande "Fichier / Dictionnaire" lance l'édition du dictionnaire, où sont déclarées les variables du projet. Les variables peuvent être globales (connues par tous les programmes) ou locales à un programme. Dans notre cas on va utiliser que les variables booléens et temporelles qu'utilise le langage SFC.

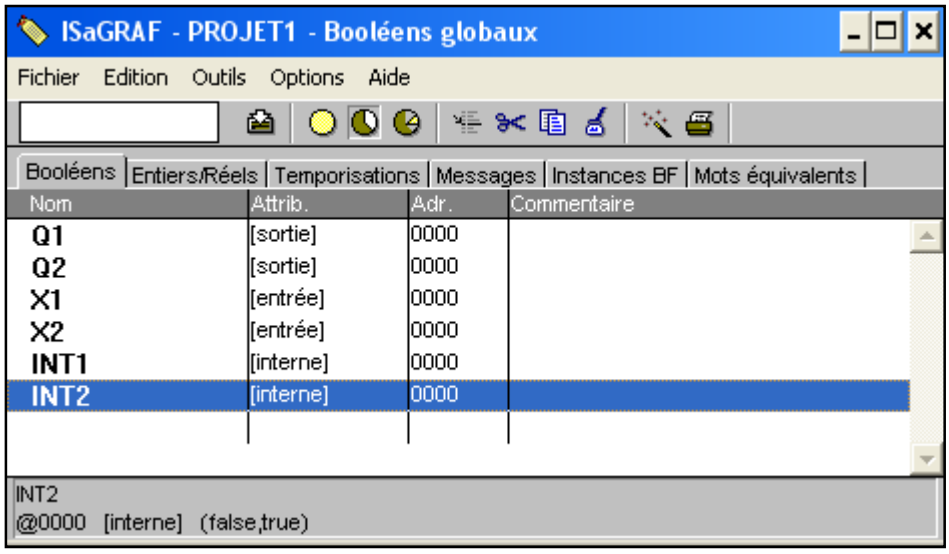

Figure II.11 *Édition du dictionnaire*

Les outils de génération de code

Les commandes du menu "Codage" lancent les outils de compilation et de génération de code, et permettent d'entrer les options et autres paramètres utilisés pendant la génération du code de l'application.

Générer le code de l'application

La commande "Générer l'application" lance la génération du code. Les options pour les cibles choisies doivent être correctement renseignées avant la génération du code, tous les programmes sont vérifiés et les éventuelles erreurs de syntaxes sont détectées.

Vérifier le programme sélectionné

La commande "Vérifier" lance la vérification de la syntaxe du programme sélectionné dans la fenêtre du Gestionnaire de Programmes. Quand un programme est vérifié, et qu'aucune erreur n'a été détectée, il n'est plus revérifié automatiquement par la génération de code, tant que son contenu, ou les objets dont il dépend (variables, définitions...) ne sont pas modifiés.

Options de compilation

Cette commande permet de sélectionner les options pour la génération et l'optimisation du code de l'application. Pour choisir une cible de compilation et/ou générer un code (programme) qui décrit totalement l'application.

Câbler les entrées / sorties

La commande "Câbler les E/S" lance l'éditeur de câblage des variables d'E/S. Cet outil est utilisé pour décrire la correspondance entre les variables et les cartes virtuelles de simulation.

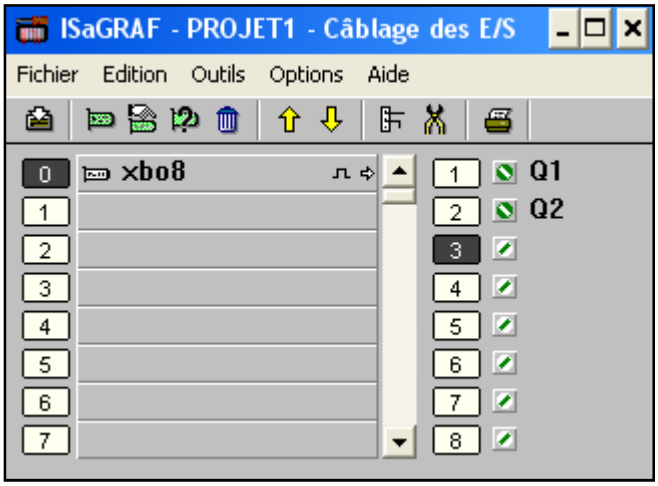

Figure II.12 *Fenêtre de câblage des E/S*

Simulation

La commande "Simuler" lance le debugger pour la simulation. Dans ce mode, une autre fenêtre est ouverte, qui simule le comportement des E/S. Cette commande permet de commencer les tests de l'application, pour vérifier le comportement du programme.

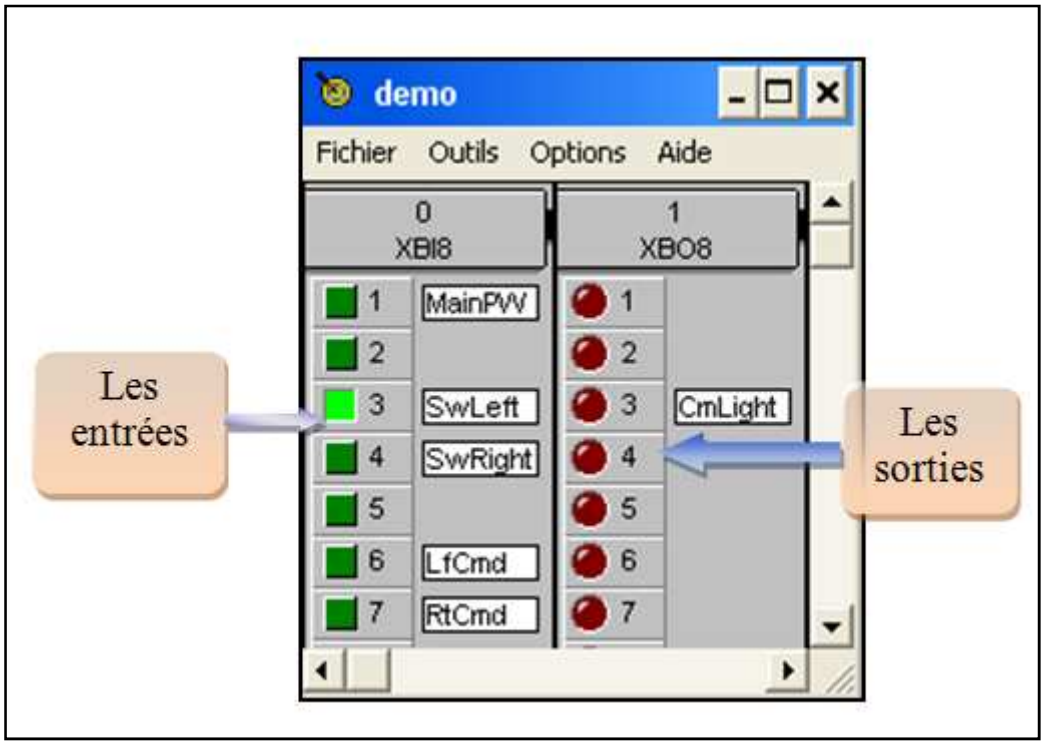

Figure II.13 *fenêtre de simulation*

Saisie du graphe SFC

Pour entrer un graphe SFC, l'utilisateur doit poser les composants significatifs du schéma. Toutes les lignes de connexion horizontales et verticales sont tracées automatiquement par l'éditeur. Pour insérer un symbole SFC, il faut déplacer la sélection sur l'emplacement désiré et sélectionner le type de symbole dans la barre d'outils.

Pour associer une variable à un état ou une transition, il suffit de cliquer deux fois sur l'état ou la transition pour que la fenêtre se divise en deux et mettre la variable désiré dans la partie droite de la fenêtre.

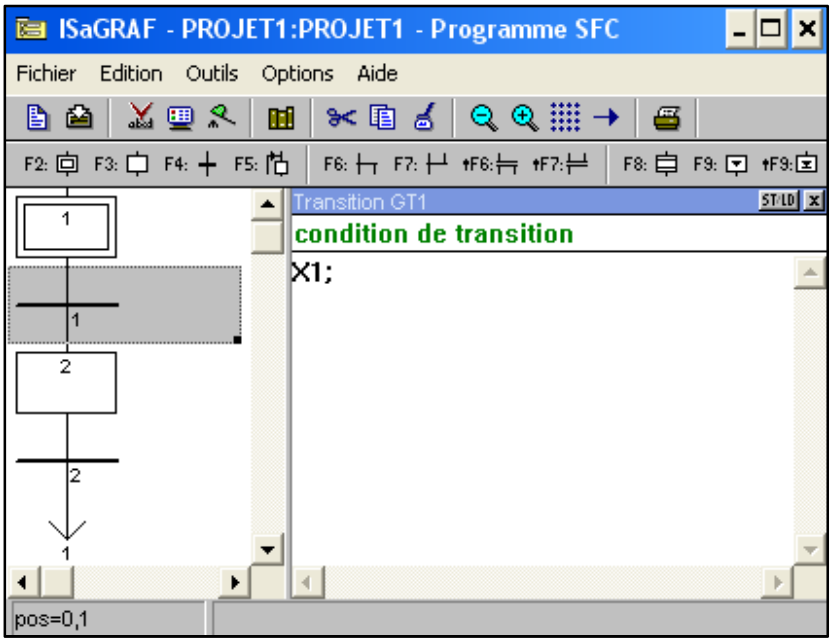

Figure II.14 *Fenêtre de saisie du graphe SFC*

Le nouveau symbole est inséré à la position courante. Les touches de fonctions suivantes peuvent aussi être utilisées:

Bouton	Signification
F2:	Insérer une étape initiale
F3:	Insérer une étape
F4:	Insérer une transition
F5:	Insérer un saut vers une étape
F6: F7:	Insérer une divergence ou convergence OU / Ajouter des branches
F6: F7:	Insérer une divergence ou convergence ET / Ajouter des branches
F8:	Insérer une macro-étape
F9: F9:	Insérer une étape de début ou de fin dans le corps d'une macro étape

Tableau II.21 *Barre d'outils de saisie du graphe SFC [13]*

L'éditeur graphique SFC montre toujours la position courante dans la matrice. La cellule sélectionnée est marquée en gris. Le petit rectangle au coin bas / droite de la sélection permet redimensionner librement les cellules de la matrice. Il permet également de changer la proportion X/Y des cellules:

Extension du langage ST dans ISAGRAF

Les énoncés et fonctions suivants sont des extensions du langage ST.

TSTART - TSTOP: contrôle des temporisations

Les énoncés suivants peuvent être utilisés dans les blocs ACTION(): ... END_ACTION; des étapes SFC.

Fonction	Signification
GSTART	lance un programme SFC
GKILL	tue un programme SFC
GFREEZE	fige un programme SFC
GRST	relance un programme SFC
GSTATUS	donne l'état d'un programme SFC

Tableau II.22 *Extension du langage ST dans ISAGRAF [13]*

Les champs suivants d'une étape SFC peuvent être testés dans des expressions:

GSnnn.x est une valeur booléenne qui représente l'activité de l'étape

GSnnn.t est une valeur temporelle qui représente le temps écoulé depuis que l'étape est active: durée d'activation.

("nnn" est le numéro de référence de l'étape SFC)

II.3.4 CONCLUSION

Nous avons tirés profit du logiciel ISAGRAF qui est un logiciel supportant les langages de la norme CEI61131-3, l'édition de nos programmes sera réalisée avec ces langages. ISAGRAF est également un outil de simulation et de contrôle puissant, nous pourrons ainsi exploiter cet aspect et exécuter les programmes simulés sous ISAGRAF, avec l'application que nous avons développés à cet effet.

Par contre le mode multifenêtre cause des blocages à l'utilisation, la sauvegarde des fichiers utilise une dizaine de fichiers pour un seul projet ce qui est lourd à gérer. Le code C généré n'est pas exploitable pour nos besoins. Le code précompilé change avec le langage utilisé, ce qui ne facilite pas son exploitation.

CHAPITRE III

DEVELOPPEMENT DE FLEXPLC

CHAPITRE III

DEVELOPPEMENT DE FLEXPLC

III.1 INTRODUCTION

Après avoir vu comment exploiter ISAGRAF, nous avons pris un cas particulier qui offre des possibilités de contrôle avancées, on exploitera des fichiers générés par ISAGRAF après édition d'un programme SFC. Nous allons entamer la partie qui décrit l'écoulement de nos efforts, nous parlerons de la compréhension du code généré, et comment nous avons développés l'application FlexPLC qui l'interprète et qui l'exécute, cette application décrite en deux aspects, l'interface utilisateur, et le cœur fonctionnel. Le code interprété et exécuté communique avec le monde extérieur, avec une carte qui nous a été fournie par Moveit Automation [19], une entreprise basée à Lausanne en Suisse. Le nom de FlexPLC est inspiré des produits et solutions basées PC de Moveit Automation dans divers domaine de contrôle commande.

III.2 COMPREHENSION DU CODE GENERE

Le code généré par ISAGRAF est sous formes de deux fichiers :

- Un fichier appelé « symboles » contenant les déclarations de toutes les variables du programme écrit en ST.
- Un fichier contenant le programme en ST qui décrit complètement le SFC édité sous ISAGRAF.

III.2.1 LE FICHIER DE DECLARATIONS

En tout premier lieu, l'utilisateur doit créer sous ISAGRAF un programme en SFC, puis en générer le code précompilé ainsi que le fichier symboles, ou élaborer un code avec les conditions requises pour le fonctionnement (la même structure générale pour les deux fichiers).

Consignes d'utilisation :

-Ouvrir ou éditer le fichier de déclarations (symboles). La structure d'un fichier de déclaration doit être la suivante :

```
@PROGRAMS,1
#<commentaire ou autre >,<nom du programme>

@BOOLEANS,<nombre de variables>
#<commentaire ou autre>,<nom variable>,<type de variable>,!0000,FALSE,TRUE
…
@ANALOGS,2
#!2001,<nom du programme>.Q,+X,!0000,I,
#!2002,<nom du programme>.S,+X,!0000,I,

@TIMERS,<nombre de timers>
#!3001,<temporel>,+X,!5001
…
```

Figure III.1 *Structure du fichier symboles*

<nom du programme> est le nom du programme en ST traduisant le SFC désiré.

<type de var> peut être, +O pour sortie,+X pour interne, +I pour entrée.

<nombre de variables> est le nombre de variables booléennes en entrée, en sortie, ou internes.

Cette structure est obligatoire, une erreur commise par l'utilisateur compromettra le bon fonctionnement du programme.

-Ouvrir ou éditer le programme correspondant au fichier de déclaration, le nom du fichier doit être obligatoirement celui désigné dans le fichier de déclarations. Ce programme doit être en ST et traduisant un comportement d'un SFC, la structure du code généré par ISAGRAF doit être prise en exemple.

III.2.2 LE FICHIER DE PROGRAMME

Le fichier programme qu'ISAGRAF génère, est un code écrit en ST et qui traduit toutes les étapes, actions, conditions, etc. pour un déroulement cyclique d'un programme en SFC, la programmation du cœur fonctionnel pour exécuter le programme ST passe par sa compréhension, afin de tirer profit de sa structure générale.

Le code généré avec ISAGRAF possède une structure bien définie, qui prends en compte des paramètres d'orientation du programme, qui sont les variables entières <nom du programme>.S, <nom du programme>.Q, et le booléen SFC_EVOL. Selon ces variables le passage d'un cycle, met en marche le programme, le gèle, fait une reprise après un gel, ou arrête le programme en cours d'exécution, ou ne fait tout simplement rien. Ces démarches sont bien définies avec des structures conditionnelles et des sauts.

Les configurations possibles pour le contrôle du comportement du programme sont définies dans le tableau III.1, ces configurations représentent le passage entre quatre états, KILL (arrêté), START (en marche), FREEZE (gelé), RESTART (reprise), au moins l'exécution d'un cycle programme est nécessaire afin d'accomplir ces passages.

Configuration	Passage
<nom du programme>.S=0, <nom du programme>.Q=1	Arrêt à marche
<nom du programme>.S=1, <nom du programme>.Q=2	Marche à gel
<nom du programme>.S=2, <nom du programme>.Q=3	Gel à reprise
<nom du programme>.S=1, <nom du programme>.Q=0	Marche à arrêt
<nom du programme>.S=2 ou 0, <nom du programme>.Q=0	Gel à arrêt

Tableau III.1 *Valeurs des variables de contrôle*

En conséquence à cette observation, nous avons mis les fonctions des boutons de l'interface de FlexPLC en prenant en compte ces paramètres. Pour démarrer, geler, reprendre, ou arrêter le programme, nous forçons les paramètres <nom du programme>.S, et <nom du programme>.Q, selon le besoin.

La variable booléenne SFC_EVOL, contrôle la validation de la transition 0 « GT0 », cette dernière existe par défaut et précède l'étape initiale dans le programme généré du programme SFC, elle est donc invisible dans ISAGRAF [13]. SFC_EVOL est forcé a « true »

si le programme est a sa première exécution, donc dans le passage de l'état « arrêt » à l'état « marche ».

Nous illustrons donc le programme généré par ISAGRAF, par l'organigramme des figures III.2 et III.3, et nous donnons un exemple de programme simple en annexe C.1.

Le contenu des éléments des ces organigrammes, est écrit dans la syntaxe du C, GT(i) sont les transitions du SFC, GX(i) sont les étapes (steps), TSTART(GS(i)), et TSTOP(GS(i)) sont respectivement la mise en marche et l'arrêt du compteur de temps d'activation.

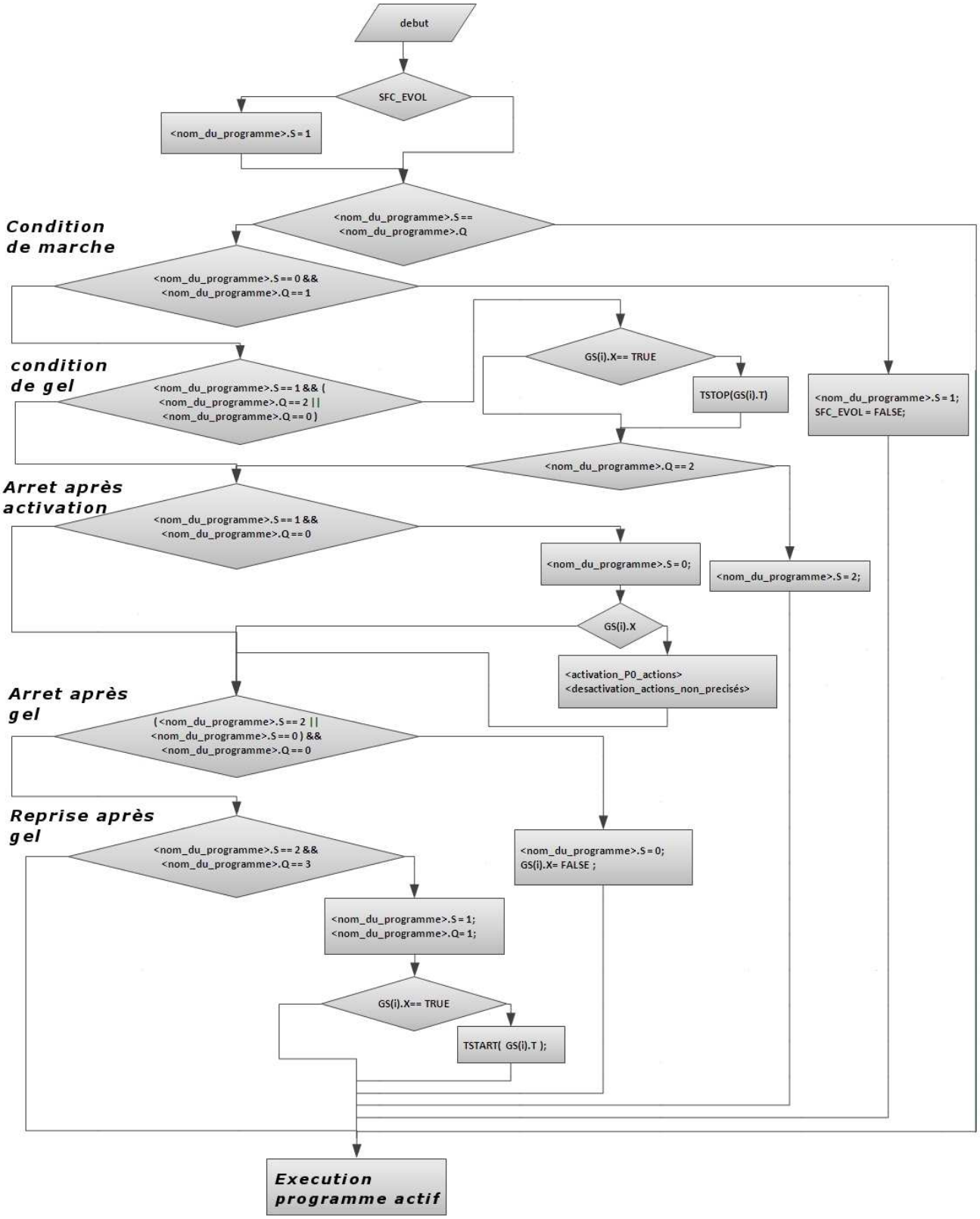

Figure III.2 *partie configuration du programme*

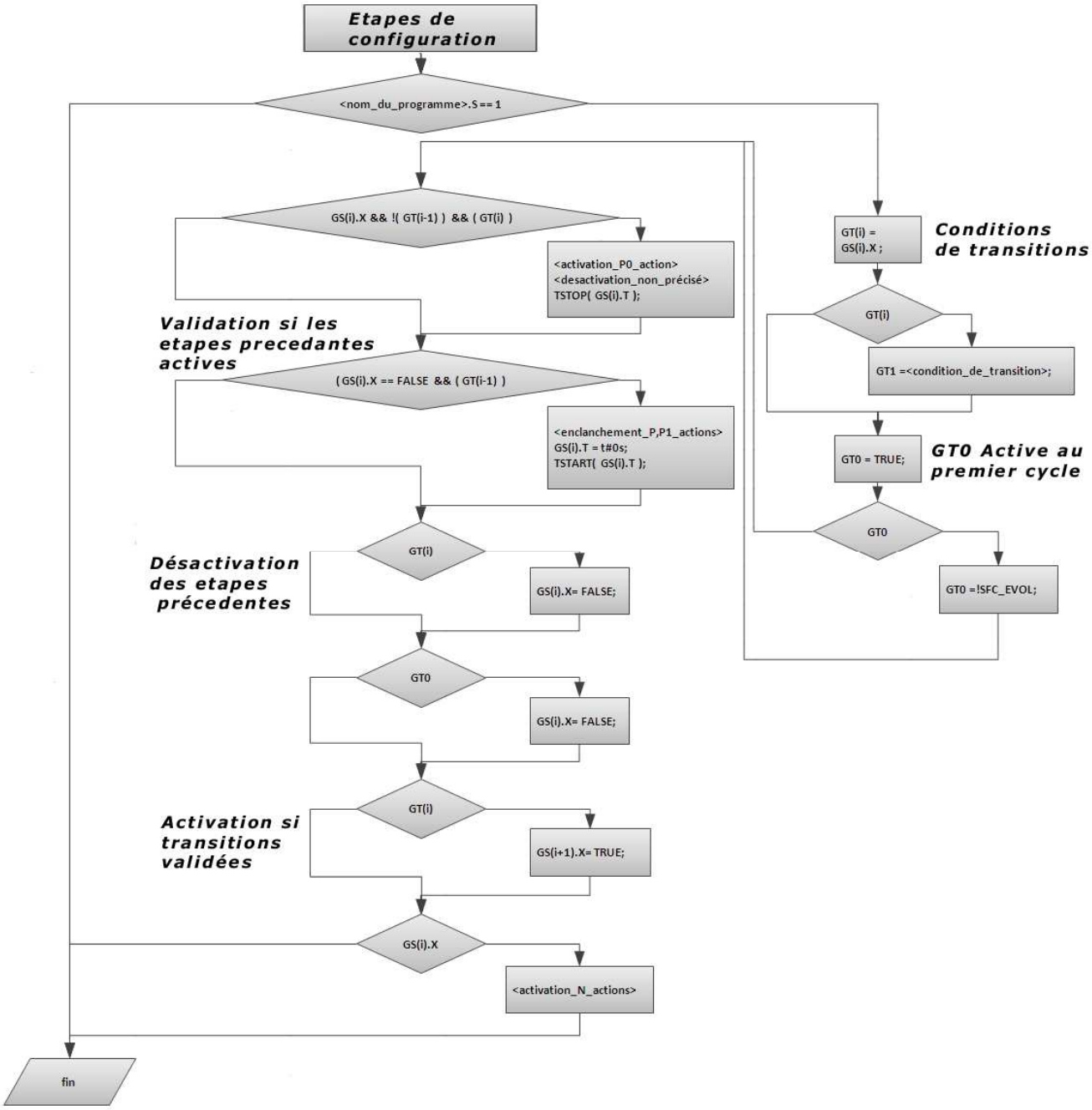

Figure III.3 *partie exécution du programme actif*

III.3 PRESENTATION DE L'APPLICATION FLEXPLC

III.3.1 INTRODUCTION

L'application FlexPLC que nous avons développés, doit avoir les spécifications suivante :

- Interprétation du code écrit en ST généré par ISAGRAF, à partir d'un SFC, et l'executer.
- Transposition du comportement des E/S sur une carte physique via le cœur fonctionnel de l'application.
- Flexibilité par rapport à la logique d'automatisation, et aux indexes des entrées/sorties et autant que possible par rapport aux cartes d'E/S.

Pour le pilotage des entrées sorties, nous avons utilisé une carte ET-PCI8255, et comme outil de développement Visual C++ 6 [4]. Dans ce chapitre nous allons montrer la conception de l'interface, et décrire comment travaille le cœur fonctionnel.

III.3.2 PRESENTATION DE L'INTERFACE

L'interface utilisateur a été conçue de telle façon à offrir le plus de clarté, et de fonctionnalités a l'utilisateur, nous allons voir quelles en sont les composants, et comment l'utiliser.

III.3.2.1 FENETRE PRINCIPALE

Pour avoir accès à la fenêtre il faut juste ouvrir le fichier FlexPLC.exe et la fenêtre de la figure III.4 apparait.

Zone de chargement des programmes

Dans cette zone l'utilisateur doit charger, et éditer au besoin, les deux fichiers générés par ISAGRAF, l'un des fichiers contient les déclarations des variables, l'autre contient le cœur du programme, il décrit la progression du programme SFC que l'utilisateur a conçu sous ISAGRAF, une illustration de la zone de chargement des programmes est donnée dans la figure III.5.

Notons que Pour ouvrir un fichier, il faut qu'il soit du type texte, et son emplacement dans le même répertoire que FlexPLC.

Lorsqu'on appuie sur le bouton enregistrer, le contenu sera enregistré avec le nom choisi de type texte et il sera mis, automatiquement, dans le même répertoire que l'exécutable s'il n'existe pas, sinon il devrait y exister.

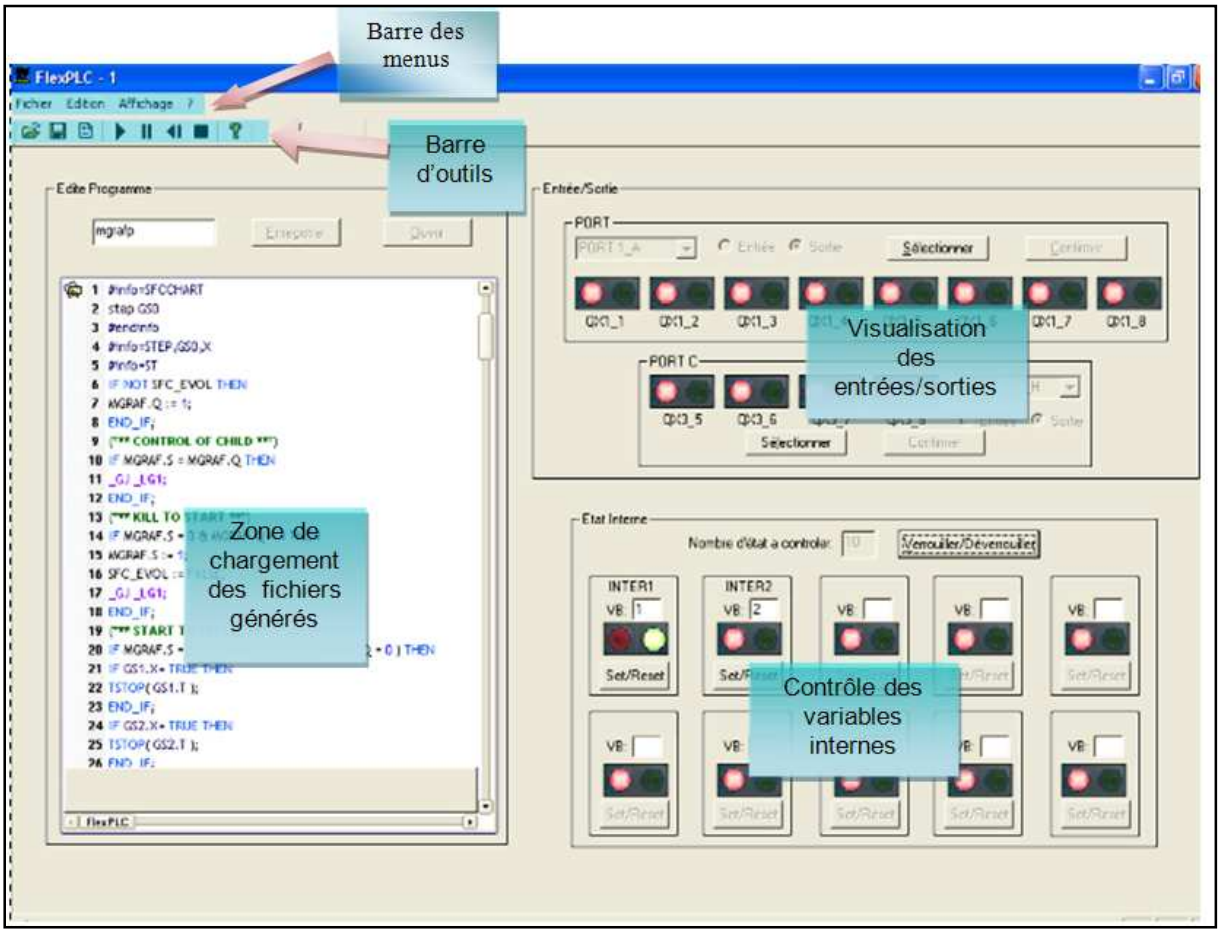

Figure III.4 *Vue d'ensemble de l'interface FlexPLC*

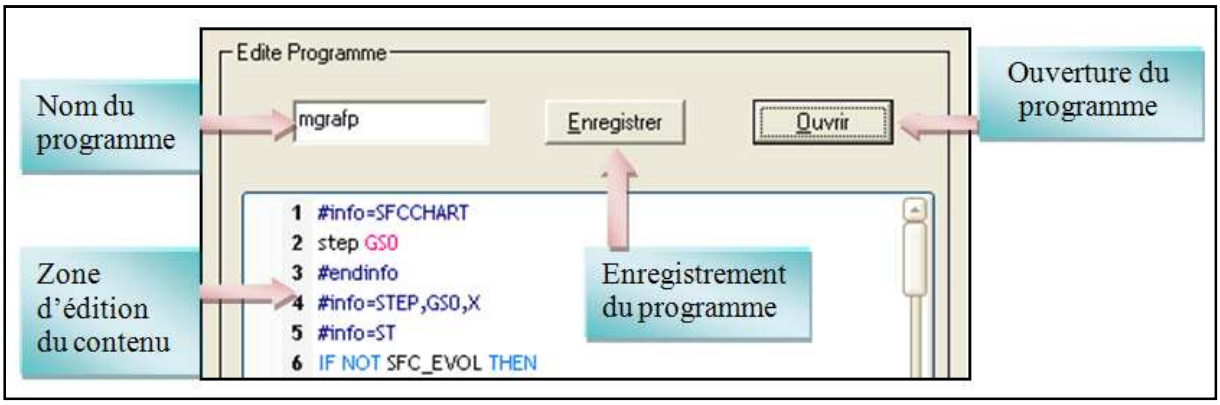

Figure III.5 *Zone de chargement des fichiers*

L'ordre de chargement des deux fichiers nécessaires n'est pas important, peu importe quel fichier l'utilisateur charge en premier, l'interface identifie la nature du fichier, s'il s'agit du fichier de déclarations, ou du programme, elle affiche âpres chargement des deux fichier, le programme en ST. L'interface détecte aussi si un fichier est vide, ou si un fichier est déjà chargé.

Le contenu des fichiers est stocké dans des variables de type string, « contenu_s » et « contenu_p », le chargement des fichiers se fait selon l'organigramme de la figure III.6.

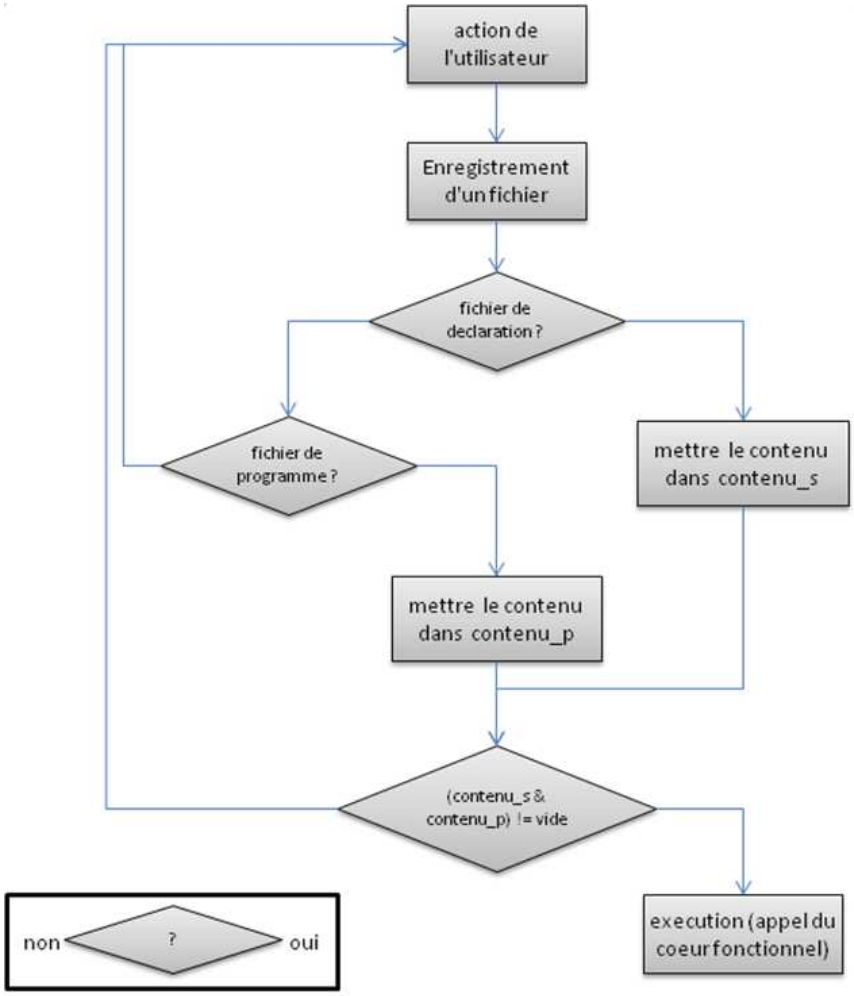

Figure III.6 *Processus de chargement de fichier*

État d'un bit ou d'une variable interne

Des que l'utilisateur enregistre les deux fichiers nécessaires, les variables internes et les E/S sont reconnues, et leurs états deviennent visibles comme illustré dans la figure III.7.

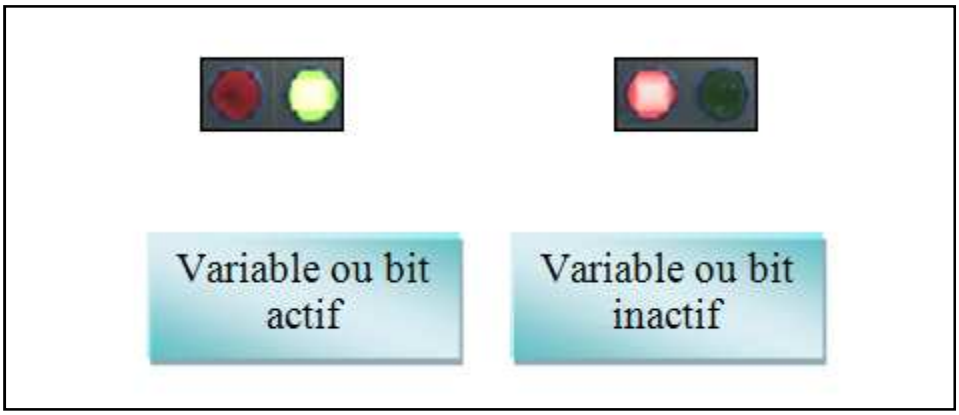

Figure III.7 *Visualisation de l'état d'une variable*

La barre d'outils

La barre d'outils de FlexPLC permet à l'utilisateur de charger, et d'enregistrer les fichiers, de connaître les noms des variables, et de contrôler son programme.

Figure III.8 *Barre d'outils*

Bouton	Signification
	Appel a la fenêtre des propriétés des E/S
	Lancement du processus
	geler le processus
	Réinitialisé processus
	Arrêter processus
	Ouvrir un programme
	Enregistrement du programme
	appel à la fenêtre help

Tableau III.2 *Barre d'outils de FlexPLC*

Remarque

Les boutons pour ouvrir et enregistrer un programme, sont actifs qu'à l'ouverture de la fenêtre principale, et après avoir appuyé sur le bouton arrêter, ils sont désactivés en cours d'exécution. Par contre les boutons appel à la fenêtre des propriétés des E/S, lancer, geler, initialiser, et arrêter processus, sont actifs qu'après avoir chargé les deux fichiers, et redeviennent inactifs à l'appui du bouton arrêter processus.

III.3.2.2 LA FENETRE HELP

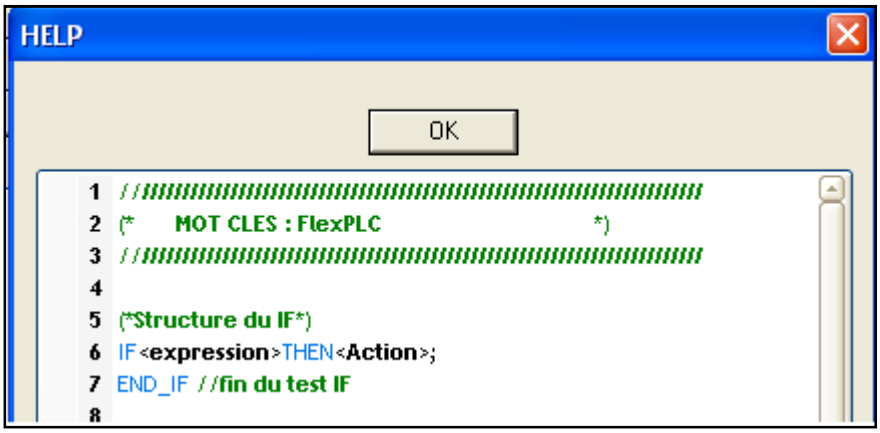

Figure III.9 *Fenêtre help*

La fenêtre contient des informations sur la bonne utilisation de l'interface ainsi que les notions de base du langage ST.

III.3.2.3 LA FENETRE DE PROPRIETES DES E/ S

La fenêtre propriétés des ports contient les informations sur a configuration des ports de la carte d'E/S (en entrée ou en sortie), et le nom de chaque bit que l'utilisateur a crée dans le dictionnaire des variable d'ISAGRAF, l'indication « (vide) » est affichée si le bit correspondant est inutilisé.

La configuration des ports est automatique pour une exploitation optimale de la carte, après avoir reçue les informations du contenu du fichier symboles le programme, commence par la configuration des entrées a partir de la zone contrôleur 8255-1 ensuite il configure les sorties, après vérification que la configuration est possible.

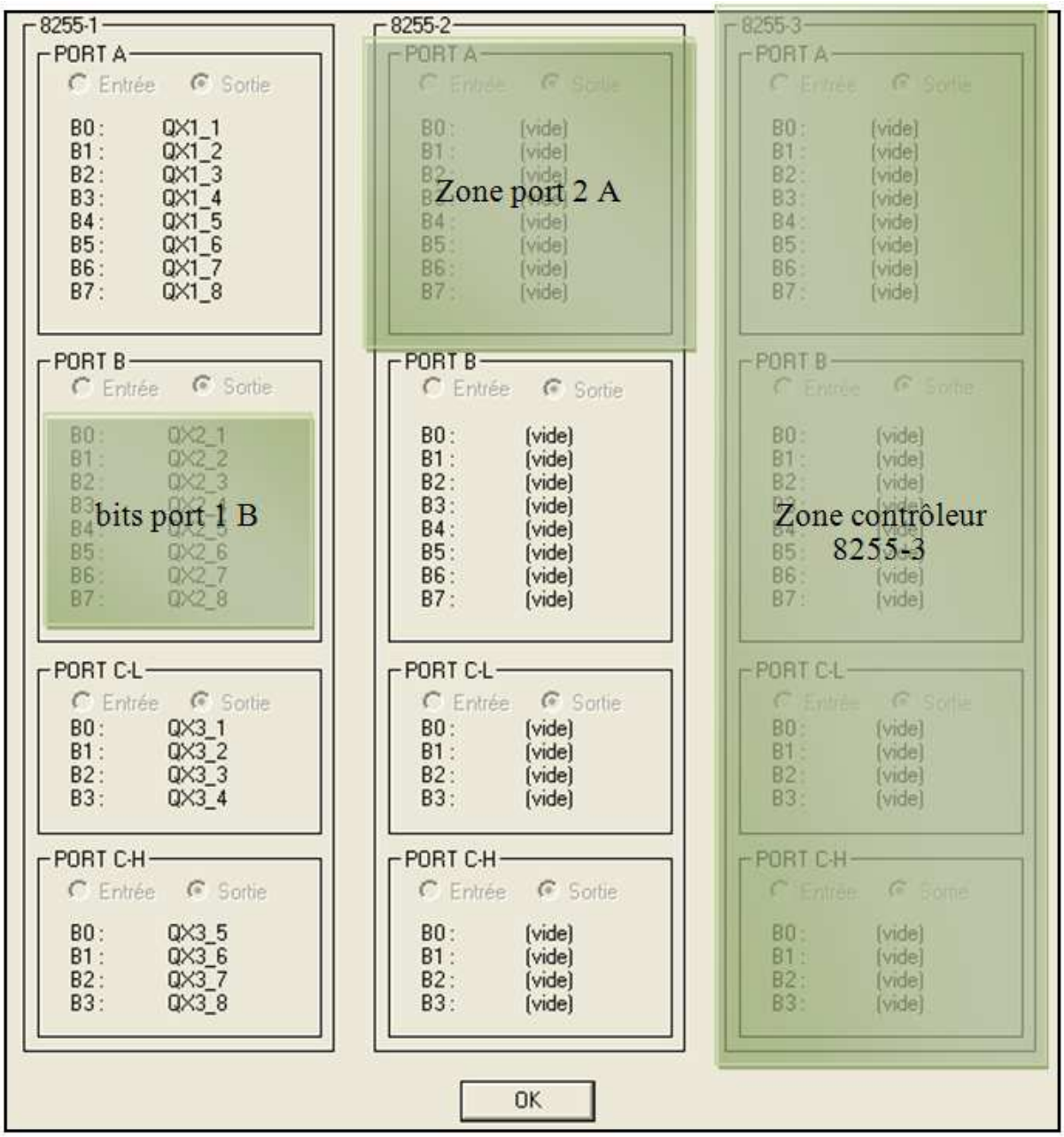

Figure III.10 *Fenêtre de propriétés des E/ S*

Zone de visualisation des entrées/sorties

Cette zone permet à l'utilisateur de visualiser les E/S de la carte ET-PCI8255 V3 en temps réel, s'est donc une transposition des états réels des ports d'E/S.

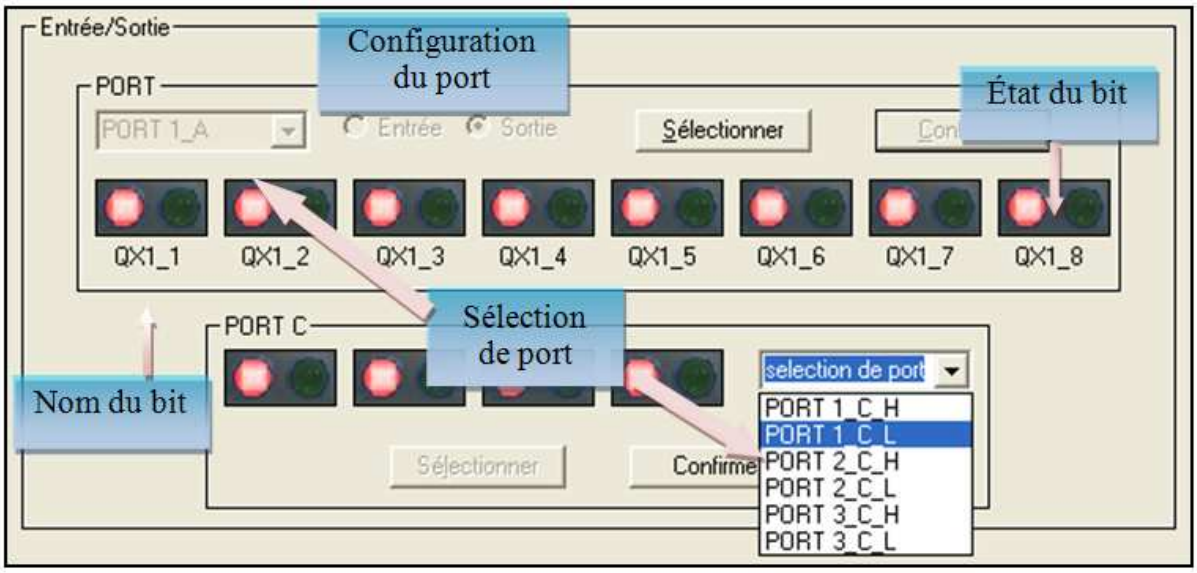

Figure III.11 *Zone E/S*

L'utilisateur peut choisir le port à visualiser, d'abord en cliquant sur le bouton Sélectionner, pour avoir accès a la sélection de port, après avoir sélectionné un port l'utilisateur doit appuyer sur le bouton confirmer, pour que les noms et les états de chaque bit du port soient visibles, ainsi que la configuration du port (en entrée ou en sortie). Cette zone est active qu'après avoir chargé les fichiers dans la zone de chargement et d'édition, et redevient inactive en cliquant sur le bouton arrêter.

Contrôle des variables internes

Cette zone est destinée à la visualisation, et/ou au forçage des variables internes, que l'utilisateur aura crée dans le programme SFC.

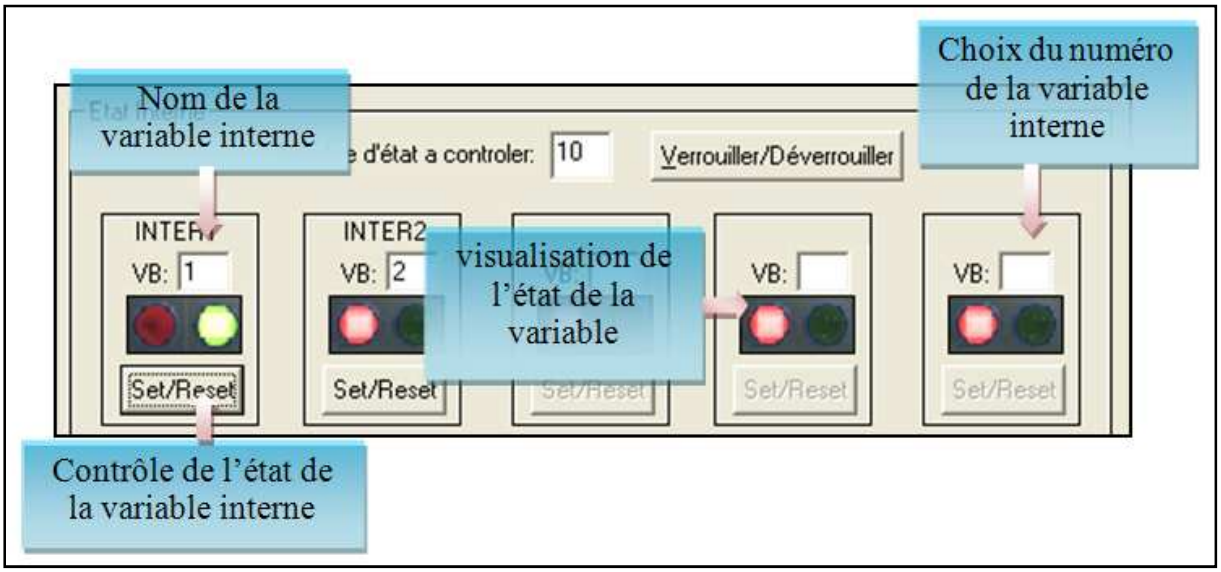

Figure III.12 *Contrôle des variables internes.*

- Le nombre d'états maximal de variables internes à visualiser simultanément, est de dix.
- Pour contrôler une variable, l'utilisateur doit mettre le numéro de la variable pour l'afficher.
- Cette zone est active qu'après avoir charger les deux programmes ST et redevient inactive en cliquant sur le bouton arrêter.

III.3.3 LE FONCTIONNEMENT GENERAL DE L'INTERFACE DE FLEXPLC

Afin de clarifier, étape par étape, à l'utilisateur comment l'interface de FlexPLC fonctionne, depuis son ouverture jusqu'à l'exécution du programme, nous illustrons le comportement général de l'interface par la figure III.13, qui met en évidence les étapes « post-ISAGRAF », nous résumons ainsi les parties précédentes de ce chapitre.

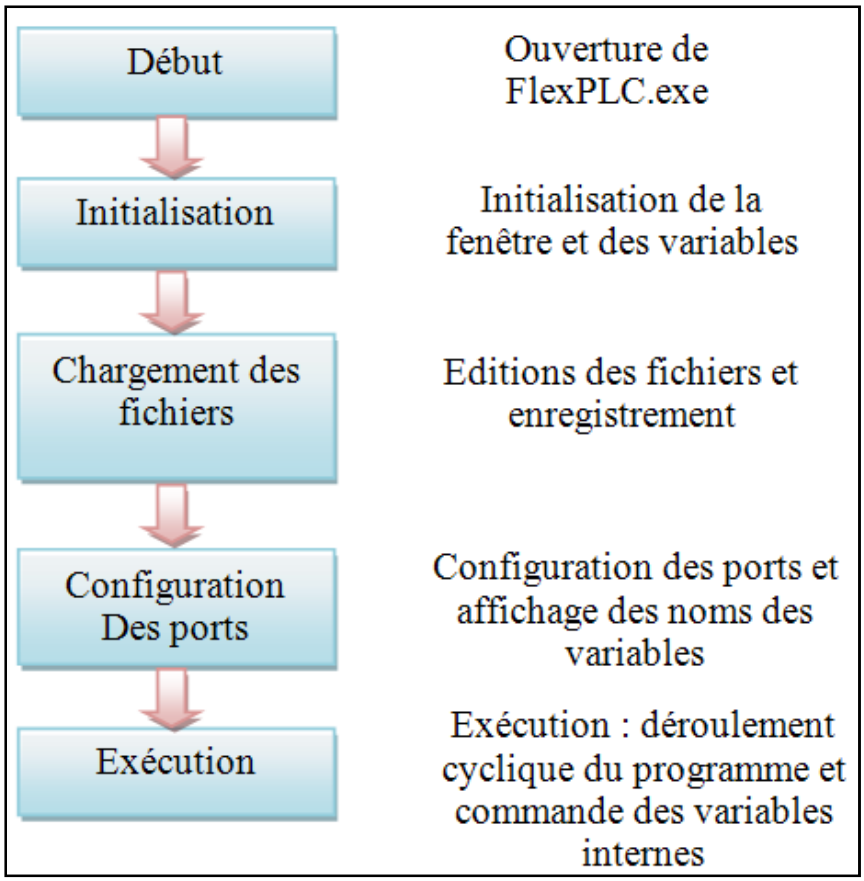

Figure III.13 *Fonctionnement de l'interface de FlexPLC*

III.4 LE CŒUR FONCTIONNEL

III.4.1 INTRODUCTION

l'aspect interprétation d'un programme en ST généré par ISAGRAF avec l'utilisation du SFC, est assurée par le cœur fonctionnel de notre travail, ce dernier se présente sous la forme d'un groupe de classes C++ que nous avons appelés globals, driver, temporel, alias, et calculette (notion de classes et programmation orienté objet [1], [2]).

Conformément aux principes du langage de programmation utilisé ces classes contiennent des variables membre, et des fonctions membres, à travers lesquelles les objets de ces classes communiquent entre elles, et avec les fichiers contenant le texte généré par ISAGRAF.

Dans un premier temps nous allons décrire chaque classe, et ses variables et fonctions membres les plus importantes, en suite nous verrons comment un programme, généré à partir d'un programme SFC sous ISAGRAF.

III.4.2 DESCRIPTIF DES CLASSES

III.4.2.1 LA CLASSE GLOBALS

La classe globals assure l'extraction, le codage des noms des E/S, et l'évaluation au besoin de leurs valeurs, aussi l'exécution du programme généré par ISAGRAF à partir de fichiers texte, et le traitement du programme en fonction des images des variables internes et physiques.

Les variables membres

Comme on le sait la carte d'E/S contient trois contrôleurs 8255 de Intel qui portent trois ports chacun, A, B et C, de 8 bits, le port C peut être divisé en deux parties de 4 bits,

tous ces ports sont configurables en entrée ou en sortie, nous avons donc à mémoriser cette configuration. Des variables booléennes ont été créées à cet effet et nommées,

select_port_<numéro du chip 8255>_<port>;

Ou <numéro du chip 8255> prend 1, 2 ou 3; et <port> sera remplacé par A, B, C_L, C_H (C_L, C_H représente les deux moitiés des ports C de chaque contrôleur 8255) (exemple : le port A du contrôleur 8255-1 correspond a select_port_1_A). On dénombre ainsi douze variables qui indiquent à l'interface utilisateur l'état de la configuration des ports, qui sera transmise à la carte et à l'utilisateur après chargement des fichiers. Quand un port est configuré en entrée la variable correspondante à une valeur "false", et "true" quand le port correspondant est en sortie.

L'état des ports étant connu, il nous faut créer un moyen d'en voir la valeur de la donnée que porte chaque port, les variables membres qui ont été créés à cet effet sont nommés,

port_<numéro du contrôleur>_<port>;

Qui sont des pointeurs de type booléens et qui ont une longueur de 8 bits pour les ports A et B de chaque contrôleur, et de 4 bits pour les moitiés des ports C. ces pointeurs ou tableaux sont les images des données présentes sur les ports physiques de la carte ils sont modifiés par le cœur fonctionnel s'ils sont en sortie, et par la donnée réels de la carte si ils sont en entrée. Une extension de ce nombre est possible nous le verrons en détails par la suite.

Aussi, comme le besoin était présent, on a créé aussi les images des variables internes que l'utilisateur peut définir, et qu'il pourrait pour des besoins de contrôle en changer la valeur, par l'intermédiaire de l'interface, ces variables sont stockées dans le tableau de booléens « intern », un maximum de 500 variables peuvent être ainsi définies. Les variables internes générées par ISAGRAF et qui ont un rôle essentiel dans le déroulement cyclique du programme, ne sont pas laissées en reste, elles ont aussi leurs images, on en distingue le tableau GSX de type booléen qui contient l'information sur l'activité des étapes d'un SFC, le tableau GT aussi booléen qui contient l'information sur la validité ou non des transition d'un SFC, le tableau GST de type temporel qui a pour rôle de compter les temps d'activation des étapes actives, la variables SFC_EVOL de type booléen, et les variables entières <nomdu programme>.Q, <nom du programme>.S, (<nom du programme> est remplacé par le nom du

programme que l'utilisateur donne à son programme) ces trois dernières variables sont utilisées dans le contrôle du déroulement cyclique d'un SFC généré par ISAGRAF.

Nous tenons à ajouter quelques remarques; quand un utilisateur conçoit un programme par ISAGRAF ou par notre éditeur de texte, les noms des variables ne doivent pas commencer par un des mots réservés qui sont données en annexe A.1.

La classe globals qui est le sujet de cette partie, contient aussi d'autres variables membres, qui sont utilisées par les fonctions de cette même classe, ont été créés pour satisfaire des besoins de programmation du cœur fonctionnel, ces variables ne seront pas citées.

Les fonctions membres

Ces fonctions membre ont pour rôle essentiel d'interpréter le contenu des fichiers générés par ISAGRAF et faire communiquer l'utilisateur avec son environnement matériel, selon la logique que l'utilisateur aura programmé dans le langage graphique, nous allons décrire très brièvement ce que fait chaque fonction.

string file_to_string(string NomDuFichier);

Fonction de type string, ouvre un fichier dont le nom est gardé dans un string NomDuFichier, et qui est transmis par l'interface à la classe globals, si le fichier existe la fonction file_to_string(string NomDuFichier) renvoi un string ayant comme valeur le contenu du fichier qui a été ouvert. Si le fichier n'existe pas ou que le nom n'a pas été transmis correctement, il y aura un message d'erreur généré par l'interface (voir annexe B.1).

string net_prg(string programme_contennu);

l'un des deux fichiers ouverts par file_to_string(string NomDuFichier) contient le programme en ST, généré par ISAGRAF, mais il contient aussi des commentaires, et des sauts de lignes, ainsi que des informations superflus, le rôle de la fonctions net_prg qui renvoi un string en retour, est de nettoyer le programme de ce superflu, le retour de cette fonction est alors un string ne contenant que des instructions ST, ou des expressions à évaluer.

void the_main();

L'autre fichier généré par ISAGRAF est le fichier symboles, contenant toutes les déclarations des variables E/S, et internes qu'elles soient propres à ISAGRAF ou définies par l'utilisateur. la fonction the_main() n'a ni renvoi ni paramètre, elle a pour rôle d'extraire les noms des variables du programme et les coder, leur nombre, et en déduire une configuration optimale des ports E/S, cette dernière est transmise à l'interface (partie III.3.2), si le nombre d'E/S dépasse les capacités du matériel utilisé, ou l'utilisateur choisit en nombre d'E/S non configurable, un message d'erreur est généré dans l'interface utilisateur, et l'utilisateur doit connaitre les limites du matériel. Dans notre cas le nombre maximal d'E/S est de 72, les configurations interdites sont celle réalisent l'équation (III.1)

$$nbr_s > 72 - \{[(nbr_e + 3)/\ 4] * 4\} \qquad \text{(III.1)}$$

nbr_s : nombre de sorties choisies par l'utilisateur.

nbr_e : nombre d'entrées choisies par l'utilisateur.

La division par 4 est uns division entière, le résultat attendu est la partie entière du nombre divisé

void replace_names();

À partir des noms donnés par l'utilisateur et qui figurent dans le fichier symboles, la fonction replace_names() remplace ces noms par un codage qui permet le traitement cyclique. Les entrées sont codées avec IX0_<numero_entrée>, et les sorties avec QX0_<numero_sortie>, <numero_entrée> et <numero_sortie> sont compris entre 0 et 71, vu qu'on dispose de 72 E/S.

Le codage des E/S est effectué par l'appel de la fonction

string replace_and_get_names(string programme_contennu,bool select_port_1_A,bool select_port_1_B,bool select_port_1_C_L,bool select_port_1_C_H,int i,Calias alias);

à laquelle on envoi le contenu du programme (programme_contennu), les configurations des ports (select_port_1_A,select_port_1_B,select_port_1_C_L,select_port_1_C_H) ainsi qu'un objet de type alias pour en manipuler ses variables internes (le contenu de la classe Calias sera décris par le suite). On aura comme valeur de retour le contenu du programme prêt à être

exécuté cycliquement. de la même façon la fonction replace_names() fait appel a string replace_and_get_names_int(string programme_contennu,Calias alias), qui traite elle, des variables internes choisies par l'utilisateur, et qui seront codées avec INTERN<numero_intern>,<numero_intern> est un entier compris entre 0 et 499, et peut être extensible selon les besoins.

void prg_cycles();

Le contenu du programme est maintenant prêt à être utilisé, avec les noms des variables codées, et contenant des actions simples et des structures conditionnelles. la fonction prg_cycles(); est appelée par l'interface pour exécuter un cycle programme, et prg_cycles() lit ce programme et en exécute les actions l'une après l'autre, chaque action est terminée par un point virgule ";".

Notons que trois cas se présentent ici :

Si une action d'affectation (ex: QX0_01:=true;), se présente prg_cycles() fait appel à la fonction,

void action_simple (string action , string nom_du_programme);

qui évalue le second membre (dans l'exemple "true") et en affecte la valeur au premier membre, et manipule en conséquence les valeurs des images des variables réelles (variables membre de globals), l'évaluation du second membre est effectuée par l'appel de la fonction string valeur_actuelle(string second_membre , string nom_du_programme); cette dernière remplace le second membre par sa valeur actuelle, après appel de la fonction int eval_net(string expression,string nom_du_programme); si besoin est.

Si une structure conditionnelle se présente on fait appel à la fonction,

string fonction_IF(string expression_konditionnelle,string nom_du_programme);

On envoi comme paramètre le morceau de code contenant toute l'expression conditionnelle, ainsi que le nom du programme, et en retour le bloc d'actions validée par la structure conditionnelle, rappelons qu'une structure conditionnelle en ST se présente sous cette forme

$$IF < condition_1 > THEN < action_1 > [ELSIF < condition_2 > THEN < action_2 >][ELSE < action_3 >]END\_IF$$

(Les expressions entre crochés sont optionnelles mais traitées)

- Si <condition$_1$> est vraie, on remplace la structure par <action$_1$>,

- Si <condition$_2$> existe, et est vraie, on remplace la structure conditionnelle par <action$_2$>,
- Si (<condition$_1$> et <condition$_2$> sont fausses) on remplace la structure par <action$_3$> si elle existe, Sinon on saute la structure conditionnelle et on passe à l'action qui suit.

Notons qu'un ou plusieurs blocs d'actions, une ou plusieurs structures conditionnelles peuvent exister, d'autres structures conditionnelles peuvent exister, et ceci est pris en considération, avec un nombre illimité d'imbrications possibles.

En fin le contenu du programme peut contenir des sauts, et des étiquettes, ces sauts sont aussi gérés, le mot clé "_GJ <étiquette>" est équivalent a un "goto <étiquette>", et "_GL <étiquette>" est perçu comme l'endroit de l'étiquette, un saut inconditionnel est effectué si une action simple se présente sous la forme d'un saut.

int eval_net(string expression,string nom_du_programme);

les second membres des actions, ainsi que les conditions des structures conditionnelles sont évalués par l'appel de la fonction int eval_net(string expression,string nom_du_programme), qui crée un objet de type calculette. Comme paramètre elle accepte une expression de type string, et retourne un entier de valeur 1 ou 0, selon la valeur de l'expression évaluée, le mécanisme de cette fonction est illustré dans la figure III.14

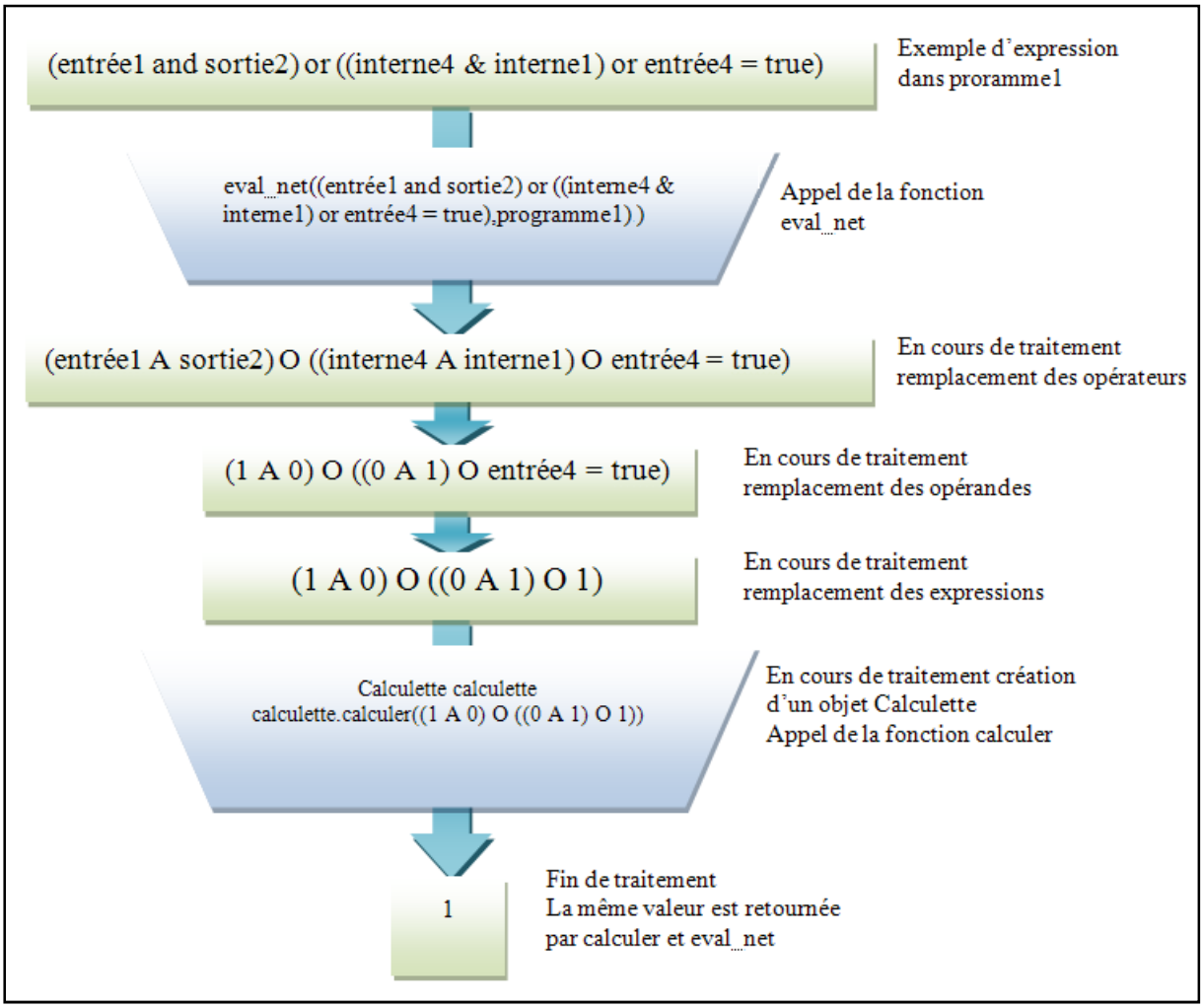

Figure III.14 *Mécanisme de traitement d'une expression*

Comme on le voit chaque variable dans l'expression est remplacée avec sa valeur, ou si une expression logique compare une variable à une autre, ou une variable à une valeur, l'expression est évaluée en tenant compte de ce critère. donc l'appel de la fonction int eval_net(string expression,string nom_du_programme), se fait lors du traitement des actions d'affectation à partir de l'évaluation du second membre ou celle des conditions des expressions conditionnelles.

Comme synthèse à la description du fonctionnement de la classe globals, nous pouvons illustrer par avec la figure III.15, certaines classes ne figurent pas sur cette figure, nous illustrons ici l'essentiel des fonctions de la classe globals et quelques dependances.

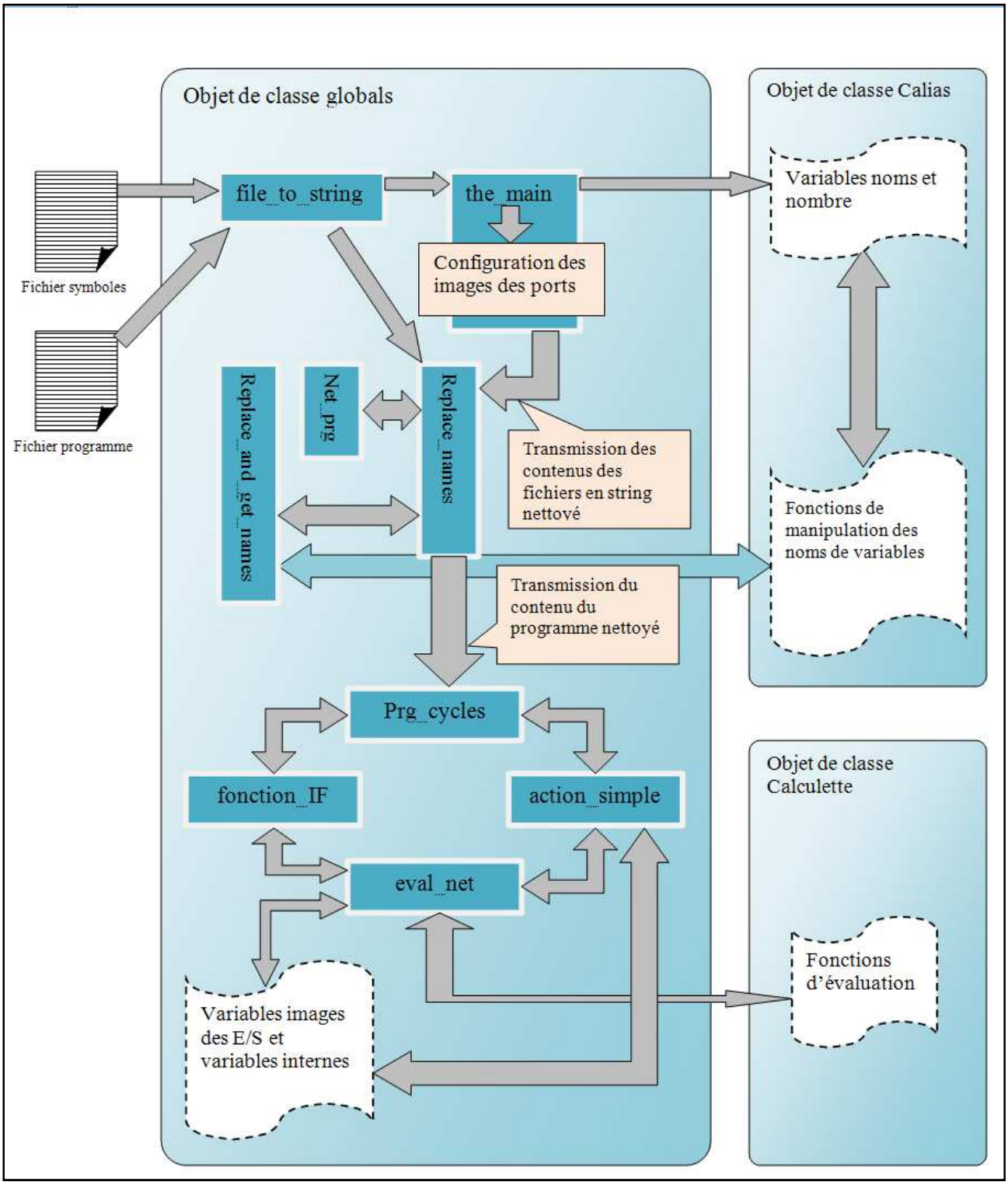

Figure III.15 *Fonctionnement de la class globals*

III.4.2.2 LA CLASSE TEMPOREL

Cette classe fait appel à la bibliothèque C++ « time.h » qui traite les variables temporelles, et permet la mesure du temps en millisecondes, cette classe a été crée pour mesurer le temps d'activation des étapes du SFC, et pour pouvoir en faire des SFC qui supportent les temporisations, les variables internes qui ont été crées pour des besoins de programmation, ne seront pas citées ici, nous allons décrire le rôle de chaque fonction membre [2].

Les fonctions membres

Un objet appartenant à la classe temporel contient quatre fonctions membres.

long int valeur() ;

Cette fonction, de type « long int » (ce type supporte une grande plage de données possibles), et n'a pas de paramètres, à son appel par un objet d'une autre classe, la fonction valeur(); retourne le temps d'activation cumulé d'une étape, une gestion des pauses et des remise en route à été prévu pour plus de performances.

void TSTART();

À l'appel de cette fonction par un objet de la classe globals, met en marche le compteur correspondant à l'objet de type temporel, pour démarrer ou reprendre le comptage du temps d'activation de l'objet du type temporel.

void TSTOP();

L'appel de cette fonction arrête le comptage du temps d'activation, mais ne le remet pas à zéro, ce qui rend une pause possible par l'utilisateur quand le programme est en cours d'exécution, sans affecter le temps d'activation cumulé.

void INIT()

En fin cette fonction remet à zéro le compteur de temps d'activation d'un objet de type temporel, pour différencier entre la première activation d'une étape, et une réactivation après une pause.

La communication avec la classe globals est faite de telle sorte qu'un objet de type globals crée un tableau de type temporel, pour donner des mesures de temps pour l'activation des étapes d'un SFC. Si l'étape GSX[i] d'un objet de classe globlas est activée (passage de "false" à "true"), GST[i] comportera le temps d'activation de cette étape.

III.4.2.3 LA CLASSE CALIAS

Cette classe assure fournit une panoplie de fonctions, et de variables internes, qui aident à extraire les noms des variables du SFC, et de les remplacer par leurs codes, les noms seront transmis à l'interface utilisateur pour indiquer à cette dernière comment sont configurés les ports et comment les variables y sont disposées. Et les codes seront transmis à l'objet de type globals, qui appel l'objet de type Calias pour coder un contenu de programme.

Les variables membres

Les strings alias_entrees; alias_sorties; alias_interns; membres de la classe Calias, sont crées avec la création d'un objet de type Calias, ils contiendront respectivement les noms des entrées, les noms des sorties, et les noms des variables internes tells que l'utilisateur les a créé avec ISAGRAF, ils y seront chargés par un objet de type globals.

Les entiers nombre_entrees; int nombre_sorties; int nombre_internes; contiennent respectivement les nombre d'entrées, de sorties, et de variables internes, s'est avec ces valeurs que l'objet de type globals configure automatiquement la sélection des ports.

Les fonctions membres

Les fonctions membres de la classe Calias, sont celles dont se servent les autres objets, pour manipuler les variables locales d'un objet de type Calias. Ces fonctions sont

string number_to_string(int number);

Fonction qui convertit un nombre en string, contenant le nombre écrit dans ce string, cette fonction est manipulée par la fonction alias_replace.

string f_extract_nom(int numero_var , string type_var) ;

Fonction qui extrait d'une des variables qui contiennent les noms des variables, le nom d'une variable particulière à partir de son numéro.

string alias_replace(int numero_var , string type_var) ;

Cette fonction en fin a pour rôle de remplacer les noms des variables, par leurs codes, selon si s'est des variables entées, sorties, ou internes. Elle fait appel aux deux premières

fonctions suscitées, elle reçoit en paramètres un nombre qui indique le numéro de la variable, et un string qui peut avoir trois valeurs « entree », « sortee », ou « interne ». Ce qui indique le type de la variable a coder.

III.4.2.4 LA CLASSE CALCULETTE

De toutes les classes que nous avons citées, ci-dessus la classe Calculette est celle qui nous a demandé le plus d'efforts, s'est en fait un petit analyseur syntaxique, son principe de fonctionnement est basé sur la récursivité [3].

Lorsqu'on veut évaluer la valeur d'une expression logique, on créer un objet de type Calculette on en appel à la fonction membre int Calculer (const string expression), cet appel déclenche une série d'appels, qui peuvent être récursifs, selon la complexité de l'expression à évaluer. On a définit au préalable des constantes qui sont les symboles des opérateurs logiques utilisées en ST (OR, AND, NOT, XOR), on note aussi que string expression et int pos sont des variables globales.

Pour illustrer le fonctionnement d'un objet de classe Calculette, nous allons nous mettre en situation, quand la fonction string valeur_actuelle(string second_membre , string nom_du_programme), trouve une variable au second membre ou une expression logique, ou que string fonction_IF(string expression_konditionnelle,string nom_du_programme); doit évaluer une expression logique, elles font appel à la fonction int eval_net(string s_c_pas_net,string nom_du_programme); qui « nettoie » l'expressions logique et en remplace chaque opérande par 1 ou 0, selon sa valeur logique, les opérateur ne sont pas laissés en reste, ils sont remplacés par leurs synonymes. On aboutira a une expression composée de « (», «) », « N », « O », « A », « X », « 1 », « 0 », uniquement, cette dernière est évaluée par un objet de classe Calculette, par l'appel de a fonction membre int Calculer (const string expression),

Si on classe les opérateurs par ordre de priorité, on aura l'ordre suivant PARENTHESE, AND, XOR, OR, nous avons considéré PARENTHESE comment le niveau de priorité le plus haut, car ce qui est entre parenthèses doit être, évalué en premier. Mais on va voir comment un objet Calculette fonctionne.

Les fonctions membres

int Calculer(const string str) retourne en faite, la valeur de retour de la fonctions membre OR() ;

int OR() a comme retour, soit le retour de la fonction XOR(), ou des opérations OR qui se suivent on évalue l'expression logique, tant qu'il y a des OR.

int XOR() a comme retour , soit le retour de la fonction AND(), ou si des XOR se suivent le résultat est évalué tant qu'il y a des XOR().

int AND() en fin retourne un le résultat qui est le AND entre deux opérandes, a droite et a gauche, les opérandes peuvent être des nombres (0 ou 1), des expressions entre parenthèses, ou des nombres ou des expressions précédés d'un NOT, si il y a des AND qui se suivent on le prends en considération. Les expressions entre parenthèses sont évaluées par l'appel de la fonction OR() ; donc elle subira le même traitement qu'une expression normale, et son retour est considéra comme une opérande, la gestion de parenthèse existe à ce niveau, mais ne génère pas d'erreur. Il peut sembler paradoxal que l'on appel au début la fonction la moins prioritaire, mais s'est pour une légèreté accrue du programme.

III.4.2.5 LA CLASSE DRIVER

Cette classe est spécialement conçue pour notre carte PCI 8255 V3, ses fonctions assurent la configuration de la carte et la communication avec le cœur fonctionnel, et avec l'interface utilisateur, donc cette classe ne peut être appelée que pour, manipuler les ports d'une carte similaire, un changement de matériel entrainera automatiquement le changement de cette classe.

La classe utilise sous Windows XP une bibliothèque de liens dynamiques, ou dynamic link library (dll), cette dernière, pour avoir l'accès aux ports physiques de la carte outrepassant le blocage de Windows, utilise un pilote en mode noyau (kernel mode driver). La bibliothèque de liens dynamiques que nous avons choisis pour des raisons de simplicité est Inpout32.dll.

Si dans une extension possible de notre solution PC, on utilise plusieurs cartes similaires, il suffira de créer plusieurs objets de type driver.

Les fonctions membres

int chargement_dll() ;

Permet de charger en mémoire la librairie Inpout32.dll, si le fichier dll n'est pas dans le même répertoire que l'application, un message d'erreur sera généré par l'interface.

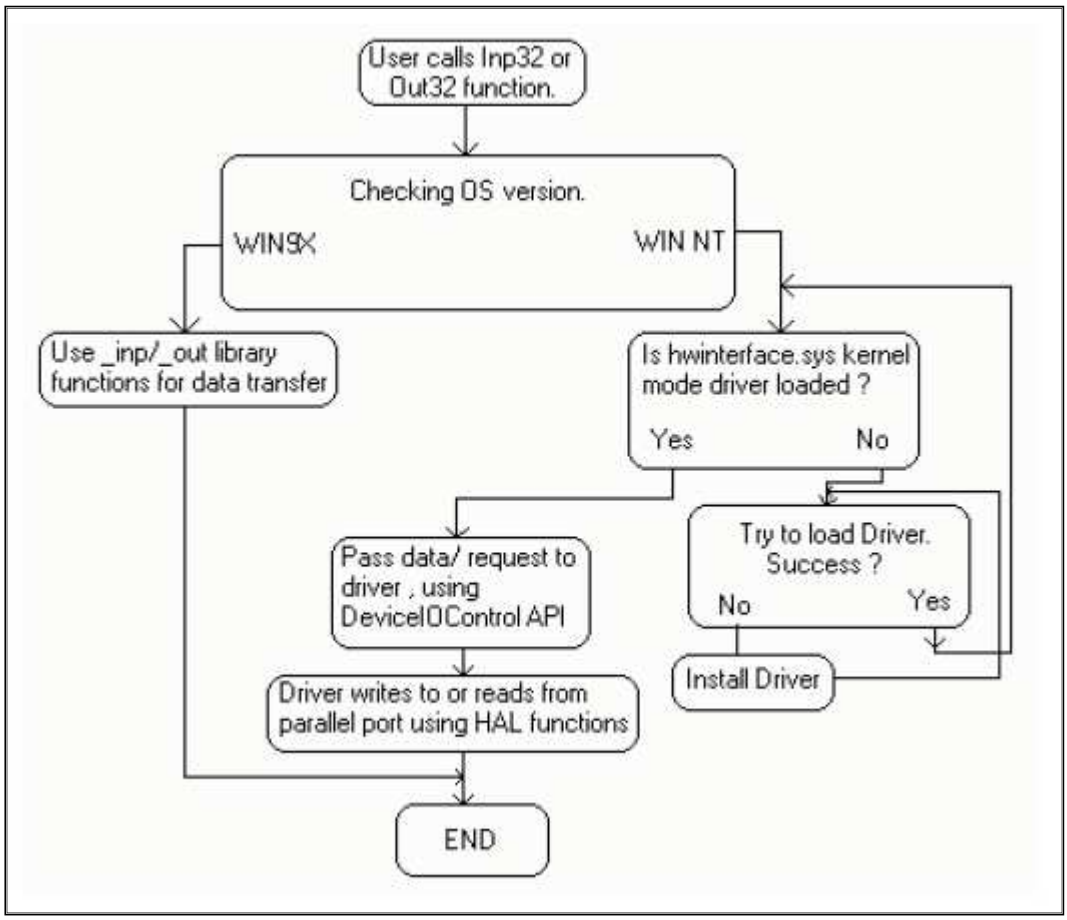

Figure III.16 *Procédure d'installation du pilote de gestion des E/S*

La figure montre comment la bibliothèque de liens dynamiques, installe le pilote de gestion des E/S physiques, selon les versions des systèmes d'exploitation windows.

void pio32_init(int base) ;

Cette fonction configure la carte, pour une communication en E/S, et la configuration des ports selon les variables de la classe globals. Ceci en prenant en compte l'adresse de base écrite en entier dans le fichier base_adress.txt, et que l'utilisateur trouvera une fois la carte installée dans la configuration matérielle de l'ordinateur.

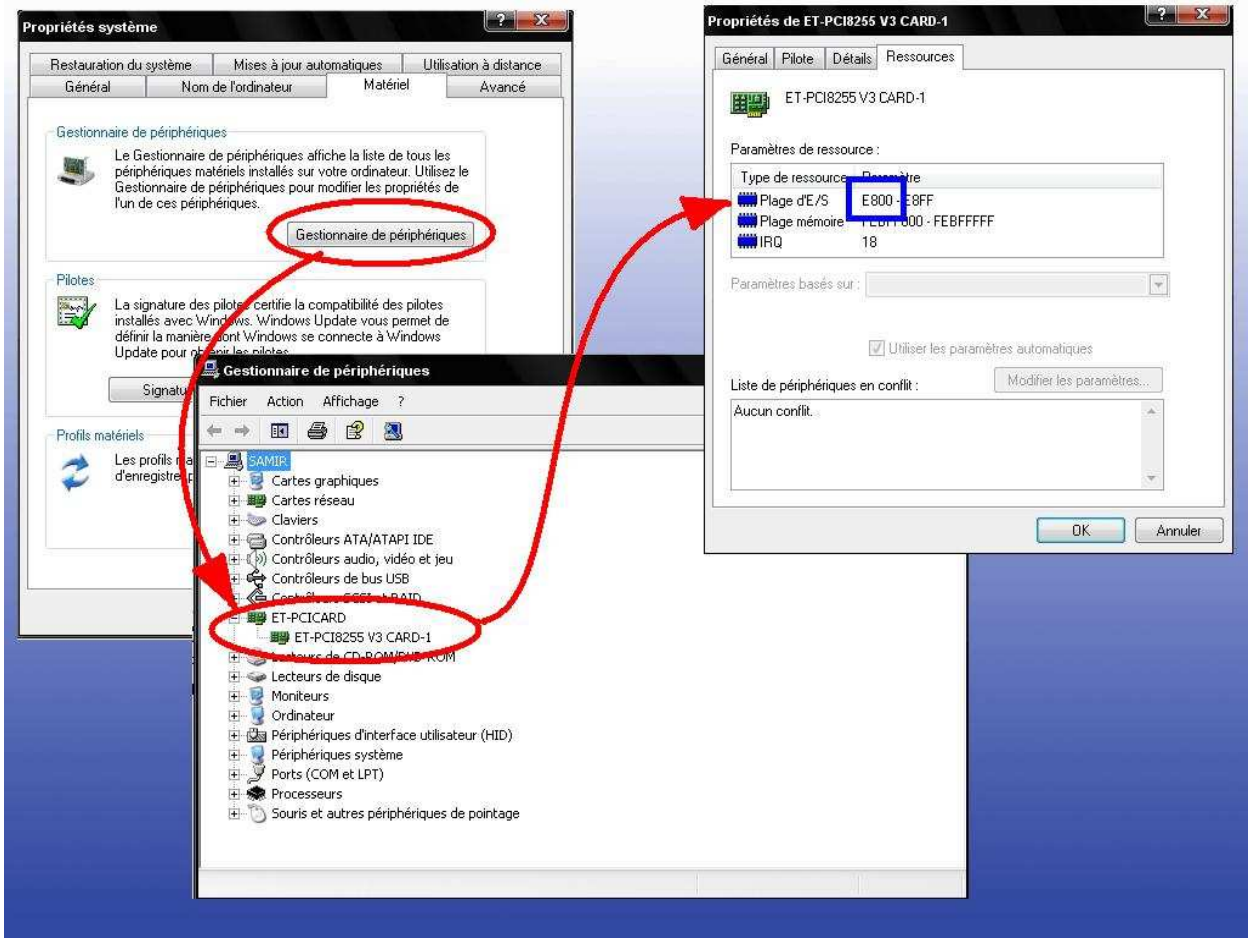

Figure III.17 *Emplacement de l'adresse de la carte*

Dans cet exemple l'adresse de base est E800 en hexadécimal, son équivalent en entier est 59392.

int pio32_read(int portx)

void pio32_write(int portx, int data)

Ces deux fonctions ont pour rôle de lire et d'écrire une valeur d'un port, cette valeur est obtenue en retour si elle est demandée en lecture avec la fonction pio32_read(portx), et est écrite en paramètre avec l'adresse du port à la fonction pio32_write (portx,data), elle devra être codée en entier.

void configure (bool A, bool B, bool CL, bool CH , int port)

Configure les ports physiques de la carte, selon la configuration des images des ports, donc selon le nombre d'E/S.

void refresh(bool port_select_A, bool port_select_B, bool port_select_CL, bool port_select_CH, bool A, bool* B, bool* CL, bool* CH, int port)*

Rafraichit les images au début et a la fin du traitement cyclique, selon l'état réel des ports physiques.

void relay()

Un relai présent sur la carte peut être activé a la demande, nous lui avons affecté le rôle d'indicateur de fonctionnement, si la carte est en bon état le relais est fermé, sinon il est ouvert.

void liberation_dll()

En fin après avoir chargé la librairie en début de traitement, on doit libérer l'espace occupé par cette dernière, ceci est fait par cette fonction.

III.4.3 COMMUNICATION ENTRE LES OBJETS DES CLASSES

Maintenant que toutes les classes sont définies, nous allons expliquer comment les objets de ces classes, communiquent entre elles pour accomplir la tache d'interpréter un programme généré par ISAGRAF, exécuter ce programme, modifier les variables, communiquer avec le matériel, et avec l'utilisateur, etc.

Une fois les fichiers préparés, l'utilisateur suivra les procédés d'utilisation de l'interface (donné dans la partie III.3.2), En arrière plan de l'interface, les objets des classes décrite dans la partie précédente sont générés, et leurs fonctions deviennent opérationnelles, elles communiquent entre elles selon le schéma de la figure III.18

Comme on le voit dans cette figure, le fonctionnement du cœur fonctionnel, l'utilisation de la bibliothèque de liens dynamiques, et les interactions entre les objets des classes, entre les objets et les fichiers, et autres, ne sont pas visibles à l'utilisateur. Une division en trois couches est alors possible, la plus haute étant les éléments qui sont à la disposition de l'utilisateur, la deuxième est celle du cœur fonctionnel, contenant les objets des classes, et en fin la couche qui interagit avec le matériel (dans notre cas la carte PCI 8255 V3), et la carte elle-même. Pour ce qui est d'un montage ou d'une application, libre cours à l'utilisateur.

Clarifions les rôles de chaque élément :

- L'interface utilisateur crée ou charge les fichiers générés, dans le répertoire courant de l'application. Prends les noms et le nombre de variables de l'objet de classe Calias. Interagit avec l'objet de classe globals pour le traitement su programme. Et envoi les ordres de configuration de la carte ainsi que les mises à jour des E/S à l'objet de classe driver selon les images des ces variables contenus dans l'objet de classe globals.
- Les objets de classes globals, temporel, Calculette, Calias, travaillent en commun pour le traitement du programme, son interprétation, et la mise à jour des images des E/S.
- La carte d'E/S reçoit les ordres de configuration, et de mise à jour des E/S, par les fonctions de l'objet de classe driver, (ces fonctions sont appelées cycliquement par l'interface en prenant en compte le fichier base_address.txt), reçoit des entrées du monde extérieur, et y envoie les sorites.

- Le comportement apparent est ordonné selon la logique d'un SFC, décrit dans le fichier programme.

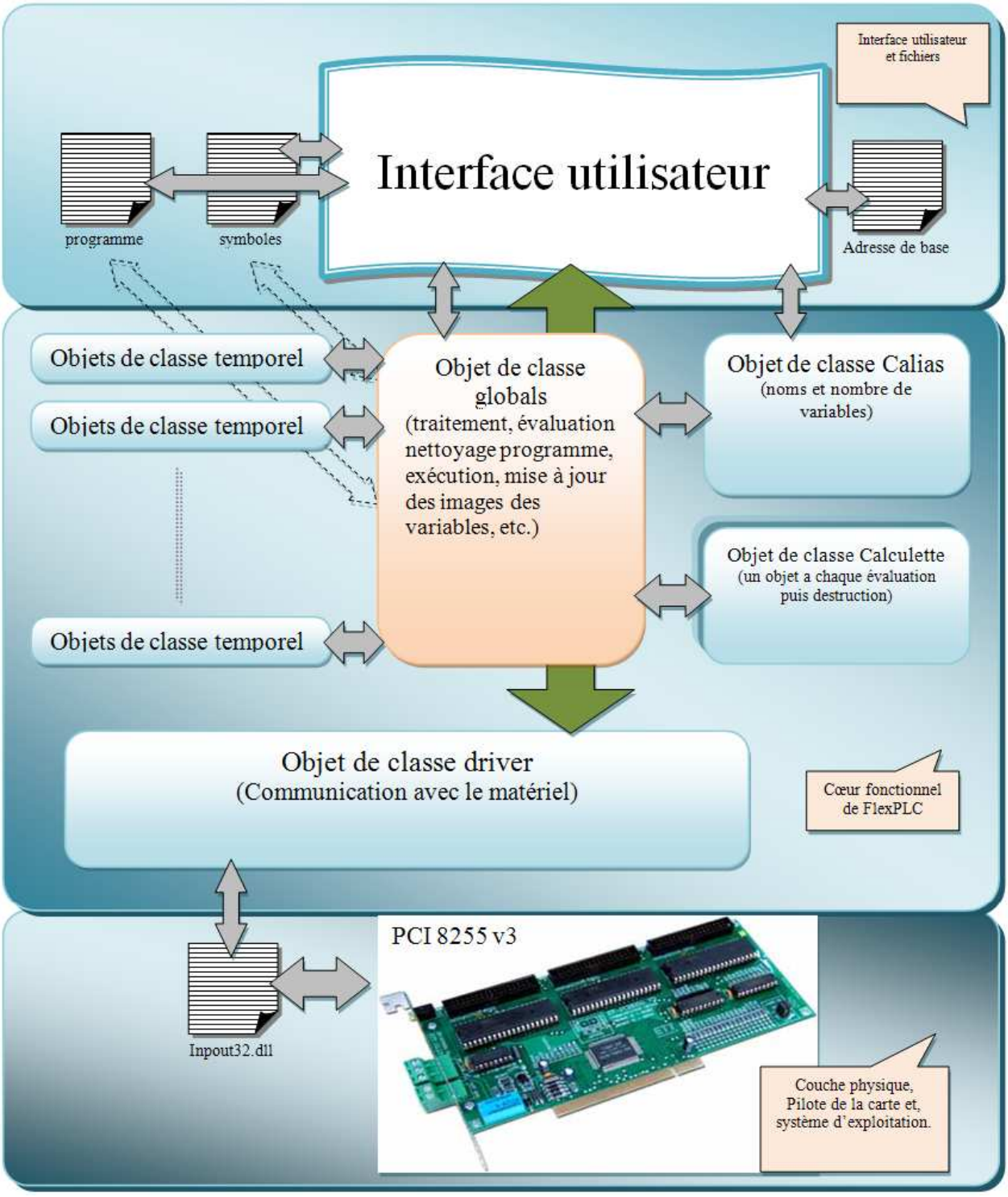

Figure III.18 *Communication entre les classes*

III.5 PRESENTATION DE LA CARTE D'E/S

III.5.1 INTRODUCTION

Nous avons vu dans les parties précédentes, comment utiliser l'interface de FlexPLC, et comment le cœur fonctionnel travaille en parallèle avec cette interface. Nous allons mettre en lumière dans cette partie l'organe de communication, la carte d'E/S, qui est une carte ET PCI8255 V3.

III.5.2 DESCRIPTION DE LA CARTE PCI8255 V3 DE ETT

La carte PCI8255 ET-V3 est une carte programmable d'entrées/sorties de 72 Bits en logique TTL. Elle a été conçue et développé pour être connectée avec un ordinateur PC, avec un bus PCI. ETT a choisit d'y implémenter un décodeur "TIGER320" Tiger Jet Network Inc, qui est spécialement conçu pour être utilisé avec le bus PCI. Ce décodeur a été testé et approuvé par PCI-SIG Organisation, il peut supporter la connexion avec un bus PCI V2.2. La carte ET-PCI8255 V3 agira donc comme un groupe d'E/S en logique TTL [14].

Trois contrôleurs programmables PIO (Programmable Input / Output) 8255, ont été choisis, chaque contrôleur a 3-ports d'E/S de 8 Bits (donc 24 Bits E/S). Chaque port peut être programmé de façon indépendante en entrée ou en sortie, ce qui fait de la carte PCI8255-V3 une carte de 9 ports E/S programmables (72 bits).

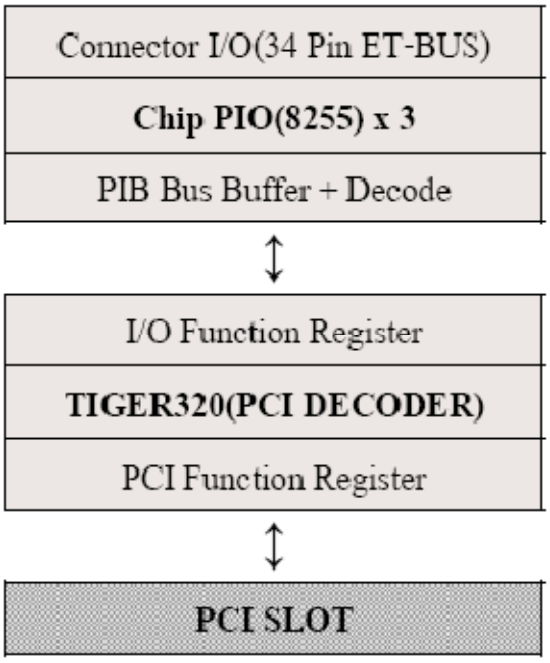

Figure III.19 *Niveaux de connexions entre un PC et la carte d'E/S [14]*

L'utilisateur devra configurer le décodeur de bus PCI (Tiger320) [15], pour qu'il puisse configurer et communiquer avec les contrôleurs 8255 à travers le bus PCI. Pour ce faire le décodeur occupe une plage de 256 adresses physiques, une partie est réservée à la configuration du décodeur (48 registres de configuration 00H à 2FH), et l'autre partie à la communication avec les contrôleurs et leurs registres d'adresses et de données, nous parleront toujours en terme d'adresse de décalage, car le PC affecte à la carte une plage d'adresses bien définie (exemple : E800H à E8FFH) l'adresse de base étant E800H, et le premier registre avec un décalage de 00H. Le bus de la carte PCI8255 V3 contient un bus de données de seulement 8 bits, a travers lesquelles le décodeur doit faire communiquer le bus PCI avec les contrôleurs.

III.5.2.1 LES REGISTRES DE CONFIGURATION

La configuration du décodeur Tiger320, qui est nécessaire sur la carte PCI8255-V3 passe par 3 registres, le registre PIB, les registres AUXC, et AUXD [14].

Le registre PIB

Ce registre occupe l'adresse de décalage 00H, il contrôle la vitesse de rafraichissement du bus de la carte, sa structure est montrée dans le tableau III.3 :

D7	D6	D5	D4	D3	D2	D1	D0
DMA Bus Mode	DMA Int. Mode	PIB Cycle Time	PIB Cycle Time	Reserve	Reset Serial Port	Reset Master	Reset EXTRST#

Tableau III.3 *Structure du registre PIB [14]*

Nous n'avons pas besoin de décrire ici le rôle de chaque bit de ce registre, mis à part les bits D4 et D5. Libre au lecteur de consulter le manuel d'utilisation de la carte.

Le bit D4 et D5 sont utilisés pour assigner une vitesse au bus de la carte

0 : 0 assigne une vitesse d'un cycle par 3 cycles de bus PCI (le plus rapide).

0 : 1 assigne une vitesse d'un cycle par 8 cycles de bus PCI.

1 : X assigne une vitesse d'un cycle par 15 cycles de bus PCI (le plus lent)

Nous avons utilisé le plus lent, nous avons assignés 1 : 1 dans l'application.

Les registres AUXC et AUXD

Les adresses de décalage pour ces registres sont 02H et 03H, ils sont assigné obligatoirement dans l'initialisation pour que le décodeur reconnaisse les positions des trois contrôleurs, ils sont aussi utilisés pour commander le relai de la carte [14].

III.5.2.2 USAGE DES CONTROLEURS 8255

En règle générale, les contrôleurs programmables d'E/S 8255, peuvent être adaptés à l'utilisation dans de nombreuses applications d'E/S. C'est des contrôleurs très connus, vu la facilité d'utilisation, on les trouve toujours dans des applications à Microprocesseurs qui requièrent davantage d'E/S, le 8255 peut être programmée dans 3 modes et chaque port peut être programmé en entrée ou en sortie. L'utilisateur peut contrôler, lire et écrire des données, par la sélection de 4 registres internes du 8255.

Nous avons utilisé dans notre application le mode 0 de chaque 8255, pour avoir le maximum d'E/S, et garder la simplicité d'utilisation. Ce mode est le plus couramment utilisé, l'utilisateur peut alors contrôler, les ports A, B, et C en entrée ou en sortie.

Dans la première étape l'utilisateur doit envoyer un mot de configuration à chaque contrôleur, au registre de contrôle de chaque 8255 (PCC1, PCC2, et PCC3), la taille de ce mot

est de 8 bits, la signification des bits de contrôle de ces registres sont montrés dans la figure III.20.

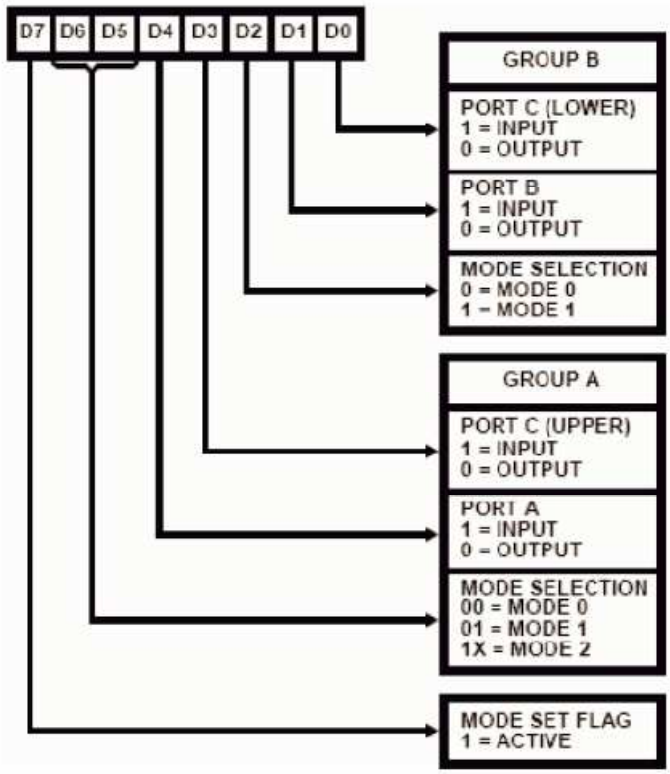

Figure III.20 *Signification des mots de contrôle des 8255[14]*

Le résultat des différentes configurations possibles de la carte est donné dans le tableau III.4

Code	**Résultat sur les ports du contrôleur 8255**			
Mot de contrôle	Port-A	Port-B	Port-Chigh	Port-Clow
80H	Sortie	Sortie	Sortie	Sortie
81H	Sortie	Sortie	Sortie	Entrée
82H	Sortie	Sortie	Entrée	Sortie
83H	Sortie	Sortie	Entrée	Entrée
88H	Sortie	Entrée	Sortie	Sortie
89H	Sortie	Entrée	Sortie	Entrée
8AH	Sortie	Entrée	Entrée	Sortie
8BH	Sortie	Entrée	Entrée	Entrée
90H	Entrée	Sortie	Sortie	Sortie
91H	Entrée	Sortie	Sortie	Entrée
92H	Entrée	Sortie	Entrée	Sortie
93H	Entrée	Sortie	Entrée	Entrée
98H	Entrée	Entrée	Sortie	Sortie
99H	Entrée	Entrée	Sortie	Entrée
9AH	Entrée	Entrée	Entrée	Sortie
9BH	Entrée	Entrée	Entrée	Entrée

Tableau III.4 *Tableau de configurations pour un contrôleur 8255[14]*

L’adressage des registres des trois contrôleurs de la carte PCI8255-V3 est donné dans le tableau III.5, pour montrer comment adresser les registres de contrôle, et les registres de données [14].

N° du 8255	Port = fonction	Adresse
8255 n°1	PA1 = port d'E/S	ADB + C0H
	PB1 = port d'E/S	ADB + C4H
	PC1 = port d'E/S	ADB + C8H
	PCC1 = contrôle du 8255	ADB + CCH
8255 n°2	PA2 = port d'E/S	ADB + D0H
	PB2 = port d'E/S	ADB + D4H
	PC2 = port d'E/S	ADB + D8H
	PCC2 = contrôle du 8255	ADB + DCH
8255 n°3	PA3 = port d'E/S	ADB + E0H
	PB3 = port d'E/S	ADB + E4H
	PC3 = port d'E/S	ADB + E8H
	PCC3 = contrôle du 8255	ADB + ECH

Tableau III.5 *adressage des registres des 8255 dans la carte PCI 8255 V3 [14]*

La signification de ADB dans le tableau d'adressage des 8255, est Adresse De Base (exemple E800H).

III.5.2.3 INSTALLATION DE LA CARTE

L'installation de la carte PCI8255-V3 n'est pas une des plus difficiles, il faut disposer d'un PC avec un slot PCI, et prendre en considération les consignes de sécurité lors de l'installation matérielle. En suite disposer des fichiers nécessaires pour que la carte soit reconnue par le système d'exploitation, ces fichiers sont livrés avec la carte pour différentes versions Windows.

L'accès aux ressources physiques est facile avec les versions Windows 95 / 98, mais n'est pas chose aisée pour les systèmes plus récents comme Windows 2000 / XP. S'est pour cela qu'on à établi une communication avec la carte a travers une librairie DLL plus de détails on été cités dans la partie description du cœur fonctionnel.

III.6 DEMONSTRATION

III.6.1 INTRODUCTION

Afin de mettre en évidence notre travail, nous avons réalisé une maquette d'une unité de tri qui sera géré avec notre solution FlexPLC et la carte d'E/S.

Nous allons décrire le fonctionnement de la maquette, et les opérations à effectuer, de la conception du programme SFC dans ISAGRAF, à l'exécution sous FlexPLC.

III.6.2 PRESENTATION DE LA MAQUETTE

III.6.2.1 CAHIER DE CHARGE

La maquette a été élabore suivant les phases d'un projet, qui sont le design, le développement et la construction. Son aspect final est l'aboutissement de plusieurs idées et astuces.

Le cahier de charge prend en compte les conditions suivantes :

-A la sortie d'un procédé de fabrication de deux types de boites, un tri est nécessaire.

-On doit concevoir un système automatisé pour ce tri.

-Le système reçoit les boites sur un tapis roulant, et met chaque type d'un coté, les deux types étant différents par la taille.

III.6.2.2 LE DESIGN

Les éléments et composants utilisés pour la construction de notre maquette, ont été choisis en fonction des paramètres, disponibilité et coût.

Pour satisfaire le cahier de charge, la maquette se compose de:

- trois tapis roulant pour le déplacement des boites.

- Un bras de robot à deux degrés de liberté pour déplacer un type de boite vers son tapis roulant.

La Figure III.21 donne une vue générale de la réalisation.

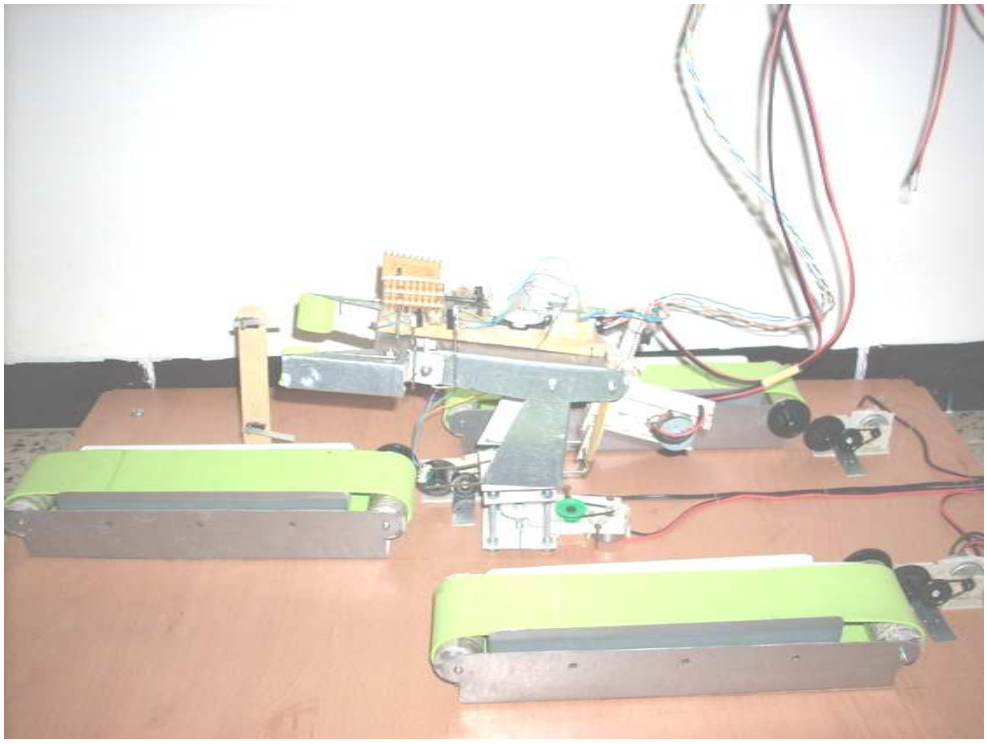

Figure III.21 *Vue générale de la maquette*

III.6.2.3 LE MONTAGE MECANIQUE

La structure du tapis roulant ainsi que celle du bras de robot ont étés fabriqués avec une tôle de 1mm d'épaisseur, pour un souci de solidité et de volume réduit. Le tout repose sur un support en bois de 70 cm sur 40 cm.

III.6.2.4 LES CAPTEURS ET ACTIONNEURS

On n'a utilisé que les fins de course comme capteurs, et les moteurs à courant continu comme actionneurs, pour des raisons de disponibilité, et de simplicité. La maquette dispose de six fins de course, et de six moteurs à courant continu avec :

- Deux fins de course sur le tapis roulant principal, pour identifier le type de boit.
- Un fin de course, et une butée mécanique pour l'articulation du bras.
- Deux fins de course pour détecter les deux positions du bras de robot (fermé ou relâché).

- Un moteur, pour chaque tapis roulant (trois en tout).
- Un moteur pour la rotation du bras.
- Un moteur pour l'articulation.
- Un moteur pour fermer ou relâcher le bras de robot.

III.6.3 LE PROGRAMME EN SFC

III.6.3.1 CREATION DU PROJET SFC

Après lancement du gestionnaire de projets d'ISAGRAF, on crée un nouveau projet avec le nom de « projet1 », un projet vide est alors généré.

On ouvre la fenêtre de gestion de programme, créer un nouveau programme avec le nom de « projet1 » et choisissait SFC pour le langage, comme le montre la figure III.22.

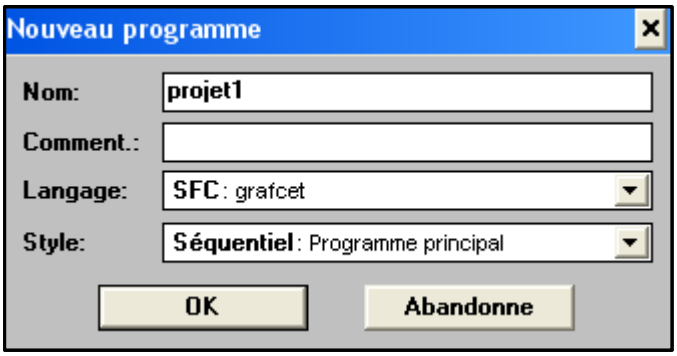

Figure III.22 *Création d'un programme SFC dans ISAGRAF*

Le symbole de SFC (voir tableau II.19) avec le nom du programme qu'on a mit, apparait sur la fenêtre gestionnaire de programme, comme le montre la figure II.10

III.6.3.2 MANIPULATION DU PROGRAMME SFC

Avant de réaliser notre programme SFC, il faut d'abord déclarer nos variables, en appuyant sur le bouton dictionnaire des variables (voir tableau II.20), la fenêtre s'ouvre, alors

on sélectionne l'onglet booléen, et on introduit les variables avec les noms qui se trouvent dans la figure III.22.

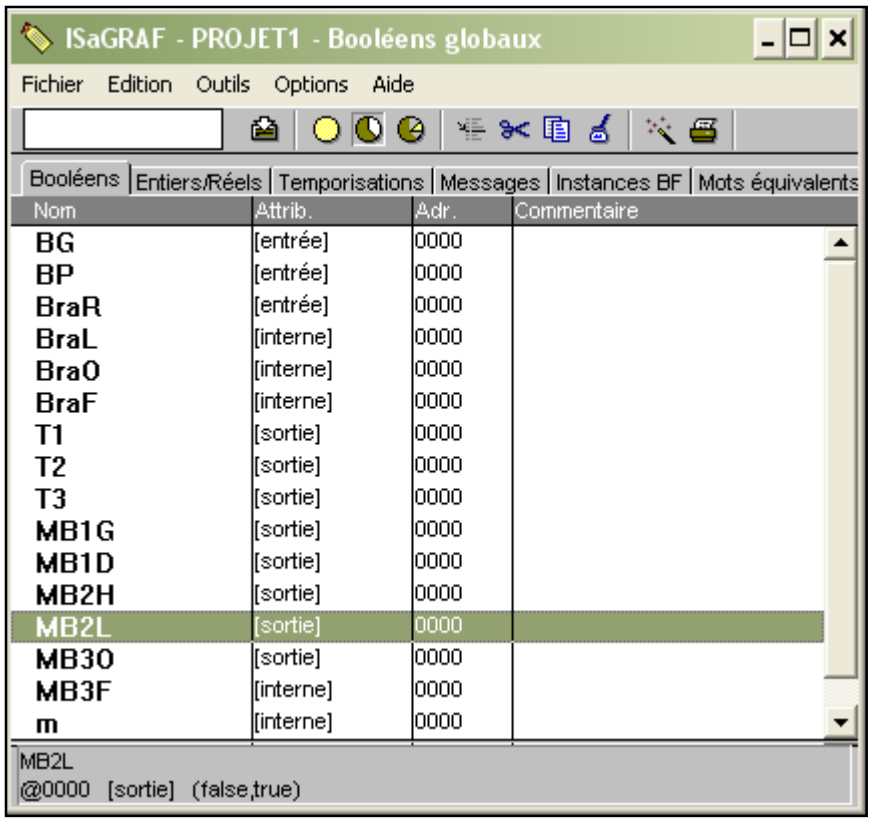

Figure III.23 *Déclaration des variables*

Après déclaration des variables, on ouvre la fenêtre de saisie du programme SFC pour réaliser le programme SFC qui satisfait le cahier de charge comme le montre la figure III.24.

Après avoir fini notre SFC, on câble les E/S, et on décoche l'option « Utilisé le moteur SFC embarqué » dans la fenêtre d'option de compilation pour obtenir un code ST structuré, comme le montre la figure III.23.

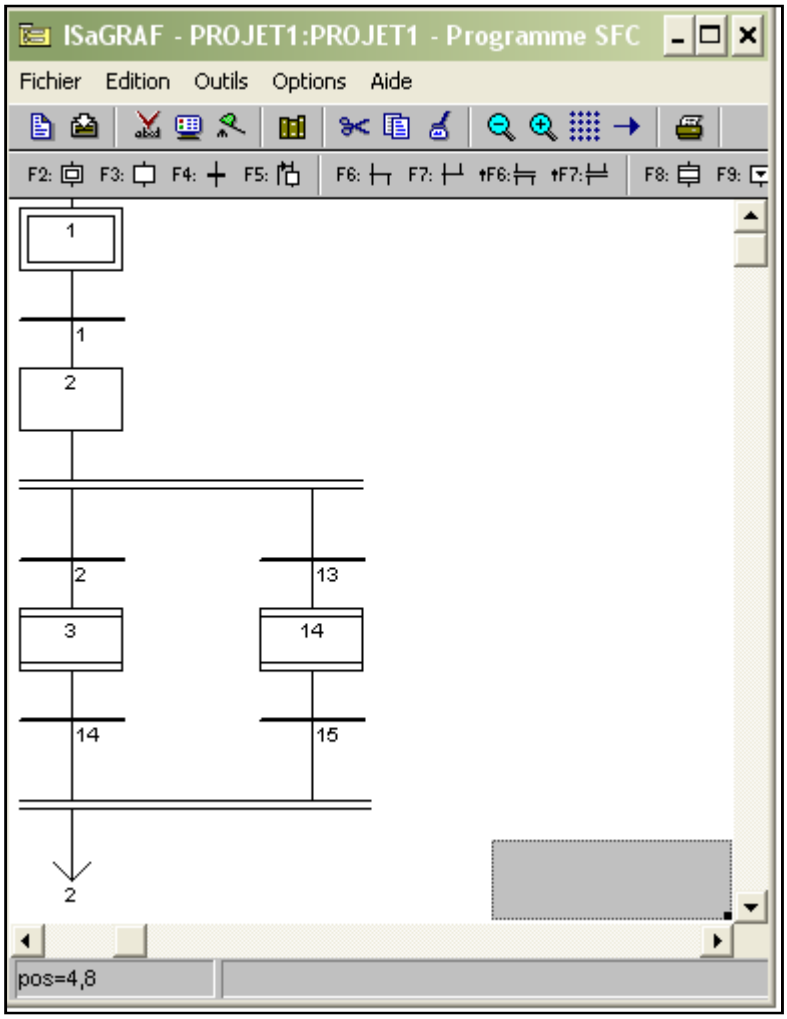

Figure II.24 *Fenêtre de saisie du graphe SFC (Démonstration)*

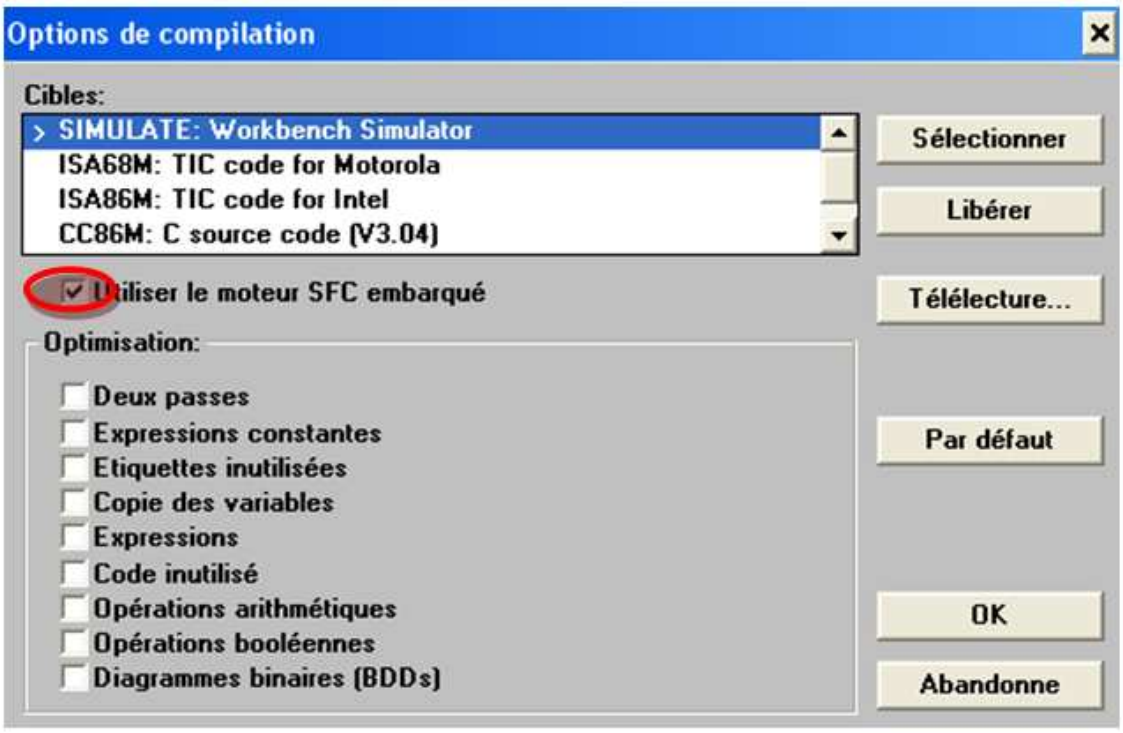

Figure III.25 *Fenêtre Option de compilation*

III.6.4 L'INTERFAÇAGE CARTE/MAQUETTE

III.6.4.1 MANIPULATION DE L'INTERFACE FLEXPLC

Après compilation et génération du code, on ouvre la fenêtre FlexPLC pour charger le contenu de symboles dans la Zone de chargement des fichiers, avec le nom « project1_s » et enregistrer. On revient à la fenêtre de génération de code d'ISAGRAF pour généré le code précompilé, et refaire la même chose de ce que nous avons fait pour le code symboles, à l'enregistrement avec FlexPLC le programme doit avoir le même nom que le programme SFC.

Après enregistrement des deux fichiers, la zone de visualisation des images des E/S, et la zone de contrôle des variables internes sont actives, et le comportement du programme est transcrit alors en actions sous FlexPLC, et les E/S sont visibles sur le 8255 #1, ainsi que sur l'interface, comme le montre la figure III.26.

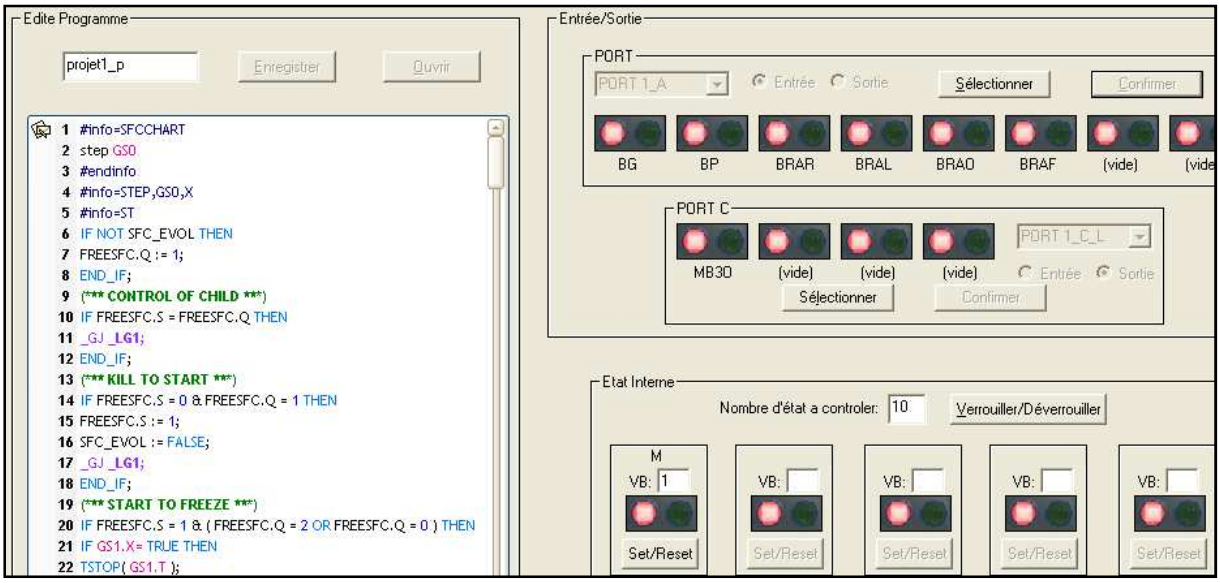

Figure III.26 *Fenêtre principale de FlexPLC (démonstration)*

CONCLUSION GENERALE

CONCLUSION GENERALE

L'objectif de ce projet était de concevoir, une solution d'automatisation basée sur un PC. La réalisation nous a poussés à approfondir nos connaissances en programmation orientée objet (POO), et à faire appel à toutes nos ressources intellectuelles acquises dans le cours de notre formation d'élève ingénieur. Ce projet nous a permis notamment de faire face a des difficultés diverses et complexes, et de les résoudre efficacement, grâce a la persévérance et au travail en équipe. Tout ceci pousse l'élève ingénieur à avoir un acquis riche et diversifié.

Le projet est passé par plusieurs phases :

- Connaissance des normes en vigueur dans le domaine de l'automatisation, notamment la norme IEC 1131.
- Compréhension du code généré en ST d'ISAGRAF.
- Développement du cœur fonctionnel qui interprète le code généré.
- Adaptation a une interface homme/machine développée en parallèle, pour former le programme FlexPLC.
- Adaptation a une carte d'E/S (PCI8255 V3) fournie au cours du développement par Moveit Automation.
- Et en fin la réalisation d'un modèle réduit d'une unité de tri, pour y appliquer la solution PC que nous avons développés.

La bonne gestion du temps et l'aboutissement de notre projet sont dues à plusieurs facteurs

- La bonne planification des taches a exécuter durant toute la durée du projet.
- La distribution des taches a effectuer sur les ressources humaines.
- L'aptitude des éléments du groupe à accomplir sa tache efficacement.

PERSPECTIVES

L'élaboration d'une solution d'automatisation basée sur un PC avec un langage orienté objet (Visual C++ 6), donne une grande flexibilité au développeur, et nous ouvrent de nombreuses perspectives :

- L'augmentation du nombre d'E/S jusqu'à 288, en ajoutant trois autres cartes PCI 8255 V3.
- Ouverture du cœur fonctionnel aux autres langages de la norme IEC 1131-3 (langage IL et LD).
- Développer un éditeur de langages IEC 1131-3 et l'intégrer à FlexPLC, pour incorporer l'environnement de programmation et de supervision, sans dépendre de logiciel tiers, ceci peut être l'objet d'un projet de fin d'études.
- Adaptation d'une librairie de cartes d'E/S, comprenant plusieurs gammes de produits.
- Envisager de charger les programmes sur des microcontrôleurs.
- En fin la création d'une entreprise axée sur le contrôle par pc des processus industriels.

ANNEXES

ANNEXES

ANNEXE A

A.1 Liste des mots réservés d'ISAGRAF

Ces identificateurs ne doivent pas être utilisés en tant pour nommer un programme, une variable, une fonction ou un bloc fonctionnel [13]. Voici la liste des mots clés réservés dans le tableau A.1.

Liste des mots réservés d'ISAGRAF
ANA, ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,
BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL,
CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS,
DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD,
ELSE, ELSIF, EN, END_CASE,END_FOR,END_FUNCTION,END_IF,END_PROGRAM, END_REPEAT, END_RESSOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT,
FALSE, FEDGE, FIND, FOR, FUNCTION,
GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT,
IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME,
JMP, JMPC, JMPCN, JMPN, JMPNC,
LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD,
MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX,
NE, NOT,
OF, ON, OPERATE, OR, OR_MASK, ORN,
PROGRAM
R, REDGE, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL,

REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REDGE, REPEAT, REPLACE, RESSOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR,
S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD, STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME, STRUCT, SUB, SYS_ERR_READ, SYS_ERR_TEST, SYS_INITALL, SYS_INITANA, SYS_INITBOO, SYS_INITTMR, SYS_RESTALL, SYS_RESTANA, SYS_RESTBOO, SYS_RESTTMR, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO, SYS_SAVTMR, SYS_TALLOWED, SYS_TCURRENT, SYS_TMAXIMUM, SYS_TOVERFLOW, SYS_TRESET, SYS_TWRITE, SYSTEM,
TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL, TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TMR, TO, TOD, TRUE, TSTART, TSTOP, TYPE,
UDINT, UINT, ULINT, UNTIL, USINT,
VAR, VAR_ACCESS,VAR_EXTERNAL,VAR_GLOBAL,VAR_IN_OUT,VAR_INPUT VAR_OUTPUT,
WHILE, WITH, WORD,
XOR, XOR_MASK, XORN

Tableau A.1 *Liste des mots réservés d'ISAGRAF*

ANNEXE B

B.1 Boites de dialogues de l'interface FlexPLC

B.1.1 Fichier manquant

Lorsque l'utilisateur lance le processus pour la première fois le programme vérifie si les fichiers nécessaires se trouvent dans le même répertoire que FlexPLC, un message d'erreur est généré par l'interface s'il y a un fichier manquant comme le montre la figure B.1.

Figure B.1 *Erreur fichier manquant*

B.1.2 Dépassement du nombre E/S autorisé

Après chargement des deux fichiers générés par ISAGRAF le programme vérifie si le nombre d'E/S dépasse les capacités du matériel utilisé, ou si l'utilisateur choisit un nombre d'E/S non configurable, en utilisant la formule (III.1). Un message d'erreur est généré par l'interface s'il y a dépassement, comme le montre la figure B.2

Figure B.2 *Nombre d'E/S non autorisé*

B.1.3 Erreur de choix du numéro de la variable interne

Lors de transmission du fichier symboles le programme compte le nombre de variable interne déclarer dans le dictionnaire des variable d'ISAGRAF par l'utilisateur, donc si l'utilisateur choisit un nombre supérieur au nombre de variable interne déclarées ou un

nombre négatif dans la zone d'édition « choix du numéro de la variable internes » (voir figure III.12) un message d'erreur est généré par l'interface comme le montre la figure B.1

Figure B.3 *Sélection de numéro de variable interne incorrecte*

Remarque : l'utilisateur, dans ce cas, a déclaré deux variables internes.

B.1.4 Sélection multiple d'une variable interne

Lorsque l'utilisateur sélectionne une variable interne déjà sélectionnée dans l'une des zones d'édition « choix du numéro de la variable interne » (voir figure III.12) le message d'erreur figure B.4 apparait.

Figure B.4 *Sélection multiple d'une variable interne*

B.1.5 Choix d'un nombre de variable interne à Contrôlé incorrecte

Si l'utilisateur met un nombre supérieur à 10 ou négatif dans la zone d'édition« choix du nombre de variable interne à Contrôler » (voir figure III.12), un message d'erreur est généré, comme le montre la figure B.5

Figure B.5 *Sélection de nombre de variable interne à contrôler incorrecte*

B.1.6 Enregistrement de fichier

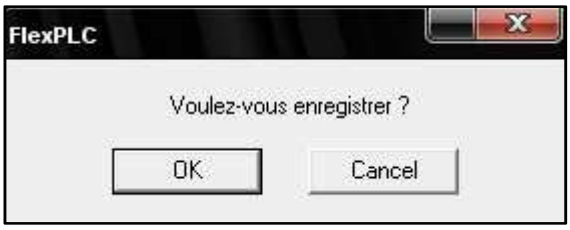

Figure B.5 *Boite de dialogue d'enregistrement de fichier*

ANNEXE C

C.1 Exemple de code généré et le fichier symboles correspondant

C.1.1 fichier symboles

```
@ISA_SYMBOLS,243406993
#NAME,graf,3.55
#DATE,21/05/2008
#SIZE,G=1,S=0,T=0,L=0,P=0,V=32
#COMMENT,wsma1tst
@PROGRAMS,1
#!5001,MGRAF
@STEPS,0
@TRANSITIONS,0
@BOOLEANS,10
#!1009,QX1_1,+O,!0000,FALSE,TRUE
#!100A,QX1_2,+O,!0000,FALSE,TRUE
#!1021,INTER1,+X,!0000,FALSE,TRUE
#!1022,INTER2,+X,!0000,FALSE,TRUE
#!1023,SFC_EVOL,+X,!5001,FALSE,TRUE
#!1024,GS1.X,+X,!5001,FALSE,TRUE
#!1025,GS2.X,+X,!5001,FALSE,TRUE
#!1026,GT1,+X,!5001,FALSE,TRUE
#!1027,GT2,+X,!5001,FALSE,TRUE
#!1028,GT0,+X,!5001,FALSE,TRUE
@ANALOGS,2
#!2001,MGRAF.Q,+X,!0000,I,
#!2002,MGRAF.S,+X,!0000,I,
@TIMERS,2
#!3001,GS1.T,+X,!5001
#!3002,GS2.T,+X,!5001
@MESSAGES,0
```

C.1.2 Fichier code précompilé

```
#info=SFCCHART
step GS0
#endinfo
#info=STEP,GS0,X
#info=ST
IF NOT SFC_EVOL THEN
MGRAF.Q := 1;
END_IF;
(*** CONTROL OF CHILD ***)
IF MGRAF.S = MGRAF.Q THEN
_GJ _LG1;
END_IF;
```

```
(*** KILL TO START ***)
IF MGRAF.S = 0 & MGRAF.Q = 1 THEN
MGRAF.S := 1;
SFC_EVOL := FALSE;
_GJ _LG1;
END_IF;
(*** START TO FREEZE ***)
IF MGRAF.S = 1 & ( MGRAF.Q = 2 OR MGRAF.Q = 0 ) THEN
IF GS1.X= TRUE THEN
TSTOP( GS1.T );
END_IF;
IF GS2.X= TRUE THEN
TSTOP( GS2.T );
END_IF;
IF MGRAF.Q = 2THEN
MGRAF.S := 2;
_GJ _LG1;
END_IF;
END_IF;
(*** START TO KILL ***)
IF NOT(MGRAF.S = 1 & MGRAF.Q = 0 ) THEN _GJ _LG4;
END_IF;
MGRAF.S := 0;
#endinfo
#info=ST
IF NOT (GS1.X ) THEN _GJ _LG5;
END_IF;
#endinfo
#info=ST
_GL _LG5;
#endinfo
#info=ST
IF NOT (GS2.X ) THEN _GJ _LG6;
END_IF;
#endinfo
#info=ST
_GL _LG6;
#endinfo
#info=ST
_GL _LG4;
(*** FREEZE TO KILL ***)
IF ( MGRAF.S = 2 OR MGRAF.S = 0 ) & MGRAF.Q = 0  THEN
MGRAF.S := 0;
GS1.X:= FALSE ;
GS2.X:= FALSE ;
_GJ _LG1;
END_IF;
(*** FREEZE TO RESTART ***)
IF MGRAF.S = 2 & MGRAF.Q = 3 THEN
MGRAF.S := 1;
MGRAF.Q := 1;
IF GS1.X= TRUE THEN
TSTART( GS1.T );
END_IF;
IF GS2.X= TRUE THEN
TSTART( GS2.T );
END_IF;
_GJ _LG1;
END_IF;
_GL _LG1;
```

```
IF NOT (MGRAF.S = 1 ) THEN _GJ _LG0;
END_IF;
#endinfo
(*** SFC_EVOL : INIT FOR FIRST PASS ***)
(*** EVALUATION OF TRANSITION RECEPTIVITY ***)
#info=ST
GT1 := GS1.X ;
IF NOT ( GT1 ) THEN _GJ _LG8;
 END_IF;
#endinfo
#info=ST
(*#GT1:*)
#endinfo
#info=ST
GT1 :=
(*#line=1*)
INTER1;
;#endinfo
#info=ST
_GL _LG8;
#endinfo
#info=ST
GT2 := GS2.X ;
IF NOT ( GT2 ) THEN _GJ _LG9;
 END_IF;
#endinfo
#info=ST
(*#GT2:*)
#endinfo
#info=ST
GT2 :=
(*#line=1*)
INTER2;
;
#endinfo
#info=ST
_GL _LG9;
#endinfo
#info=ST
GT0 := TRUE;
IF NOT ( GT0 ) THEN _GJ _LG10;
 END_IF;
#endinfo
#info=ST
(*#GT0:*)
#endinfo
#info=ST
GT0 :=
(*#line=1*)
NOT(SFC_EVOL);
#endinfo
#info=ST
_GL _LG10;
#endinfo
(*** REALIZATION OF ACTIONS ***)
(*** REALIZATION OF Z ACTION ***)
#info=ST
IF NOT ( GS1.X AND NOT( GT2 OR GT0 )  AND ( GT1 )  ) THEN _GJ _LG11;
 END_IF;
#endinfo
```

```
#info=ST
TSTOP( GS1.T );
_GL _LG11;
#endinfo
#info=ST
IF NOT ( GS2.X AND NOT( GT1 )  AND ( GT2 )  ) THEN _GJ _LG12;
 END_IF;
#endinfo
#info=ST
TSTOP( GS2.T );
_GL _LG12;
#endinfo
(*** REALIZATION OF PULSE ACTIONS ***)
#info=ST
IF NOT( GS1.X =FALSE  AND ( GT2 OR GT0 )  )  THEN _GJ _LG13;
 END_IF;
#endinfo
#info=ST
(*#GS1:*)
#endinfo
#info=ST
(*#line=1*)
QX1_2:=false;
#endinfo
#info=ST
(*#line=2*)
QX1_1:=true;
#endinfo
#info=ST
GS1.T := t#0s;
TSTART( GS1.T );
_GL _LG13;
#endinfo
#info=ST
IF NOT( GS2.X =FALSE  AND ( GT1 )  )  THEN _GJ _LG14;
 END_IF;
#endinfo
#info=ST
(*#GS2:*)
#endinfo
#info=ST
(*#line=1*)
QX1_2:=true;
#endinfo
#info=ST
(*#line=2*)
QX1_1:=false;
#endinfo
#info=ST
GS2.T := t#0s;
TSTART( GS2.T );
_GL _LG14;
#endinfo
(*** MOVE TOKENS : change step.X value ***)
#info=ST
IF NOT ( GT1 ) THEN _GJ _LG15;
 END_IF;
GS1.X:= FALSE;
 _GL _LG15;
IF NOT ( GT2 ) THEN _GJ _LG16;
```

```
 END_IF;
GS2.X:= FALSE;
 _GL _LG16;
IF NOT ( GT0 ) THEN _GJ _LG17;
 END_IF;
_GL _LG17;
IF NOT ( GT1 ) THEN _GJ _LG18;
 END_IF;
GS2.X:= TRUE;
_GL _LG18;
IF NOT ( GT2 ) THEN _GJ _LG19;
 END_IF;
GS1.X:= TRUE;

_GL _LG19;
IF NOT ( GT0 ) THEN _GJ _LG20;
 END_IF;
GS1.X:= TRUE;
 _GL _LG20;
#endinfo
(*** REALIZATION OF NONE STORED ACTION ***)
#info=ST
IF NOT ( GS1.X ) THEN _GJ _LG21;
 END_IF;
#endinfo
#info=ST
_GL _LG21;
SFC_EVOL := TRUE;
#endinfo
#info=ST
IF NOT ( GS2.X ) THEN _GJ _LG22;
 END_IF;
#endinfo
#info=ST
_GL _LG22;
SFC_EVOL := TRUE;
#endinfo
(*** NOTHING TO DO ***)
#info=ST
_GL _LG0;
#endinfo
#endinfo
```

BIBLIOGRAPHIE CONSULTEE

[1] I.HORTON, « Visual C++6 », édition Eyrolles, 2001

[2] M.MOALRET, «C/C++ et programmation objet », édition ARMAND COLIN 1989.

[3] A .STEVENS « BASIC 6 ; FORMATION VISUELLE », Edité par First, 2000.

[4] R.NANKO « Microsoft visual c++ 6.0 » édition maulde et reneu 2000.

[5] D.CHAPMAN « Visual c++ 6 » édition CampusPress 1998.

[6] P.JARGOT, « Langages de programmation pour API. Norme IEC 1131-3 », Techniques de l'ingénieur, Vol. S8030.

[7] G.DÉCHENAUX, « API et PC : solutions concurrentes ou complémentaires», Techniques de l'ingénieur, Vol. R8022.

[8] M.ROUX, « Cahier des charges des automatismes. Analyse fonctionnelle », Techniques de l'ingénieur, Vol. S8095.

[9] H.CHEKIREB, « Electronique de Puissance » *Notes de Cours de 4ème Année Automatique de l'Ecole Nationale Polytechnique,* Année universitaire 2006-2007.

[10] A.BERKOUK, « Informatique Industriel » *Notes de Cours de 5ème Année Automatique de l'Ecole Nationale Polytechnique,* Année universitaire 2007-2008.

[11] A.ABRICHE, « Réalisation et gestion d'un prototype de station de pompage à base d'automates programmables industriels SIEMENS » projet de fin d'étude, Ecole Nationale Polytechnique, 2007.

Manuels

[12] ISAGRAF, « Manuel utilisateur ».

[13] ETT, « User Manual For ET-PCI8255 V3 Card ».

[14] TIGER JET NETWORK **«** Tiger320 Datasheet ».

Internet

[15] www.PLCopen.org

[16] www.isagraf.com

[17] www.pefformancesw.com/index.html

[18] www.siemens.com

[19] www.moveit-automation.com

NOTES

Résumé :

Ce travail a pour but la conception, et la réalisation d'une solution d'automatisations basée sur un PC, en ayant au point de départ une carte de communication, un outil de développement Visual C++6, et un logiciel de programmation ISAGRAF conforme aux normes internationales dans le domaine de l'automatisation. La solution basée PC ainsi conçue comporte un programme FlexPLC, et une carte de communication PCI 8255 V3. En dernier lieu la solution est appliquée sur un modèle réduit d'une unité de tri.

Mots clés : Conception, application, Automatisation, Visual C++, ISAGRAF, Normes, Solution basée PC, maquette.

ملخص :

يهدف هذا العمل إلى تصميم و إنجاز حل تأليات يرتكز على حاسوب شخصي، نقطة انطلاقه بطاقة للاتصال و آلة تطوير Visual C++6 و كاتب البرمجيات ISAGRAF مطابق للمقاييس الدولية في ميدان الآلية . يتألف الحل المرتكز على الحاسوب الشخصي، المصمم هكذا، من برنامج FlexPLC و بطاقة الاتصال PCI 8255 V3. في آخر الأمر، يطبق الحل على نموذج مصغر لوحدة فرز.

الكلمات الأساسية : **التصميم، التطبيق، التآلية، Visual C++، ISAGRAF، المقاييس، الحل المرتكز على الحاسوب الشخصي، النموذج المصغر.**

Abstract :

This work aims to design, and implementation of a PC based solution for automation, with a communication card, a development tool Visual C + +6, and software programming ISAGRAF conform to international standards in the field of automation, at the starting point. The designed PC based solution contains a program FlexPLC, and a communication card PCI 8255 V3. Lastly the solution is applied on a small scale model of a unit of sorting.
Keywords: design, implementation, automation, Visual C + +, ISAGRAF, Standards, PC based solution, small scale model.