UNIVERSITÉ DE LIÈGE

AUTOMATISATION ET ROBOTISATION DE LA PRODUCTION

NOTES DE LABORATOIRE

Partim: Introduction à la Programmation des Automates et des Robots

Pierre Duysinx
Geoffray Hutsemekers
Henri Lecocq

Notes Provisoires
Année académique 2009-2010

TABLE DES MATIERES

1. PROGRAMMATION DE L'AUTOMATE SIEMENS S7-400

Le laboratoire de robotique et automatisation dispose de plusieurs automates ou PCs équipés de carte PLC programmables sous le logiciel Siemens S7.

Les caractéristiques principales des automates programmables industriels (API) Siemens S7-400 sont les suivantes :
Il s'agit d'un matériel multiprocesseur :

- un processeur logique (bit processor)
- un processeur pour les opérations arithmétiques (word processor)
- un processeur dédicacé à la régulation de type PID
- un processseur dédicacé à la gestion des communications

Le logiciel Siemens S7 permet une programmation multi language, c'est-à-dire qu'il peut être programmé dans plusieurs langages différents, qui peuvent être même mélangés dans un même programme (mais pas dans une même sous-routine) :

- Liste d'instructions ou Intruction List (IL)
- Langage à contacts ou Ladder diagramm (CONT)
- Logigramme ou Functional Block (LOG)

Le mode séquentiel est accessible :
- soit en utilisation une programmation en GRAFCET directement
- soit en créant une séquence d'exécution.

1.1 PRINCIPE DE LA PROGRAMMATION EN S7

1.1.1 Structure du programme

La programmation structurée permet la rédaction claire et transparente de programmes. Elle permet la construction d'un programme complet à l'aide de modules qui peuvent être échangés et/ou modifiés à volonté.

Pour permettre une programmation structurée confortable, il faut prévoir plusieurs types de modules : les modules d'organisation (OB), de programmes (FB), fonctionnels (FC), de pas de séquences (SB), de données (DB).

Les modules de programmes (FC) servent à subdiviser le programme en parties fonctionnelles et/ou orientées vers le "processus".

Les modules de données (DB) contiennent des données variables, textes, valeurs de temporisations ou de comptage, résultats de calculs, etc. et sont aceesibles et actualisables à tout moment.

Les modules séquentiels (SB) sont spécialement utilisés pour effectuer des séquences selon Grafcet. Les paramètres d'entrées y seront les conditions d'avancement d'un pas de séquence et les paramètres de sorties, les ordres à exécuter lorsque ces conditions seront vérifiées.

Les modules d'organisation (OB) sont, comme leur nom l'indique, utilisés pour l'organisation interne du programme et forment ainsi un moyen puissant et essentiel pour la programmation structurée.
Ils servent par exemple au déroulement cyclique du programme principal, à l'exécution de programmes d'interruption par des fonctions d'alarmes ou de temps, ou par des fonctions diagnostic interne autant du point de vue hardware que software du système complet.

Ce dernier point est surtout un élément essentiel pour des systèmes complexes.
Ainsi, une chute de tension, une défectuosité des cartes d'entrées/sorties, un dépassement du temps de cycle, des erreurs d'adressage, etc. peuvent être détectés, signalés et la réaction du système suite à ces défauts, peut être librement programmée.

Les modules fonctionnels (FB) sont librement paramétrables spécialement conçus pour la standardisation de fonctions complexes et revenant souvent.
S'il faut commander par exemple une cinquantaine de vannes à l'aide d'un automate, on ne programmera qu'une fois ce programme de commande et de surveillance de vannes avec des paramètres symboliques dans un module fonctionnel. Ensuite, on appellera 50 fois ce module dans le programme principal et à chaque fois on y adjoindra d'autres entrées et d'autres sorties étant donné le caratère de substition des paramètres.

En résumé on distingue plusieurs types de blocs:

- Les blocs d'organisation : OB

On retiendra principalement l'OB1 qui est examiné à chaque cycle d'automate. C'est donc à partir de ce bloc que l'on fera les appels aux différents blocs de programmes.
L'OB100 et l'OB101 sont uniquement appelés aux démarrage (respectivement à chaud et à froid). On y appellera donc les blocs traitant les initialisations.

- Les fonctions : FC

C'est dans ces blocs que l'on va mettre les instructions à exécuter. La numérotation est libre (de 0 à 255). Ces blocs n'ont pas de mémoire.

- Les blocs de fonction : FB

Ces blocs sont *paramétrables*. On peut passer des données en créant des DB d'instance associés à un seul FB pour le passage de paramètres. La numérotation est libre (de 0 à 255). Ils peuvent être très utiles pour réduire le code en créant des DB d'instance associés à un seul FB avec passage de paramètres.

- Fonctions systèmes SFC, les blocs fonctionnels systèmes SFB, les blocs fonctionnels de communication CFB

1.1.2 La programmation structurée

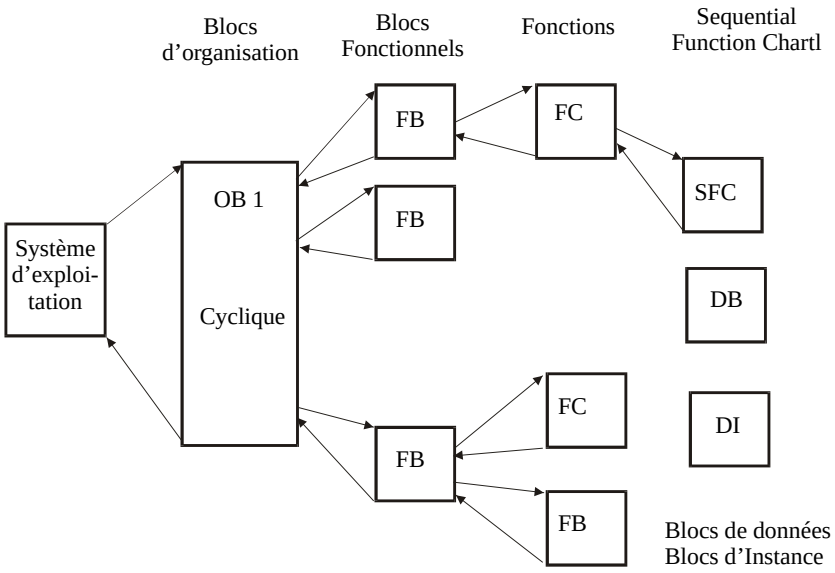


Fig. 1.1 : Architecture des programmes en S7

Pour mettre plus de clarté dans un programme, on le découpe en plusieurs sections affectées chacune à une fonction technologique. On est donc amené à programmer différents blocs (FB ou FC). L'ordre chronologique d'appel et de traitement des différents blocs est défini dans le bloc d'organisation (OB 1). Chaque bloc ainsi appelé peut lui-même contenir une instruction de saut vers un autre bloc. A la fin de l'exécution du bloc ainsi appelé, le traitement se poursuit automatiquement au lieu de départ du saut. De cette manière, il est possible d'obtenir une "imbrication" de 15 blocs.

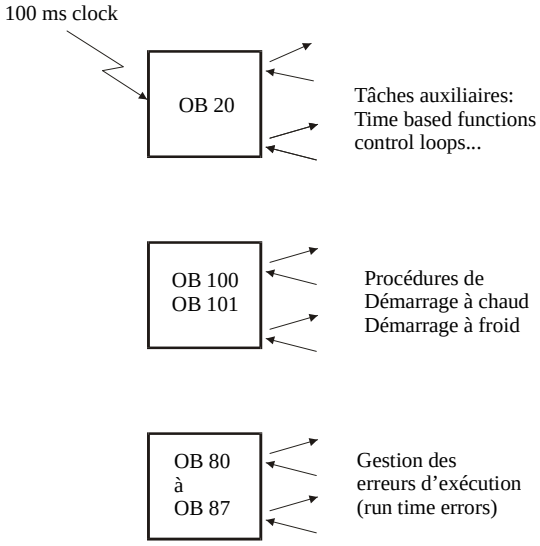


Fig 1.2. Blocs d'organisation (OB)

1.1.3 Les structure des données

- Les blocs de données: DB

Il s'agit de blocs de données (et pas des blocs d'instructions !) composés de mot de 16 bits (DW) dans lesquels on peut lire et écrire des données.

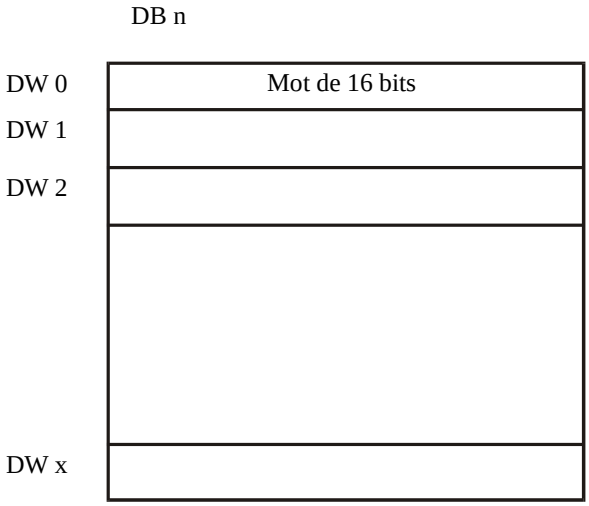


Fig. 1.3 : Bloc de données DB en Step 7

Un bloc de données ouvert reste valide jusqu'à ce qu'un autre bloc de données soit appelé ou qu'un bloc de code appelant soit terminé avec un BE ou un BEB.

Voici un exemple. Dans un programme on a:

AUF	DBn	ouvre la DBn et en fait le bloc de données courant
L	DW5	chargement du mot 5 dans la DB courante (ici DBn)
AUF	DBm	ouvre la DBm et en fait le bloc de données courant (et donc ferme la DBn précédemment ouverte)
L	DW5	chargement du mot 5 dans la DB courante (ici DBm)

1.2 TYPES DE VARIABLES

Les différents types de variables sont donnés dans le tableau qui suit. Dans le cadre de cette première approche, on fera plus particulièrement attention aux variables suivantes:

- Entrées (E)
- Sorties (A)
- Mémentos (Flag) (M)
- Temporisations (T)
- Compteurs (Z)

Bit variables

Input	E0.0 à 127.7	(#byte.#bit)
Output	A 0.0 à 127.7	
Flag	M 0.0 à 255.7	

Data	D 0.0 à 255.7
Tempo	T 0 à 127
Compteur	Z 0 à 127

Byte variables (=8 bits)

Input	EB 0 à 127
Output	AB 0 à 127
Flag	MB 0 à 255
Data (left byte)	DL 0 à 255
Data (right byte)	DR 0 à 255

Word variables (= 16 bits)

Input		EW	0 à 126
Output		AW	0 à 126
Flag	(mémo interne)	MW	0 à 254
Data		DW	0 à 255

Double word variables (=32 bits)

Input		ED	0 à 126
Output		AD	0 à 126
Flag	(mémo interne)	MD	0 à 254
Data		DD	0 à 254

Constantes

Bit	TRUE/FALSE 1/0
Integer unsigned	B#(0,0) à B#(255,255)
Integer signed	-32768 à +32767
Real	-1.175494 E -38 à 3.402823 E 38
Hexadecimal (8 bits)	B#16#0 à B#16#FF
(16 bits)	W#16#0 à W#16#FFFF
Bit pattern	2#0 à 2#11111111 11111111
ASCII characters	‘abcd ’
Time Constant (S5Time)	S5T#0ms, S5T2h46m30s
Time Constant (CPU TIME)	T#-24d20h31m23s746ms
	T#-65535ms à T#+65535ms
Counter	C#0 à C#999
Pointer	P#x.y

1.3 OPERATIONS ET INSTRUCTIONS DE BASE

1.3.1 Opérations combinatoires sur bits

Instructions communes au STEP 5 et au STEP 7

U	AND ou chargement de l’accumulateur du registre logique (RLG) si vide
UN	AND NOT
O	OR
ON	OR NOT
U(	AND sur expression entre parenthèses

O(	OR sur expression entre parenthèses
)	fin parenthèse
S	SET à ‘ 1 ’ de l’opérande (permanente)
R	RESET à ‘ 0 ’ de l’opérande (permanente)
=	assignation de l’opérande à la valeur du RLG (1 cycle)

Instructions spécifiques au STEP 7

CLR	mise à ‘ 0 ’ du RLG
SET	mise à ‘ 1 ’ du RLG
NOT	négation du registre logique
X	OU EXCLUSIF
XN	(OU NON) EXCLUSIF
FN	front montant d’une variable
FP	front descendant d’une variable

1.3.2 Instructions de base sur bytes, words et constantes

On travaille sur les accumulateurs arithmétiques (ACCU1, ACCU2)

Instructions de chargement

AUF	Ouverture DB	
L	Chargement:	ACCU1 → ACCU2, Opérande → ACCU1
T	Transfert	ACCU1 transféré dans opérande

Instructions arithmétiques

+I/+D/+R	Addition:	ACCU1 = ACCU2 + ACCU1
-I/-D/-R	Soustraction:	ACCU1 = ACCU2 - ACCU1
×I/×D/×R	Multiplication:	ACCU1 = ACCU2 × ACCU1
÷I/÷D/÷R	Division:	ACCU1 = ACCU2 ÷ ACCU1
	I si integer 16 bits, D si integer 32 bits, R si réels 32 bits	

Instructions de comparaison sur entiers et réels

Le bit de l’accumulateur logique est mis à ‘1’ si :

==D/ ==I/ ==R	ACCU2 = ACCU1
><D/ ><I/ ><R	ACCU2 ≠ ACCU1
>D/ >I/ >R	ACCU2 > ACCU1
>=D/ >=I/ >=R	ACCU2 ≥ ACCU1
<D/ <I/ <R	ACCU2 < ACCU1
<=D/ <=I/ <=R	ACCU2 ≤ ACCU1

avec D pour entier 32 bits, I pour entiers 16 bits, R pour réels 32 bits

Opérations combinatoires sur mots

OD/OW	OU sur MOT DOUBLE (32 bits) ou MOT (16 bits)
UD/UW	ET sur MOT DOUBLE (32 bits) ou MOT (16 bits)
XOD/XOW	OU EXCLUSIF sur MOT DOUBLE ou MOT

1.3.3 Opération d'appel des blocs

Appel des blocs

UC	FC/FB	Saut inconditionnel, appel du bloc
CC	FC/FB	Saut conditionnel, appel du bloc si bit accumulateur RLG =1
CALL	FC/FB/SFC/SFB	saut avec transfert de paramètres

exemples:
call FCn
call SFCn
call FBn1,DBn2
call SFBn1,DBn2

Le bloc DBn2 est appelé bloc d'instance, car sa valeur est définie lors de l'appel.

Instructions de fin de bloc

BE	Fin de bloc (requis)
BEB	Saut conditionnel (si RLG=1) à la fin du bloc
BEA	Saut inconditionnel à la fin du bloc

Opérations de saut

SPA	saut (JUMP) inconditionnel
SPB	saut (JUMP) conditionnel si RLG=1
LOOP	boucle de programme

1.3.4 Utilisation des tempos (timers)

Quelques types de tempos:

SI: Temporisation sous forme d'impulsion.

La valeur logique de cette temporisation passe à 1 durant le temps de la temporisation et retombe ensuite à 0 lorque celle-ci est terminée.
Cette opération démarre la temporisation indiquée si le résultat logique RLG passe de 0 à 1. La durée programmée s'écoule tant que le RLG égale 1. Si le RLG passe à 0 avant que cette durée n'ait expiré, la temporisation s'arrête. Le démarrage de la temporisation ne s'exécute que si la valeur de temps et la base de temps figure en format DCB dans l'accumulateur 1-L.

SS: Temporisation sous forme de retard à la montée mémorisé

La valeur logique de cette temporisation reste à 0 pendant que la temporisation s'écoule. Elle monte ensuite à 1 lorsque le temps est écoulé.
Cette opération démarre la temporisation indiquée si le résultat logique RLG passe de 0 à 1. La durée programmée s'écoule même si le RLG passe entretemps à 0. Si le RLG passe de 0 à 1 avant que cette durée n'ait expiré, la temporisation redémarre. Le démarrage de la temporisation ne s'exécute que si la valeur de temps et la base de temps figure en format DCB dans l'accumulateur 1-L pour que le temps redémarre.

SV: Temporisation sous forme d'impulsion prolongée

La valeur logique de cette temporisation passe à 1 durant le temps de la temporisation et retombe ensuite à 0 lorque celle-ci est terminée.
Cette opération démarre la temporisation indiquée si le résultat logique RLG passe de 0 à 1. La durée programmée s'écoule même si le RLG passe entretemps à 0. Si le RLG passe de 0 à 1 avant que cette durée n'ait expiré, la temporisation redémarre. Le démarrage de la temporisation ne s'exécute que si la valeur de temps et la base de temps figure en format DCB dans l'accumulateur 1-L pour que le temps redémarre.

SE: Temporisation sous forme de retard à la montée

La valeur logique de cette temporisation reste à 0 pendant que la temporisation s'écoule. Elle monte ensuite à 1 lorsque le temps est écoulé.
Cette opération démarre la temporisation indiquée si le résultat logique RLG passe de 0 à 1. Si le RLG passe à 0 avant que cette durée n'ait expiré, la temporisation s'arrête. Le démarrage de la temporisation ne s'exécute que si la valeur de temps et la base de temps figure en format DCB dans l'accumulateur 1-L pour que le temps redémarre.

SA: Temporisation sous forme de retard à la retombée

La valeur logique de cette temporisation passe à 1 durant le temps de la temporisation et retombe ensuite à 0 lorque celle-ci est terminée.
Cette opération démarre la temporisation indiquée si le résultat logique RLG passe de 0 à 1. La durée programmée s'écoule tant que le RLG reste à 0. Si le RLG passe à 1 avant que la durée n'ait expiré, la temporisation s'arrête. La valeur de temps et la base de temps doivent figurer en format DCB dans l'accumulateur 1-L pour que la temporisation démarre.

Exemple d'utilisation d'une temporisation
Départ:

```
U     E/A/M
L     S5T#3S
SS    Tn
```

Utilisation:

```
U/O/UN/ON   Tn
```

1.3.5 Mode d’adressage direct et indirect

On peut adresser indifféremment les variables soit par leur adresse directe ou bien par un pointeur. La valeur du pointeur peut être définie de manière dynamique. Illustrons le principe sur un exemple.

Mode direct:

```
L     4
L     MW7
```

Mode indirect:

```
L     5
T     MW2
L     T[MW2]      équivalent à   L     T5
```

```
L     P#8.7
L     MD2
U     E[MD2]      équivalent à   U E8.7
```

1.3.6 Nouveaux types de variables

Array: ARRAY[x1..x2,y1..y2,z1..z2]

Structure:
```
STRUCT
:
:
END_STRUCT

<name-struct.name-var>
```

1.3.7 Passage de données à un bloc fonctionnel

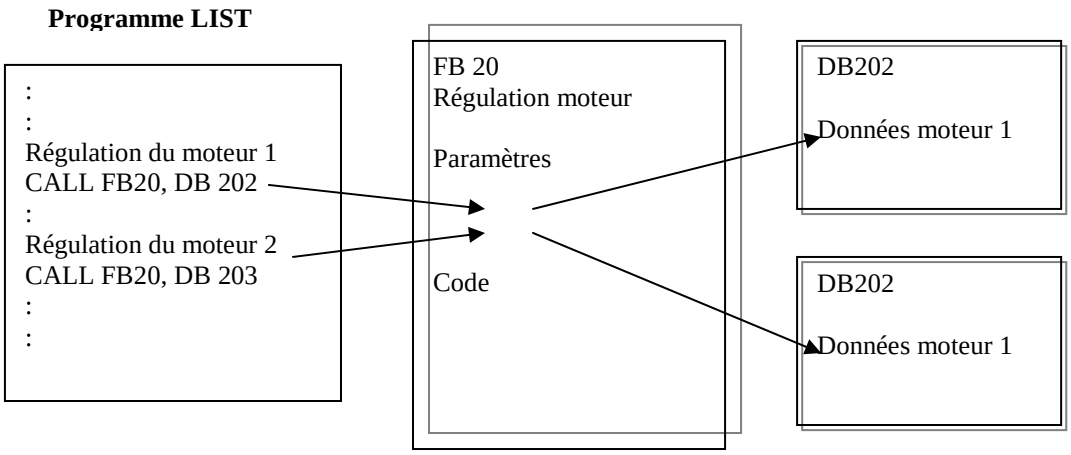


Fig 1.4: Exemple de passage des données à un bloc fonctionnel en Siemens S7

1.4 TRANSPOSITION D'UN GRAFCET EN STEP 7

1.4.1 Principe de la transposition d'un Grafcet en Liste d'Instructions sous Siemens S7

En partant d'une application séquentielle écrite en GRAFCET, on transpose de manière systématique en STEP 7 de la façon suivante. On divise le programme en trois parties:

1. *Calcul des réceptivités.* On associe un bit interne (mémento) à chaque réceptivité. Celui-ci est mis à 1 si la réceptivité est vraie.

2. *Évolution des étapes.* On utilise des bits d'étapes. A chaque étape est associé un bit interne qui est mis à 1 quand l'étape est active. On passe à l'étape suivante quand l'étape qui précède est active (validation de la transition) et que la réceptivité est vraie, ce qui correspond à mettre le bit d'étape suivante à 1 (set) et celui de l'étape précédente à 0 (reset).

3. *Actions associées aux étapes.* Il suffit d'imposer comme condition d'activation d'une action le bit correspondant à l'étape dans laquelle cette action doit être exécutée.

1.4.2 Exemple:

Grafcet de l'exemple

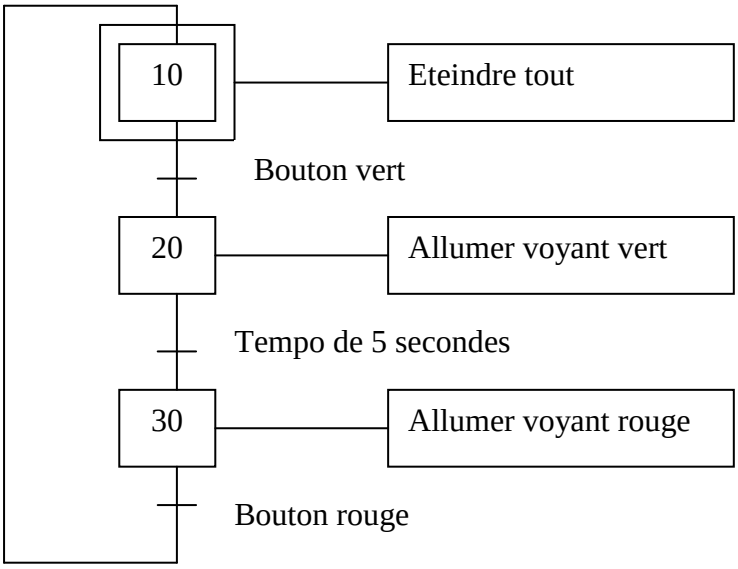


Fig1.5 Grafcet de l'exemple

Listing de l'exemple

OB100

```
:CALL FC 1
:BE
```

OB101

```
:CALL FC 1
:BE
```

FC1 **activation de l'étape initiale**

```
:O      M 0.0
:ON     M0.0
:S      M 10.0
:BE
```

OB1 **appel des sous routines**

```
:CALL FC 2
:CALL FC 3
:CALL FC 4
:BE
```

FC2 **calcul des réceptivités**

```
:U      E 0.4
:=      M 10.1
:***

:UN     T 1
:=      M 20.1
:***

:UN     E 0.5
:=      M 30.1
:BE
```

FC3 **évolution du grafcet**

```
:U      M 10.0
:U      M 10.1
:S      M 20.0
:R      M 10.0
:BEB
:***

:U      M 20.0
:U      M 20.1
:S      M 30.0
:R      M 20.0
:BEB
:***

:U      M 30.0
:U      M 30.1
:S      M 10.0
:R      M 30.0
:BE
```

FC4 **actions associées aux étapes**

```
:U      M 20.0
:L      S5T# 5s
:SV     T1
:***

:U      M 20.0
:=      A0.2
:U      M 30.0
:=      A 0.6
:BE
```

Remarques.
Lorsque l'on veut traduire du GRAFCET en STEP7, il est très important de bien *structurer* son programme, sinon celui-ci devient très vite lourd et impossible à débugger. On utilisera par exemple un FC pour calculer toutes les réceptivités, un autre pour la mise à jour des bits d'étapes et un troisième FC pour exécuter les actions proprement dites. Également pour une question de clarté et quand le nombre d'étapes le permet, on utilise le mémento dont le numéro correspond à celui de l'étape.

Il faut également faire attention à *ne pas franchir plusieurs étapes d'un même grafcet pendant un seul cycle d'automate*. Pour cela, on procède de la façon suivante.

On effectue un **saut en fin de bloc BEB** (dans le bloc gérant l'évolution du grafcet) quand la transition à été faite. Exemple: Voir listing ci-avant.

Pour le bloc où sont exécutées les actions, on **trie par action.** On parle de combinatoire des actions. On utilise un segment par action, dans lequel on affecte le résultat de l'équation booléenne des étapes dans lesquelles cette action est exécutée, à la sortie en question. Cette méthode permet de voir directement à quelles étapes telle sortie est activée et est donc intéressante pour le débuggage.

1.5 UTILISATION DE LA CONSOLE

Les automates de la série 7 se programment grâce au logiciel STEP 7 sous Windows (95, 98, NT ou 200). La liaison entre la console et l'automate se fait au moyen du réseau MPI.

Lancement du logiciel de programmation:
Il faut lancer le SIMATIC Manager.

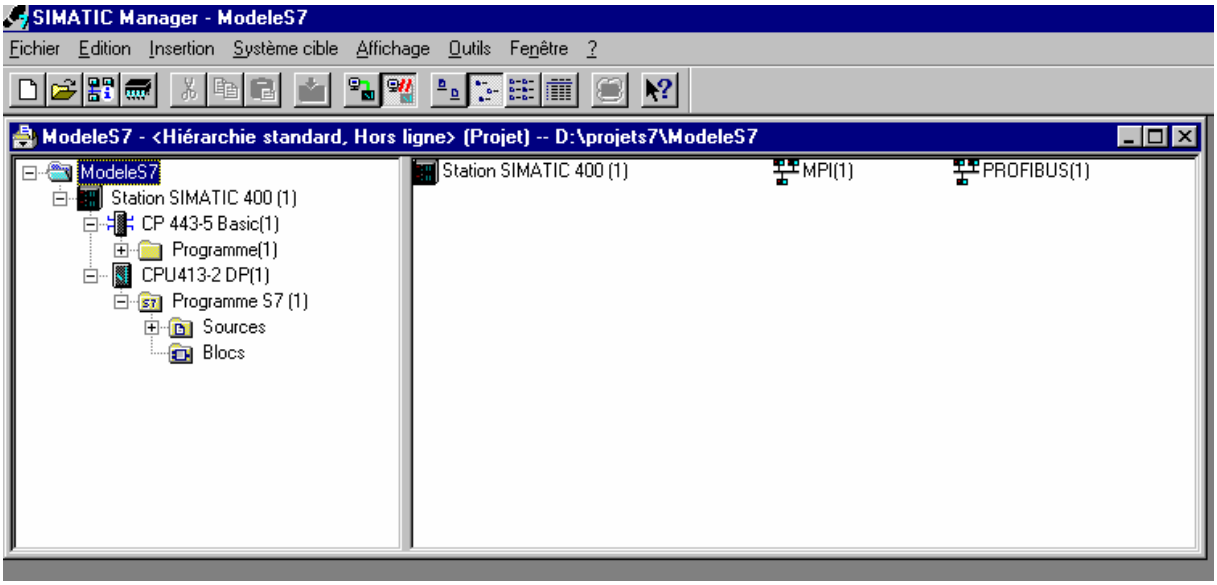


Fig 1.6: Simatic S7 manager

Les programmes sont mémorisés et regroupés en PROJETS. Dans un projet, il y a plusieurs

niveaux:

- Au niveau 1, on trouve le nom de projet (ex : balle97). Le projet contient d'une part les stations connectées (Station SIMATIC 400 1) et le ou les réseaux configurés (réseau MPI (1)).

- Au niveau 2, on accède à la définition de la station : le type de CPU (CPU 413-2 DP) et la station de configuration (Matériel).

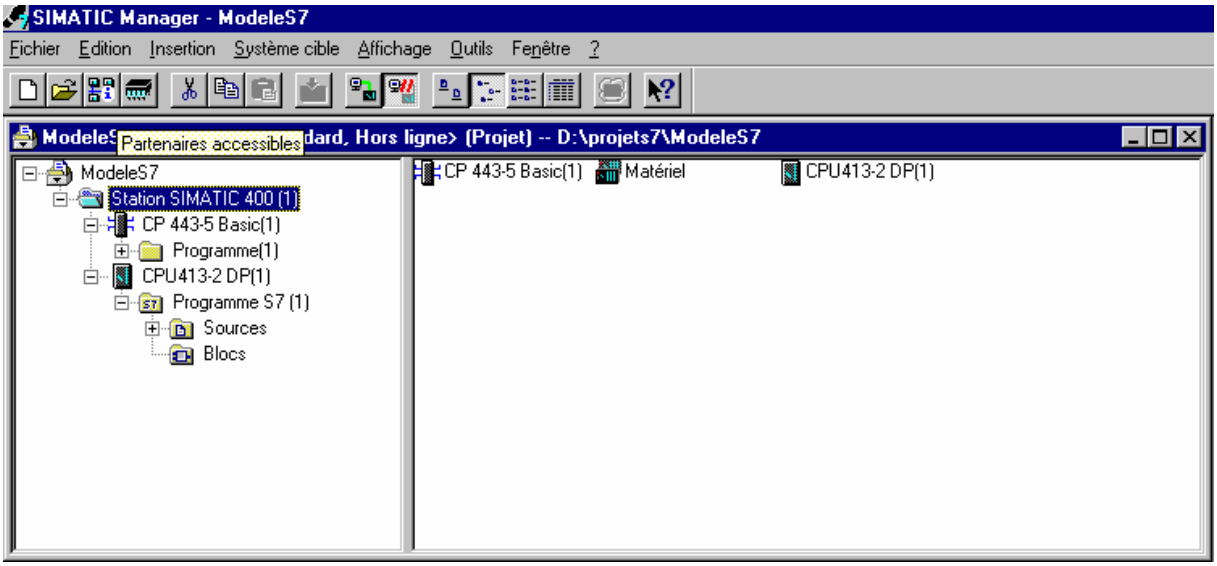


Fig 1.7: Niveau 2, configuration de la station

- Au niveau 3, on accède aux différents programmes (Programmes S7) et à la table des connexions définies pour le réseau (Liaisons).

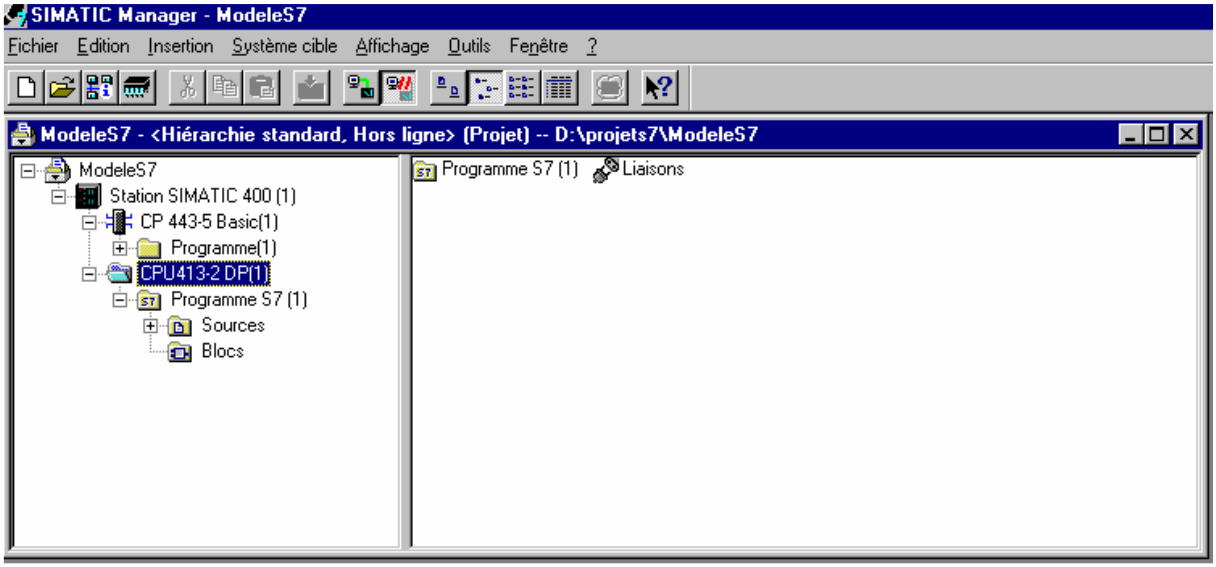


Fig 1.8: Niveau 3, niveau de la CPU

- Au niveau 4, on accède à

- aux sources externes éventuelles (répertoire Sources) comme des fichiers textes en listes d'instructions, d'anciens programmes S5 convertis...
- à la table des mnémoniques (fichier Mnémoniques)

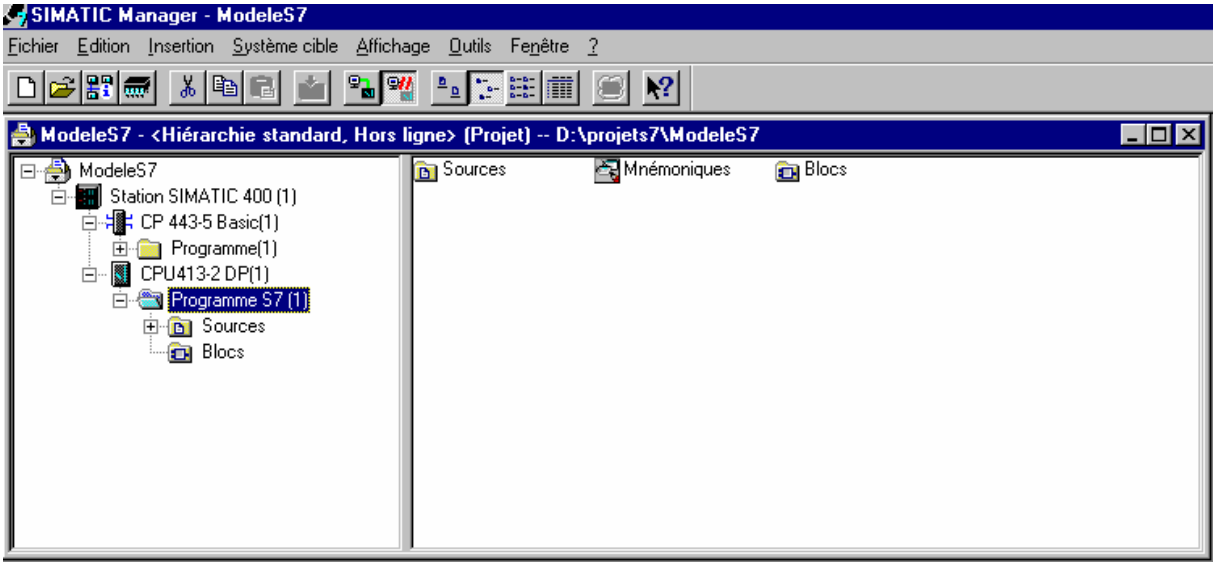


Fig 1.9: Niveau 4, niveau des programmes

- Au niveau 5, on accède au programme proprement dit (blocs d'instructions OBs, FCs...), tables de variables (VAT), types données utilisateur (UDT), blocs de données (DBs)...

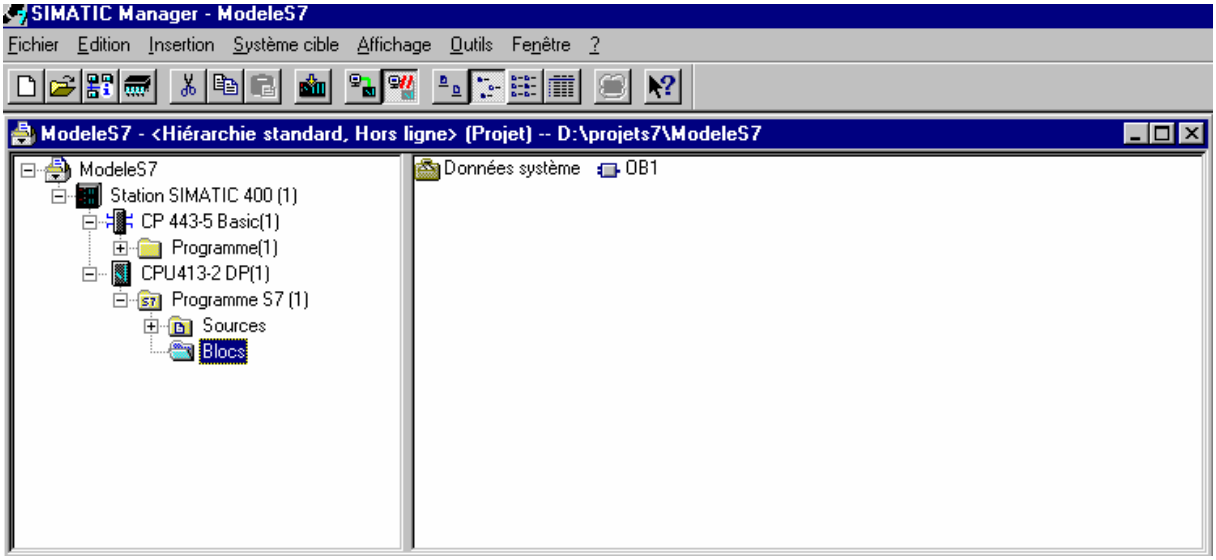


Fig 1.10: Niveau 5, les blocs de programme

A ce niveau pour créer un nouveau bloc, il faut choisir insérer dans la barre des menus du SIMATIC Manager et sélectionner blocs S7.

Si l'on crée un nouveau projet, il ne faut pas oublier de définir la configuration matérielle du niveau 2 qui permet de configurer l'automate lors du transfert du programme dans celui-ci. Cette configuration doit correspondre évidemment à la configuration matérielle réelle qui consiste à détailler la position et le type des différentes cartes enfichées sur le bus fond de panier.

Configuration de la station
La configuration consiste à sélectionner un emplacement libre et à choisir un équipement dans le catalogue du matériel. En ce qui concerne la configuration de la station S7-400 que nous avons au laboratoire, on se référera à l'annexe située en fin de chapitre.

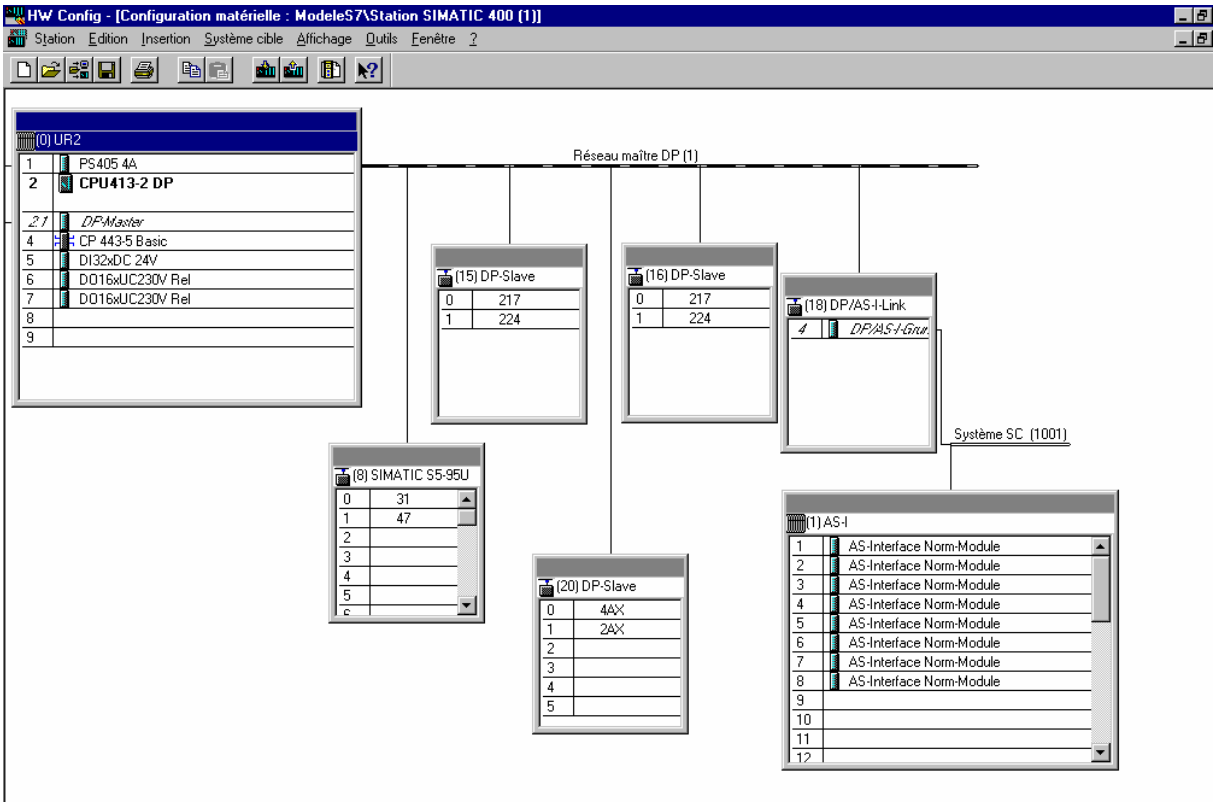


Fig 1.11: Menu de la configuration de la station

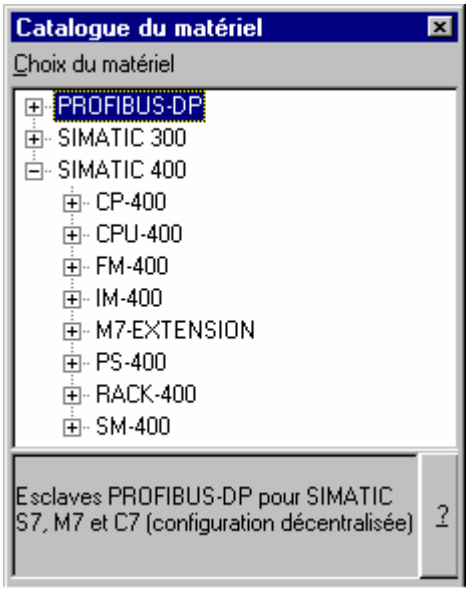


Fig 1.12: Catalogue du matériel

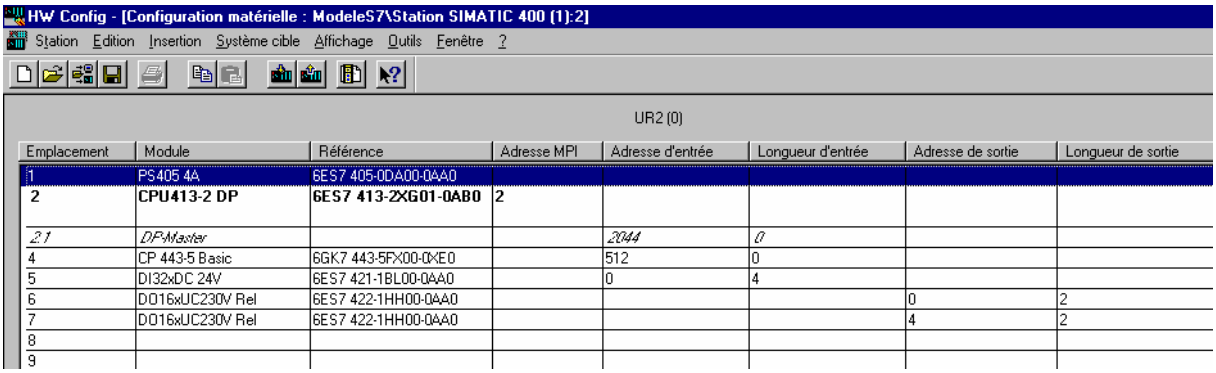


Fig 1.13: Configuration matérielle de la station S7-400 du laboratoire

Édition

On peut maintenant passer en mode édition dans le fichier programme (niveau 5). Il faut sélectionner un bloc et double-cliquer sur l'icône du fichier, du bloc de données, etc. On accède alors a l'éditeur de programme.

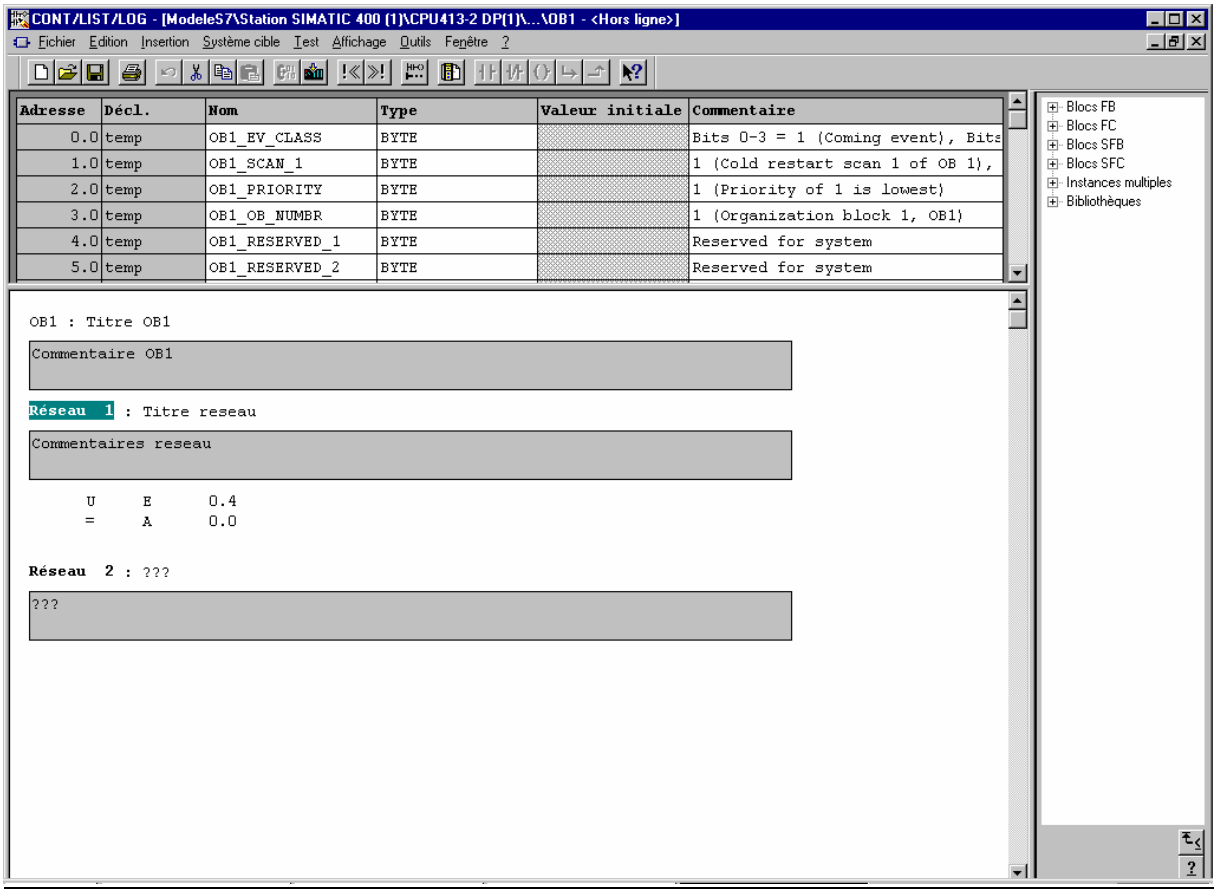


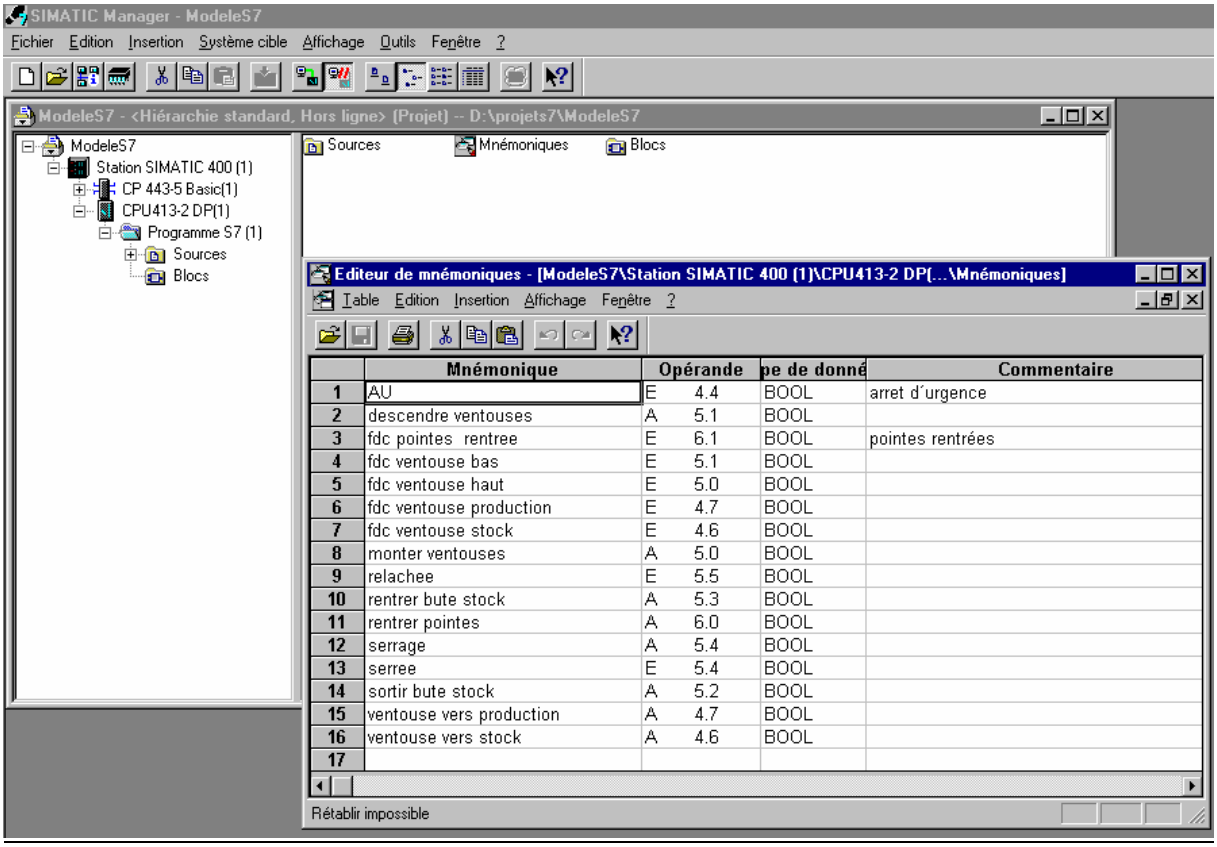
Fig 1.14: Editeur de bloc de programme

La partie supérieure contient des informations systèmes sur le bloc et sert aussi à définir les variables à passer comme paramètres pour les FB.

La partie inférieure commence par le nom du bloc (OB1) et un nom éventuel ainsi qu'un commentaire.

Le bloc est divisé en réseaux, ce qui permet de clarifier le programme et d'insérer des commentaires. Il est possible d'ajouter de nouveaux réseaux en faisant *insertion / réseau*

Edition des mnémoniques
Les mnémoniques sont des noms que l'on peut donner aux variables afin de facilliter la programmation en affectant des noms parlant plus faciles à retenir. Ils améliorent la lisibilité du programme et servent également de documentation. Les mnémoniques se définissent au niveau 4 en double-cliquant sur l'icône "Mnémonique". On accède alors à l'éditeur des mnémoniques.



	Mnémonique	Opérande	pe de donné	Commentaire
1	AU	E 4.4	BOOL	arret d´urgence
2	descendre ventouses	A 5.1	BOOL	
3	fdc pointes rentree	E 6.1	BOOL	pointes rentrées
4	fdc ventouse bas	E 5.1	BOOL	
5	fdc ventouse haut	E 5.0	BOOL	
6	fdc ventouse production	E 4.7	BOOL	
7	fdc ventouse stock	E 4.6	BOOL	
8	monter ventouses	A 5.0	BOOL	
9	relachee	E 5.5	BOOL	
10	rentrer bute stock	A 5.3	BOOL	
11	rentrer pointes	A 6.0	BOOL	
12	serrage	A 5.4	BOOL	
13	serree	E 5.4	BOOL	
14	sortir bute stock	A 5.2	BOOL	
15	ventouse vers production	A 4.7	BOOL	
16	ventouse vers stock	A 4.6	BOOL	
17				

Fig 1.15: Edition des mnémoniques

Pour utiliser les mnémoniques à la place des adresses physiques dans un programme, il faut les encadrer d'apostrophes U E0.4 devient U ‘bpvert'

Transfert entre la console et l'automate
Une fois le Programme terminé hors-ligne (‘offline’), il faut le transférer dans l'automate. Pour cela il faut d'abord arrêter l'automate. Dans le programme 'en ligne' et au niveau de la CPU, on va dans le menu *système cible*, puis dans *Etat de fonctionnement de la station* (en raccourci Ctrl+I). On arrête alors la station. On peut alors commencer à télécharger le programme présent sur la console PC dans l'automate. On peut y arriver de plusieurs manières:

- soit en utilisant le menu *Système cible*, (accessible aussi bien à partir du SIMATIC Manager qu'à partir de l'éditeur de blocs),
- soit de manière équivalent via l'icône de commande ‘*Charger*’ dans la barre de menu général
- soit via l'icône '*Charger*' dans l'éditeur de bloc de programme.

Dans ce dernier cas, on ne peut charger que le module qui est édité, tandis que dans les deux premiers cas on peut charger soit un module en venant se positionner sur l'icône correspondante, soit tous les modules en se positionnant sur le répertoire.

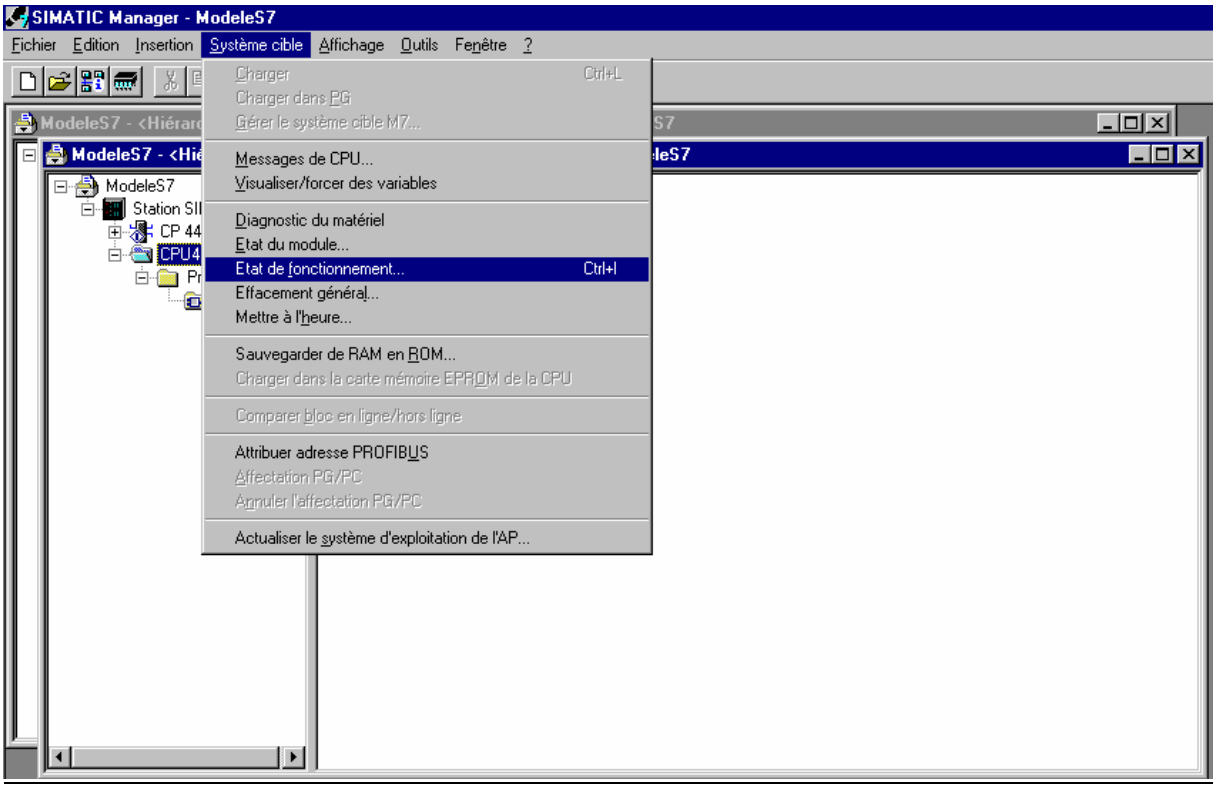


Fig 1.16: Accès à l'état de fonctionnement de la station

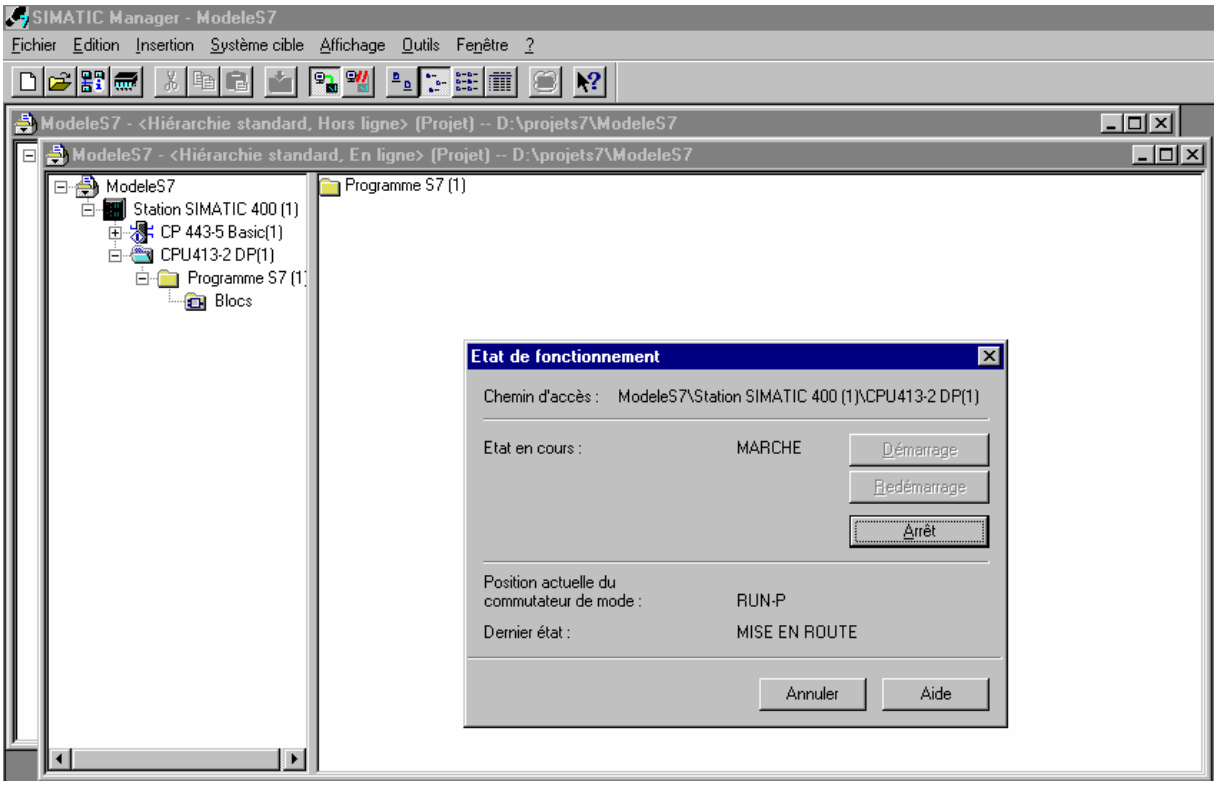


Fig 1.17: Arrêt de l'automate

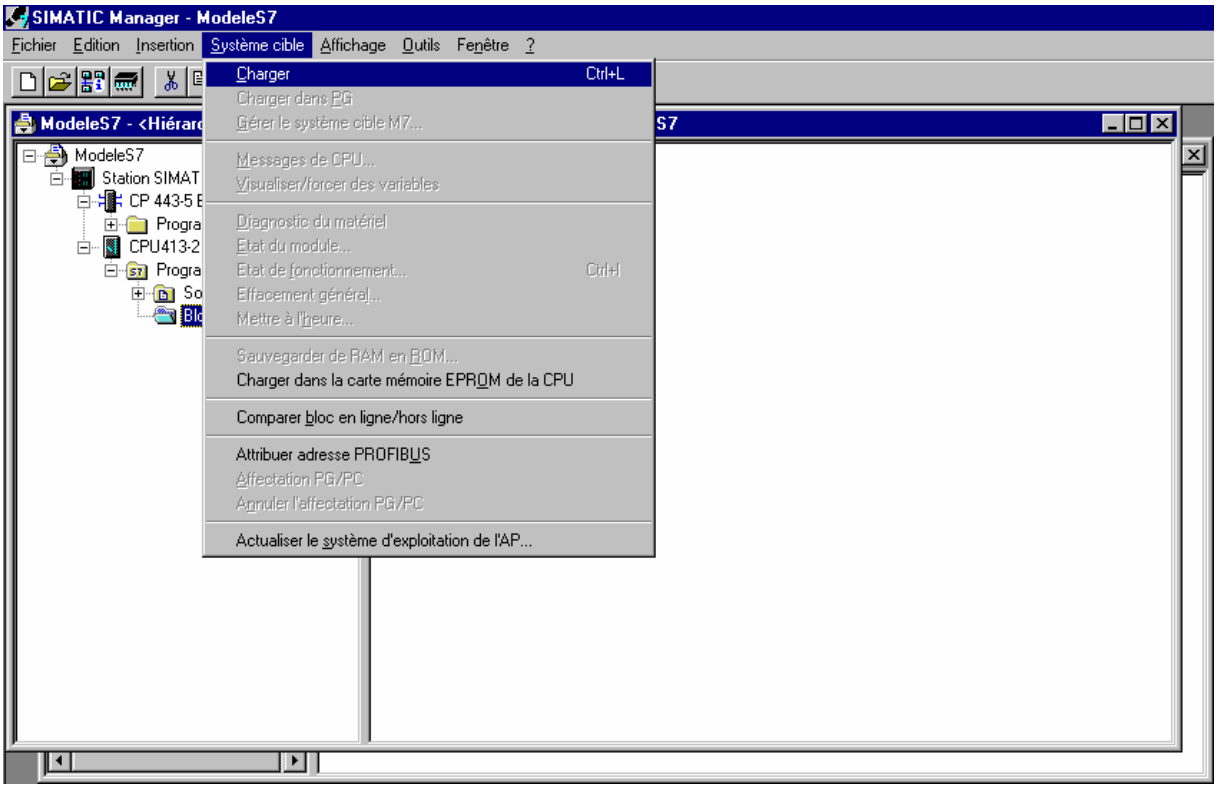


Fig 1.18: Charger le ou les modules dans l'automate

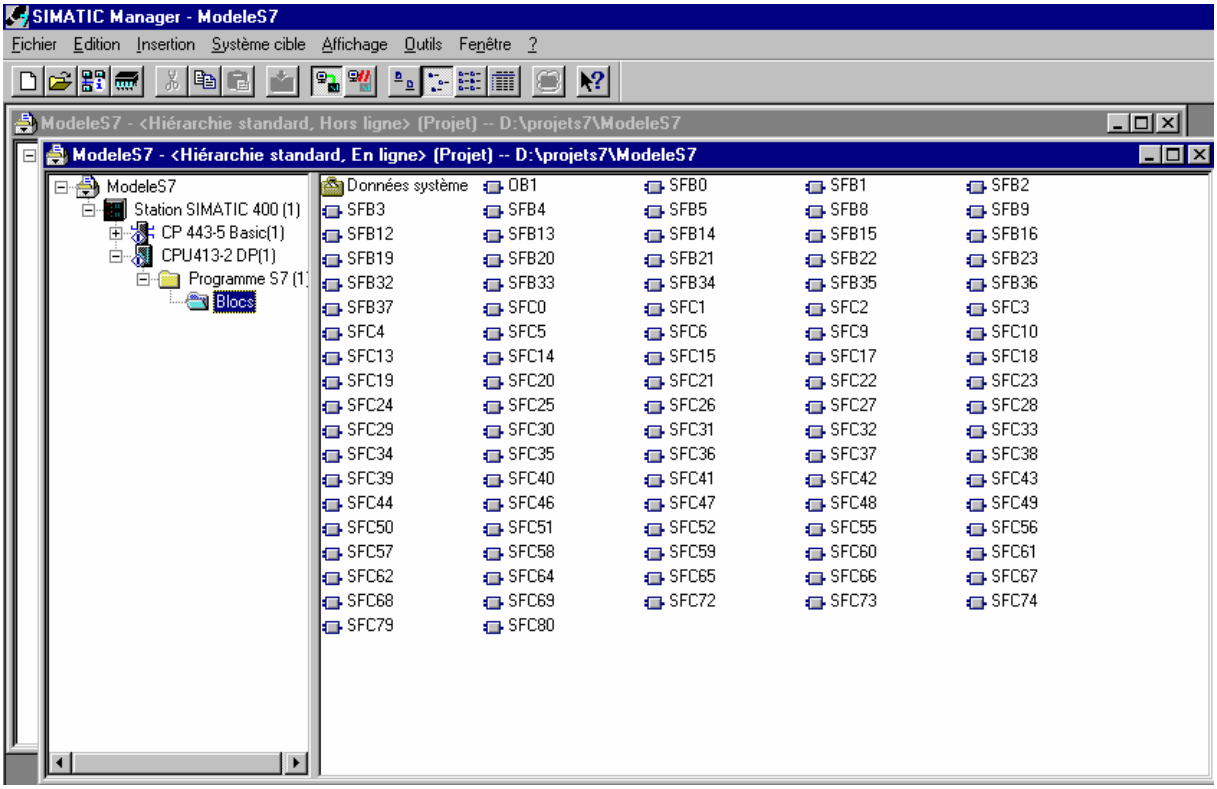


Fig 1.19: Etat du projet 'on line' après chargement

Après chargement, le système cible, c'est-à-dire l'automate contient notre ou nos modules plus toute une série de modules systèmes (SFC ou SFB) chargés spontanément dans l'automate pour gérer la configuration matérielle, les communications entre les différentes cartes,

modules, etc.

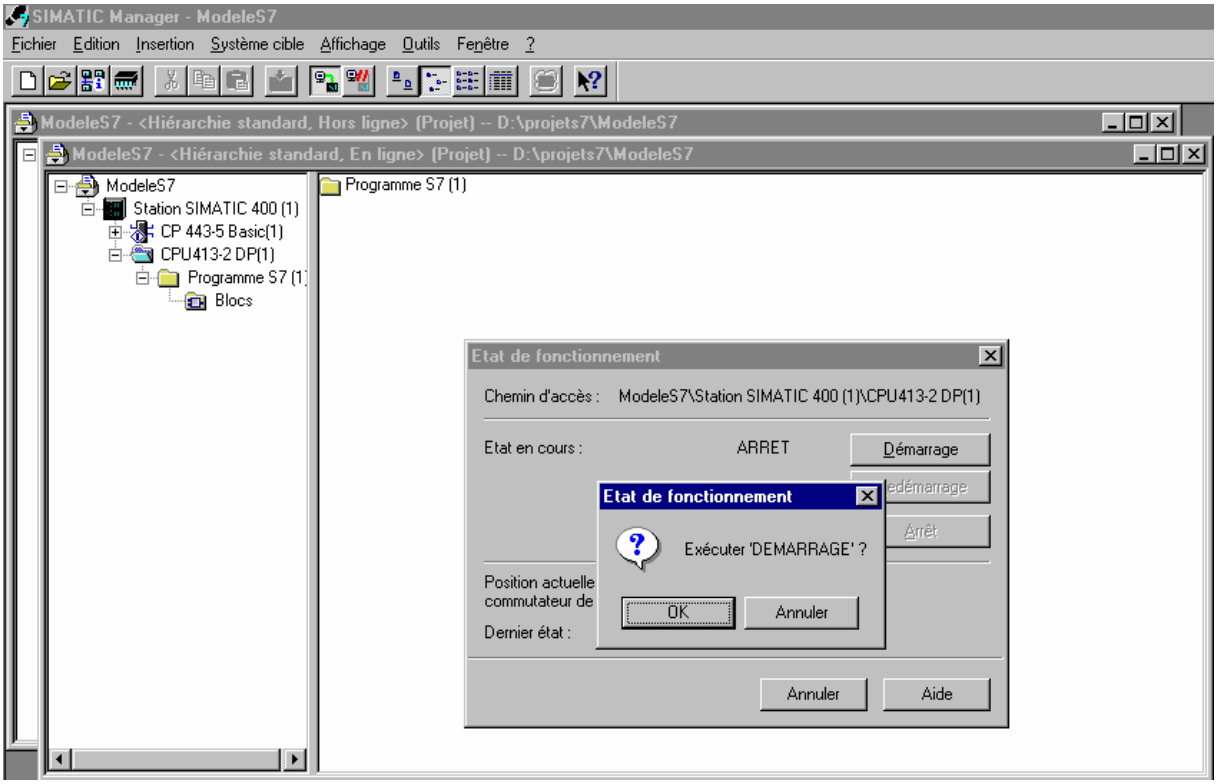


Fig 1.20: Redémarrage (à chaud) de l'automate

On peut alors redémarrer l'automate pour effectuer les tests. On parle dans ce cas d'un démarrage à chaud.

Dans le menu '*Système Cible*' on a encore d'autres fonctions intéressantes:
- Etat du module
- Etat de fonctionnement
- Effacement général
- Messages de la CPU
- Visualiser / Forcer les variables.

L'utilisation de ceux-ci est expliquée au points suivants.

Etat du module
Cet outil permet d'obtenir des informations concernant le CPU, la taille de la mémoire libre et occupée, les durées de temps de cycle, les fonctions d'horodatage...

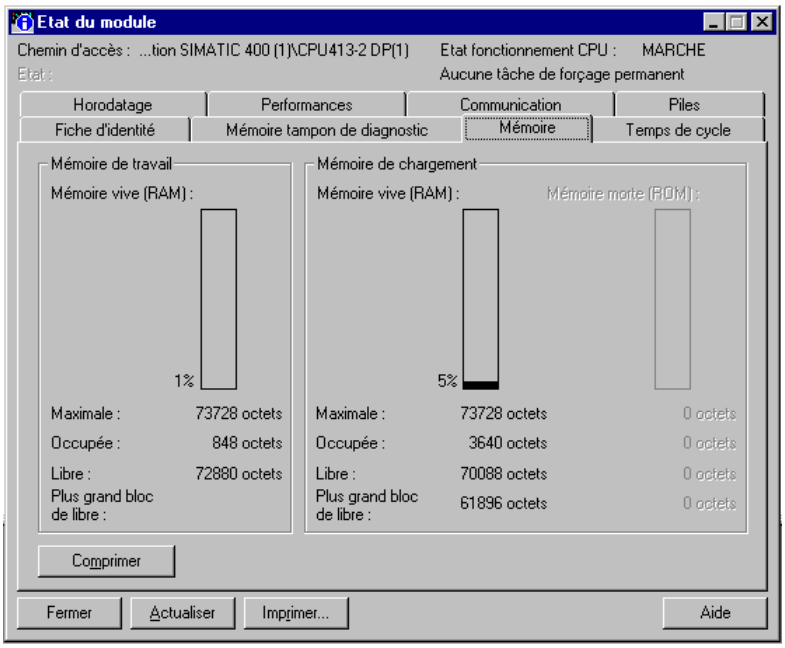


Fig 1.21: Etat du module: onglet mémoire

Dans l'onglet mémoire, on offre la possibilité de comprimer la mémoire pour réduire l'espace utilisé. On peut également surveiller le temps de cycle de l'automate dans l'onglet correspondant.

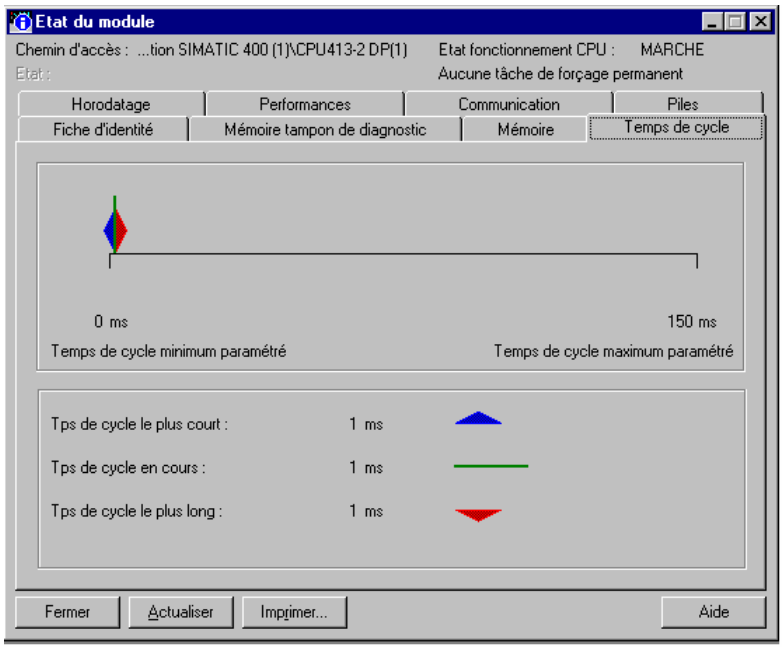


Fig 1.22: Etat du module: onglet temps de cycle

Etat de fonctionnement
Comme cela a été expliqué lors de la manœuvre de transfert de fichier, le menu *Etat de fonctionnement* permet d'arrêter et de redémarrer l'automate.

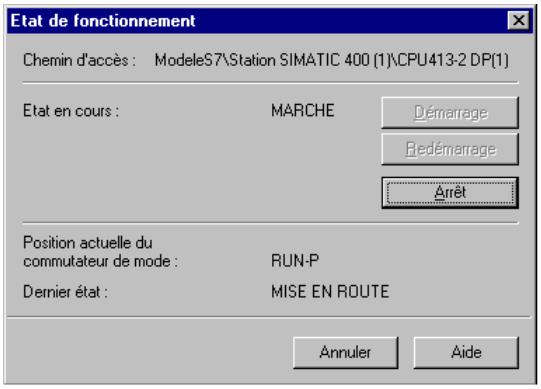


Fig 1.23: Démarrage / arrêt de l'automate

Visualisation et forçage des variables

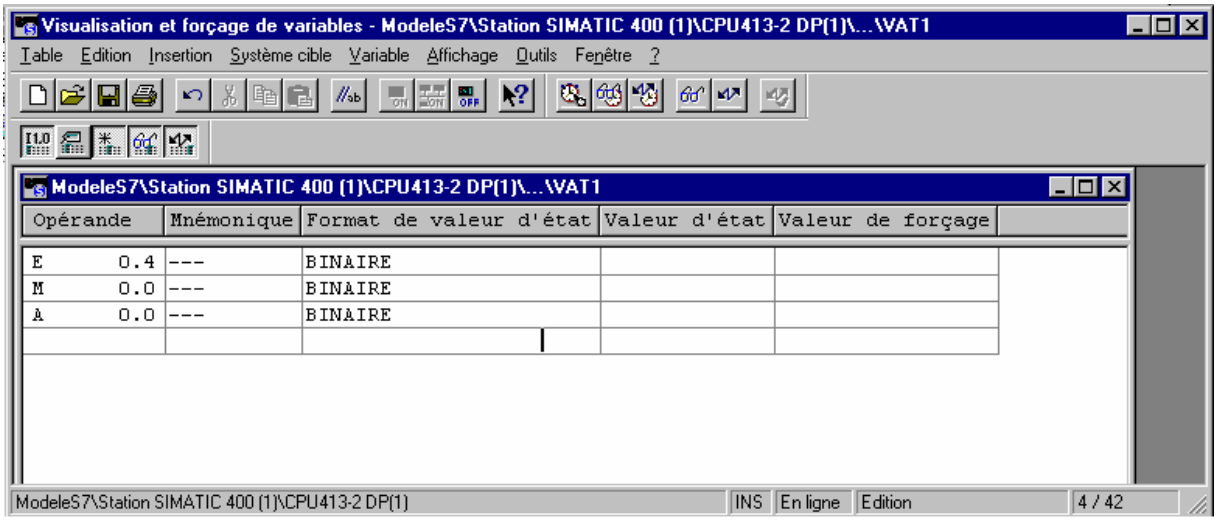


Fig 1.24: Visualisation et forçage des variables

Cet outil permet de visualiser en ligne (online) la valeur des différentes variables (entrées, sorties...) et de forcer la valeur des sorties et des variables internes. La table peut être sauvegardée avec un nom commençant impérativement par VAT. Compte tenu d'un problème connu dans le programme STEP7, il faut faire attention au répertoire où on sauve la table (nom de projet et niveau!), car celui-ci sauve souvent la table dans le dernier projet visité… mais peut-être pas par vous!

Visualisation dynamique du programme.
La visualisation dynamique du programme est accessible à partir de l'éditeur de blocs : *Menu / Test / Visualisation.*

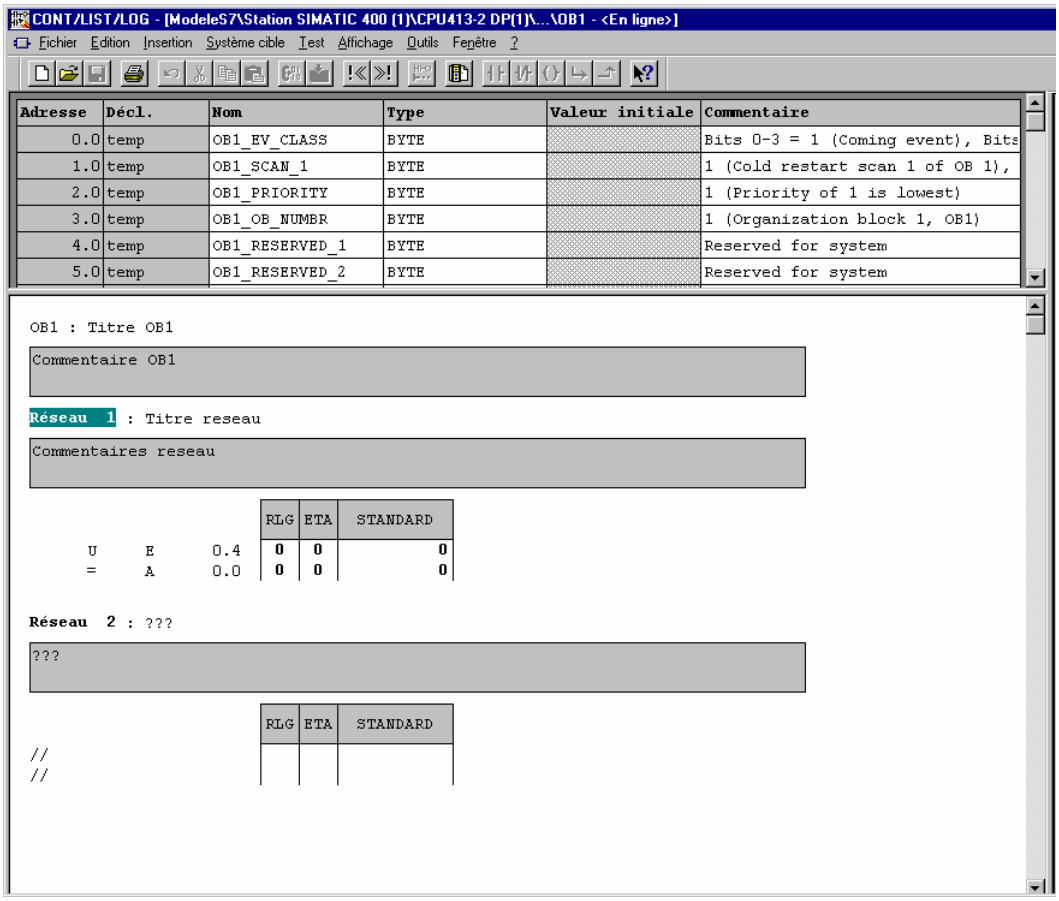


Fig 1.25: Visualisation dynamique des variables

Données de références

A partir du menu OUTILS, on peut afficher les *données de référence :*

- la structure du programme,
- la table des références croisées,
- les variables utilisées...

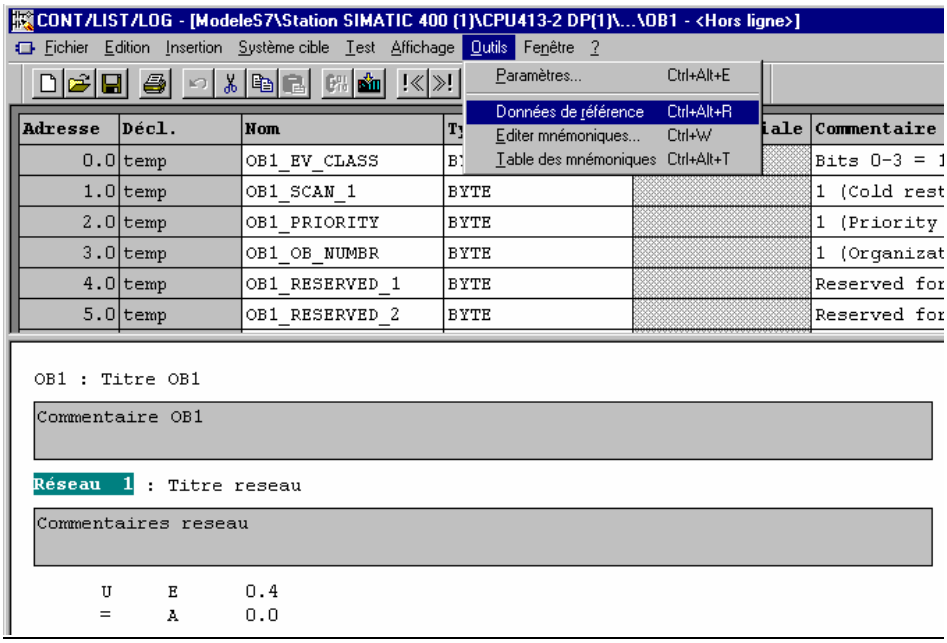


Fig 1.25: Visualisation dynamique des variables

1.6 ANNEXE : LISTE DES ENTREES ET SORTIES DES AUTOMATES

Annexe 1.6.1 : Configuration de l'automate S7-400:

Emplacement	**Module**	**N° de référence**	**Type**
0	UR2 (chassis universel)	6ES7 400-1JA00-0AA0	RACK-400
1	PS-405 4A (alimentation)	6FS7 405-0DA00-0AA0	PS-400
2	CPU4I3-2DP (CPU + DP master)	6ES7 413-2xG01-0AB0	CPU-400
	[emplacement implicite du DP master]		
4	CP443-5A (CP pour Profibus FMS)	60K7 443-5FX00-0XE0	CP-400
5	DI32xDC 24V (module d'entrées)	6ES7 421-lBL00-0AA0	SM-400 → DJ-400
6	DO16xUC23OV Rel (module de sorties)	6ES7 422-1LIH00-0AA0	SM-400→ DO-400
7			
8			
9			

Annexe 1.6.2 : Liste des entrées et sorties de l'automate S7-400:

Adresse physique	Signification
Sorties	
A0.0	voyant rouge 1
A0.1	voyant orange 1
A0.2	voyant vert 1
A0.3	voyant rouge 2
A0.4	voyant orange 2
A0.5	voyant vert 2
Entrées	
E0.0	sélecteur 1 à gauche
E0.1	sélecteur 1 à droite
E0.2	sélecteur 2 à gauche
E0.3	sélecteur 2 à droite
E0.4	bouton poussoir vert 1
E0.5	bouton poussoir rouge
E0.6	bouton poussoir vert 2

Annexe 1.6.3 : Liste des entrées et sorties du Win LC "Poste Intercalaire"

Entrées	
Adresse physique	**Signification**
E0.0	Sélecteur 1 à gauche
E0.1	Sélecteur 1 à droite
E0.2	Sélecteur 2 à gauche
E0.3	Sélecteur 2 à droite
E0.4	Bouton vert 1 armoire (NO)
E0.5	Bouton rouge armoire (NF)
E0.6	Bouton vert 2 armoire (NO)
Sorties	
Adresse physique	**Signification**
A0.0	Voyant rouge 1 armoire
A0.1	Voyant orange 1 armoire
A0.2	Voyant vert 1 armoire
A0.3	Voyant rouge 2 armoire
A0.4	Voyant orange 2 armoire
A0.5	Voyant vert 2 armoire

Annexe 1.6.4 : Liste des entrées et sorties du Win LC "Poste Balles"

Entrées	
Adresse physique	**Signification**
E3.0	Bouton vert
E3.1	Bouton rouge armoire (NF)
E3.2	Sélecteur à droite
E3.3	Sélecteur à gauche
Sorties	
Adresse physique	**Signification**
A3.0	Voyant rouge
A3.1	Voyant orange
A3.2	Voyant vert

Annexe 1.6.5 : Liste des entrées et sorties du Win LC "Poste Robot"

Carte de sorties directes S95U	A2.0	Voyant rouge armoire
	A2.1	Voyant orange armoire
	A2.2	Voyant vert armoire
	A2.3	Non connecté
	A2.4	Non connecté
	A2.5	Non connecté
	A2.6	Non connecté
	A2.7	Non connecté
Carte d'entrées directes S95U	E2.0	Bouton vert armoire
	E2.1	Sélecteur à gauche
	E2.2	Sélecteur à droite
	E2.3	Bouton rouge armoire
	E2.4	Bouton d'arrêt d'urgence armoire
	E2.5	Non connecté
	E2.6	Non connecté
	E2.7	Non connecté

2. PROGRAMMATION DES AUTOMATES SIEMENS S5

Le laboratoire de robotique et automatisation dispose de plusieurs automates de type S5 programmable en STEP 5. Il s'agit d'un automate S5-135U et d'un automate S5-95U.

2.1 CARACTÉRISTIQUES PRINCIPALES DES PLC SERIE 5

Il s'agit d'un matériel multiprocesseur :

- un processeur logique (bit processor)
- un processeur pour les opérations arithmétiques (word processor)
- un processeur dédicacé à la régulation de type PID
- un processseur dédicacé à la gestion des communications (communication processor, ici ethernet)
- un processeur pour le contrôle d'axe (axis control processor)
- un processeur pour l'écran couleur (color screen processor)

On dispose également de d'une capacité multi langage

- Intruction List (IL) ou liste d'instructions, relativement proche du langage machine
- Ladder ou langage à contacts (CONT)
- Functional Block ou blocs fonctionnels (LOG)

La réalisation de la programmation en mode séquentiel peut se faire

- en utilisant une programmation GRAFCET directement
- en utilisant un mode d'exécution séquentiel.

2.2 STRUCTURE DES PROGRAMMES ET DES DONNEES

2.2.1 La programmation structurée en S5

La structuration des programmes et des données est assez semblable à celle existant sur les matériels de la génération S7, mais des différences significatives (et parfois déroutantes) sont à observer.

La structure du programme est schématisée à la figure Fig 2.1. La programmation est organisée ici sous forme de *blocs.* On distingue plusieurs types de blocs. On y distingue:

•Blocs d'organisation OB

Ces blocs définissent la structure du programme application. Le programme système appelle un bloc d'organisation en présence de certains événements, à savoir:

- le bloc d'organisation OB1 à intervalles réguliers d'une manière cyclique. C'est à partir de ce bloc que l'on fera appels aux différents blocs de programmes,
- le bloc d'organisation OB 21 pour le redémarrage manuel de l'automate (commutation du sélecteur de mode de "STOP " sur "RUN") opération appelée démarrage à chaud,
- le bloc d'organisation OB 22 pour le redémarrage automatique après une coupure de la tension secteur (démarrage à froid),
- le bloc d'organisation OB 34 en cas de défaillance de la batterie.

Les blocs d'organisation (OB) font office d'interfaces entre le programme système (programme dans le CPU qui réalise les fonctions internes) et le programme d'application rédigé par l'utilisateur et définissent la structure programme. Ils font partie programme d'application au même titre que les blocs de programme, les blocs et de donnees.

Le bloc d'organisation OB1 permet de définir l'ordre chronologique d'exécution des blocs de programme PB. A cet effet, on programme une liste des appels de blocs. Au démarrage de l'automate, le traitement du programme commence par l'exécution du bloc d'organisation OB1.

> *Remarques:* si un programme comporte des blocs autres que le bloc de proramme PB1, la présence du bloc d'organisation OB1 est obligatoire. Si on veut un programme linéaire, on peut seulement utiliser le PB1. La programmation de l'OB1 est alors inutile.

Les blocks OB21, OB22 qui sont examinés uniquement au démarrage à chaud et à froid de l'automate. Ils servent à l'initialisation des données. On y appelle les blocs traitant l'initialisation. Remarque : l'automate *S5-95U* ne possède pas le bloc d'organisation OB22.

La liste n'est évidemment pas exhaustive, il y encore d'autres types, comme par exemple l'OB13 qui est scruté avec une période de 100 ms (tâches auxiliaires)…

•**Blocs de fonctions PB**Ces blocs sont des blocs fonctionnels (nécessairement sans paramètres). Ils contiennent le programme ou des partie du programme, c'est-à-dire que l'on va y mettre les instructions à exécuter. La numérotation est libre (de PB0 à 255 sur le 135U et PB0 à PB53 sur le 95U).

•Blocs de programmes FB
Il s'agit également de blocs de programme comme les PB, mais ceux-ci disposent d'un jeu d'opérations élargi par rapport a celui des blocs de programme: ils acceptent des paramètres. La numérotation est libre (de 0 à 255). Il peuvent être très utiles et sont utilisés de préférence aux blocs PB.

•Blocs systèmes SB
Les modules séquentiels (SB) sont spécialement utilisés pour effectuer des séquences selon Grafcet. Les paramètres d'entrées y seront les conditions d'avancement d'un pas de séquence et les paramètres de sorties, les ordres à exécuter lorsque ces conditions seront vérifiées.

Pour les automates S5-100U, les types de blocs peuvent avoir une longueur maximale de 2 K octets (environ 1024 instructions).

Pour mettre plus de clarté dans un programme, on le découpe en plusieurs sections affectées chacune a une fonction technologique. On est donc amené à programmer différents blocs (PB ou FB). L'ordre chronologique d'appel et de traitement des différents blocs est défini dans le bloc d'organisation (OB 1). Chaque bloc ainsi appelé peut lui-même contenir une instruction de saut vers un autre bloc. A la fin de l'exécution du bloc ainsi appelé, le traitement se poursuit automatiquement au lieu de départ du saut. De cette manière, il est possible d'obtenir une "imbrication" de 15 blocs.

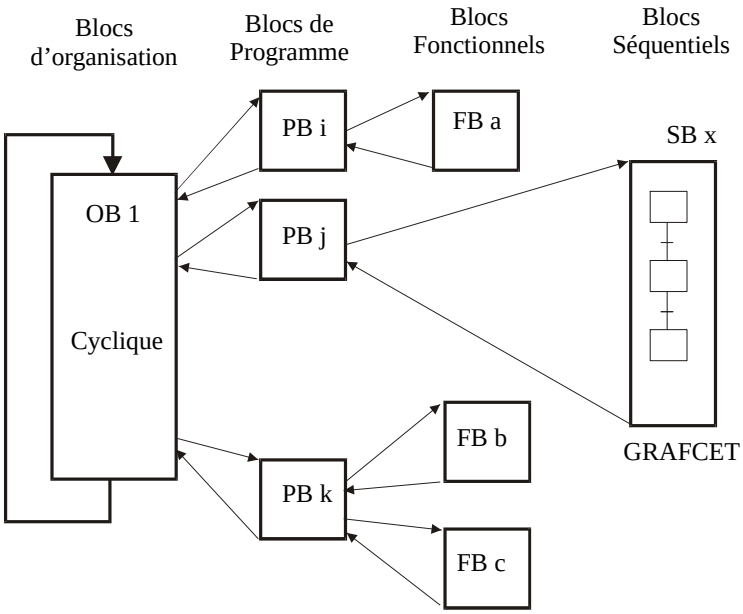


Fig 2.1 Structure des programmes en STEP 5

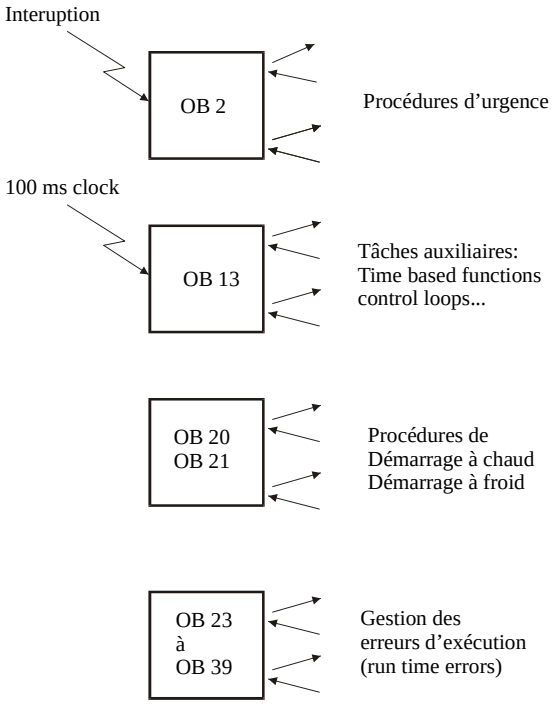


Fig 2.2. Blocs d'organisation en STEP5

Les appels de bloc ("SPA" et "SPB") permettent d'appeler les blocs de programme pour execution. Nous pouvons programmer des appels de blocs dans des blocs d'organisation (OB), des blocs de programme PB et des blocs fonctionnels. Les appels de blocs sont comparables à des appels à un sous-programme. Chaque "saut" équivaut à un changement de bloc. Il existe deux types de "saut":

- Saut absolu ou inconditionnel "SPA" (Sprung Absolut). Le bloc de programme appelé est exécuté indépendamment du résultat de la combinaison logique précédente et mémorisé dans le registre logique;
- Saut conditionnel : SPB (Sprung Bedingt). Le bloc de programme n'est exécuté que si le résultat de la combinaison logique précédente (registre logique) est égal à "1".

Les blocs de données

Les blocs de données sont des mots composés de 16 bits (DW) dans lesquels on peut lire et écrire des données.

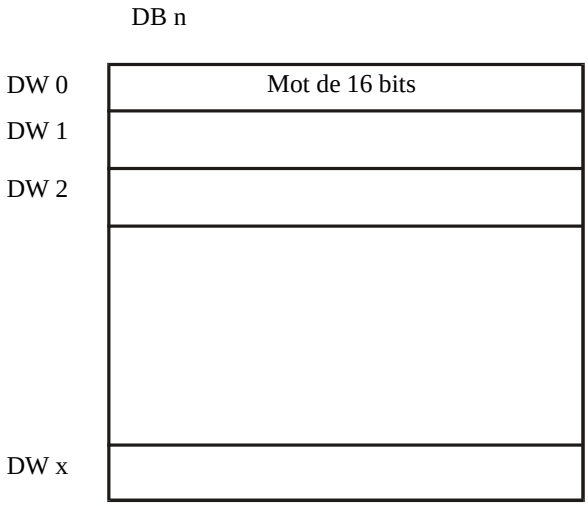


Fig 2.3 Block de données en STEP5

Comme en STEP7, un bloc de données ouvert reste valide jusqu'à ce qu'un autre bloc de données soit appelé ou qu'un bloc de code appelant soit terminé avec un BE ou un BEB. Par exemple dans un programme:

```
A    DBn     Ouvre/charge la DBn en mémoire
L    DW5     Travaille avec le DW5 de la DB courante (= DBn)
A    DBm     Ouvre la DBm et remplace la DBn
L    DW5     Transfert du mot (word) DW5 de la DBm dans ACCU1
```

Attention cependant:
Pour utiliser la DBn, il ne faut pas oublier de le créer au début du programme! Comment la créer?
–Soit la créer comme un bloc de programme
–Soit l'initialiser dans l'OB20/OB21 avec l'instruction:

```
E    DBn
```

2.3 PRINCIPALES INSTRUCTIONS DU LANGAGE STEP5

2.3.1 Les types de variables

Bit Variables

Input	E	0.0 à 127.7 (#byte . #bit)
Output	A	0.0 à 127.7
Flag (mémo interne)	M	0.0 à 255.7
Data	D	0.0 à 255.7
Timer Output	T	0 à 127
Counter	Z	0 à 127

Byte variables (=8 bits)

Input	EB	0 à 127
Output	AB	0 à 127
Flag	MB	0 à 255
Data (left byte)	DL	0 à 255
(right byte)	DR	0 à 255

Word variables (= 16 bits)

Input	EW	0 à 126
Output	AW	0 à 126
Flag (mémo interne)	MW	0 à 254
Data	DW	0 à 254

Constantes et formats

Byte	KB	0 à 255
Integer	KF	-32768 à +32767
Floating Point	KG	± 0,1469368 E-38 à ± 0,1701412 E 39
Hexadecimal	KH	0 à FFFF
Bit pattern (16 bits)	KM	0000000 0000000 à 11111111 11111111
ASCII characters (2)	KC	
2 bytes	KY	0 à 255 , 0 à 255
Time Constant	KT	0.0 à 999.3 (Delay value . Delay Unit)
		Delay Units: 0=10 ms, 1=100 ms, 2=1s, 3=10 s

2.3.2 Instructions logiques de base (sur bit variables)

U	AND ou chargement de l'accumulateur du RLG si vide
UN	AND NOT
O	OR
ON	OR NOT
U(	AND sur expression entre parenthèses
O(	OR sur expression entre parenthèses
)	fin parenthèse
S	SET, mise à vrai (permanente)
R	RESET, mise à faux (permanente)
=	assignation de la valeur de l'accumulateur du RLG (1 cycle)

2.3.3 Instructions de base sur bytes, words et constantes

On travaille sur les accumulateurs arithmétiques (ACCU1, ACCU2, etc.)

Instructions de chargement

A	Ouverture DB	
L/LW	Chargement:	ACCU1 dans ACCU2, Opérande dans ACCU1
T	Transfert	ACCU1 transféré dans opérande

Instructions arithmétiques

+F/+G	Addition:	ACCU1 = ACCU2 + ACCU1
-F/-G	Soustraction:	ACCU1 = ACCU2 - ACCU1
´F/´G	Multiplication:	ACCU1 = ACCU2 ´ ACCU1
¸F/¸G	Division:	ACCU1 = ACCU2 ¸ ACCU1

F si fixed point arithmetic (16 bits)
G si floating point arithmetic (32 bits)

Instructions de comparaison

!=F	ACCU2 = ACCU1
><F	ACCU2 ¹ ACCU1
>F	ACCU2 > ACCU1
>=F	ACCU2 ³ ACCU1
<F	ACCU2 < ACCU1
<=F	ACCU2 £ ACCU1

2.3.4 Operation d'appel des blocs

Instruction de saut vers un bloc

SPA	PB/FB/SB	Saut inconditionnel
SPB	PB/FB/SB	Saut conditionnel si bit accumulateur registre logique (RLG) =1

Instructions de fin de bloc

BE	Fin de bloc (requis)
BEB	Saut conditionnel (si RLG=1) à la fin du bloc
BEA	Saut inconditionnel à la fin du bloc

2.3.5 Utilisation des timers

Départ:

U	E/A/M	
L	KT	X.Y
SE	Tn	SE: timer type (on) retard à la montée

Utilisation:

U/O/UN/ON	Tn

2.3.6 Résumé des principales instructions:

U	AND
UN	AND NOT
O	OR
ON	OR NOT
S	SET de bit
R	RESET de bit
=	Affectation de la valeur RLG
L	Chargement accu arithmétique
SPA	Saut inconditionnel vers un block
SPB	Saut conditionnel (si RLG=1) vers un block
BE	Fin de bloc
BEA	Saut inconditionnel vers la fin du bloc
BEB	Saut conditionnel (si RLG=1) vers la fin du bloc

2.4 TRANSPOSITION D'UN GRAFCET EN STEP 5

2.4.1 Principe de la transposition d'un Grafcet en Liste d'Instructions sous Siemens S7

En partant d'une application séquentielle écrite en GRAFCET, on transpose de manière systématique en STEP 5 de la façon suivante. On divise le programme en trois parties

1. *Calcul des réceptivités.* On associe un bit interne (mémento) à chaque réceptivité. Celui-ci est mis à un Si la réceptivité est vraie.

2. *Evolution des étapes.* On utilise des **bits d'étapes.** A chaque étape est associé un bit interne qui est mis *à* 1 quand l'étape est active. On passe à l'étape suivante quand l'étape qui précède est active et que la réceptivité est vraie, ce qui correspond à mettre le bit d'étape suivante à 1 (set) et celui de l'étape précédente à 0 (reset).

3. *Actions associées aux étapes.* Il suffit d'imposer comme condition d'activation d'une action le bit correspondant à l'étape dans laquelle cette action doit être exécutée.

Lorsque l'on veut traduire du GRAFCFT en STEP5, il est très important de bien structurer son programme, sinon il devient très vite lourd et impossible à débugger. On utilisera par exemple un FB pour calculer toutes les réceptivités, un autre pour la mise à jour des bits d'étapes et un troisième FB pour exécuter les actions proprement dites. Egalement pour une question de clarté et quand le nombre d'étapes le permet, on utilise le mémento dont le numéro correspond à celui de l'étape.

2.4.2 Exemple:

Grafcet de l'exemple

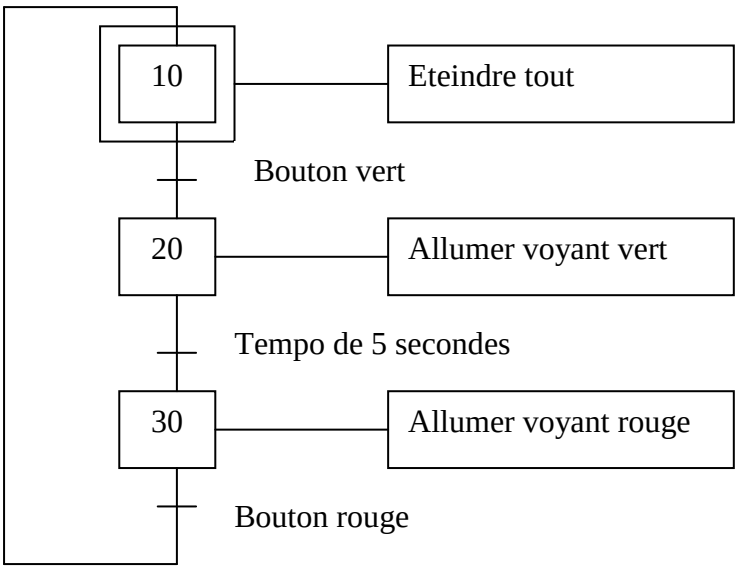


Fig1.5 Grafcet de l'exemple

Listing de l'exemple

0B20

:SPA PB 1
:BE

0B21

:SPA PB 1
:BE

0B22

:SPA PB 1
:BE

PB1 **activation de l'étape initiale**

:O M 0.0
:ON M0.0
:S M 10.0
:BE

OB1 **appel des sous routines**

```
:SPA FB 1
:SPA FB 2
:SPA FB 3
:BE
```

FB1 **calcul des réceptivités**

```
:U E 0.4
:= M 10.1
:***

:UN T 1
:= M 20.1
:***

:UN E 0.5
:= M 30.1
BE
```

FB2 **évolution du grafcet**

```
:U M 10.0
:U M 10.1
:SM 20.0
:RM 10.0
:BEB
:***

:U M 20.0
:U M 20.1
:S M 30.0
:R M 20.0
:BEB
:***

:U M 30.0
:U M 30.1
S M 10.0
:R M 30.0
:BE
```

FB3 **actions associées aux étapes**

```
:U M 20.0
:L KT 005.2
:SV T 1
:***
```

```
:U M 20.0
:=A0.2
:***

:U M 30.0
:=A0.6
:BE
```

Il faut également faire attention à ne pas franchir plusieurs étapes d'un même grafcet pendant un seul cycle d'automate. Pour cela, on procède de la façon suivante:

- Dans le bloc gérant l'évolution du grafcet, on effectue un **saut en fin de bloc** (BEB) quand la transition a été faite. Exemple: Voir listing ci avant.

OU

- On utilise, en plus de bits d'étapes et de réceptivités, des bits de transitions. Ceux-ci sont vrais quand la réceptivité de l'étape active est vraie. Ces bits servent alors de condition dans le franchissement d'une étape.

Exemple d'utilisation des bits de transition:

FB 1 Calcul des réceptivités:

Rien de changé par rapport au listing ci avant.

FB2: Calcul des transitions

```
:U M 10.0       Etape 10 active
:UM 10.1        Réceptivité étape 10 à 20
:M 10.2         Bit transition étape 10 à 20
:***

:UM20.0         Etape 20 active
:U M 20.1       Réceptivité étape 20 à 30
:= M 20.2       Bit transition étape 20 à 30
:***

:U M30.0        Etape 30 active
:U M 30.1       Réceptivité étape 30 à 10
:=M 30.2        Bit transition étape 30 à 10
:BE
```

FB3: Evolution du grafcet

```
:UM 10.2        Transition 10 à 20 0K
:S M 20.0       Activer étape 20
:R M l0.0       Désactiver étape 10
:***

:U M 20.2       Transition 20 à 30 0K
:S M 30.0       Activer étape 30
```

```
:R M20.0        Désactiver étape 20
:***

:U M 30.2       Transition 30 à 10 0K
:S M 10.0       Activer étape 10
:R M 30.0       Désactiver étape 30
:BE
```

FB4: Actions

Rien de changé par rapport au listing ci avant.

Pour le bloc où sont exécutées les actions, deux possibilités d'organisations sont possibles:

1. On trie par étapes Grafcet. On utilise un segment par étape de Grafcet, et la condition d'exécution est le bit d'étape. Si une même action est exécutées à plusieurs étapes, elle sera donc notée plusieurs fois. Ce type de classement permet de bien garder en vue la séquence du processus.

2. On **trie par action.** On utilise un segment par action, dans lequel on affecte le résultat de l'équation booléenne des étapes dans lesquelles cette action est exécutée, à la sortie en question. On parle de combinatoire des sorties. Cette méthode permet de voir directement à quelles étapes telle sortie est activée et est donc intéressante pour le débuggage. On perd cependant la vue de la séquence.

2.5 UTILISATION DE LA CONSOLE:

2.5.1 Le principe

On programme et on modifie un bloc à la fois. On peut choisir de travailler directement sur l'automate ou sur la console. Pour la création du programme, il est cependant conseillé de programmer offline et d'utiliser la programmation on-line pour le debuggage. Il faudra donc effectuer des transferts de blocs entre la console et l'automate, que ce soit pour sauver le programme sur le disque ou pour charger le programme dans l'automate.

2.5.2 Lancement du logiciel de programmation

Taper S6 dans le DOS.
Le logiciel démarre avec le dernier projet créé. Il faut en recréer un nouveau en faisant Sauvegarder sous ... avec le nouveau nom. On valide avec 'INS'. On peut maintenant modifier les présélections du nouveau projet. On accède à cet écran par 'F4' (attention, il y 2 pages). On introduit le nom du fichier programme, du fichier symbolique, du fichier imprimante, on valide l'utilisation du mode symbolique et on choisi de se mettre online ou offline. Attention, ne pas oublier de sauver avec F6.

2.5.3 Edition

On peut maintenant passer en mode édition dans le fichier programme. Les blocs sont divisés en segments. Il convient d'utiliser cette division pour clarifier le programme au maximum. Lorsque l'on modifie une ligne, il faut valider une première fois pour valider le segment modifié et une seconde fois pour valider le bloc tout entier. On passe d'un segment à l'autre

avec les touches Page Up/ Page Down.

On édite les *mnémoniques* en tapant F7 (ou edit... Liste des assignations). Après avoir introduit les mnémoniques et les commentaires, valider F7 et sauver F6.

2.5.4 Remarques sur l'éditeur:

- Le clavier numérique n'est pas actif
- La touche END permet d'insérer une ligne.
- La touche INS permet de valider une ligne après modification.
- Lorsque l'on a chargé un bloc dans l'éditeur, il faut appuyer sur F6 (Editer) pour le modifier.

2.5.5 Transfert

Les fonctions de transfert sont accessibles par le menu OBJET/BLOCS/... ou par les touches F5 et SHIFT F5.

2.5.6 Menu TEST

C'est par ce menu que l'on a accès aux fonctions de l'automate: Run Stop. Il faut être en mode on-line (voir présélection). On peut également visualiser dynamiquement l'état d'un bloc ou des variables. Pour celles-ci, on les introduit une par une et on valide l'activation de la visualisation dynamique avec F6 (Activer). Il est également possible d'enregistrer une liste de variables (F2, Mémoriser) et de la rappeler par après (Fl Appel). Le nom d'une telle liste est BB x, où x est un numéro.
Pour les commandes de forçage, il faut appeler les mots (AB, MB au format KM) et pas les bits un par un. On se place ensuite sur le bit que l'on veut modifier. Après la modification, il faut valider une fois pour exécuter le forçage et une fois pour remettre à jour la visualisation dynamique. Le forçage des sorties n'est autorisé que quand l'automate est en stop.

2.6 ANNEXE : LISTE DES ENTREES ET SORTIES:

Pour l'automate 135U

Adresse physique	Signification
A0.0	voyant rouge 1
A0.1	voyant orange 1
A0.2	voyant vert 1
A0.3	voyant rouge 2
A0.4	voyant orange 2
A0.5	voyant vert 2
E0.0	sélecteur 1 à gauche
E0.1	sélecteur 1 à droite
E0.2	sélecteur 2 à gauche
E0.3	sélecteur 2 à droite
E0.4	bouton poussoir vert 1
E0.5	bouton poussoir rouge (NF)
E0.6	bouton poussoir vert 2

Pour l'automate 95U

Adresse physique	Signification
A32.0	voyant rouge
A32.1	voyant orange
A32.2	voyant vert
E32.0	bouton poussoir vert
E32.1	sélecteur à gauche
E32.2	sélecteur à droite
E32.3	bouton poussoir rouge (NF)

3. AUTOMATES ALLEN BRADLY CONTROL LOGIX 5550

3.1 CARACTÉRISTIQUES PRINCIPALES

Processeurs :
– logique Logix 5550 avec 512 KB de RAM
– de communication DeviceNet 1756-DNB

Communications
– RS 232 pour la programmation

Réseaux:
– DH+
– RIO (Remote I/O)
– Ethernet TCP/IP
– DeviceNet
– ControlNet

Langages :
– LD (ladder)
– SFC (Sequential Function Chart basé sur la norme IEC 1131-3)
– Function Blocks (FB)

3.2 ORGANISATION DES PROGRAMMES

- Tasks
  - Main tasks
    - Main Program
      - Program Tags
      - Main routine_0
      - Routine_1
      - Routine_2
      - ...
  - Unscheduled Program
    - Program Tags
    - Main routine_A
    - Routine_B
    - ...

3.3 LES VARIABLES

3.3.1 Adressage des variables

L'adressage complet

L'adressage se fait en suivant la logique du matériel.

Par exemple, dans la variable Local: 2:I.Data.11, on a:

- 2 = numéro de l'emplacement de la carte sur le rack (sur notre matériel 1 = Devicenet, 2 = carte d'entrées, 3 = cartes des sorties)
- I est mis pour entrées et O pour sorties,
- Data = données
- 11 = numéro du bit (sur notre matériel, 0 à 31)

Utilisation de « tags »

Compte tenu de la longueur de l'adressage il est possible (et vivement recommandé) de définir des noms qui « pointent » vers l'adresse physique et qui évoquent la signification de la variable.

3.3.2 Définitions des données

La définition des données est réalisée dans « Controller Tags » pour tous les programmes ou bien dans « Programme Tags » si la définition doit être valable uniquement dans le programme considéré.

La démarche de définition d'une nouvelle variable est assez semblable à celle que l'on trouve pour les langages de programmation structures des ordinateurs (Pascal, Fortran, C) : définition du nom de la variable, de son type et de son champ si c'est un tableau. De manière pragmatique, on utilise les lignes des tableaux mis à disposition.

Les variables qui seront visibles par OPC seront définies dans « Controller Tags ».

3.3.3 Types de données

Les types de données suivants sont disponibles :

BOOL Booléen, c'est-à-dire 1 bit
Valeur 0/1

SINT Entier court codé sur 7 bits + S:C (bit de report), soit 1 octet
Valeur de -128 à +126

INT Entier codé sur 15 bits + S:C (bit de report), soit 2 octets
Valeur de -32768 à +32767

DINT Entier double codé sur 31 bits + S:C (bit de report), soit 4 octets
Valeur de - 2 147 483 648 à + 2 147 483 647

REAL Nombre (réel) en virgule flottante codé sur 4 octets
Valeurs de -3,402823E38 à -1,1754944E-38 (valeurs négatives), 0
et de 1,175499E-38 à 3,402823E38 (valeurs positives)

Remarque: Les types de données sont conformes aux types définis dans la norme IEC 1131-3

Conversions

Il est possible de convertir

- des SINTs ou des INTs en REALs (sans perte de précision)
- des DINTs en REALs (avec éventuelle perte de précision à cause de l'exposant du REAL)
- des REALs en un nombre entier avec arrondi

Il est impossible de convertir à partir de ou vers le type de donnée BOOL

3.3.4 Concept de tableau

Definition d'un tableau à 1, 2 ou 3 dimensions

– one_d_array DINT[3]
– two_d_array DINT[4,5]
– three_d_array DINT[2,3,4]

Les dimensions sont basées sur zéro (i.e. les indices commencent à zéro).
Par exemple, dans one_d_array on a les éléments suivants: one_d_array[0], one_d_array[1], one_d_array[2].

Références statiques et dynamiques

La référence aux éléments peut se fait de manière statique:
Exemple: one_d_array[1]
ou de manière dynamique
Exemple: on met (instruction MOV) la valeur 2 dans la variable *position* et puis on utilise one_d_array[*position*]

Remarque importante

Assurez-vous que tout indice de tableau est dans les limites de définition du tableau. Les instructions qui lisent les tableaux produisent un défaut majeur si l'indice dépasse la dimension déclarée.

Adressage d'un bit dans un tableau

On peut adresser des bits dans des éléments d'un tableau à l'aide du « . »
Exemple: one_d_array[0].3

Stockage des donnees d'un tableau

Pour un tableau à deux dimensions, le stockage s'effectue ligne par ligne.

Allocation de memoire

Les bits de poids fort sont placés en premier lieu dans les adresses de la mémoire.

3.3.5 Concept de structure

Adressage des variables dans les structures

Un exemple typique de structure est fournit par les temporisations. L'accès aux différentes composantes est réalisé grâce au ".".

ex. Bit DNT4 : 0.DN (Bit done)
Word PV T4 : 0.PRE
Word AC T4 : 0.ACC3.3.6 Divers

Les bits systèmes

Ils nous renseignent sur l'état du système.
Un bit très important: le bit S : 1 / 15 qui est mis à ‘ 1 ’ lors du premier cycle d’automate.

3.4. PRINCIPALES INSTRUCTIONS DU LANGAGE LADDER

Le langage de type LADDER est un langage de type graphique. Il s'agit là de son principal avantage. Par contre l'explication et la description du langage ne s'en trouvent que plus ardues.

3.4.1 Principales instructions sur bits

A -] [-	A	A -] / [-	non A
A -( )-	A = 1		
A -(L)-	set A	A -(U)-	reset A
-[ONS]-	Valide les sorties quand les entrées précédents l’instruction ONS passent de 0 à 1		

3.4.2 Instructions de type relais

Test circuit fermé	XIC	(et)
Test circuit ouvert	XIO	(et pas)
Activation de sortie	OTE	(=)
Accrochage de sortie	OTL	(set)
Décrochage de sortie	OTU	(reset)
Entrée immédiate	IIN	
Sortie immédiate	IOT	

3.4.3 Instructions logiques

ET logique	AND
OU logique	OR
OU exclusif	XOR
Complément	NOT

3.4.5 Instructions de temporisation et de comptage

Temporisation au travail:	TON
Temporisation au repos:	TOF
Temporisation à mémoire	RTO
Compteur	CTU
Décompteur	CTD
Remise à zéro	RES

3.4.6 Instructions de comparaison

Comparaison	CMP
Egalité	EQU
Différence	NEQ
Supériorité	GRT
Supériorité ou égalité	GEQ
Infériorité	LES
Infériorité ou égalité	LEQ

3.4.7 Instructions de calcul

Calcul	CPT
Addition	ADD
Soustraction	SUB
Multiplication	MUL
Division	DIV
Moyenne	AVE
Effacement	CLR
Négation	NEG
Racine carrée	SQR
Tri	SRT
Ecart type	STD

3.4.8 Instructions de transfert

Transfert d'un mot:	MOV
Transfert masqué	MVM

3.4.9 Instructions de contrôle de programme

Saut	JMP
Saut au sous-programme	JSR
Etiquette	LBL
Retour	RET
Boucle	FOR, NXT, BRK
Fin temporaire	TND

3.4.10 Activation du module DeviceNet

|----------() Local :1 :O.CommandRegister.Run
(à placer dans la routine Main)

3.4.11 Instructions de communications

MSG

3.4.12 Principales instructions de programmation

Comparaison: Bloc EQU EQUAL source A et source B

Affectation: Bloc MOV MOVE source A vers source B

Temporisation: Bloc TON TIMER ON DELAY

Compteur Bloc CTU COUNT

99
Saut: -(JMP)-
99
Label: -[LBL]-

Appel sous routine: Bloc JSR JUMP TO SUBROUTINE

Fin de sous routine: Bloc RET RETURN

3.5 Utilisation du logiciel RS Logix 5550

3.5.1 Architecture et hiérarchie des projets

La programmation de l'automate se fait via la réalisation de plusieurs tâches (*Task*). La tâche principale (*Main Task*) contient le programme à exécuter. Les autres tâches sont relatives à des missions auxiliaires. La tâche principale de l'automate peut être décomposée en plusieurs programmes (*Program*) qui tournent de manière concurrente. Chaque programme contient lui-même plusieurs routines. Pour chaque programme, c'est la routine « *Main program* » qui est le nœud maître. De cette routine, on appellera toutes les autres suivant une arborescence plus ou moins compliquée. En ce qui concerne les variables du programme, on peut avoir de variables globales si on les définit dans la rubrique « *Controller Tags* » ou bien des variables locales valables uniquement au niveau du programme si elles sont définies au niveau du « *Program Tags* ».

La présentation de l'arborescence est réalisée à l'aide d'un explorateur type windows. Les relations d'inclusions de chaque entité sont alors parfaitement visibles (Voir figure 3.1).

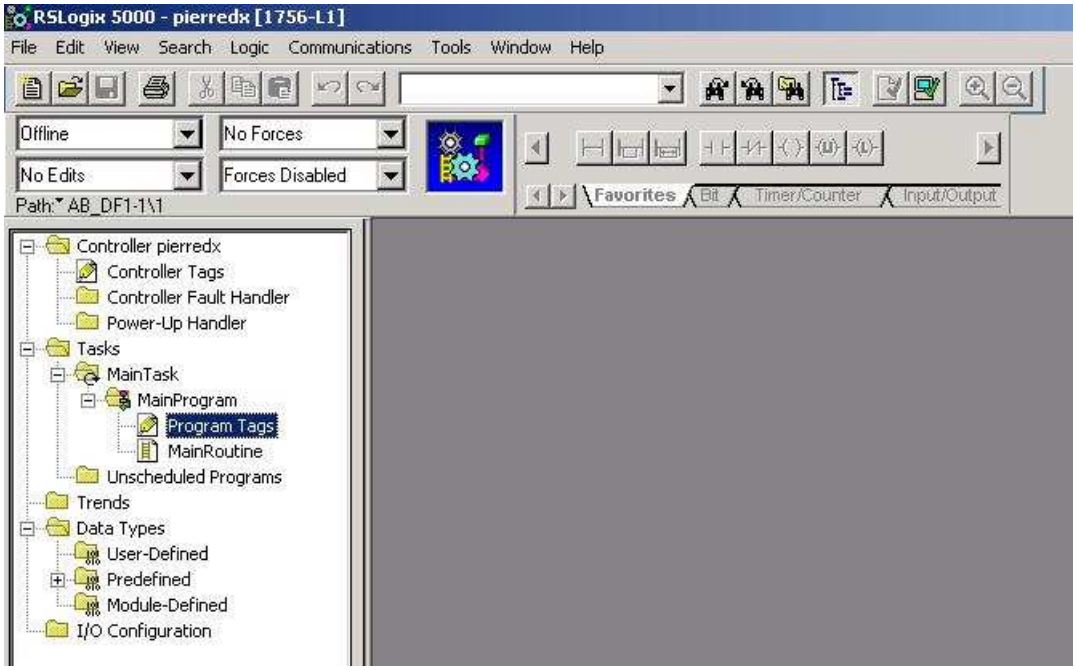


Fig 3.1: Architecture des projets RS Logix

3.5.2 Définition et introduction d'une variable

Les variables portent le nom de « tags ». Les tags sont soit des variables du systèmes (entrées – sorties directes ou déportées) soit des variables internes (pointeur d'étape, etc.).

En ce qui concerne les variables du système, leur définition, basée sur l'architecture matérielle est assez difficile d'utilisation dans la programmation. Pour éviter cette difficulté, on peut recourir à ce qu'on appelle des « tags » c'est-à-dire des étiquettes qui pointent vers les adresses physiques. On remplace alors toutes les occurrences de la variable par le tag sélectionné. Cette opération se fait dans le même tableau que la définition des variables. Une colonne est spécialement réservée à cet effet.

Les variables de programmation sont également définies dans les tags. Lors de leur définition, on doit spécifier le nom de la variable, son type (BOOL, INT, TMER, etc.) et même sa valeur initiale lors des démarrages à chaud ou à froid. L'introduction d'une nouvelle variable est réalisée en ouvrant l'onglet « Edit tags » dans les répertoires « *Controller tags* » et « *Program Tags* ».

Les variables et les tags se définissent dans les ongles « *Controller tags* » ou bien dans « *Program Tags* ». Les variables définies dans « *Controller tags* » sont ainsi définies de manière globale pour tous les programmes. C'est là également que sont définies les variables qui seront visibles par l'extérieur (OPC ou superviseur). Les variables définies dans « *Program task* » ont une portée limitée au cadre restreint du programme.

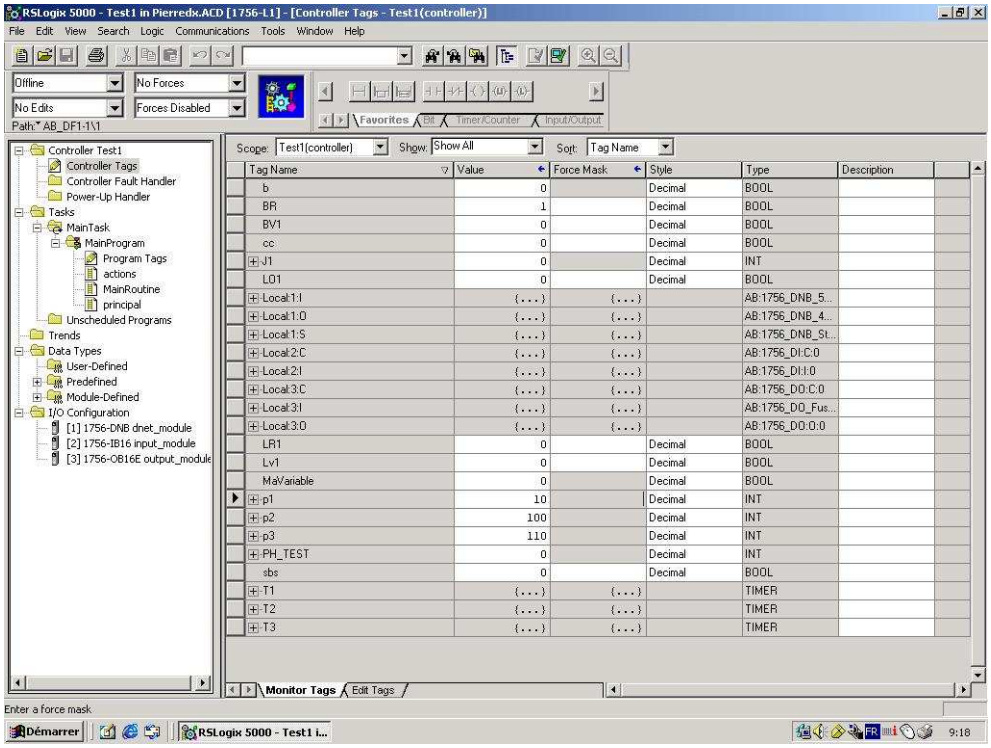


Fig 3.2 : Définitions des variables dans "Controller tags"

3.5.3 Programmes et routines

Insérer un nouveau programme ou une nouvelle routine

Pour insérer un nouveau programme ou une nouvelle routine, il suffit de se placer au niveau supérieur à l'entité à créer (soit au niveau tâche si on veut insérer un nouveau programme ou bien au niveau programme si on veut introduire une nouvelle routine) et faire un traditionnel « click droit de souris ». L'insertion d'un nouvel élément est alors proposée.

Edition d'une routine

On peut alors éditer de manière très simple les routines en procédant par un click sur le nom de la routine.

Insertion et édition des rungs

Ici encore on procède manière assez intuitive quand on se place dans un environnement de type windows. En se plaçant sur le *rung* à éditer ou sur le *rung* à côté duquel on veut insérer un nouveau, on fait un click droit de souris. Un menu apparaît. On sélectionne l'action désirée.

Etant donné la nature graphique du langage « ladder », on peut insérer des instructions (lorsqu'on est en mode édition du *rung*) en allant rechercher les éléments dans une bibliothèque graphique proposée au-dessus de la fenêtre d'édition. La composition du programme se fait alors de façon assez naturelle.

On note également l'aide en ligne proposé au programmeur lors de l'entrée des paramètres des instructions, par exemple le choix d'une variable du programme.

3.5.4 Communications entre l'automate et la console de programmation

Le menu « *Communications* » permet d'échanger des informations entre la console de programmation et l'automate programmable.

Une fois le programme terminé, on peut le décharger de la console vers l'automate au moyen de la commande « *Download* ». De manière plus risquée, on peut reprendre un programme se trouvant dans l'automate et le recharger dans la console au moyen de la commande « *Upload* ». Cette opération ne permet cependant pas d conserver les commentaires introduits dans le programme original.

Le passage du mode hors-ligne « off-line » (c'est-à-dire sur la console de programmation) vers le mode en ligne « on-line » (c'est-à-dire sur l'automate) ou vice-versa peut se faire avec les menus « *Go online* » ou bien « *Go Offline* » respectivement.

Le menu offre encore la possibilité de travailler selon différents « modes ». Le mode « *Run* » est celui qui correspond à l'exécution du programme dans des conditions normales de production. Pour la mise au point, on travail en modes « *Remote* ». Le mode « *Remote Run* » permet de décharger un nouveau programme, de la tester sur l'automate et même de le modifier sur l'automate. Par contre parfois il est intéressant de simuler une modification sans pour autant faire fonctionner réellement l'automate. C'est le cas d'une application industrielle dangereuse, où il n'est pas loisible de se permettre des erreurs importantes. Dans ce cas, on travaille d'abord en more « *Remote Test* » qui permet de faire tourner le programme de manière virtuelle sur le PC de programmation et détecter des erreurs sans que les sorties de la partie opérative soient actionnées.

3.5.5 Mise au point et débogage

La mise au point des programmes est extrêmement facile dans cet éditeur. En effet en se plaçant « on-line », on voit que l'édition des programmes laisse apparaître des jeux de couleur verte lorsque les bits deviennent vrais. On peut ainsi assister au déroulement du programme.

3.6 Exemple

On reprend ici l'exemple précédent développé pour le S7 de Siemens et on en donne sa version en ladder diagram. Etant donné le caractère graphique du langage, le programme est donné sous forme d'impressions d'écrans.

Grafcet de l'exemple

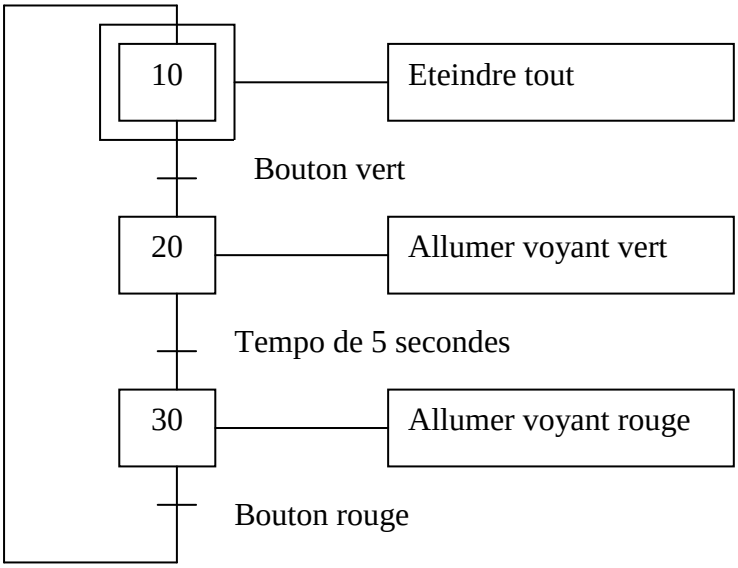


Fig 3.3: Grafcet de l'exemple

Programme type:

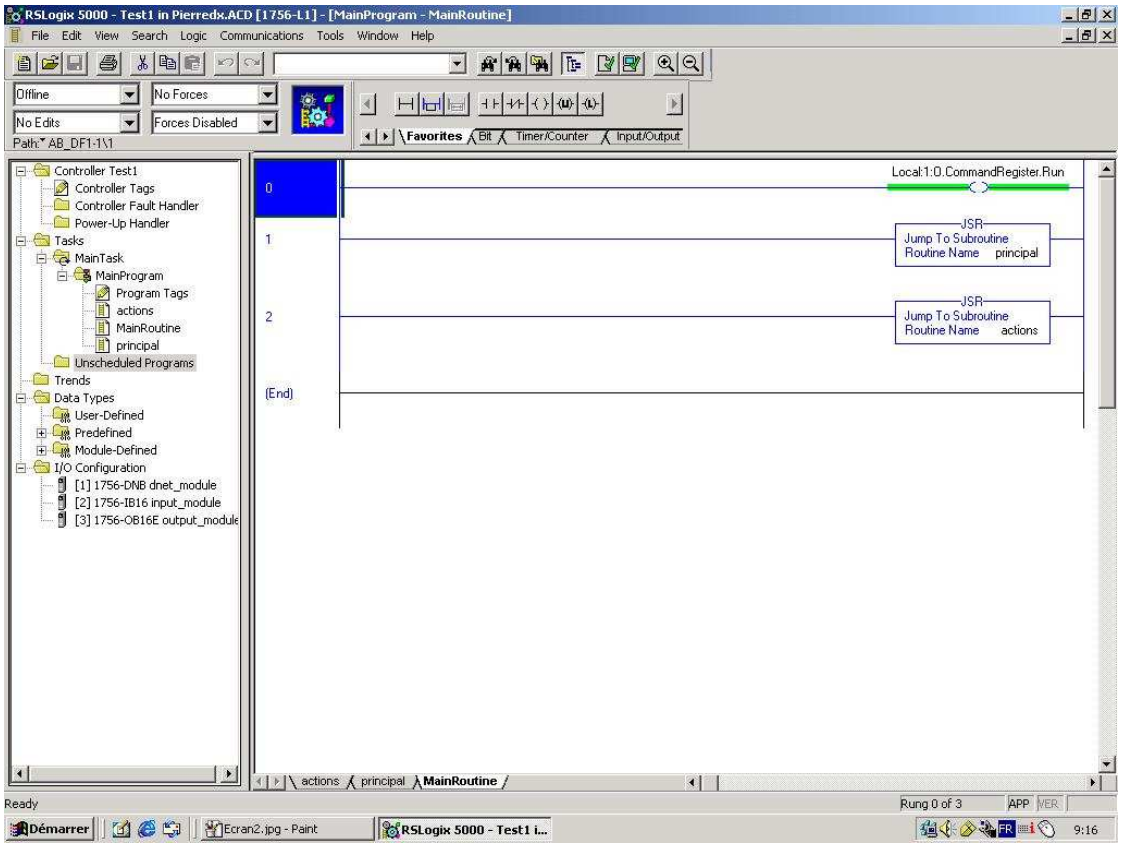


Fig 3.4 : Programme principal (MainRoutine)

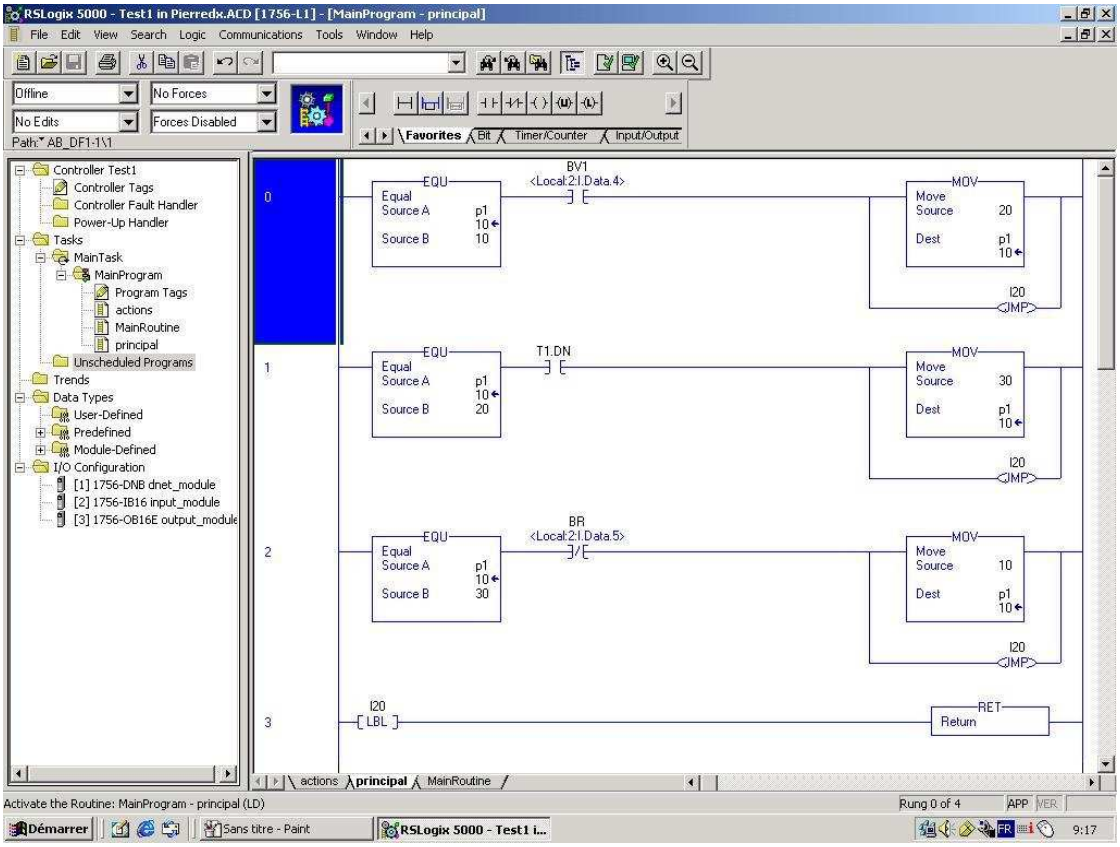


Fig 3.5 : Recéptivités – transitions (Principale ici)

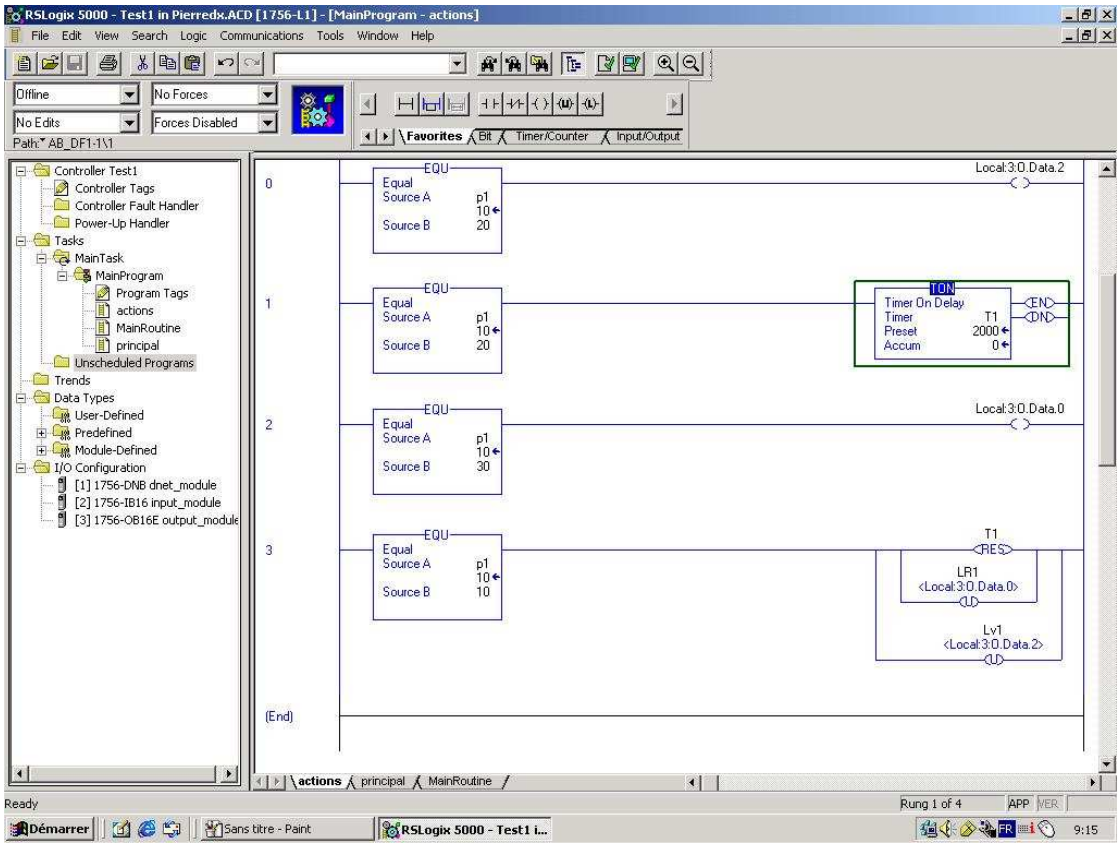


Fig 3.6 : Actions (routine du même nom ici)

3.7 Annexe : Liste des entrées et sorties de l'automate

Carte de sorties directes	Local:3:O.Data.0	Voyant rouge 1
	Local:3:O.Data.1	Voyant orange 1
	Local:3:O.Data.2	Voyant vert 1
	Local:3:O.Data.3	Voyant rouge 2
	Local:3:O.Data.4	Voyant orange 2
	Local:3:O.Data.5	Voyant vert 2
	Local:3:O.Data.6	Non connecté
	Local:3:O.Data.7	Non connecté
	Local:3:O.Data.8	Connexion vers S7-400
	Local:3:O.Data.9	Connexion vers S7-400
	Local:3:O.Data.10	Connexion vers S7-400
	Local:3:O.Data.11	Connexion vers S7-400
	Local:3:O.Data.12	Connexion vers S7-400
	Local:3:O.Data.13	Connexion vers S7-400
	Local:3:O.Data.14	Connexion vers S7-400
	Local:3:O.Data.15	Connexion vers S7-400
Carte d'entrées directes	Local:2:I.Data.0	Sélécteur 1 à gauche
	Local:2:I.Data.1	Sélecteur 1 à droite
	Local:2:I.Data.2	Sélecteur 2 à gauche
	Local:2:I.Data.3	Sélecteur 2 à droite
	Local:2:I.Data.4	Bouton poussoir vert 1
	Local:2:I.Data.5	Bouton poussoir rouge
	Local:2:I.Data.6	Bouton poussoir vert 2
	Local:2:I.Data.7	Non connecté
	Local:2:I.Data.8	Connexion vers S7-400
	Local:2:I.Data.9	Connexion vers S7-400
	Local:2:I.Data.10	Connexion vers S7-400
	Local:2:I.Data.11	Connexion vers S7-400
	Local:2:I.Data.12	Connexion vers S7-400
	Local:2:I.Data.13	Connexion vers S7-400
	Local:2:I.Data.14	Connexion vers S7-400
	Local:2:I.Data.15	Connexion vers S7-400

4 AUTOMATE SCHNEIDER – TELEMECANIQUE

Nous disposons au laboratoire d'un automate Schneider-Telemecanique TSX Premium 57.

4.1 Caractéristiques principales de l'automate

Alimentation de type PSY 2600M (100 …240 VAC)

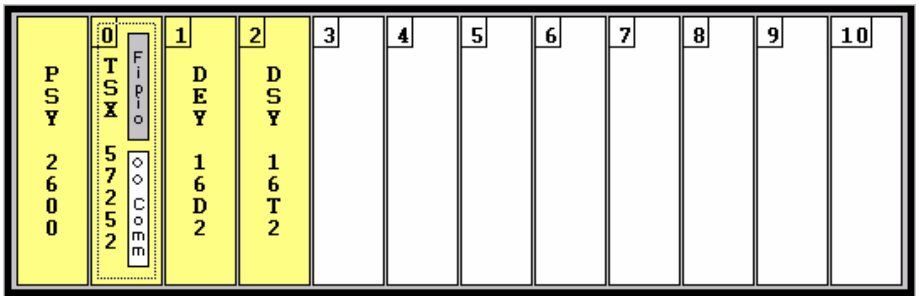
Processeur TSX P 57 252 V3.3

- Mémoire RAM interne : 64 K mots
- Nombre maximum de voies E/S TOR directes : 1024
- Temps typique d'exécution pour 1 K instructions : 0.78 ms
- Coupleur bus/réseau Fipio intégré (connecteur SUB-D 9 contacts)
- Communication par prise terminal (TER et AUX) en mode Uni-Telway
- Emplacement pour une carte d'extension mémoire PCMCIA type I
- Emplacement pour une carte de communication PCMCIA type III
- Langage de programmation : LD (Ladder), IL (Instruction List), ST (Structured Text) et SFC (Sequential Function Chart)

Rack de type TSX RKY 12 EX (12 positions et extensible V2)

Module d'entrée TOR de type TSX DEY 16D2 (24 VCC)

Module de sorite TOR de type TSX DSY 16T2 (24 VCC 0.5 A)



4.2. Le réseau Fipio

Le bus de terrain Fipio permet la connexion de 127 équipements à partir du point de connexion intégré au processeur. Il est conforme à la norme WorldFip et est destiné au déport d'entrées/sorties jusqu'à 15 km.

Sur ce réseau, on peut trouver :

- Le TSX Premium qui est gestionnaire du réseau
- Une passerelle Fipio/AS-i de type TBX SAP 10
- Un module d'entrées déportées de type TBX DES 16C22

- Un module de sorties déportées de type TBX DSS 16C22

4.3. Le réseau AS-i

A la passerelle AS-i du réseau Fipio est connecté un réseau AS-i sur lequel sont greffés 5 modules :

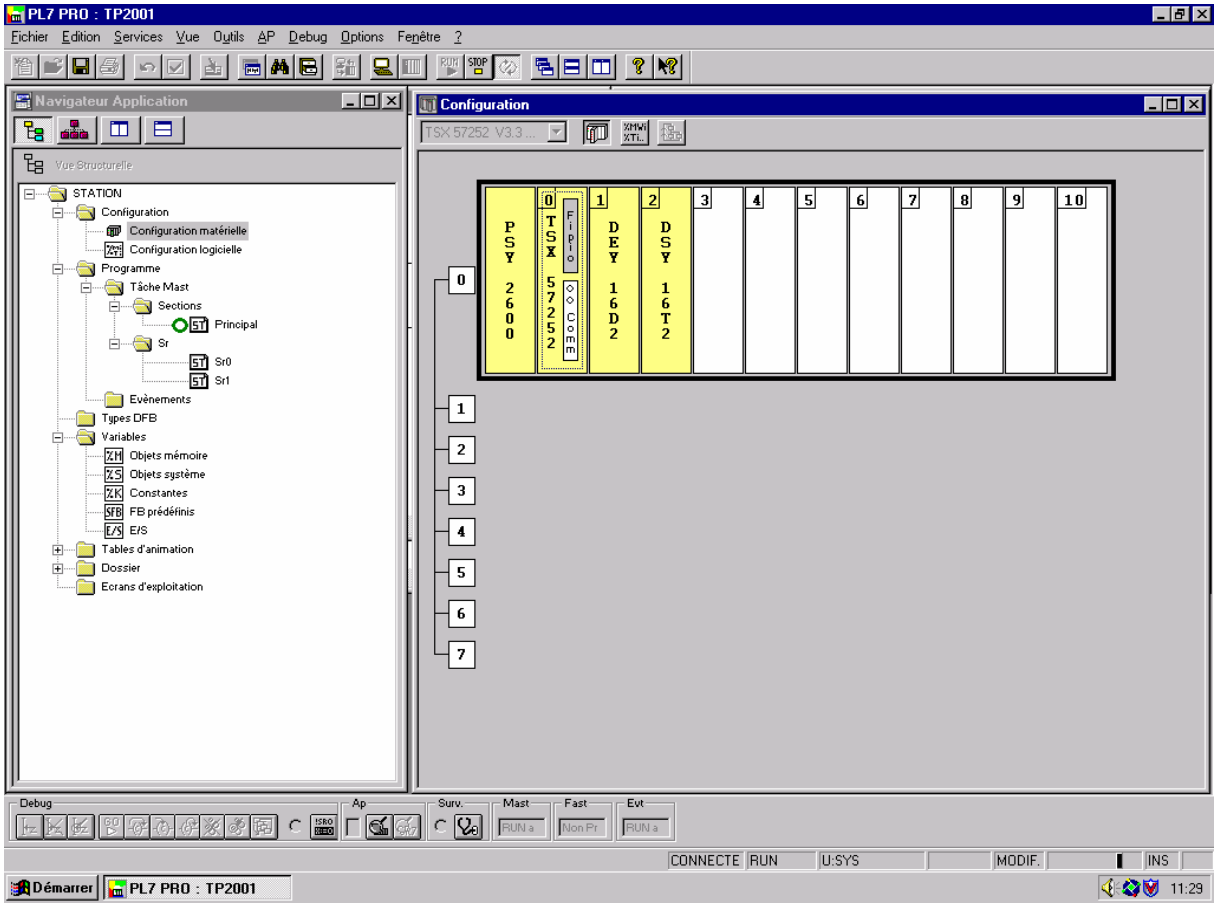
- 1 module IFM 2 boutons et 2 voyants (adresse AS-i : 16)
- 1 module IFM 4 entrées/4 sorties (adresse AS-i : 17)
- 3 modules FESTO (adresse AS-i : 18, 19 et 20)

4.4. Utilisation de PL7 Pro v3.4 pour configurer l'automate

4.4.1. Configuration de l'automate

Il faut commencer par définir la configuration matérielle de l'automate. Pour ce faire :

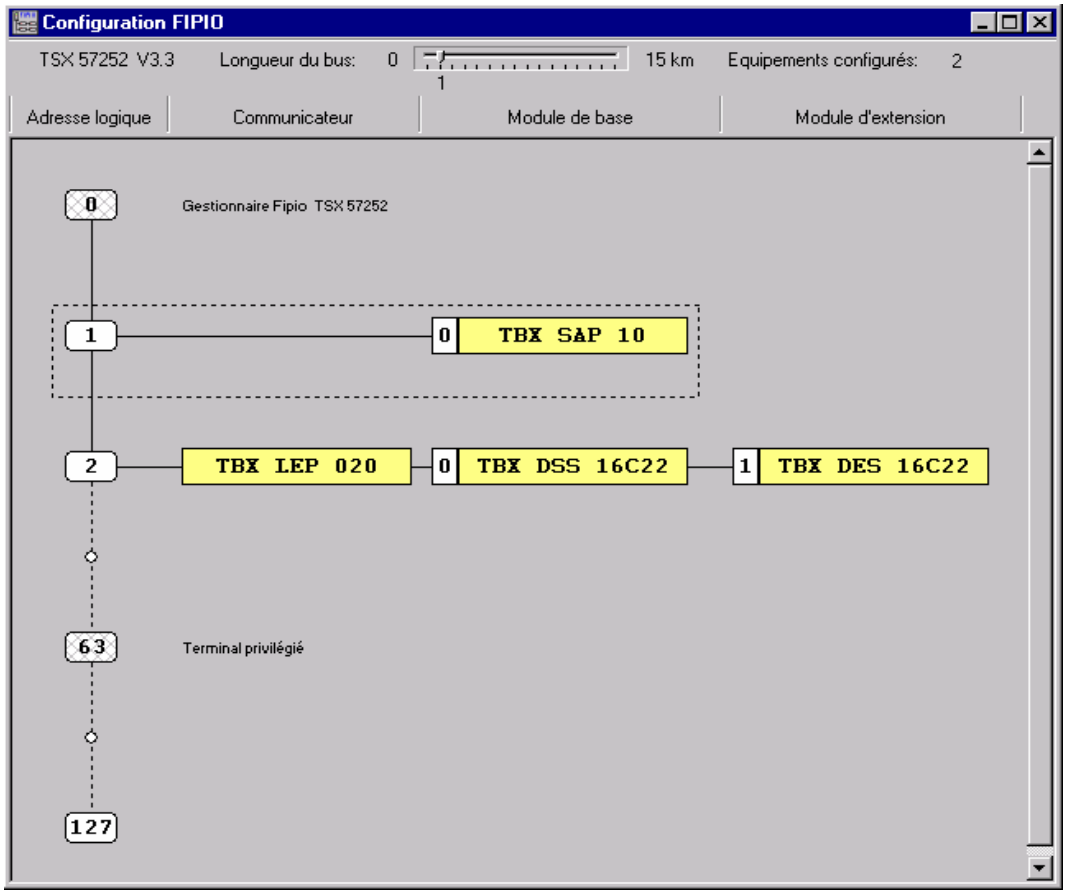
1. Lancer le logiciel PL7 Pro.
2. Choisir le type de processeur, ce qui provoque l'ouverture du « Navigateur Application ».
3. Cliquer sur STATION, Configuration, Configuration matérielle. Une fenêtre « Configuration » s'ouvre.



4. Définir le type exact de rack en cliquant sur le carré blanc situé devant le rack proposé par défaut.
5. Sur le premier emplacement du rack (cet emplacement n'est pas numéroté), vérifier que l'alimentation proposée est correcte.
6. Sur le deuxième emplacement (numéroté 0), on trouve le processeur avec un rectangle blanc qui indique le type de coupleur bus/réseau intégré au processeur (en l'occurrence, Fipio) et un autre rectangle blanc (oo comm) qui représente la liaison avec la console de programmation. En cliquant sur ces rectangles, on accède à des fenêtres de configuration du réseau Fipio et de la liaison PC/Automate (voir § 4.2.).
7. Sur les emplacements suivants (numérotés de 1 à …), on peut configurer des cartes d'E/S, de communication, etc. Il faut respecter l'ordre physique des cartes réellement présentes sur le rack de l'automate. Pour placer une carte sur un emplacement, il suffit de cliquer sur cet emplacement. Ceci provoque l'ouverture d'une fenêtre qui permet de choisir le type de carte.

4.4.2. Configuration du réseau Fipio

On peut ensuite définir la configuration du réseau Fipio en cliquant sur le rectangle blanc du processeur. La fenêtre suivante s'ouvre :



L'adresse 0 est celle du gestionnaire de réseau, en l'occurrence, le TSX Premium. Lorsqu'on veut ajouter un module au réseau, il suffit de cliquer sur l'endroit désiré et de sélectionner le type de module adéquat. Dans notre cas, il s'agit d'une passerelle AS-i TBX SAP 10 (nœud

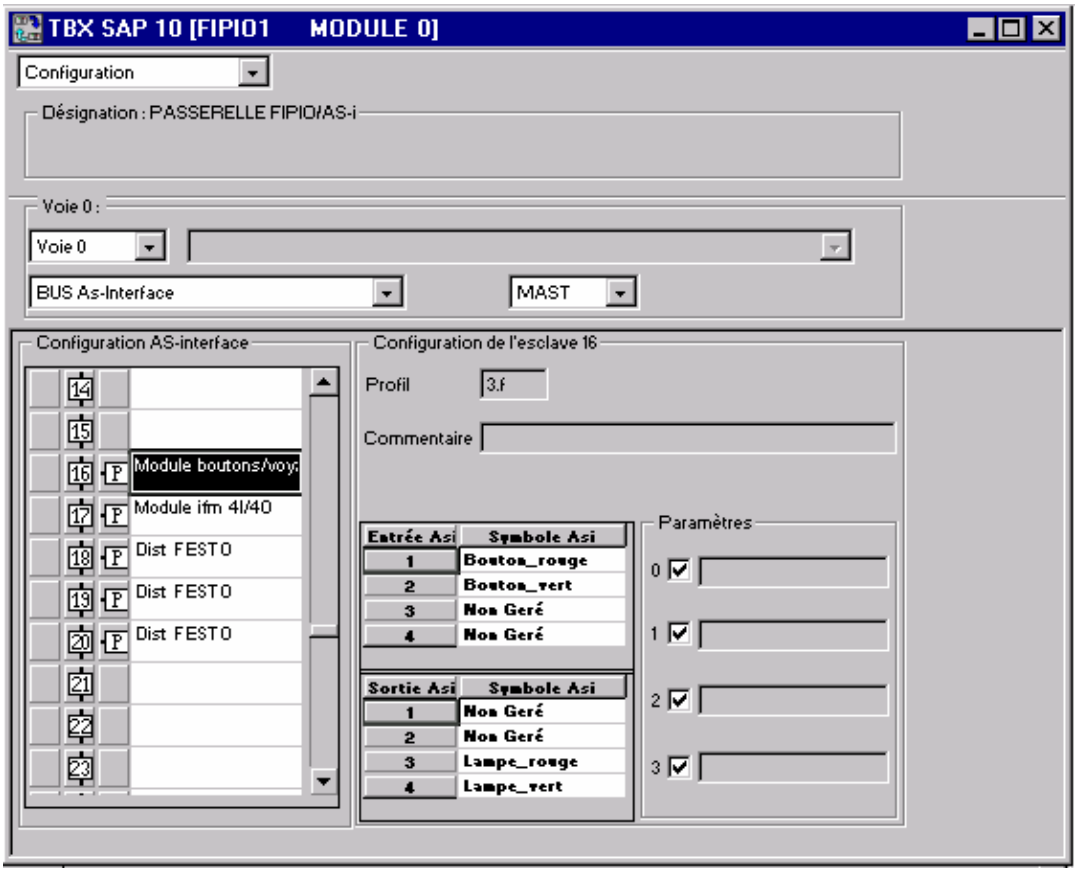
1), d'un module de sorties déportées TBX DSS 16C22 et d'un module d'entrées déportées TBX DES 16C22 (nœud 2).

Remarques importantes :

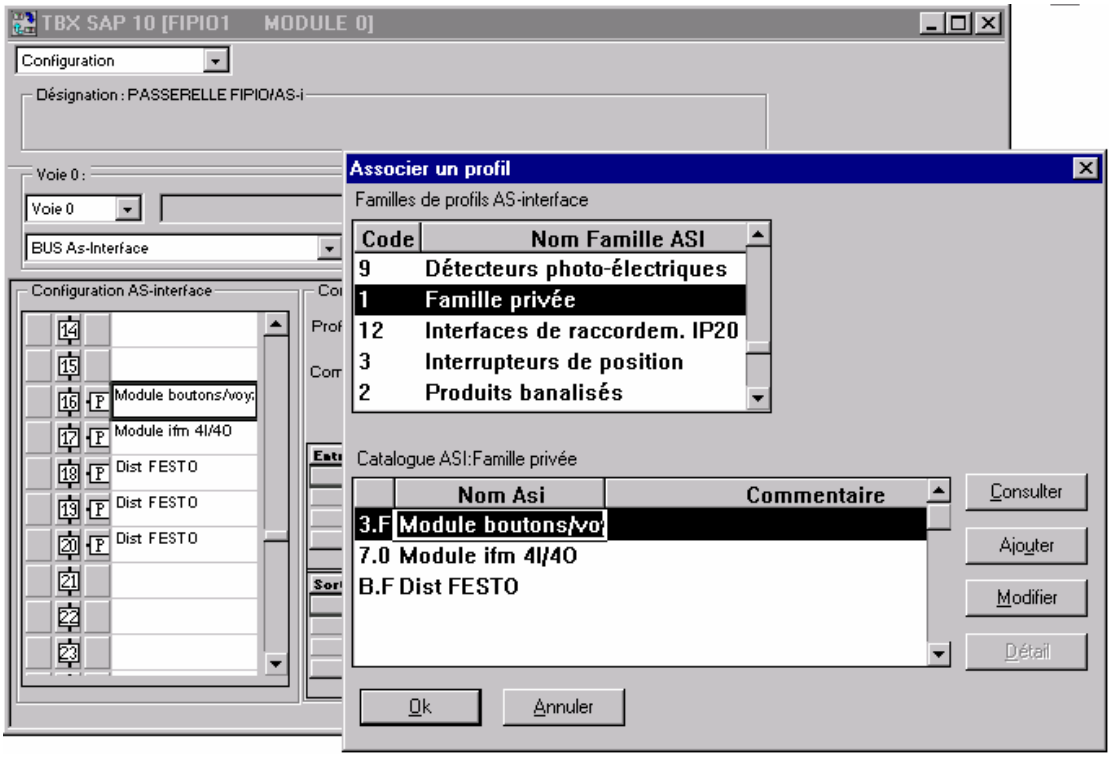
- Les deux modules d'entrées et de sorties n'occupent qu'un seul nœud du réseau car ils sont reliés l'un à l'autre, ils vont par paire ; c'est pourquoi ils portent chacun un numéro supplémentaire (0 et 1) qui permet de les distinguer.
- Au nœud 2 du réseau où sont reliés les deux modules d'E/S, on trouve un module TBX LEP20 qui est un module fictif de communication ajouté par le logiciel lorsque l'on choisit les modules d'E/S.

4.5. Configuration de la passerelle et du réseau AS-i

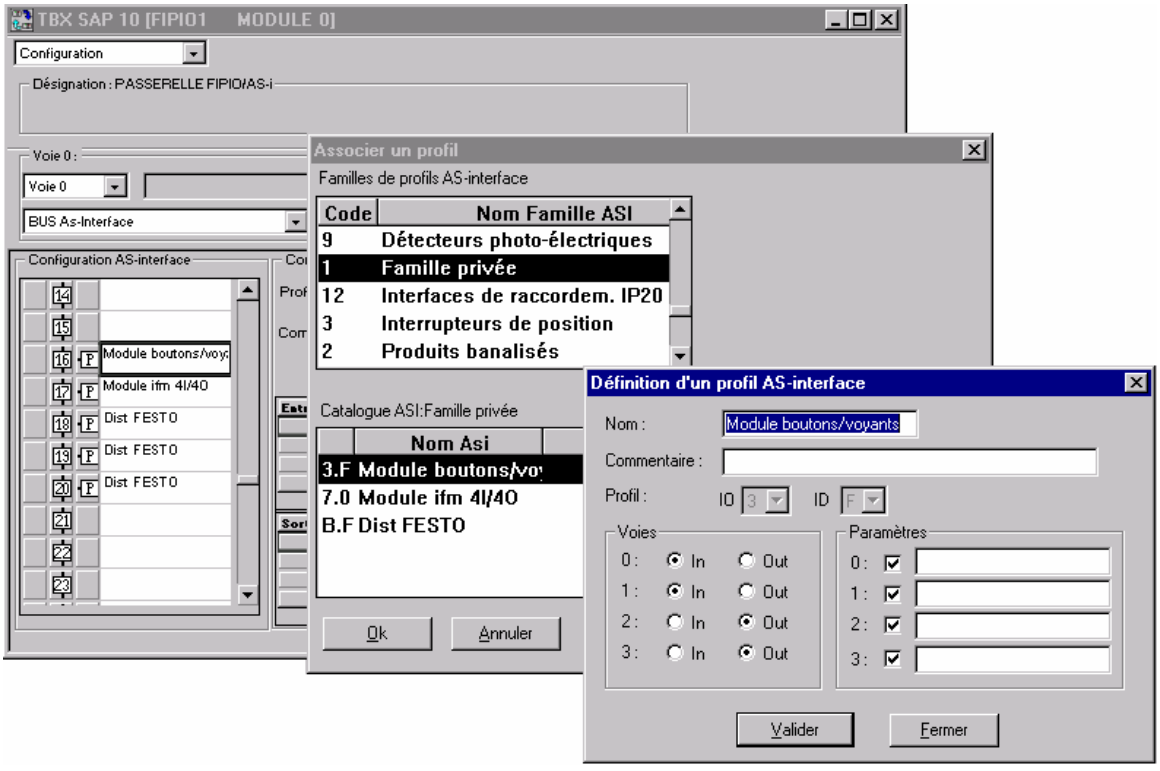
Lorsque l'on clique sur la passerelle, il est possible de configurer le réseau AS-i qui lui est attaché.



Il suffit d'ajouter les modules AS-i présents sur le réseau dans le cadre « Configuration AS-Interface ». Pour ce faire, on clique sur le carré blanc contenant le numéro correspondant à l'adresse AS-i du module (configurée physiquement dans le module) et on sélectionne le type de module adéquat dans la fenêtre « Associer un profil » qui apparaît. Si on ne trouve pas le module recherché dans les familles de produits proposées, il est possible de définir ses propres modules (c'est ce que nous avons fait).



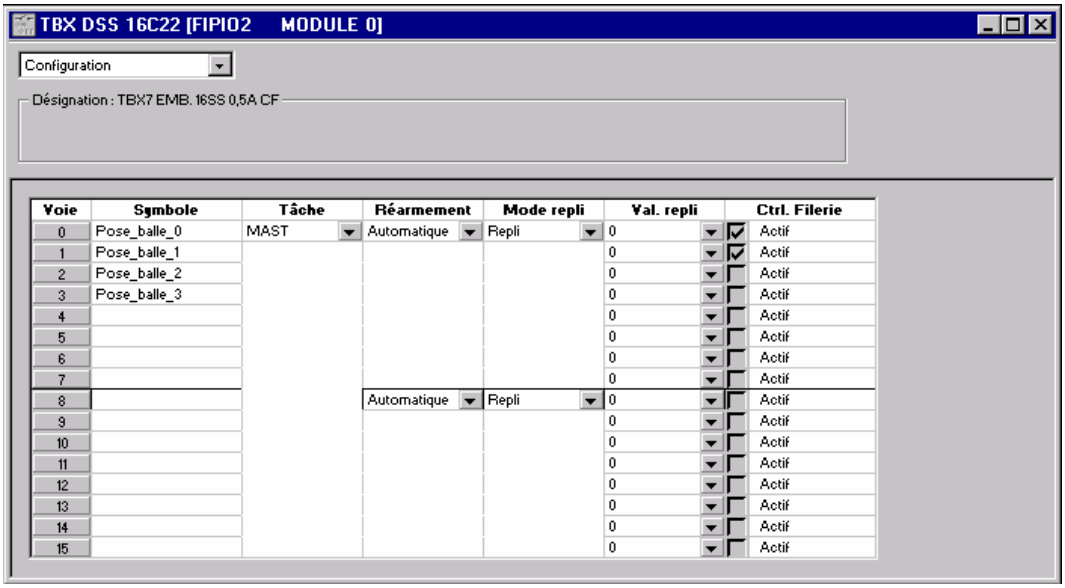
Pour définir ses propres modules AS-i, on sélectionne la famille privée et on clique sur « Ajouter » pour faire apparaître la fenêtre « Définition d'un profil AS-interface ». On sélectionne alors les voies d'entrées et de sorties du module en fonction de ses caractéristiques (voir annexe pour les caractéristiques des trois types de modules que nous avons défini).



4.6 Les modules d' entrées / sorties déportées

4.6.1 Les modules de sorties déportées TBX DES 16C22

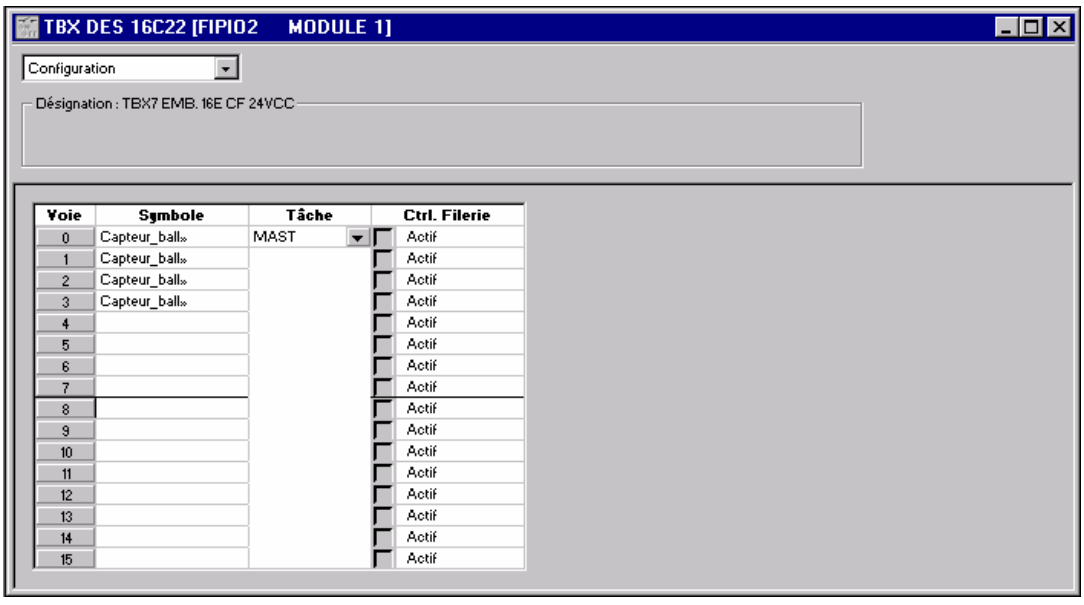
Lorsque l'on clique sur un module de sorties déportées, la fenêtre suivante apparaît :



On peut y définir des mnémoniques et y activer des contrôles de filerie qui permettent de détecter les anomalies de raccordement (coupure de fil, etc).

4.6.2 Les modules d'entrées déportées TBX DES 16C22

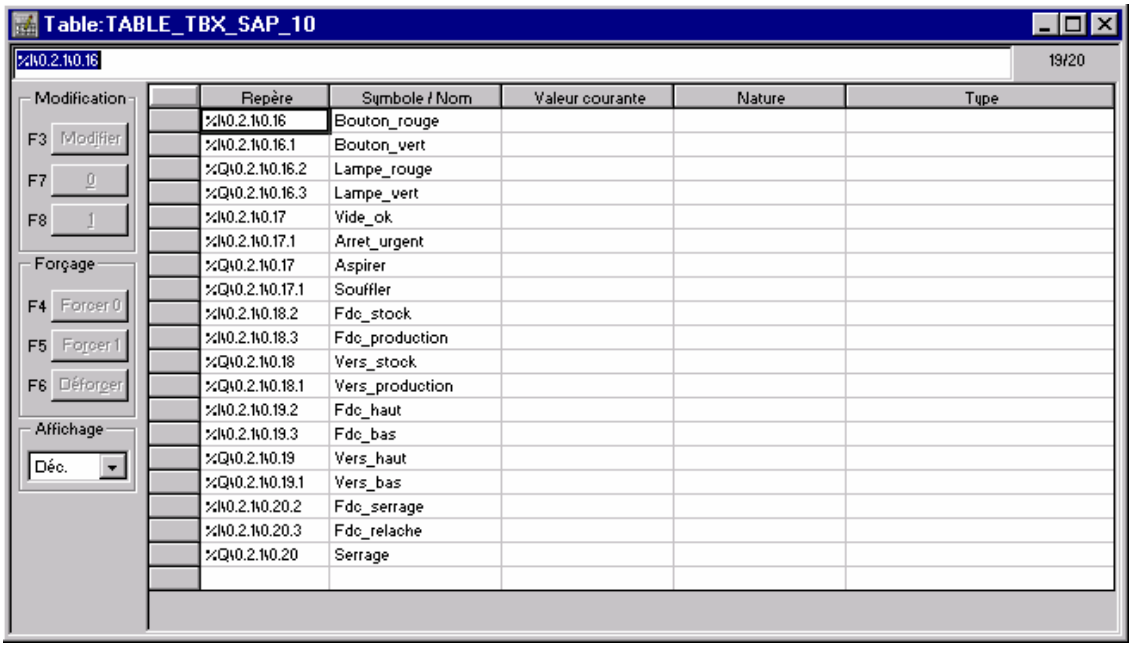
Lorsque l'on clique sur un module de sorties déportées, la fenêtre suivante apparaît :



On peut y définir des mnémoniques et y activer des contrôles de filerie qui permettent de détecter les anomalies de raccordement (**ces contrôles ne fonctionnent pas avec les capteurs présents sur le réseau**, donc nous ne les utiliserons pas).

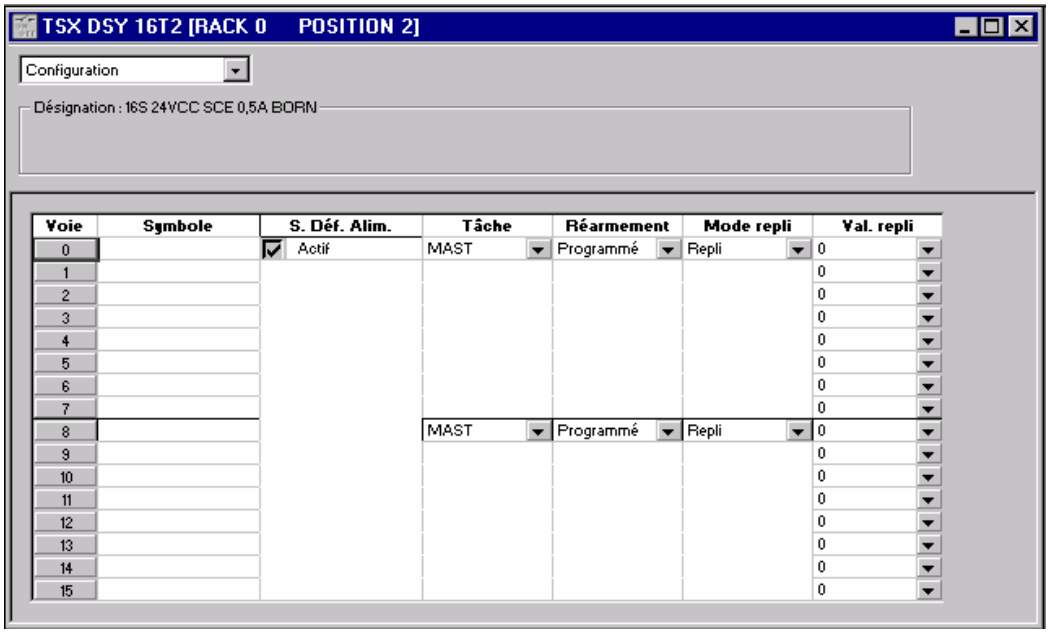
4.7. Visualisation et forçage de variables

Sous la rubrique « Table d'animation » du Navigateur Application, il est possible de définir des tableaux de variables dont on peut ensuite visualiser ou forcer les valeurs.

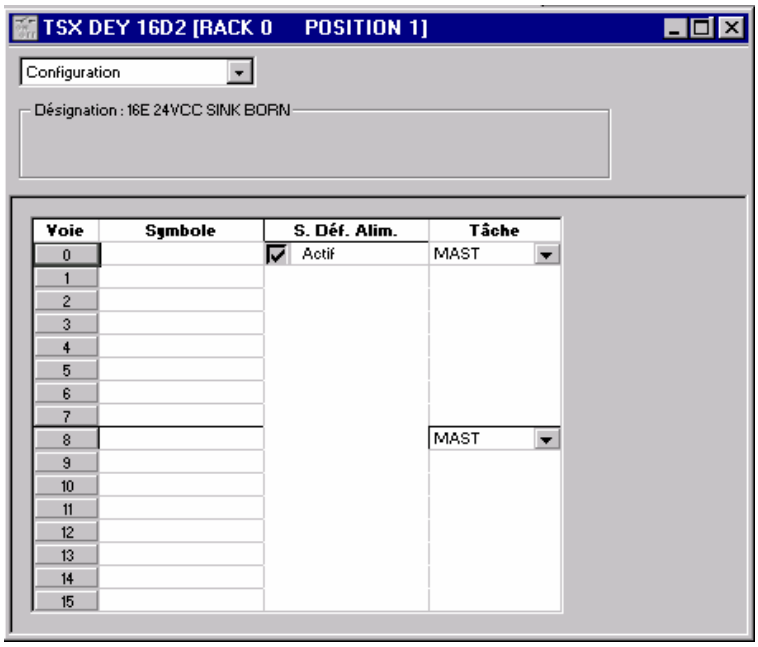


4.8. Annexes

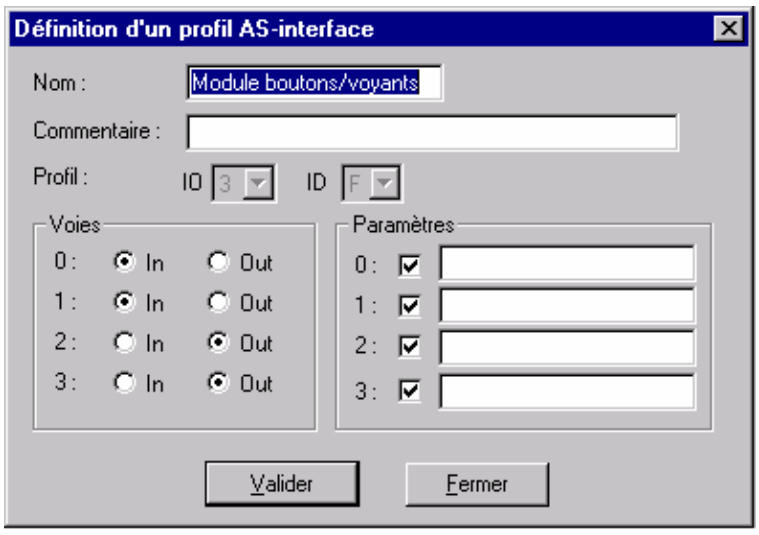
Configuration du module des sorties directes



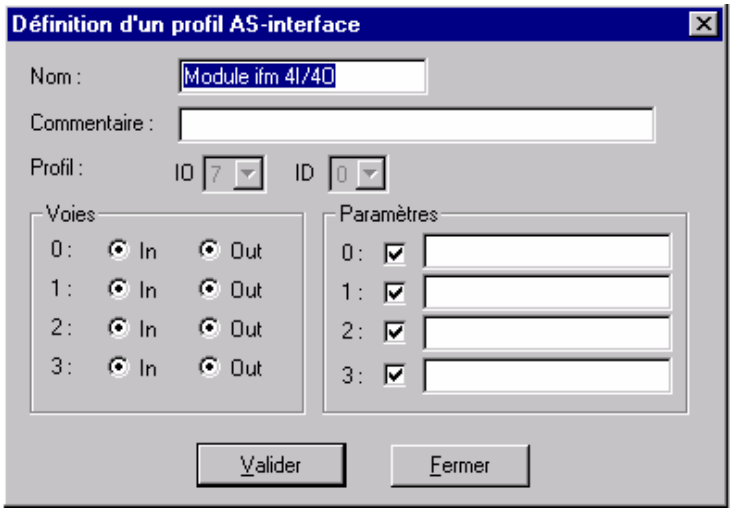
Configuration du module d’entrées directes



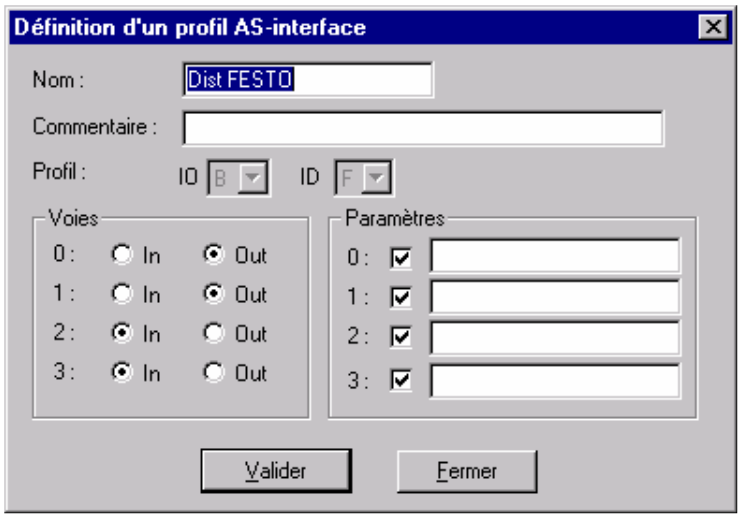
Définition du module AS-i IFM 2 boutons/2 voyants



Définition du module AS-i IFM 4 I/4 O



Définition du module AS-i Distributeur FESTO



Liste des variables Fipio

Entrées déportées

Repère	Symbole / Nom
%I0.2.2\1.0	Capteur_balle_0
%I0.2.2\1.1	Capteur_balle_1
%I0.2.2\1.2	Capteur_balle_2
%I0.2.2\1.3	Capteur_balle_3
%I0.2.2\1.4	
%I0.2.2\1.5	
%I0.2.2\1.6	
%I0.2.2\1.7	
%I0.2.2\1.8	
%I0.2.2\1.9	
%I0.2.2\1.10	
%I0.2.2\1.11	
%I0.2.2\1.12	
%I0.2.2\1.13	
%I0.2.2\1.14	
%I0.2.2\1.15	

Sorties déportées

Repère	Symbole / Nom
%Q\0.2.2\0.0	Pose_balle_0
%Q\0.2.2\0.1	Pose_balle_1
%Q\0.2.2\0.2	Pose_balle_2
%Q\0.2.2\0.3	Pose_balle_3
%Q\0.2.2\0.4	
%Q\0.2.2\0.5	
%Q\0.2.2\0.6	
%Q\0.2.2\0.7	
%Q\0.2.2\0.8	
%Q\0.2.2\0.9	
%Q\0.2.2\0.10	
%Q\0.2.2\0.11	
%Q\0.2.2\0.12	
%Q\0.2.2\0.13	
%Q\0.2.2\0.14	
%Q\0.2.2\0.15	

Variables AS-i

Repère	Symbole / Nom
%I\0.2.1\0.16	Bouton_rouge
%I\0.2.1\0.16.1	Bouton_vert
%Q\0.2.1\0.16.2	Lampe_rouge
%Q\0.2.1\0.16.3	Lampe_vert
%I\0.2.1\0.17	Vide_ok
%I\0.2.1\0.17.1	Arret_urgent
%Q\0.2.1\0.17	Aspirer
%Q\0.2.1\0.17.1	Souffler
%I\0.2.1\0.18.2	Fdc_stock
%I\0.2.1\0.18.3	Fdc_production
%Q\0.2.1\0.18	Vers_stock
%Q\0.2.1\0.18.1	Vers_production
%I\0.2.1\0.19.2	Fdc_haut
%I\0.2.1\0.19.3	Fdc_bas
%Q\0.2.1\0.19	Vers_haut
%Q\0.2.1\0.19.1	Vers_bas
%I\0.2.1\0.20.2	Fdc_serrage
%I\0.2.1\0.20.3	Fdc_relache
%Q\0.2.1\0.20	Serrage

4.9 PRESENTATION DU LANGAGE TEXTE STRUCTURE (ST)

4.9.1 Les types de variables

Adressage des bits

Type	*Adresse*	*Nombre maximum*	*Accès en écriture*
Bits d'entrée	% Ix.i		
Bits de sortie	%Qx.i		
Bits internes	%Mi		oui
Bits systèmes	%Si	128	
Bits extraits de mots	ex. %MW10:X5		
Bits d'étapes (progr. grafcet)	%Xi		oui

x=emplacement de la carte d'E/S sur le rack de l'automate

Adressage des mots

%	**M, K ou S**	**B, W, D ou F**	**i**
Symbole	**Type d'objet**	**Format**	**Numéro**
	M = interne	**B** = octet	
	K = constante	**W** = mots	
	S = système	**D** = double mots	
		F = flottants	

Adressage des objets de modules entrées/sorties dans des racks

Dans les équipements FIPIO, l'adressage des principaux objets bits et mots d'entrée/sortie est défini par les caractères suivants:

%	**I,Q, M** ou **K**	**X, W, D** ou **F**	**\x.y.z**	**\m.v.r**
Symbole	**Type d'objet**	**Format**	x = n° emplacement du processeur (tjrs 0 ou 1)	m = n° du module dans l'équipement (0 pour la base, 1 pour l'extension)
	I = entrée	X = booléen	y = n° de la voie FIPIO intégrée (tjrs 2)	
	Q = sortie	W = mots		
	M = information en lecture-écriture	D = double mots	z = n° du point de connexion de l'équipement déporté (adresse FIP)	v = n° de la voie dans le module
	K = information en configuration	F = flottants		r = rang de l'objet

Exemple:	%I\0.2.1\0.16	bouton rouge sur notre poste balle
	%Q\0.2.1\0.16.2	lampe rouge correspondante

Syntaxe

- Phrases LITTERALE (Literal language) comportant 512 caractères au maximum.
- La phrase commence toujours par un point d'exclamation "!" qui apparaît lorsqu'elle est validée.
- La phrase se termine par un point-virgule ";".
- Une phrase d'actions peut comporter une ou plusieurs actions séparées par un point virgule ";".

4.9.2 Instructions sur bit

NOT	Inversion logique du Bit
AND	ET logique
OR	OU logique
( )	Parenthèses
:= Bit	Assignation au Bit
SET Bit	Set (=mise à ‘ 1 ’) du Bit
RESET	Reset (=mise à ‘ 0 ’) du Bit
RE(Bit)	Détection du front montant du Bit
FE(Bit)	Détection du front descendant du Bit

4.9.3 Instructions sur mot

Word	Accès au mot
Word:=value	Assignation à un mot
INC	Incrémentation du mot
DEC	Décrémentation du mot

Comparaisons

>, >=, <, <=, =, <>

Attention: mettre les arguments entre [ ]

Opérations arithmétiques

+, -, *, /

Opérations logiques sur les mots

AND, OR, XOR, CPL (compléments)

4.9.4 Temporisations

Arrêt de la tempo

STOP %TM0

Armement de la tempo

%TM0.P:= 20; DOWN %TM0

Démarrage tempo

START %TM0

Utilisation de la tempo

%Q2.2:=(%MW11=20);

4.9.5 Appel aux sous routines

Routine principale

MAIN

Sous routines

SRn

Appel aux sous routines

SRn;

Fin de sous routine (retour)

RETURN

4.9.6 Sauts et labels

```
JUMP        %L{Label de ligne}
…
%L{Label de ligne}
```

4.9.7 Structures de contrôle

Conditions:

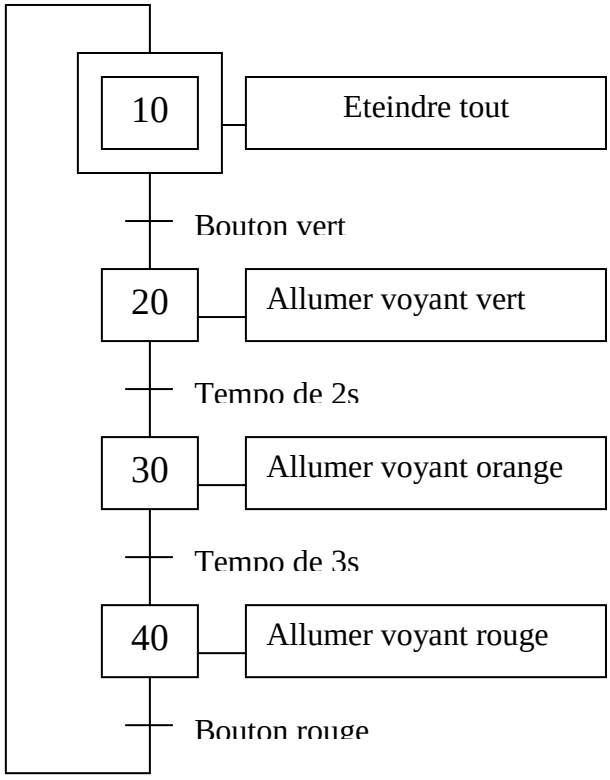
```
!  IF  Condition  THEN  Action; END_IF;
!  IF  Condition  THEN  Action 1  ELSE  Action2; END_IF;

Exemples:  !  IF  Condition  THEN  SET B6; END_IF;
           !  IF  Condition  THEN  SET B6  ELSE  RESET B4; END_IF;
```

Structure itérative

```
! WHILE  Condition  DO  Action; END_WHILE;
```

4.10 Exemple de grafcet

4.10.1 Grafcet de l'exemple



4.10.2 Listing de l'exemple

Tâche maître

SR0 ;
SR1 ;

Sous-routine SR0 : Evolution du grafcet

```
%M1 :=%TM1.Q ;
%M2 :=%TM2.Q ;
IF %S1 OR %S0 OR %S13 THEN
     %MW11:=10;
END_IF;

IF (%MW11=10) AND %I1.13 THEN
     %MW11:=20;JUMP %L0;
END_IF;

IF (%MW11=20) AND NOT %M1 THEN
     %MW11:=30;JUMP %L0;
END_IF;
```

```
IF (%MW11=30) AND NOT %M2 THEN
     %MW11:=40;JUMP %L0;
END_IF;

IF (%MW11=40) AND NOT %I1.14 THEN
     %MW11:=40;JUMP %L0;
END_IF;

%L0:
RETURN;
```

Sous-routine SR1 : Action

```
IF (%MW11=10) THEN
     %TM0.P:=2;DOWN %TM1;
     %TM1.P:=3;DOWN %TM2;
END_IF;

IF (%MW11=20) THEN
     START %TM0;
END_IF;
%Q2.2:=(%MW11=20);

IF (%MW11=30) THEN
     START %TM1;
END_IF;
%Q2.1:=(%MW11=30);

%Q2.0:=(%MW11=40);
```

5. ROBOT ABB IRB1400

5.1 PRÉAMBULE

Le texte qui suit est une présentation relativement succincte du robot ABB IRB1400. Il est évident que pour des informations complètes le lecteur doit se référer aux manuels du constructeur. Pour un accès rapide au manuel de référence, on se souviendra que

- les chapitres importants sont les chapitres 6, 7 et 8,
- les aspects de sécurité sont couverts au chapitre 3,
- les types de données, les instructions et les fonctions sont donnés par ordre alphabétique.

5.2 PRESENTATION DU ROBOT

5.2.1 Structure mécanique

Le robot dont on dispose au laboratoire est de type anthropomorphe à 6 degrés de liberté. La structure anthropomorphe est formée d'une chaîne cinématique ouverte simple et de 6 degrés de liberté placés aux axes de 6 liaisons de type rotoïde (charnière) comme représenté sur la figure suivante.

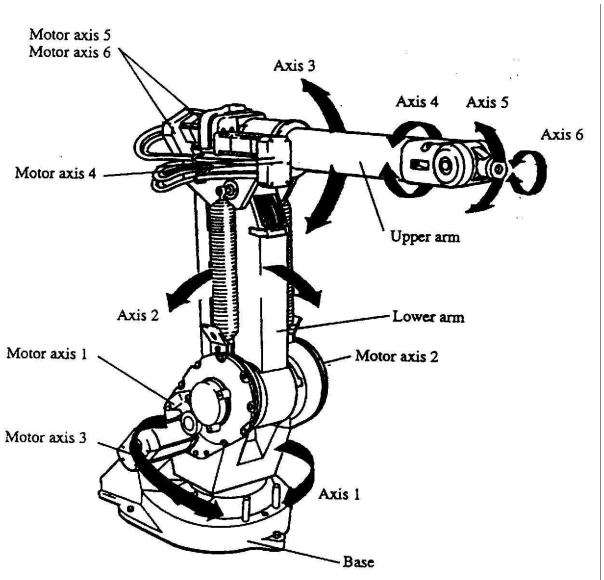


Fig 5.1 Axes du robot IRB 1400

Les trois premiers degrés de liberté en partant de la base forme le *bras*. Le bras a pour objet de positionner le poignet.

Les trois derniers degrés de liberté sont les degrés de liberté du *poignet*. Ils ont pour objet de fixer l'orientation de l'outil dans une configuration désirée.

L'*effecteur* ou l'*outil* est le dispositif qui se trouve à l'extrémité du robot et qui est destiné à réaliser le travail envisagé. Par exemple il peut s'agir d'une pince, d'une torche de soudage, etc.

5.2.2 L'espace de travail:

L'*espace de travail* d'un robot est l'ensemble des positions et orientations accessibles par un repère particulier lié à l'organe terminal du robot lorsque les paramètres articulaires prennent toutes les valeurs permises.

Il s'agit donc d'un espace à 6 dimensions (3 pour la position et 3 pour décrire l'orientation). Cet espace est donc difficile à visualiser. C'est pourquoi on rencontre souvent des représentations de l'espace de *travail en position (*ou *reachable workspace* en anglais*)* qui est l'ensemble des positions accessibles qu'un point particulier de l'organe terminal du robot peut atteindre pour au moins une orientation lorsque les paramètres articulaires prennent toutes les valeurs permises. Cet espace est donc plus grand que l'espace de travail car il suffit que l'on puisse atteindre le point avec une orientation au moins.

Il convient donc de bien distinguer

- L'espace primaire (*dexterous workspace*) qui est l'ensemble des positions de l'espace de travail qui peuvent être atteintes avec toutes les orientations possibles.
- L'espace secondaire qui est l'ensemble des positions de l'espace qui ne peuvent être atteintes que pour certaines orientations données.

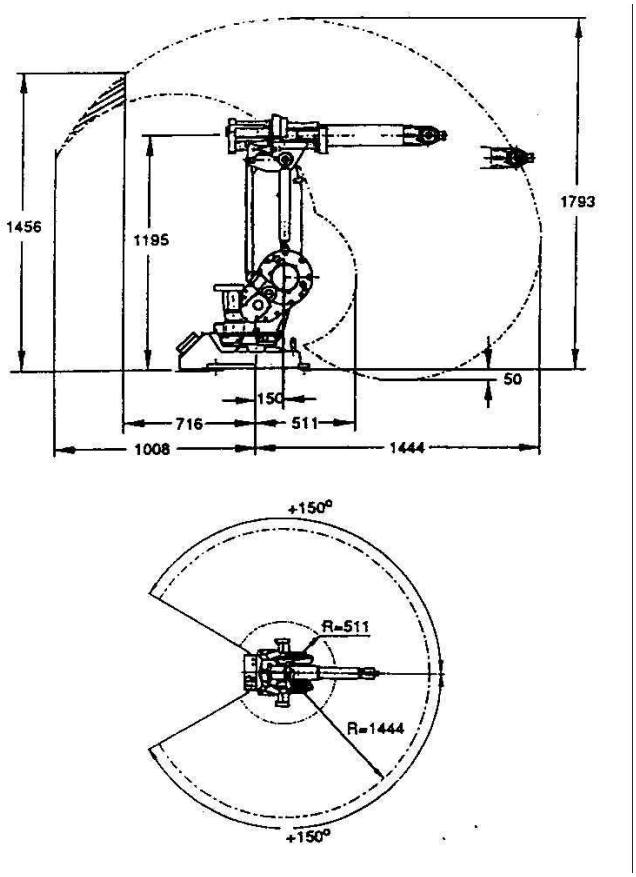


Fig 5.2 : Espace de travail du robot IRB1400

Lorsque le travail demande au robot de pouvoir atteindre des points avec une certaine orientation, on parle d'espace de travail à orientation donnée.

Pour le robot IR1400, l'espace de travail en position est donné aux figures 5.2.

5.2.3 Diagramme de charge

La charge de travail du robot est évidemment limitée par sa constitution mécanique et par la capacité de ses actuateurs et des éléments du système de transmission. La figure suivante montre la capacité du robot IRB1400. Bien évidemment la charge, mais aussi son positionnement (excentricité par rapport au poignet) sont limitatifs.

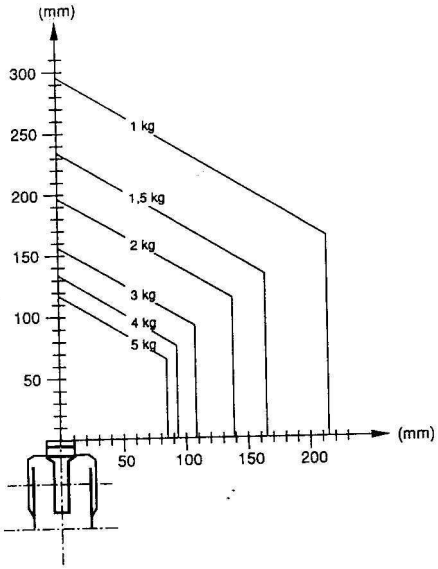


Fig 5.3: Diagramme de charge du robot IRB1400

5.3 SYSTEMES DE COORDONNEES

5.3.1 Repère de l'outil

L'accomplissement du travail robot nécessite la connaissance à tout instant de la positions et de l'orientation de l'outil monté à l'extrémité du bras du robot. Plus exactement on contrôle la position et l'orientation d'un repère particulier sur l'outil appelé le centre d'outil (CDO) ou "Tool Center Point" (TCP). Pour réaliser le travail, la position et l'orientation de ce repère outil doivent être rapportées dans différents systèmes de coordonnées.

Le Centre d'Outil est un point de référence de l'outil, en rapport avec sa géométrie ou mieux en repart avec sa fonction de travail (centre de pinçage, pointe de la torche soudure, etc.) . En ce qui concerne le repère de l'outil, on appelle:

- l'axe "z" de l'outil, l'axe d'approche de l'outil qui représente l'axe selon lequel on réalise une approche de l'outil pour effectuer un travail.
- l'axe "y" de l'outil, l'axe d'orientation de l'outil qui permet de définir son orientation par rapport à la pièce sur laquelle on veut travailler.
- l'axe "x" est obtenu par ortho normation des axes "z" et "y", autrement en effectuant le produit vectoriel "y x z".

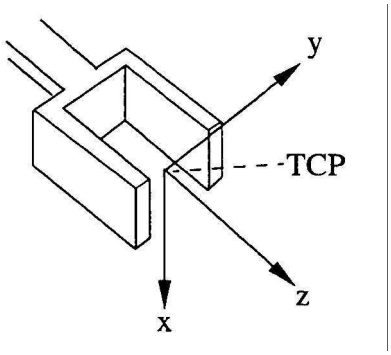


Fig 5.4 Définition du centre d'outil et du repère de l'outil
(Illustration dans le cas d'une pince)

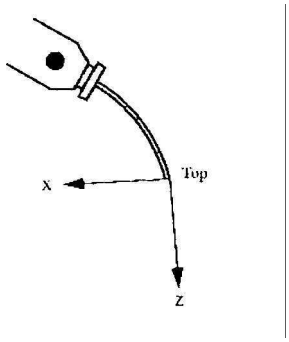


Fig 5.5 Définition du centre d'outil et du repère de l'outil
(Illustration dans le cas d'une torche de soudage)

5.3.2 Repère Base

Dans des applications simples, les coordonnées du TCP peuvent être directement définies par rapport au repère attaché à la base du robot.

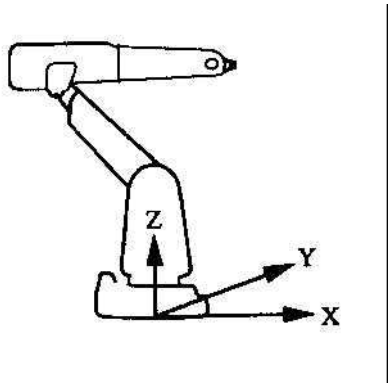


Fig 5.6 : Repère "Base" du robot IRB1400

5.3.3 Repère des coordonnées universelles ou World ou Atelier

Le repère lié à la base du robot est défini par rapport au repère de l'atelier. Dans le cas où celui-ci n'a pas été défini, les deux coïncident.

Définir un Repère Atelier peut s'avérer fort utile, par exemple, dans le cas de robots se déplaçant sur rail, de robots suspendus ou lorsqu'une communication entre robots travaillant dans le même espace doit être réalisée.

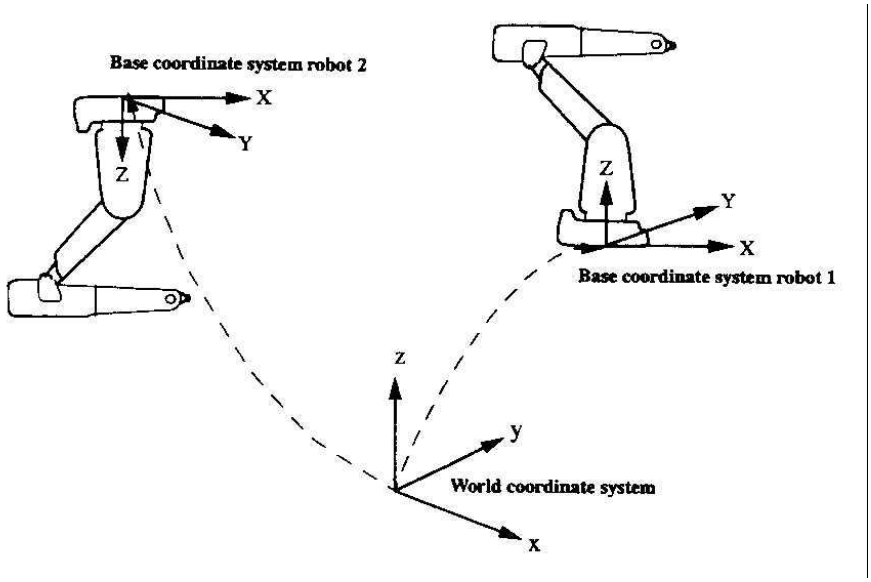


Fig 5.7: Repère atelier et repère lié au robot

5.3.4 Repère Utilisateur

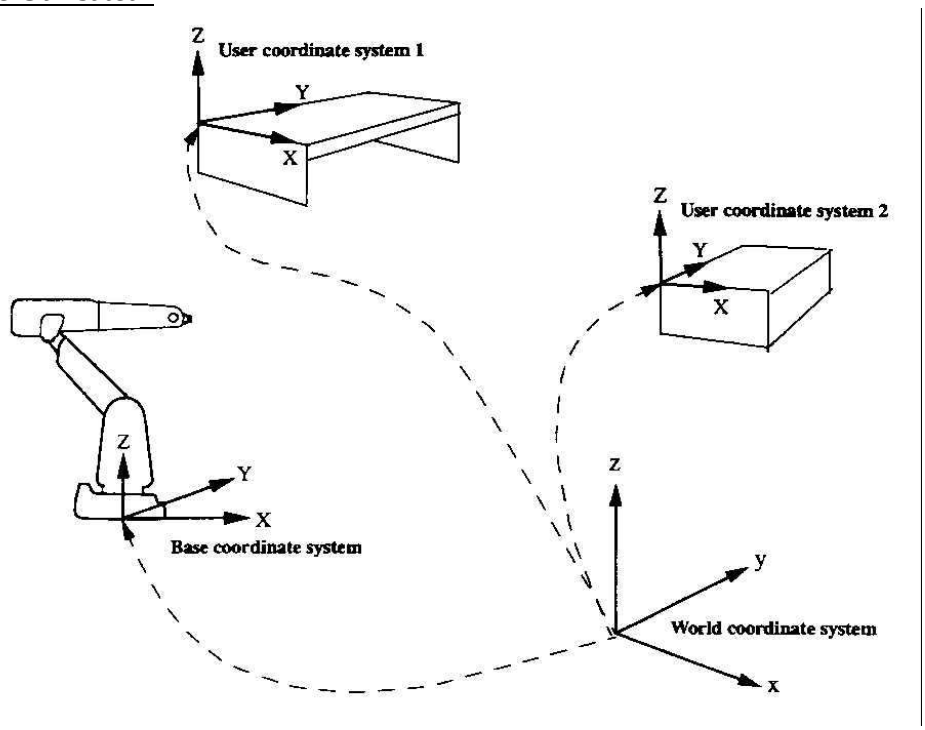


Fig 5.7 Repère utilisateur ou repère de travail

Un Repère Utilisateur servira principalement à définir un plan de travail. Une même séquence pourra ainsi être effectuée dans différents plans de travail, sans que le programme ne doive être réécrit.

5.3.5 Repère Objet

Le repère lié à l'objet sera défini par rapport au Repère Utilisateur. Ainsi, par exemple, une même routine de soudage pourra être appliquée à plusieurs pièces fixées sur un même plan de travail.

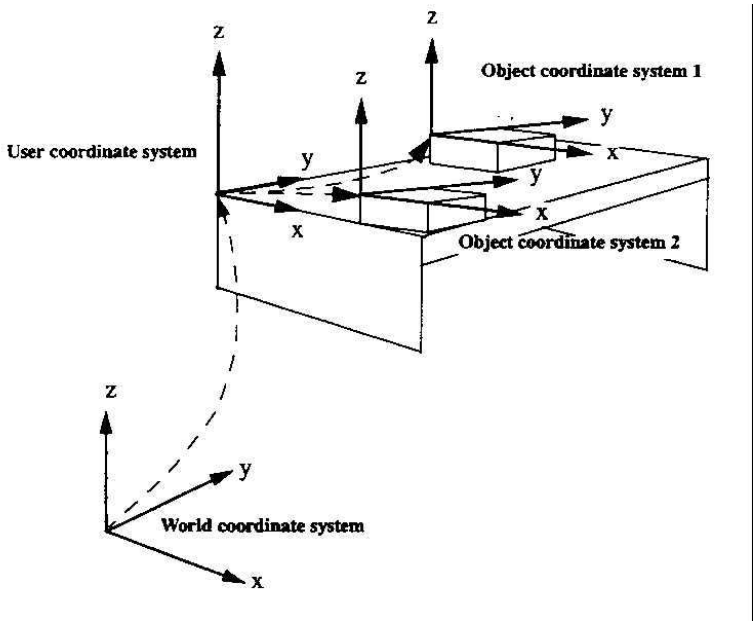


Fig 5.8: Repère objet

5.3.6 Définition des Repères Objet et Utilisateur

Les Repères Objet et Utilisateur sont définis par une méthode "trois points". Le système de coordonnées est défini par 3 points (deux sur l'axe x, un sur l'axe y) que l'on approche à l'aide du CDO (TCP) d'un outil préalablement défini dans la mémoire du robot.

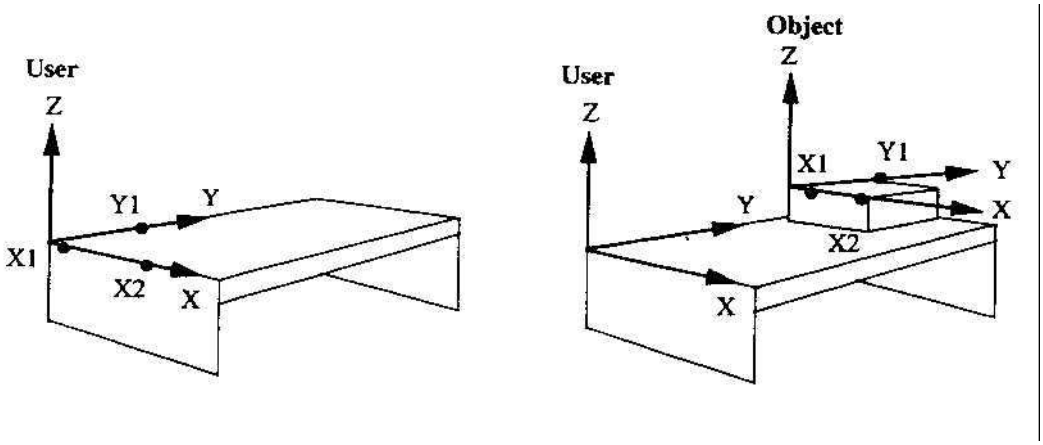


Fig. 5.9: Définition des repères Utilisateur et des repères Objet

5.4 DÉFINITION DE L 'OUTIL

5.4.1 Position de l'outil

Rappelons que la position de l'outil définit le système de coordonnées attaché à cet outil, c'est-à-dire son TCP et son orientation.

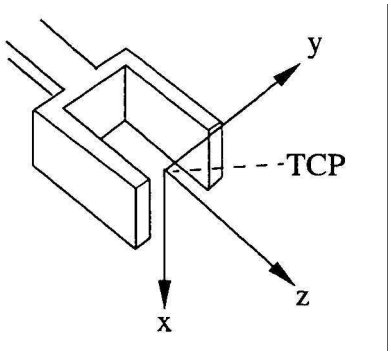


Fig 5. 10 Centre d'outil et du repère de l'outil

Par défaut, le centre d'outil et un repère sont placés au centre du flasque du poignet. Ce repère et ce système d'axes sont appelés *tool0*.

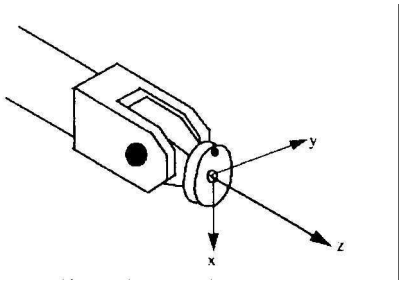


Fig. 5.11 Axes de référence situés sur le flasque du poignet

La procédure de définition de définition de l'outil revient à définir la position et l'orientation opérationnelle de l'outil par rapport au repère par défaut, c.-à-d. celui du flasque.

Si on connaît la géométrie de l'outil, on peut introduire manuellement les données relatives au système de coordonnées de l'outil dans la mémoire du robot.

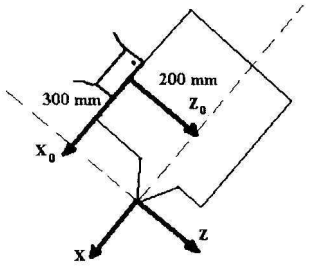


Fig. 5.12: Introduction manuelle du système de l'outil

Souvent, les dimensions de l'outil sont inconnues mais elles peuvent être identifiées et déterminées par une procédure expérimentale utilisant l'aide du robot employé comme d'outil de mesure. La définition expérimentale du système de coordonnées du robot se fait à l'aide d'un point fixe (World Fixed Tip) placé dans l'espace de travail du robot. Le CDO est alors déterminé en approchant le point fixe selon différentes orientations. Ces différentes positions sont appelées les points d'approche. Il existe 3 méthodes pour définir le TCP (CDO et on orientation).

Pour définir entièrement l'orientation de l'outil, deux autres positions doivent encore être définies: l'une sur l'axe "z", l'autre sur l'axe "x". Ces positions sont appelées les points d'élongation.

Méthode à 4 points

Dans ce cas, on réalise une translation (déplacement parallèle) du système d'axes *tool0*. L'orientation du repère reste celle du flasque (à défaut).

Fig. 5.13 : Méthode à 4 points – déplacement du CDO

Méthode à 5 points

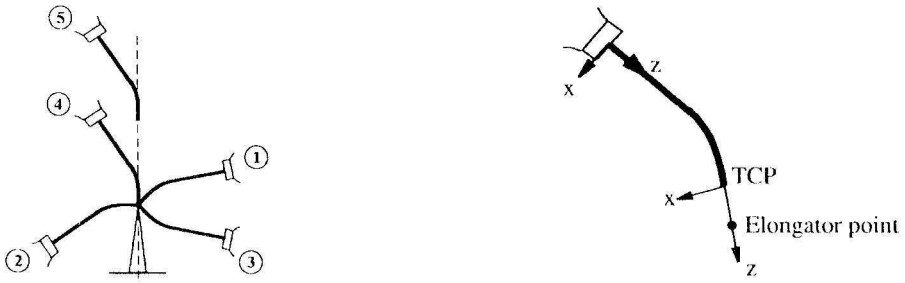


Fig 5.14: Méthode à 5 points: déplacement du CDO et axe d'approche "z"

Dans la méthode à 5 points, 4 servent au déplacement du CDO et tandis que le point d'élongation sert à définir l'axe "z." Les axes "x" et "y" de l'outil seront pris aussi proches que possible de ceux du flasque.

Méthode à 6 points

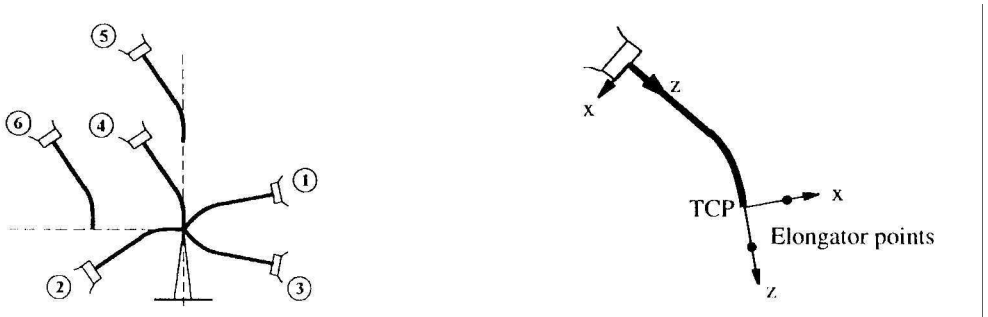


Fig 5.15: Méthode à 6 points: déplacement du CDO, axes d'approche "z" et axe "x"

Dans la méthode à 6 points, les deux points d'élongations servent à définir les axes "z" et "x". L'axe "y" est défini par ortho normation.

Comment créer un nouveau TCP (CDO - Tool)?

1. On va dans la fenêtre "Jogging"
2. On place le curseur sur "tool"
3. On enfonce ensuite la touche "enter"
4. On fait "New" (attention à la déclaration!)
5. On choisit "Define"
6. On choisit une des différentes méthodes.

5.4.2 Propriétés mécaniques de l'outil

Les propriétés mécaniques de l'outil regroupent toutes données suivantes:

- Poids
- Coordonnées du centre de gravité dans le référentiel outil
- Moments principaux d'inertie
- Moments d'inertie giratoire

5.5 CONSOLE DE CONTROLE

La console de contrôle, aussi appelée Teach Pendant (TP). Elle est l'outil principal de dialogue avec le robot dans la pratique.

Remarquez
- La fonction main morte et le bouton d'arrêt d'urgence
- Les touches de menu
- Les touches de mouvement

The teach pendant is outlined briefly in Figure 4.

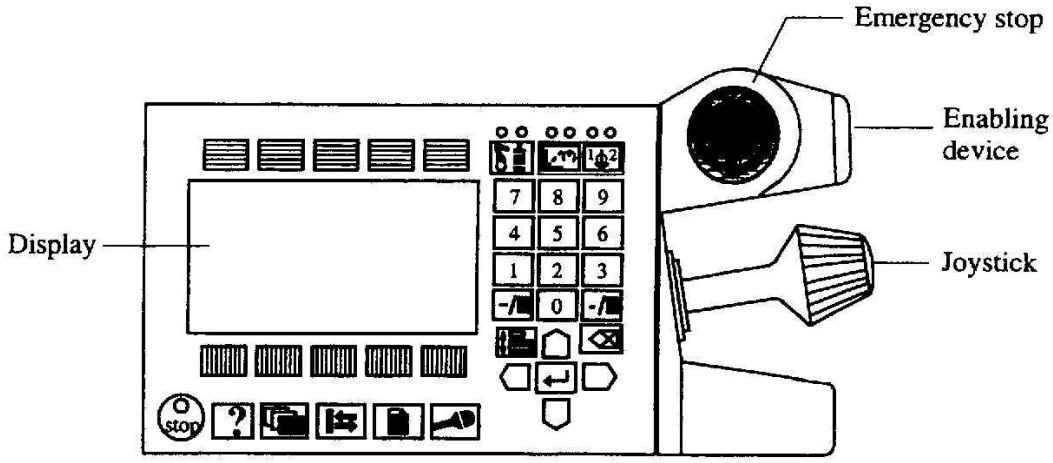
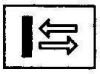


Figure 4 The teach pendant is used to operate the robot.

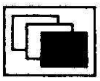
Fig 5.16: Teach pendant

Jogging: Used to jog the robot.

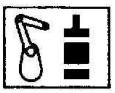
Program: Used to program and test.



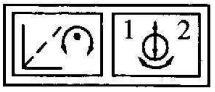
Inputs/Outputs: Used to manually operate the input and output signals connected to the robot.



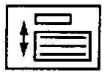
Misc.: Miscellaneous; other windows, i.e. the System Parameters, Service, Production and File Manager windows.



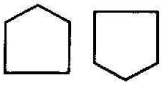
MotionUnit: Press to jog the robot or other mechanical units.



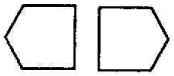
MotionType: Press to select how the robot should be jogged.



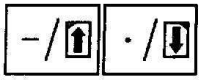
List: Press to move the cursor from one part of the window to another (normally separated by a double line).



Up and Down arrows: Press to move the cursor up or down.



Right and Left arrows: Press to move the cursor to the right or left.



Previous/Next page: Press to see the next/previous page.

Stop: Stops program execution.

Help: Press to call up a help menu directly to the display (no information in this version of the robot).

Enter: Press to input data.

Delete: Deletes the data selected on the display.

Menu keys: Press to display menus containing various commands.

Function keys: Press to select the various commands directly.

Fig 5.16 : Explication des touches du Tecah Pendant

5.6 PROGRAMMATION DU ROBOT (Langage rapid et S4C)

5.6.1. Structure d'un programme

La mémoire du robot est structurée de la manière suivante:

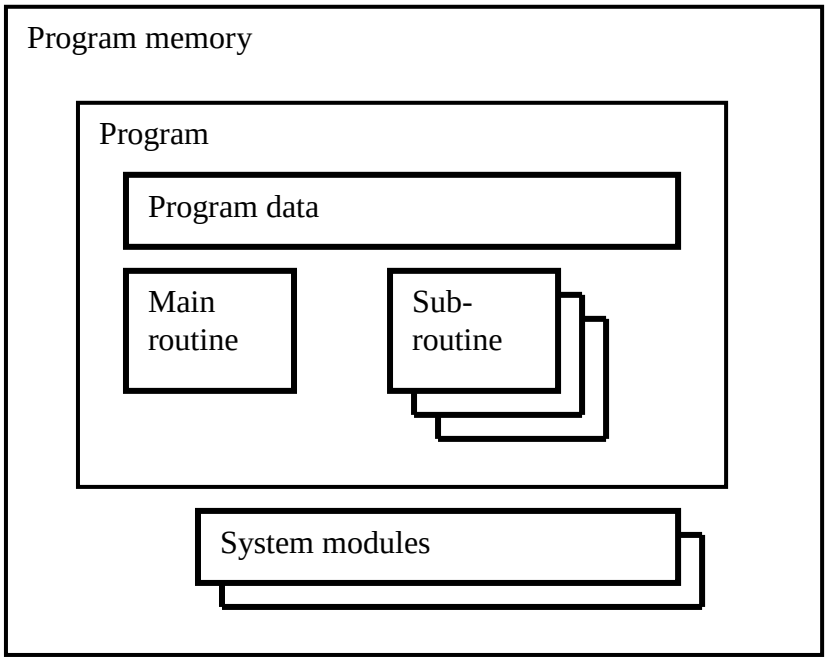


Fig 5.18 : Structure de la mémoire et des programmes

La section *Program data* contient toutes les positions, les variables ou systèmes de coordonnées utilisées dans le programme.

La *Main routine* est la routine de départ du programme ou programme principal. De la routine principale, on peut appeler les autres sous-routines éventuellement en cascade.

Les *Sub-routines* permettent de structurer la programmation de manière à rendre le programme modulaire, structuré. Cela facilite aussi la compréhension et les phases de test ou plus tard de modification. Cela permet aussi d'isoler des parties de code récurrentes et de les appeler à plusieurs reprises sans devoir multiplier la programmation. On peut tester chaque sous-routine individuellement. Pour cela il suffit de se placer dans la sous routine (avec le menu *voir*), puis de sélectionner le menu *tester*.

Les *System modules* sont des programmes qui sont toujours présents dans la mémoire du robot (Routines et paramètres d'installation). Les modules systèmes sont utilisés pour définir des données et des routines communes, spécifiques au système lui-même tels que les outils. Ils ne sont pas inclus dans le programme sauvé. Ceci signifie que toute modification des données et des routines du module système affecte l'ensemble des programmes existants ou chargés par la suite dans la mémoire programme!

5.6.2 Les routines

Les routines contiennent généralement une séquence d'instructions que l'on peut isoler. Les routines sont particulièrement efficaces pour structurer le code. Cela permet aussi une économie de programmation en cernant des opérations récurrentes.

Il existe trois types de routines: procédure (*Proc*) , fonctions (*Func*) et trap routines (*Trap*).

- Les *procédures* peuvent se définir comme une séquence d'instructions qui réalisent une tâche spécifiée.
- Les *fonctions* renvoient en plus une valeur vers la routine d'appel. Le type des données (num, booléen, etc.) renvoyées par la fonction doit être spécifié lors de sa définition.
- Les *trap routines* sont utilisées pour traiter les interruptions.

Une routine se compose de 4 parties : la déclaration, les données, les instructions et un *Error handler*.

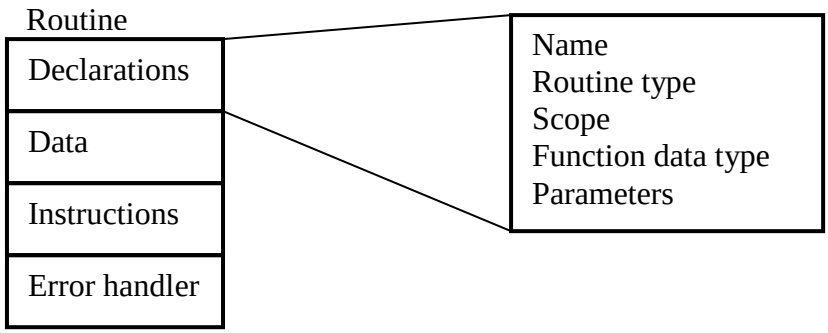


Fig 5.19 : Structure d'une routine

Comment voir les sous-routines existantes?

On va dans la fenêtre **Programme → View → Routines**

Comment créer une sous-routine?

On va dans la fenêtre **Programme → View → Routines → New.**
On change le nom et les différentes caractéristiques en, modifiant les champs appropriés.

Comment appeler une sous-routine?

A l'aide de l'instruction *ProcCall*, on peut appeler une routine dans un programme.

Exercice

Créer une routine, copier les instructions du « main » dans cette routine puis appeler la routine du main.

5.6.3 Instructions de base

Une instruction définit une tâche que le robot doit effectuer lorsque l'instruction est exécutée. Par exemple:

- Déplacer le TCP,
- Activer une sortie,
- Calculer une valeur,
- Faire un saut dans le programme.

Une instruction est composée d'un nom d'instruction (mot clé) et d'une série d'arguments. Le nom désigne l'objet général de l'instruction qui doit être effectuée tandis que les arguments spécifient les caractéristiques de cette tâche. Certains arguments sont requis (compulsory), d'autres peuvent être optionnels (optional). Ces derniers peuvent être omis et une valeur par défaut est spécifiée par le nom général de l'instruction
Les arguments peuvent être:

- des valeurs numériques, e.g. 1
- des chaînes de caractères, e.g. "Waiting for machine"
- des données, e.g. reg2
- des appels à des fonctions, e.g. Abs(reg2)
- des expressions à évaluer, e.g. reg2 + reg3 / 5

Exemple:

SetDO \Sdelay := 0.5,do2,1.
Mettre la sortie do2 à 1 après 0.5 secondes. \Sde1ay est un argument optionnel.

5.6.2.1. Instructions de déplacements

A cause de sa cinématique complexe et fortement non linéaire, effectuer un déplacement d'un robot n'est pas une opération simple. Par exemple, l'opération qui permet d'effectuer un déplacement linéaire de l'outil selon une ligne droite demande un effort important pour la génération et le contrôle du suivit de la trajectoire.

En conséquence, en robotique, on distingue d'abord plusieurs sortes de déplacements.

Le *déplacement articulaire* (MOVEJ avec J pour Joint) est le plus simple et le plus économique. Il consiste à réaliser une simple interpolation linéaire des coordonnées articulaires (axes des moteurs) entre les points de départ et d'arrivée. Le mouvement qui en résulte dans l'espace physique est par contre lui totalement non linéaire. Comme on ne possède aucun contrôle sur la trajectoire effectivement parcourue dans l'espace physique, ce type de mouvement est inadmissible en présence d'obstacles ou pour des mouvements de précision.

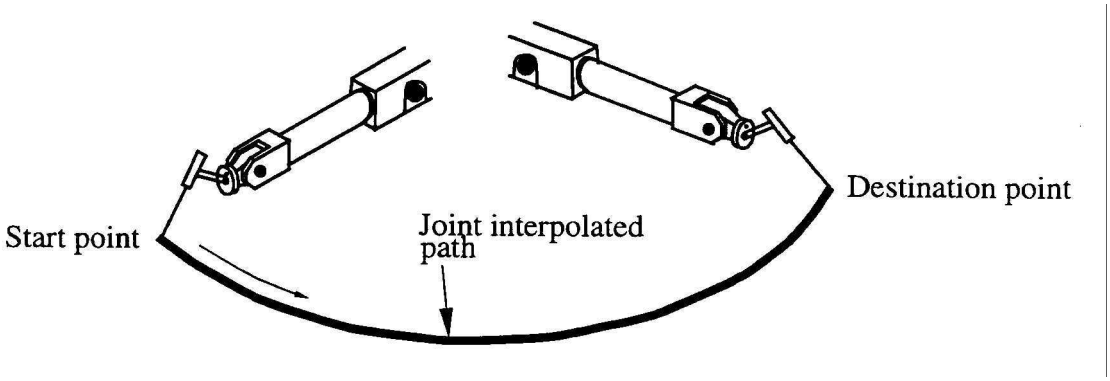


Fig 5.20: Trajectoire articulaire de l'outil (MoveJ)

Le *déplacement linéaire* (MOVEL avec L pour linear) est plus compliqué. Il exige un calcul de cinématique inverse pour générer une trajectoire linéaire au niveau de l'effecteur. Une coordination des mouvements des différentes articulations est requis. Un contrôle de trajectoire est activé. Cette opération est évidemment plus coûteuse qu'une génération de trajectoire articulaire. Elle est néanmoins nécessaire pour s'assurer de la trajectoire parcourue afin d'éviter des collision par exemple.

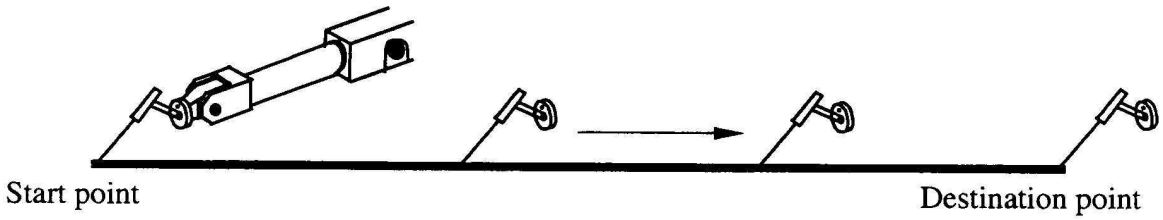


Fig 5.21 : Trajectoire linéaire de l'outil (moveL)

Le *mouvement circulaire* (MOVEC avec C pour circular) est lui aussi un mouvement dans l'espace physique. Il possède donc des caractéristiques semblables au mouvement linéaire. Il est toutefois encore plus coûteux à cause de la courbure de la trajectoire à générer. Il est nécessaire pour certaines opérations de fabrication (soudage de pièces cylindriques, etc.).

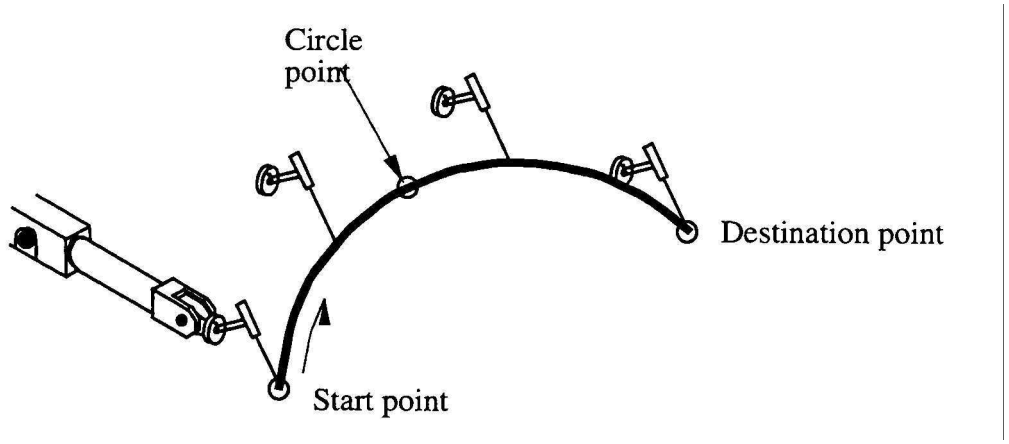


Fig 5.22 : Trajectoire circulaire de l'outil (MoveC)

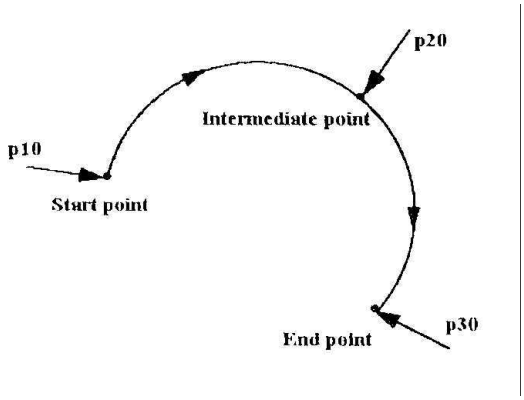


Fig 5.23 : Paramètres d'une commande de trajectoire circulaire MoveC

Syntaxe d'une opération de déplacement

Les instructions suivantes réalisent le déplacement du CDO et de son système d'axes (définis pour l'outil mentionné) à partir du point courant vers le point d'arrivée spécifié (position + orientation).

MOVEL p1, v100, z10, tool1

Instructions: MOVEL pour mouvement linéaire,
MOVEJ pour un mouvement articulaire
MOVEC pour un mouvement circulaire

Point de destination:
- p1 numéro d'un point (robtarget, c'est-à-dire position plus orientation) mémorisé
- * le point d'arrivée est mémorisé dans l'instruction

Vitesse sur la trajectoire en mm/s et degré/s
- v100 ici v=100 mm/s et degrés/s

Taille de la zone de transition pour un point de passage (fly-by point) en mm et en degrés
- z10 , ici z = 10 mm

Outil utilisé
- Tool1 outil 1

Remarques:
- un point mémorisé peut être réutilisé à plusieurs reprises dans le programme
- si on change d'outil, l'instruction génère une trajectoire telle que le CDO et son repère arrive exactement à la même destination.

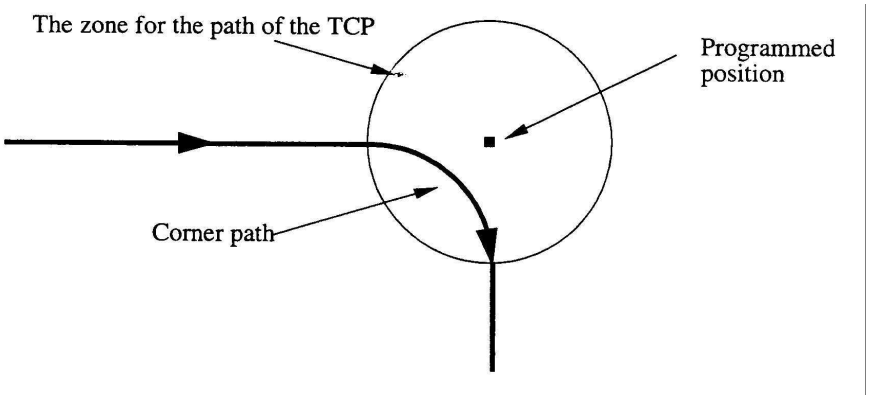


Fig 5.24 : Notion de point de passage et de zone de transition (z)

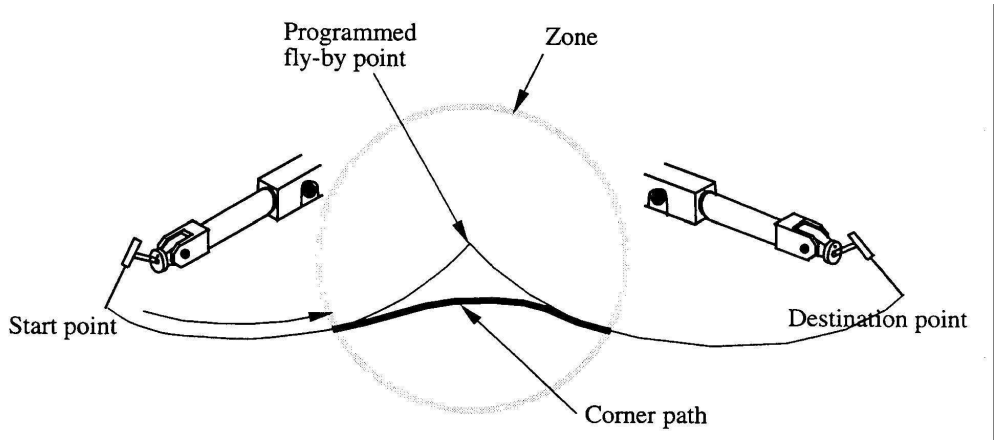


Fig 5.25 : Zone de transition entre deux trajectoires articulaires

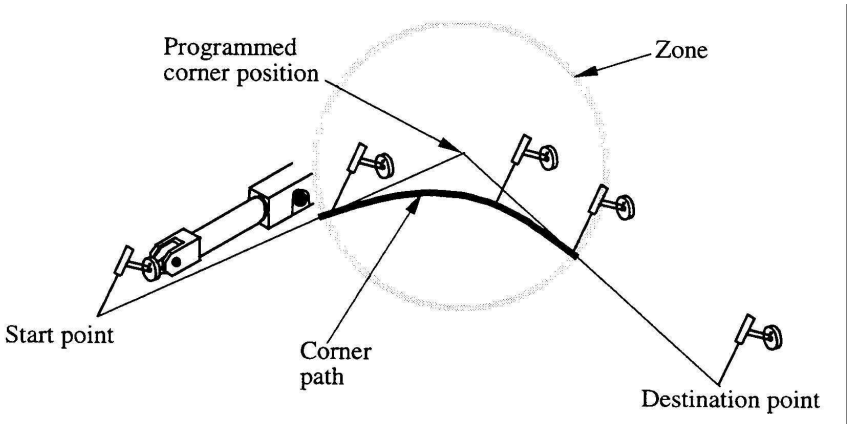


Fig 5.26 : Zone de transition entre deux trajectoires linéaires

Mouvements relatifs

Une manière très agréable de travailler et de construire des procédures récurrentes consiste à utiliser les mouvements relatifs.

La commande "Offs" permet de définir des mouvements relatifs en coordonnées cartésiennes.

MOVEL Offs (point, deltaX, deltaY, deltaZ), v100, z10, tool1

Les écarts deltaX, deltaY, deltaZ, par rapports au point de référence "point" sont donnés en mm. Seuls des écarts en translation par rapport au système d'axes courant (le système de coordonnées de base ou utilisateur) peuvent être définis.

Exemple:

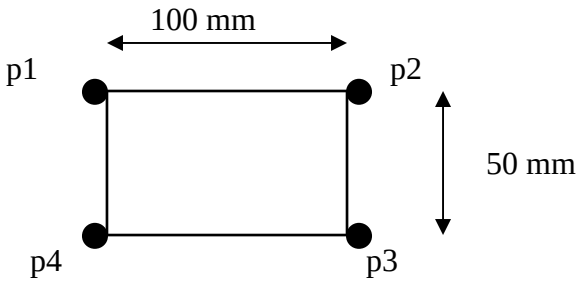


Fig 5.27 : Exemple d'utilisation des mouvements relatifs

MoveL p1, …	MoveL p1, …
MoveL p2, …	MoveL Offs (p1, 100,0, 0), …
MoveL p3, …	MoveL Offs (p1, 100, 50, 0), …
MoveL p4, …	MoveL Offs (p1, 0, 50, 0), …
MoveL p1, …	MoveL p1, …

La commande "Reltool" permet de définir des mouvements relatifs dans le systèmes de coordonnées relatives de l'outil.

MOVEL Reltool (point, Dx, Dx, Dz, [\Rx], [\Ry], [\Rz]), v100, z10, tool1

La commande définit un déplacement de l'outil autour du point de référence "point"
Les déplacements relatifs sont de Dx, Dy, Dz mm le long du système d'axes de l'outil tool1.
De manière optionnelle, on peut également spécifier des rotations Rx, Ry et Rz autour des axes de l'outil.

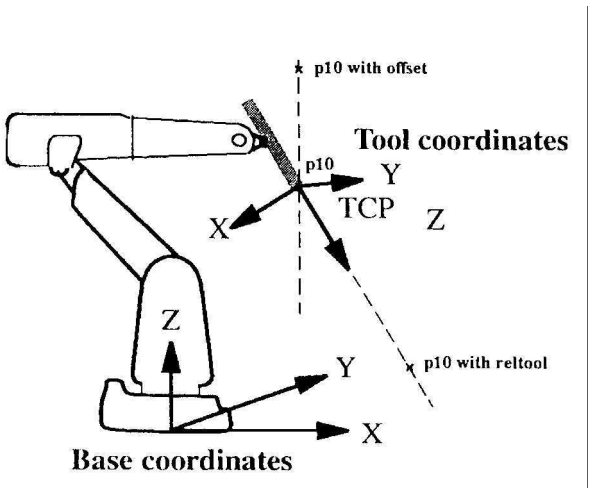


Fig 5.28 : Mouvements relatifs

Exemple: comparaison de Offs et de Reltool

MoveL Offs (p10, 0, 0, 100), v1000, fine, tool1

Le point p10 va trouver 100 mm en z par rapport au point d'origine suivant le système de coordonnées de base (dans ce cas ci) ou utilisateur.

MoveL Reltool (p10, 0, 0, 100), v1000, fine, tool1

Le point p10 va trouver 100 mm en z par rapport au point d'origine en suivant le système de coordonnées de l'outil tool1.

Exercices:

1. Créer 3 routines (carré, cercle, triangle) puis appeler les trois routines à partir du programme principal (main).
2. Demander au robot d'exécuter les trois routines uniquement si l'entrée DI1 est à 1

5.6.2.2 Modifier une sortie

Reset do1

Set ou Reset permet de mettre de façon permanente la sortie digitale do1 à 1 (Set) ou à 0 (Reset).

SetDO do1, high

Met une sortie de type déterminée (DO=digitale, AO = analogique, GO = un groupe) à la valuer désirée (ici high)

Exemples:

Set dol ;	Met la sortie dol à 1.
Reset dol ;	Met la sortie dol à 0.
SetDO do1, 1 ;	Met la sortie dol à 1.
SetDO do l, 0 ;	Met la sortie dol à 0.
lnvertDO dol ;	Inverse l'état de sortie dol
PulseDO dol ;	Inverse l'état de sortie dol pendant 0,2 sec.
PulseDO \Plength :=1, dol ;	Inverse l'état de sortie dol pendant 1 sec (Plength max = 32 sec).

5.6.2.3 Attendre le signal d'une entrée ou un délais de temporisation

WaitDI di1, 1

Attend que l'entrée digitale di1 passe à la valeur vrai.

WaitUntil DInput(di1) = 1

Attend qu'une entrée de type digitale (DInput), analogique (AInput), ou de groupe (GInput) passe à la valeur désirée (ici 1).

WaitTime 0.5

Attend une durée spécifiée (valeur numérique en secondes).

Exemples:

```
WaitDl di1, 1 ;     Attend que l'entrée dil passe à 1
WaitTime 2 ;        Attend 2 sec. avant de continuer.
                    Type de données numérique.
```

5.6.4 Programmation structurée

5.6.5 Structure des données

ANNEXE 5.1 FAIRE UN BACKUP SUR LES ROBOTS ABB

Les sauvegardes doivent être faites sur une disquette format 1.44.
Les fichiers suivants peuvent être sauvegardés:

1. Les paramètres systèmes
2. Les programmes
3. Les modules (program module & system module)

Il y a 2 possibilités:

1. Backup toutes les données sont alors sauvegardées en une seule opération.
2. Possibilité de sauvegarder les paramètres systèmes, les programmes et les modules de manière individuelle.

Attention : !! ! le robot doit être en mode manuel pour faire des sauvegardes.!!!

1ère possibilité:

1. Enfoncer la touche divers ***(Misc).***
2. Sélectionner ***service*** et enfoncer ***enter.***
3. Enfoncer dans le coin supérieur gauche la touche ***file.***
4. Sélectionner 3 ***Backup.***
5. Le robot propose un directory par défaut. Si le directory proposé est correcte, enfoncer alors la touche ***OK*** sinon allez au point 6.
6. Enfoncer la touche ***NewDir*** dans le coin inférieur droit pour créer un nouveau directory.
7. Nommer le nouveau directorv souhaité (ex. : BU991217) et enfoncer la touche ***OK.***
8. Enfoncer l'un à la suite de l'autre 2 x ***OK.***

Le Backup commence.

2ème possibilité:

1. Paramètres systèmes:

1. Enfoncer la touche divers ***(Misc).***
2. Sélectionner ***systemparameters*** et enfoncer ***enter.***
3. Enfoncer dans le coin supérieur gauche la touche ***file.***
4. Sélectionner 3 ***Save All as.***
5. Si le directory ***SYSPAR*** se trouve sur la disquette alors allez au point 6, sinon enfoncer la touche ***NewDir*** dans le coin inférieur droit. Il propose de manière automatique le nom ***SYSPAR.*** Enfoncer alors ***OK*** pour confirmer.
6. Si le directory ***SYSPAR*** n'est pas sélectionné, utiliser alors les flèches pour déplacer le curseur sur ce dernier et enfoncer la touche ***OK.*** Si le robot indique «*File already exist. Overwrite?*» enfoncer alors la touche ***OK*** pour écraser les fichiers existants.

2. Programmes

1. Enfoncer la touche ***Pro gram.***
2. Enfoncer dans le coin supérieur gauche la touche ***file***.
3. Sélectionner ***4 Save Pro gram As.***
4. Vérifier si le lecteur de disquette est sélectionné, sinon utiliser la touche ***Unit*** en bas à gauche pour sélectionner le lecteur.
5. Enfoncer le touche ***OK*** pour sauvegarder le programme.

3. Modules

1. Enfoncer la touche ***Program.***
2. Enfoncer la touche ***view*** et puis ***6 Modules.***
3. Sélectionner le module que l'on veut sauvegarder.
4. Enfoncer dans le coin supérieur gauche la touche ***file.***
5. Sélectionner ***0 Save Module As.***
6. Vérifier si le lecteur de disquette est sélectionné, sinon utiliser la touche ***Unit*** en bas à gauche pour sélectionner le lecteur.
7. Enfoncer le touche ***OK*** pour sauvegarder le module.

4. Vérifier si tout a été sauvegardé

1. Enfoncer la touche divers ***(Misc).***
2. Sélectionner ***filemanageret*** enfoncer ***enter.***
3. Enfoncer la touche ***view.***
4. Sélectionner ***floppy.***

Remarque: La fonctionnalité de back-up complet n'est pas disponible sur le robot du laboratoire qui est équipé d'un système trop ancien.

ANNEXE 5.2 LISTE DES INSTRUCTIONS DISPONIBLES

:=	Assigas a value
AccSet	Reduces the acceleration
ActUnit	Activates a mechanical unit
Add	Adds a numeric value
ArcC	Arc welding with circular motion
ArcL	Arc welding with linear motion
Clear	Clears the value
ClkReset	Resets a clock used for timing
ClkStart	Starts a clock used for timing
ClkStop	Stops a clock used for timing
Close	Closes a file or serial channel
comment	Comment
Compact IF	If a condition is met, then... (one instruction)
ConfJ	Controls thee configuration during joint movement
ConfL	Monitors the configuration during linear movement
CONNECT	Connects an interrupt to a trap routine

Deactunit	Deactivates a mechanical unit
Decr	Decrements by 1
EOffsOff	Deactivates an offset for external axes
EOffsOn	Activates an offset for external axes
EOffsSet	Activates an offset for external axes using a value
EXIT	Terminates program execution
FOR	Repeats a given number of times
GOTO	Goes to a new instruction
GripLoad	Defines the payload of the robot
IDelete	Cancels an interrupt
IDisable	Disables interrupts
IEnable	Enables interrupts
IF	If a condition 15 met, then ...; otherwise...
Incr	Increments by 1
lnvertDO	Inverts the value of a digital output signal
ISignalDI	Orders interrupts from a digital input signal
ISleep	Deactivates an interrupt
ITimer	Orders a timed interrupt
IWatch	Activates an interrupt
label	Line name
LimConfL	Defines thee permitted deviation in the-e configuration
MoveC	Moves the robot circularly
MoveJ	Moves thee robot by joint movement
MoveL	Moves thee robot linearly
Open	Opens a file or serial channel
PDispOff	Deactivates program displacement
PDispOn	Activates program displacement
PDispSet	Activates program displacement using a value
ProcCall	Calls a new procedure
PulseDO	Generates a pulse on a digital output signal
RAISE	Calls an error handler
Reset	Resets a digital output signal
RestoPath	Restores the path after an interupt
RETRY	Restarts following an error
RETURN	Finishes execution of a routine
SearchC	Searches circularly using the robot
SearchL	Searches linearly using the robot
Set	Sets a digital output signal
SetAO	Changes the value of an analog output signal
SetDO	Changes the value of a digital output signal
SetGO	Changes the value of a group of digital output signals
SingArea	Defines interpolation around singular points
StartMove	Restarts robot motion
Stop	Stops program execution
StopMove	Stops robot motion
StorePath	Stores the path when an interupt occurs
TEST	Depending on the value of an expression...
TPErase	Erases text printed on the teach pendant
TPReadFK	Reads function keys
TPReadNum	Reads a number from the teach pendant

TPWrite	Writes on the teach pendant
VelSet	Charges the programmed velocity
WaitDI	Waits until a digital input is met
WaitTime	Waits a given amount of time
WaitUntil	Waits until a condition is met
WHILE	Repeats as long as...
Write	Writes to a character-based file or serial channel
WriteBin	Writes to a binary serial channel

ANNEXE 5.3 CONFIGURATIONS DU ROBOT

Il est généralement possible d'obtenir la même position et la même orientation de l'outil du robot de diverses manières c'est-à-dire en donnant des valeurs différentes aux angles des articulations du robot. On parle de l'existence de plusieurs postures (dans la littérature française) ou configurations (dans la littérature anglaise). Par exemple, au centre de l'espace de travail, il est possible d'atteindre une localisation (position + orientation) avec le "coude en haut" ou bien le "coude en bas" (voir figure 5.29). Sur l'une de ces configurations, le bras est tourné vers le haut. Pour obtenir la seconde configuration, on devra faire tourner l'axe 1 de 180°.

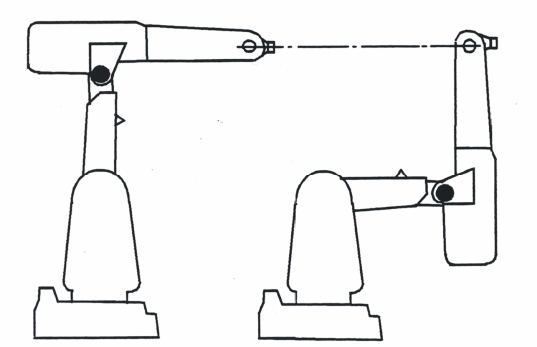


Fig.5.29 : Configurations coude en haut et coude en bas
donnant la même localisation de l'outil

De même pour le poignet il est possible de trouver plusieurs postures en faisant tourner la première partie du poignet (axe 4) tout en faisant tourner les axes 5 et 6 pour conserver la position et l'orientation souhaitée.

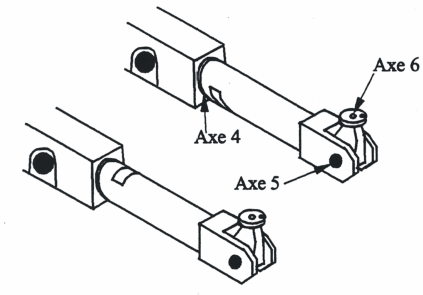


Fig.5.30 : Configurations alternatives du poignet

Généralement on souhaite conserver la même posture du robot tout en cours d'exécution d'un mouvement. Pour cela il faut spécifier dans la programmation la posture (configuration) afin que le programme vérifie la configuration qui va être adoptée. Si la posture n'est pas vérifiée, on peut assister à des mouvements assez brusques et inattendus du bras et des articulations robot et même arriver à des collisions.

La vérification de la configuration demande que l'on spécifie le quart de tour approprié pour les axes 1, 4 et 6. Si le robot et la position programmée font état du même quart de tour pour ces axes, la configuration du robot est correcte.

Lors du déplacement selon une trajectoire linéaire, le robot se déplace toujours vers la configuration la plus proche possible. Si la vérification de la configuration est active, l'exécution du programme s'arrête dès qu'un des axes sort des limites fixées.

ANNEXE 5.4 SINGULARITES DU ROBOT

On introduit d'abord le concept de *redondance*. On dit qu'un manipulateur est redondant lorsque le nombre de degrés de liberté (ddl) nécessaires pour spécifier la configuration du robot est strictement inférieur au nombre de degrés de liberté conférer à l'outil. La propriété de redondance est une propriété de l'architecture du robot (donc permanente en quelque sorte).

On peut alors définir le concept de *configuration singulière* comme suit. Un sous-ensemble de degrés de liberté du robot étant fixé par la définition d'une famille de configurations, le sous-ensemble des degrés de liberté restants présente une redondance, c'est-à-dire que le nombre de degrés de liberté conférés à l'outil est inférieur au nombre de degrés de liberté restant libres.

Une autre manière de définir le concept de singularité peut être faite en passant par la relation qui lie les vitesses articulaires (aux actionneurs) $\dot{\theta}$ à la vitesse de l'outil $\dot{x}$. Pour un manipulateur en chaîne ouverte, on peut montrer que cette relation peut toujours se mettre sous la forme:

$$\dot{x} = J(\vartheta)\,\dot{\theta}$$

où ***J*** est la matrice jacobienne des relations du modèle géométrique du robot. Dans le cas de configurations singulières, on a:

$$\det(J(\theta)) = 0$$

Pour les configurations solutions de cette équation, on peut montrer que les lignes de la matrice $J(\vartheta)$ étant linéairement dépendantes, il existe des vitesses particulières de l'outil qui requièrent des vitesses infinies aux actionneurs. Cette situation constitue donc une perte de commandabilité du manipulateur puisque certaines manœuvres deviennent impossibles. En outre pour les vitesses qui sont accessibles, il existe un nombre infini de combinaisons des vitesses articulaires qui donnent lieu à la vitesse prescrite à l'outil. Dans un cas comme dans l'autre on aboutit à une situation problématique. Les points dits de singularités (et leur voisinage) sont donc des points à éviter lors des manœuvres du robot.

Pour les manipulateurs anthropomorphes comme le robot IR1400, le robot comporte 3 types de singularités:

- la singularité bras tendu lorsque l'on veut atteindre la frontière de l'espace de travail en position,
- la singularité d'épaule (voir figure 5.31),

- la singularité de poignet (voir figure 5.32).

La singularité d'épaule survient dans les configurations où le centre du poignet (intersection des axes 4, 5 et 6) se trouve directement au-dessus de l'axe 1 (voir figure 5.31).

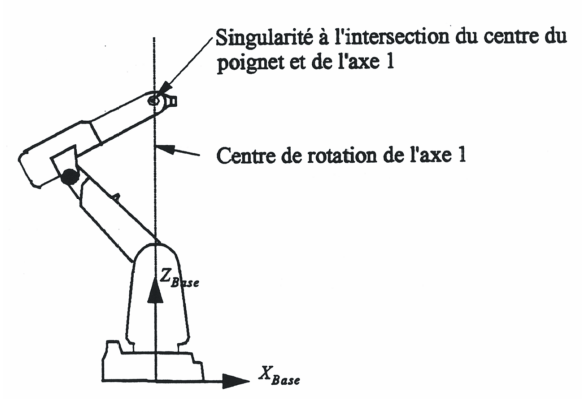


Fig.5.31 : Singularité du bras ou d'épaule

Les singularités dites de poignet sont les configurations où l'axe 4 devient colinéaire avec l'axe 6, c'est-à-dire que l'angle de l'axe 5 est égal à 0°.

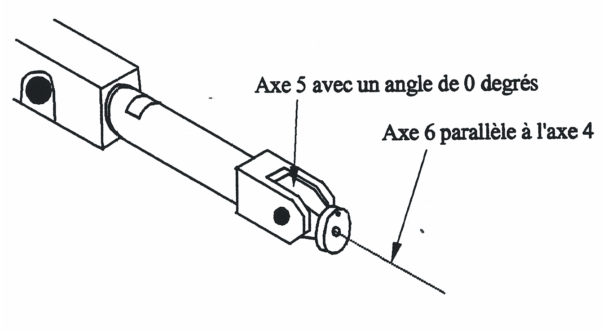


Fig.5.32 : Singularité de poignet

Lors d'un mouvement articulaire, le robot n'éprouve aucun problème à traverser des points singuliers. En effet la génération de trajectoire étant définie sur la base de vitesses articulaires uniquement il n'y a pas de risque de solliciter des vitesses de l'outil qui pourraient requérir des vitesses infinies aux actionneurs.

Lorsque l'on veut générer un mouvement dans l'espace de travail (trajectoire linéaire ou circulaire), le robot qui traverse des points singuliers peut exciter des vitesses infinies aux actionneurs. Ce problème est évité par une procédure de sauvegarde. Le robot traverse le voisinage de points singuliers à vitesse réduite et l'exécution est contrôlée en permettant par exemple une légère erreur de l'orientation de l'outil à proximité des points singuliers.

De plus amples informations peuvent être obtenues sous la rubrique *SingArea* du manuel.

ANNEXE 5.5 REPRESENTATION DES ROTATIONS

L'effecteur, considéré comme un corps rigide, nécessite la connaissance des coordonnées de 3 points non alignés. L'hypothèse de rigidité implique 3 contraintes de distances entre chacun de ceux-ci. Il en résulte que la localisation de l'effecteur peut être définie univoque ment par la connaissance de 6 paramètres. Par la mécanique rationnelle, on sait que 3 paramètres définissent la position d'un point particulier du repère : le centre d'outil ici. Les 3 autres paramètres sont nécessaires pour décrire l'orientation de l'outil.

Nous connaissons déjà des systèmes de paramètres permettant de définir une rotation quelconque en termes de 3 grandeurs: *les angles d'Euler*. Néanmoins cette paramétrisation est peu employée en robotique, compte tenu des difficultés suivantes:

- utilisation de fonctions transcendantales pour la construction de l'opérateur de rotation, ce qui implique un coût numérique élevé pénalisant pour des applications temps réel
- l'existence de difficultés liées à la présence d'une singularité lors de l'identification des angles d'Euler à partir de la connaissance de la matrice de rotation

En robotique, on leur préfère souvent un autre système de paramétrisation : *les paramètres d'Euler*. On définit les paramètres d'Euler à partir du théorème d'Euler qui dit que n'importe quelle rotation ou combinaison de rotations peut être écrite comme une rotation unique d'amplitude angulaire ϕ autour d'un axe de rotation **n** = (n_x, n_y, n_z) (**n** et ϕ à déterminer). Une manière extrêmement puissante d'utiliser ces 4 paramètres ϕ, n_x, n_y, n_z consiste à définir les quatre paramètres suivants appelés paramètres d'Euler:

$$q_1 = \cos\frac{\phi}{2} \quad q_2 = n_x \sin\frac{\phi}{2} \quad q_3 = n_y \sin\frac{\phi}{2} \quad q_4 = n_z \sin\frac{\phi}{2}$$

Seuls 3 paramètres parmi ceux-ci sont indépendants, car ils sont liés par la relation de normalisation suivante:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$$

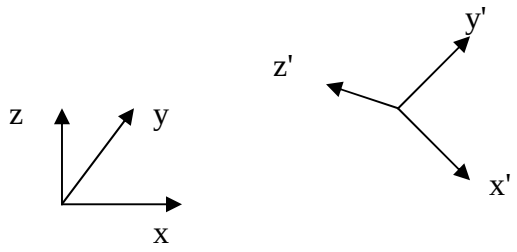
La matrice de rotation s'écrit en n'utilisant que des opérations de types arithmétiques simples:

$$R = \begin{bmatrix} 1-2(q_3^2+q_4^2) & 2(q_2q_3-q_1q_4) & 2(q_2q_4+q_1q_3) \\ 2(q_2q_3+q_1q_4) & 1-2(q_2^2+q_4^2) & 2(q_3q_4-q_1q_2) \\ 2(q_2q_4-q_1q_3) & 2(q_3q_4+q_1q_2) & 1-2(q_2^2+q_3^2) \end{bmatrix}$$

Le processus d'identification des paramètres d'Euler à partir d'un changement d'orientation s'effectue aussi selon une procédure aussi puissante qu'efficace.
La matrice de rotation permettant de ramener les coordonnées du système local (ayant subi une rotation) dans les coordonnées du système de référence est la suivante:

$$R = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}$$

où $(x_1, x_2, x_3)^T$ sont les cosinus directeurs du vecteur x' du système de référence local dans le système de coordonnées global, etc.



Les quaternions de la rotation sont donnés par :

$$q_1 = \frac{1}{4}\sqrt{x_1 + y_2 + z_3 + 1}$$

$$q_2 = \frac{1}{4}\sqrt{x_1 + y_2 + z_3 + 1} \quad sign\ q_2 = sign(y_3 - z_2)$$

$$q_1 = \frac{1}{4}\sqrt{x_1 + y_2 + z_3 + 1} \quad sign\ q_3 = sign(z_1 - x_3)$$

$$q_1 = \frac{1}{4}\sqrt{x_1 + y_2 + z_3 + 1} \quad sign\ q_4 = sign(x_2 - y_1)$$