SIEMENS

FIRST STEP WITH STEP7

DEMARRER AVEC STEP7 MANAGER

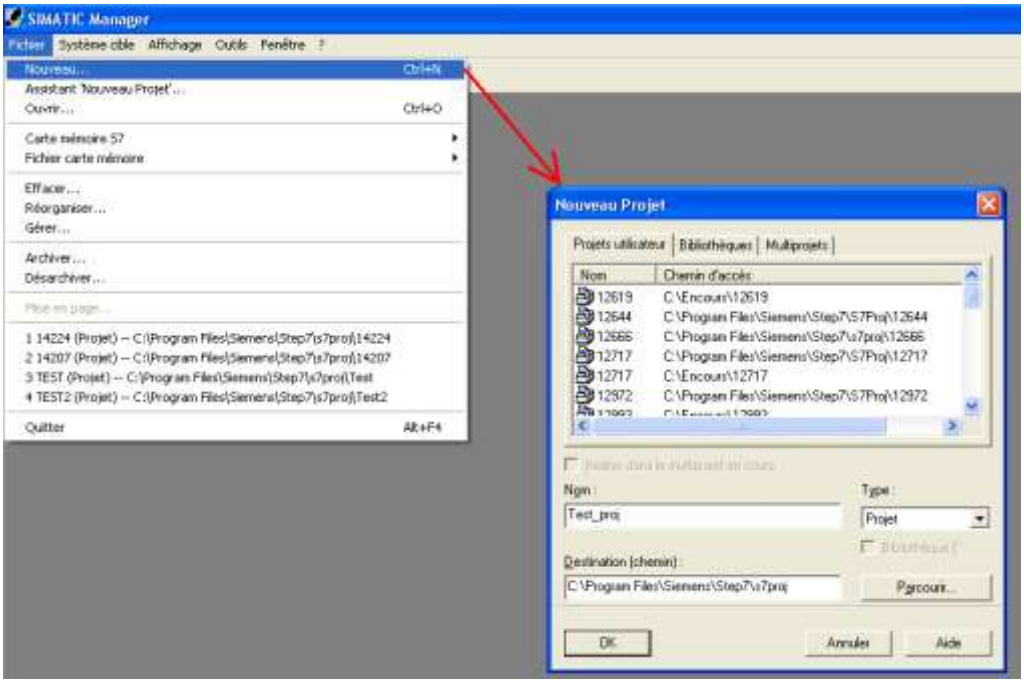
REV	DATE	ETAT	REVISION	ETABLI
06	18/03/13	UP	MISE A JOUR (langage LIST, …)	M.AIDEL
05	02/05/11	UP	MISE A JOUR	M.AIDEL
04	20/09/07	UP	MISE A JOUR	M.AIDEL
03	08/08/07	UP	MISE A JOUR	M.AIDEL
02	30/06/05	UP	MISE A JOUR	M.AIDEL
01	06/06/05	UP	MISE A JOUR	M.AIDEL
00	18/08/04	PRE	EDITION ORIGINALE	M.AIDEL

SOMMAIRE

1 CREATION D'UN PROJET

Dans le cas où l'on n'utilise pas l'assistant (Wizard):

- Lancer SIMATIC STEP7,
  Remarque: le logiciel s'ouvre sur le ou les derniers projets qui ont été fermé.
- Dans la barre d'outils sélectionner : *"Fichier > Nouveau …"*,
- Donner un nom au projet (ici "Test_proj"),
- Valider par le bouton OK.

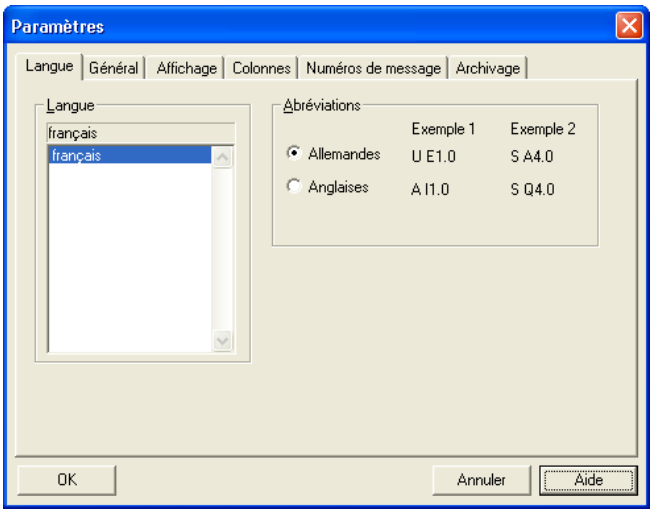


Le projet un maintenant créé. Une nouvelle fenêtre apparaît:

- avec à gauche une arborescence projet présentant tous les constituants de votre projet,
- à droite le détail du constituant de votre projet en surbrillance.

1.1 REGLAGES PRELIMINAIRE

Choix du type d'abréviations:



	Anglaises	Allemandes
Entrées	I0.0	E0.0
Sorties	Q0.0	A0.0
Compteurs	C0	Z0

2 CONFIGURATION MATERIEL

2.1 DEFINITION DU MATERIEL

Avant de commencer la programmation, il faut tout d'abord déclarer le matériel du projet.

Nous allons prendre l'exemple d'un S7-300 connecté sur un réseau Ethernet, on y ajoutera une station déportée type ET200 sur réseau PROFIBUS.
Remarque: En théorie, pour pouvoir programmer, il nous suffit simplement d'insérer un Rack (Rail) et une CPU (qui contiendra le programme).

Dans l'arborescence projet, faire un clic droit sur l'icône (qui est le seul pour le moment et qui représente votre projet) et choisissez "*Insérer un nouvel objet > Station SIMATIC 300*". Vous venez d'insérer un automate (ou station) S7-300, vous pouvez renommer votre automate si besoin.

2.2 INTRODUCTION CONFIGURATION DU MATERIEL

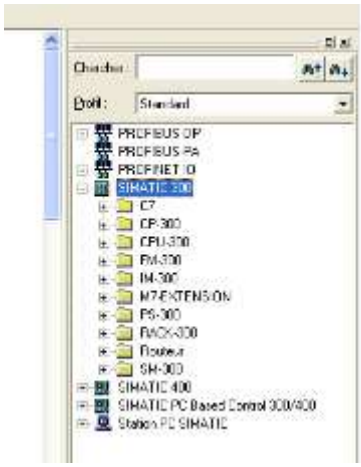
Pour commencer, dans l'arborescence projet, sélectionnez l'automate. Ceci fait, on peut voir apparaître dans la fenêtre de droite tous ce qui concerne l'automate sélectionné. Pour le moment seul l'icône "Matériel" est disponible, double-cliquez sur cette icône pour lancer l'application "HW config" (**H**ard**w**are config, utilitaire pour gérer la configuration matériel de l'automate).

2.2.1 VUE D'ENSEMBLE

Cette application comprend trois parties:
- à gauche : l'implantation du matériel (automates, réseaux, esclaves, …),
- à droite : le catalogue des constituants de la gamme SIMATIC (à valider à travers la barre d'outils : "*Affichage > Catalogue*" si elle n'est pas présente),
- en bas : un tableau comprenant les informations de l'objet sélectionné via la zone de gauche (implantation du matériel).

2.2.2 INSERTION D'UN AUTOMATE AVEC SES MODULES

Pour constituer notre automate nous allons nous servir du catalogue (droite de l'écran), et plus particulièrement la partie "**SIMATIC 300**" du catalogue qui contient les modules pour notre automate S7-300.



a) Il faut commencer tous d'abord par insérer un Rack (Rail) pour supporter les modules automates, pour cela allez dans le répertoire "**RACK-300**" dans le catalogue et faites glisser/déposer l'objet "profilé support" dans la partie implantation du matériel (gauche de l'écran). Une petite fenêtre avec en titre "(0) UR" (châssis n°0) est maintenant disponible pour recevoir vos modules.

b) Insérons maintenant une CPU, pour cela allez dans le répertoire "**CPU-300**" dans le catalogue et faites glisser/déposer la CPU de votre choix dans le Rack posé précédemment à l'emplacement n°2 (le seul disponible pour l'insertion d'une CPU). Valider par OK la fenêtre sans faire de modification nous reviendrons sur la configuration de la CPU plus tard.

Les deux étapes précédentes (insertion du rail et de la CPU) sont le minimum pour pouvoir programmer. Rendez-vous au chapitre 4 pour débuter la programmation.

c) Insérons maintenant une alimentation pour alimenter la CPU et les différents modules de notre automate, pour cela allez dans le répertoire "**PS-300**" (**P**ower **S**upply) dans le catalogue et faites glisser/déposer l'alimentation de votre choix dans le Rack à l'emplacement n°1 (le seul disponible pour l'insertion d'une alimentation).

d) Insérons maintenant un coupleur Ethernet pour pouvoir du réseau Ethernet (remarque certaines CPU disposent d'une interface Ethernet intégrée), pour cela allez dans le répertoire "**CP-300**" (**C**ommunication **P**rocessor = coupleur) dans le catalogue et faites glisser/déposer le coupleur Ethernet de votre choix dans le Rack à l'emplacement n°4. Valider par OK la fenêtre sans faire de modification nous reviendrons sur la configuration du coupleur plus tard.

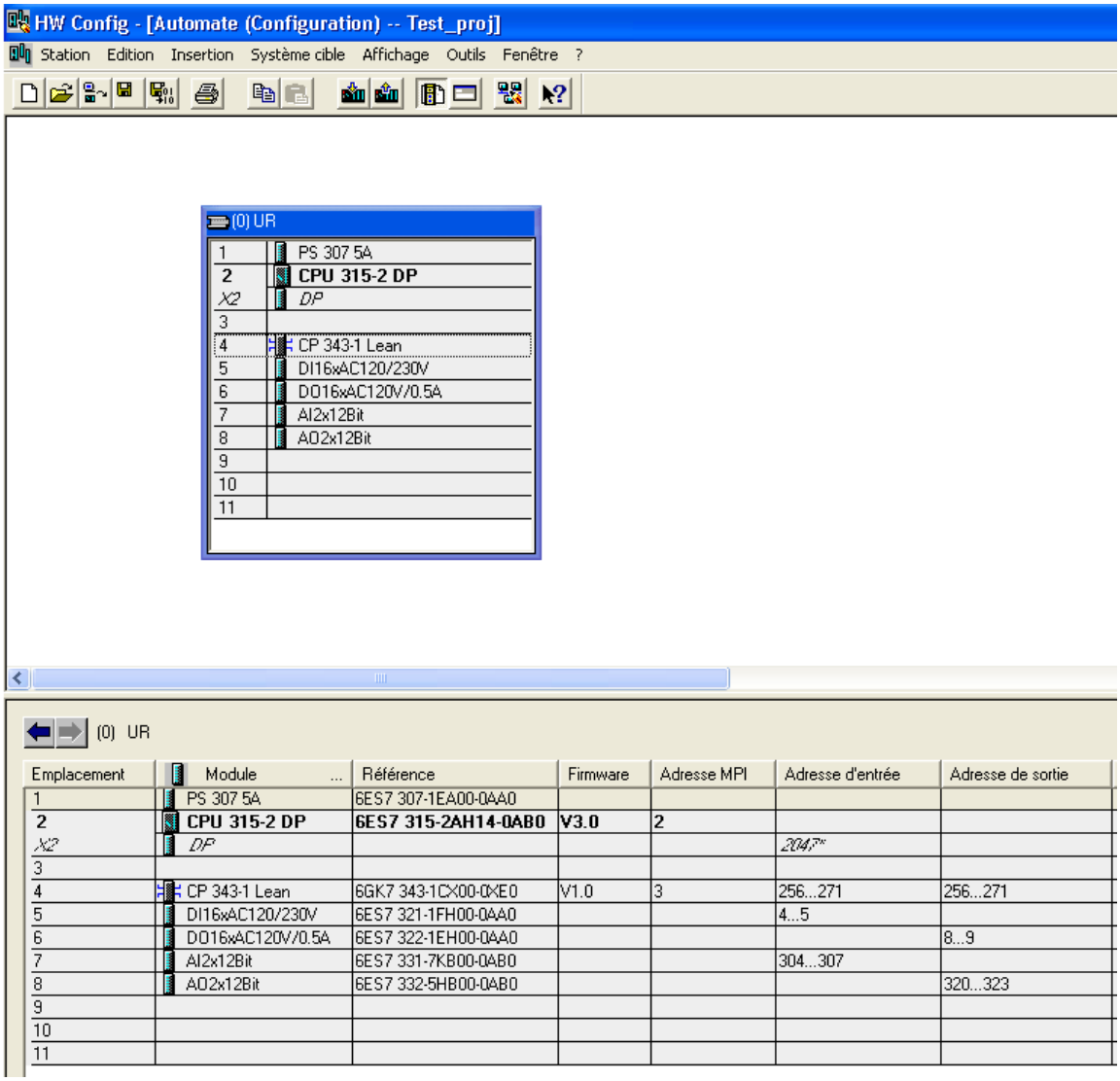
e) Insérons maintenant des modules d'Entrées et de Sorties (TOR ou Analogique), pour cela allez dans le répertoire "**SM-300**" (**S**ignal **M**odules) dans le catalogue et faites glisser/déposer le de votre choix dans l'emplacement de votre choix à partir de l'emplacement n°4 (le n°3 est réserver au coupleur d'extension **IM-300**).

Concernant les modules on retrouve:
- **les DI (Digital Input) :** Entrées TOR,
- **les DO (Digital Output) :** Sorties TOR,
- **les DI/DO :** TOR Mixte (Entrées et Sorties TOR),
- **les AI (Analog Input) :** Entrées Analogiques,
- **les AO (Analog Output) :** Entrées Analogiques,
- **les AI/AO:** Analogique Mixte (Entrées et Sorties Analogiques).

Attention : comme vu plus haut, tous modules ne peuvent pas se mettre n'importe où dans le rack, une alimentation ne peut être insérée que dans l'emplacement n°1, une CPU dans l'emplacement n°2, …
Lorsque l'on est en train de glisser/déposer un élément venant du catalogue, on peut apercevoir en vert sur le rack le ou les emplacements disponible pour accueillir cet élément.

Voici un exemple de configuration pour un automate S7-300 :

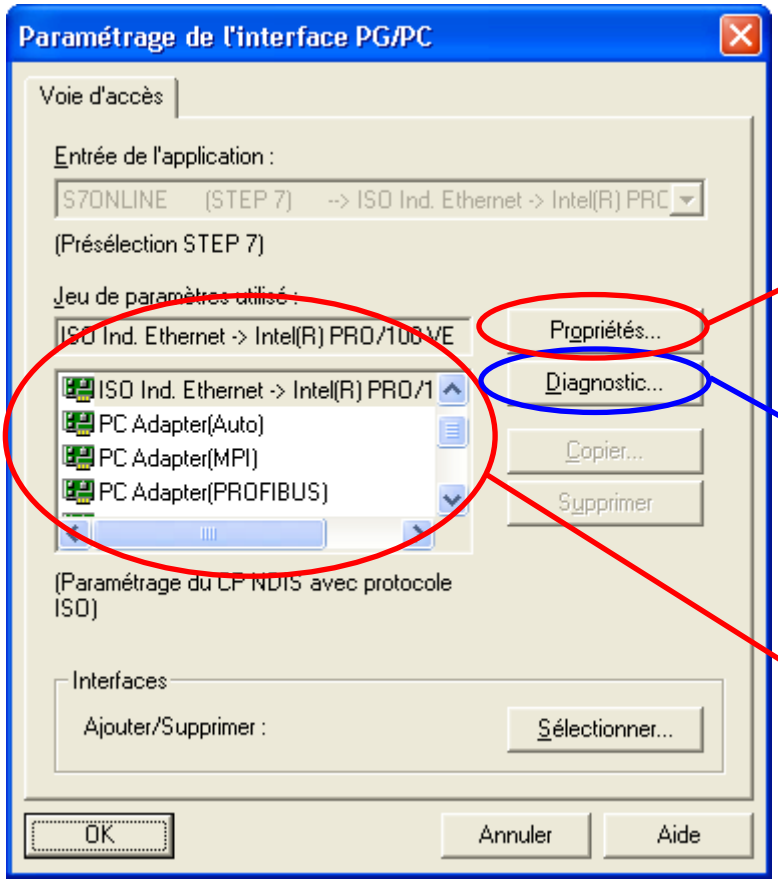
- une alimentation 5A,
- une CPU 315-2DP,
- un coupleur Ethernet CP-343,
- une carte de 16 Entrées TOR,
- une carte de 16 Sorties TOR,
- une carte de 2 Entrées Analogiques,
- une carte de 2 Sorties Analogiques.



2.3 REGLAGE DE L'INTERFACE PG/PC

Il est important de vérifier la configuration de la carte PG/PC (dans la barre d'outils de la fenêtre projet: "*Outils→ Paramétrage de l'interface PG/PC …*"

Ceci est très important pour le chargement de votre programme car on peut le transférer aussi bien par liaison MPI, PROFIBUS ou ETHERNET **il faut donc s'assurer du paramétrage de la PG/PC Interface**



Adapter la configuration (vitesse de transmission, …)

Permet de tester si la connexion est OK et de voir les stations sur le réseau

Choisir l'Interface MPI, PROFIBUS ou ETHERNET en fonction du port utilisé sur l'automate pour le chargement du PC → Automate.

Câbles de programmation

PC Adapter	Câble Ethernet	Carte CP5512 (PCMCIA)
 Existe en **USB** ou connexion **Série**		

2.4 CONFIGURATION DE LA CPU ET DU RESEAU PROFIBUS

Une fois ceci fait, faites clic droit sur la CPU et faites "Propriétés de l'objet". Une fenêtre vous permet de configurer la CPU. Pour commencer il faut lui indiquer si vous voulez la connecter au réseau MPI et lui donner son adresse (par défaut elle n'est pas connecté au réseau MPI et elle possède l'adresse "2")

Dans l'onglet "Cycle/mémento de Cadence" il est utile d'autoriser un octet qui renverra des bits clignotant de période différentes, pour cela cocher la case "Mémento de Cadence " et indiquer lui l'octet réservé à cet effet (si par exemple adresse = 1 → M1.0 à M1.7 = bits clignotant à différentes cadences). Ces peuvent être utiles pour l'animation de voyant clignotants.

Bits de l'octet du mémento de cadence	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Période (s)	2,0	1,6	1,0	0,8	0,5	0,4	0,2	0,1
Fréquence (Hz)	0,5	0,625	1	1,25	2	2,5	5	10

L'onglet "Alarmes Cyclique" permet de régler le temps de cycle des OBs cycliques.

Remarque: l'OB1 n'est pas présent dans la liste car son temps de cycle dépends de la charge du programme.

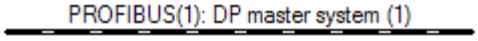
100ms			ex : 15ms	100ms			ex : 15ms
OB1	OB1	OB1	OB35	OB1	OB1	OB1	OB35

exemple d'interruption de bloc par rapport à l'OB1

Comme on peut le voir sur le tableau ci-dessus, l'OB35B est appelé cycliquement toutes les 100ms (on prendra comme exemple que le programme à l'intérieur de l'OB35 s'exécute en 15ms) et vient interrompre l'appel de l'OB1.

Remarque : L'OB1 à la plus basse priorité, il peut être interrompu par tous les autres OBs.

Après avoir validé ces modifications, faites un clic droit en dessous de la CPU (sur 

) et faites " Propriétés de l'objet". Une fenêtre vous permet de savoir si vous voulez connecter la CPU au réseau Profibus-DP et lui donner son adresse (par défaut elle n'est pas connecté au réseau Profibus et possède l'adresse "2"). Pour notre cas il faut connecter le réseau Profibus, pour ce faire cliquer sur "Propriétés", ensuite sur la fenêtre suivante faire "Nouveau" et validez ensuite les fenêtres jusqu'à revenir sur la page principale (on remarque le réseau Profibus présent → 

).

2.5 CONFIGURATION DU RESEAU ETHERNET

Important: Avant tout raccordement il faut définir quelle type de câble Ethernet l'on va utiliser dans notre application, 2 cas se présentes :

- si on relie directement l'automate au PC de développement il faut un câble dit ***croisé***
- si on relie l'automate au PC à travers un HUB (ou Switch) il faut un câble dit ***droit***

Il faut configurer la PG/PC Interface en "Ethernet" (TCP/IP (Auto) -> Intel® PRO/100VE Network. (voir chapitre 2.3)

Vérifier que l'adresse IP du PC qui charge la config soit du même type que celle que l'on attribuer au PLC (exemple :

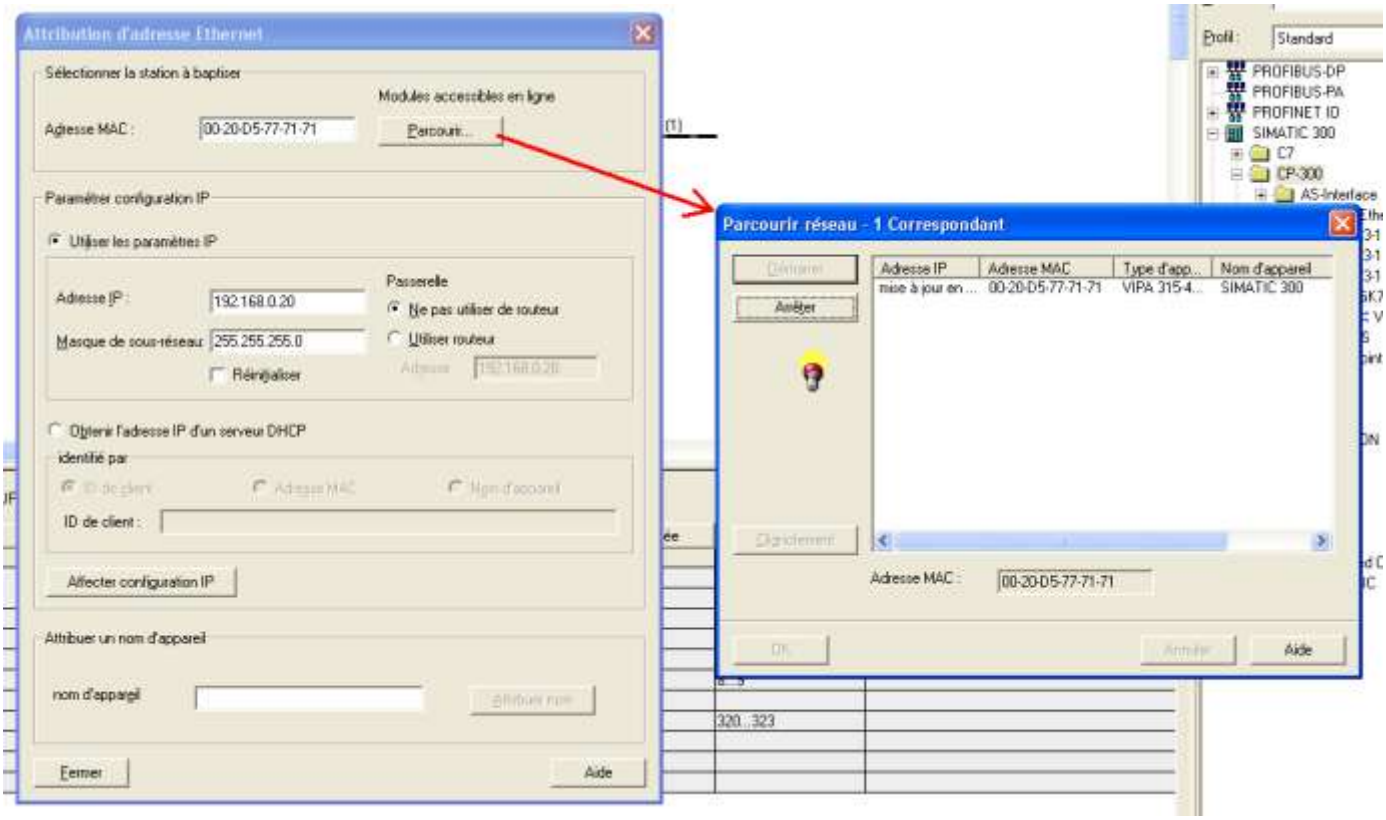
Nom station	Adresse IP station
PC de développement	192.168.0.**1**
Automate	192.168.0.**20**

* Mask de sous réseau : 255.255.255.**0**

Ensuite il faut affecter une adresse IP à votre coupleur, il faut pour cela rechercher l'adresse MAC de votre Coupleur, pour ce faire 2 solutions:

- soit la relever sur l'étiquette collée sur le coupleur
- soit faire une recherche par logiciel, pour détecter tout partenaire sur le réseau. Cette solution est meilleure car vous êtes certains de la communication et vous obtenez du même coup l'adresse MAC).

Pour faire une recherche par logiciel, dans la barre d'outil de l'utilitaire de configuration matériel (HW Config) cliquer sur "*Système Cible* > *Ethernet* > *Attribuer une adresse Ethernet …*". Une fenêtre comme celle-ci apparaît:



La première chose à faire est de cliquer sur le bouton "Parcourir" pour rechercher le partenaire accessible. Une fois le coupleur trouvé, vous devez affecter une adresse IP à ce coupleur, ainsi que le masque de réseau (255.255.255.0). Valider en cliquant sur le bouton "Affecter configuration IP".

Une fois ceci fait, faire clic droit sur le coupleur Ethernet (position slot n°4 dans notre exemple) et faire "Propriétés de l'objet". Une fenêtre vous permet de configurer le coupleur.

Affecter l'adresse que l'on a défini (192.168.0.**20**) puis créer le réseau Ethernet de la même manière que l'on a créé le réseau MPI et Profibus (voir chapitre 2.4).

2.6 CONFIGURATION DES MODULES E/S TOR ET ANA

Lorsque l'on met en surbrillance dans la zone de gauche un module E/S TOR ou ANA on peut apercevoir dans la zone du bas, un tableau qui récapitule les informations de celui-ci. Le plus important à prendre en compte sont les adresses.

L'adressage par défaut des modules se fait en fonction de son emplacement physique dans le rack. Il est néanmoins possible de changer l'adressage, pour ce faire, cliquer droit sur le module et faites "Propriétés de l'objet". Une fenêtre s'ouvre, choisir l'onglet "adresses" et modifier vos adresses comme vous le désirez, il faut pour cela avoir décocher la case "valeur par défaut système".

Remarque: il est possible de changer les adresses via le tableau de la zone du bas (voir dernière photo dans chapitre 2.2.2), ainsi il n'est pas nécessaire d'ouvrir la fenêtre relative aux informations du module.

2.7 AJOUTER UNE STATION DEPORTEE TYPE ET200 SUR LE RESEAU PROFIBUS

Dans le catalogue (zone droite de la fenêtre principale), dossier "**PROFIBUS-DP"**, choisir dans le dossier ET200M, l'interface: "IM 153" par exemple, faire un glisser/déposer sur la représentation graphique du câble du réseau PROFIBUS, valider les informations (nom, choix adresse, …) de la fenêtre qui vient de s'ouvrir.

2.8 AJOUT DE MODULE E/S TOR ET ANA SUR UNE STATION DEPORTEE ET200M

Pour insérer des modules E/S TOR ou ANA il suffit de rechercher dans le catalogue: dossier "*PROFIBUS > ET200M > IM153*" les modules que l'on désire ajouter à notre station déportée ET200M (AI,AO,DI ou DO). Cliquer sur votre ET200M de votre config et faire un glisser/déposer du module souhaité dans le tableau de la zone du bas, répéter ceci pour chaque module. Tout comme l'automate, il est possible de configurer les adresses des modules.

2.9 SAUVEGARDE, COMPILATION ET CHARGEMENT DE LA CONFIG DANS LE PLC

Une fois ceci terminé, dans la barre d'outils choisir : "*Station > Enregistrer et compiler*" afin de conserver dans votre projet la config matériel de votre PLC et être sûre qu'il n'y a pas d'erreur (grâce à la compilation).

Ensuite dans la barre d'outils sélectionner "*Système cible > Charger dans module …*" pour charger la configuration matérielle dans l'automate. (bien vérifier la configuration de la carte PG/PC → voir chapitre 2.3. **N.B. : au premier chargement il est recommandé de sélectionner la carte PG/PC en AUTO**)

IMPORTANT: la configuration établit sur SIMATIC STEP7 Manager doit être **identique** à celle de l'automate que l'on utilise pour notre application (emplacement des cartes, types de cartes, références des cartes, …)

3 ARCHITECTURE DU PROGRAMME

Après avoir quitté l'application "HW config" vous voilà de retour dans la fenêtre projet. Dans l'arborescence vous pouvez retrouver certains constituants de votre config matériel comme la CPU et le Coupleur.

Tout d'abord nous retrouvons dans l'arborescence de l'automate, sous la CPU un dossier "Programme S7 (1)" regroupant le programme sous deux dossiers différents : les codes "Sources" et les "Blocks".
Dans le dossier Sources, on a tous les codes sources des blocks programmés en SCL ou STL.
Dans le dossier Blocks, on retrouve l'OB1 ainsi que tous les Blocks appelés par celui-ci (FB, FC, DB, …)
Remarque : dans la description (voir fenêtre de droite) du dossier "Programme S7(1)", un fichier "Mnémoniques" est présent et comprends la liste des variables (E/S, Mémento, OB, FB, FC, DB,…).

4 VARIABLES

5 OBJETS OB, FC, FB ET DB

5.1 UTILITAIRE CONT/LIST/LOG

C'est cet utilitaire qui nous permet de réaliser la programmation des OB, FC, FB et DB Global (pas les DB d'instances).

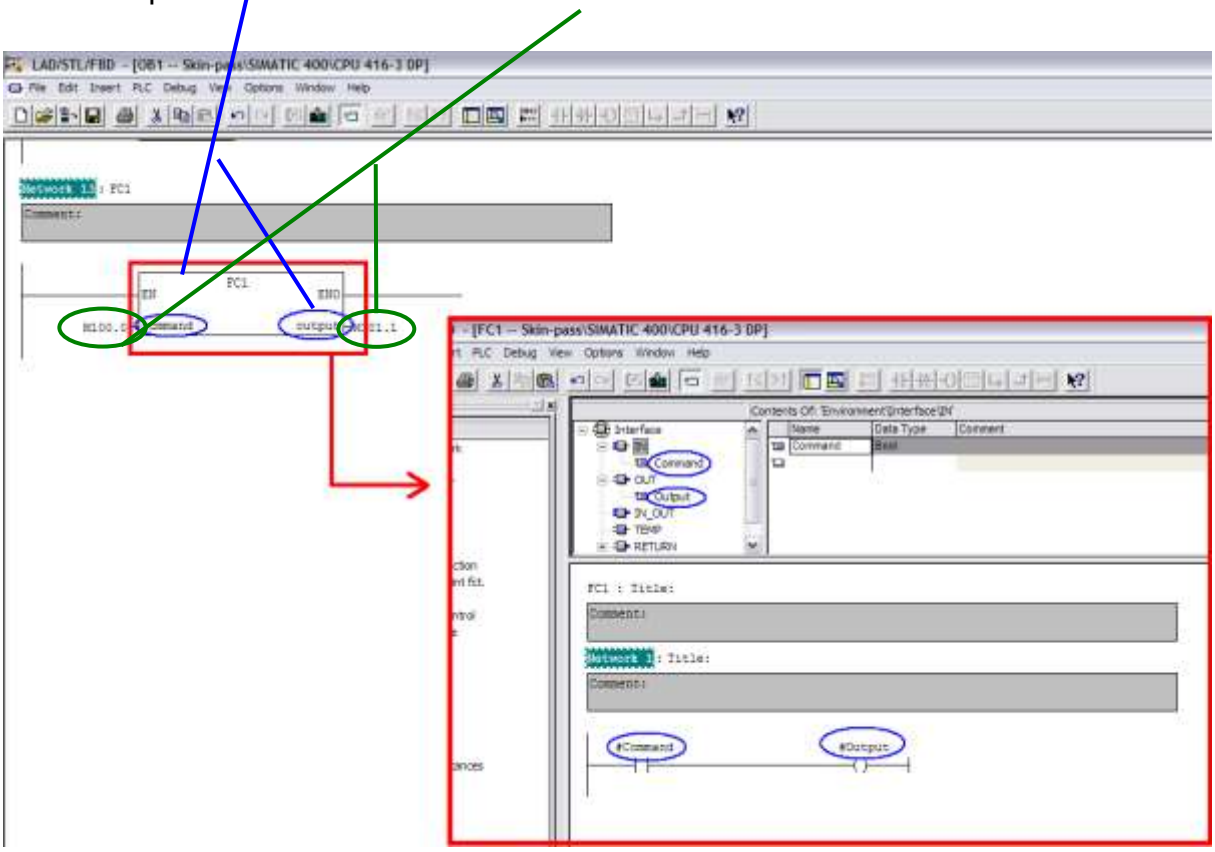
5.2 LES OB

L'OB1 gère le programme en fonctionnement normal, on peut utiliser l'OB1 pour programmer directement à l'intérieur de celui-ci des ordres ou alors appeler des Blocks (FCs ou FBs) qui permettront de structurer le programme. Ainsi on pourra créer un Block pour un moteur avec tous ses défauts, ses statuts (HMI), ses modes opératoires (Auto/Manu), ces commandes (Marche/Arrêt), ceci est valable pour des vannes où autres.

5.3 LES FC

Les FCs sont des fonctions permettant la programmation de sous-programmes. Ils peuvent être appeler via un OB, un FB ou même un autre FC.

Le bloc utilise des variables temporaires à l'intérieur du bloc pour ces calculs et aussi pour l'affectation de ces E/S. En appelant ce FC dans un autre endroit du programme (FC, OB, …), alors celui-ci exécutera son contenu à l'aide de ses paramètres d'entrées et modifiera des valeurs de sorties. On raccordera aux E/S du FC les variables faisant partie de notre table de mnémonique.



5.4 LES FB

Les FBs obéissent aux mêmes principes que le FC, à la différence qu'un FB doit être impérativement associé à une DB d'instance. A chaque fois que l'on utilisera un FB, une DB différent devra lui être associé.

La différence entre un FB et un FC c'est que le FB permet de mémoriser les variables que l'on aura besoin au prochain appel de ce FB dans des variables statiques (STAT) dans un DB, alors que le FC ne peut qu'avoir des variables internes (TEMP) qui seront écrasé et ne pourront donc pas être utilisé au prochain tour.

Conclusion : Les valeurs tels que les compteurs, des temporisations, …ne peuvent donc pas être stockés dans un FC car si il est appelé plusieurs fois, les valeurs TEMP du FC ne le permettent pas)

5.5 LES DB

Les DBs (**D**ata **B**lock) permettent de stocker des valeurs. Une DB peut être de deux types différents:

- d'instance (propre à un FB ou à un SFB)
- global (interrogeable dans le programme n'importe où)

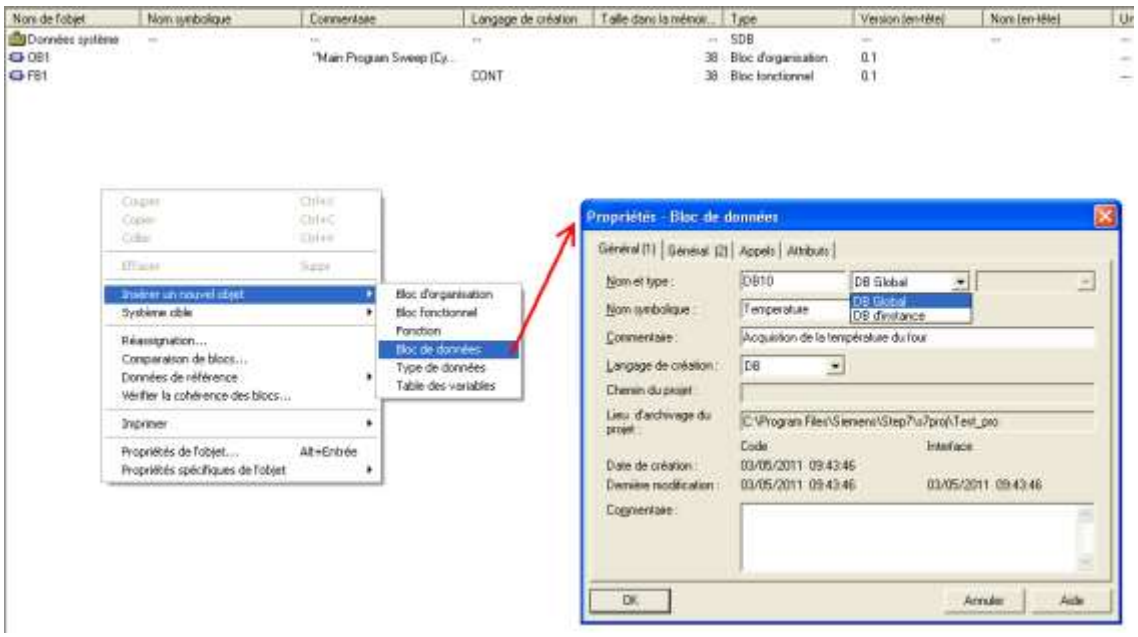
Comme les mémentos, les Blocs de Données (DBs) permettent d'enregistrer des valeurs mais contrairement aux mémentos :

- Les valeurs des DBs sont toutes sauvegardées Hors Tension
- Le nombre et la taille des DBs ne sont pas fonction de la CPU mais de la taille mémoire utilisateur. En effet les DBs prennent de la place mémoire, de la même manière que les FCs, FB,s, OBs.

On peut y stocker tous types de variables: BOOL, BYTE, WORD, INT, DWORD, DINT, REAL, ….

5.5.1 CREATION D'UNE DB

Dans la liste des objets du programme, faire un clic droit et choisir "*Insérer un objet > Bloc de données*" via le menu contextuel.

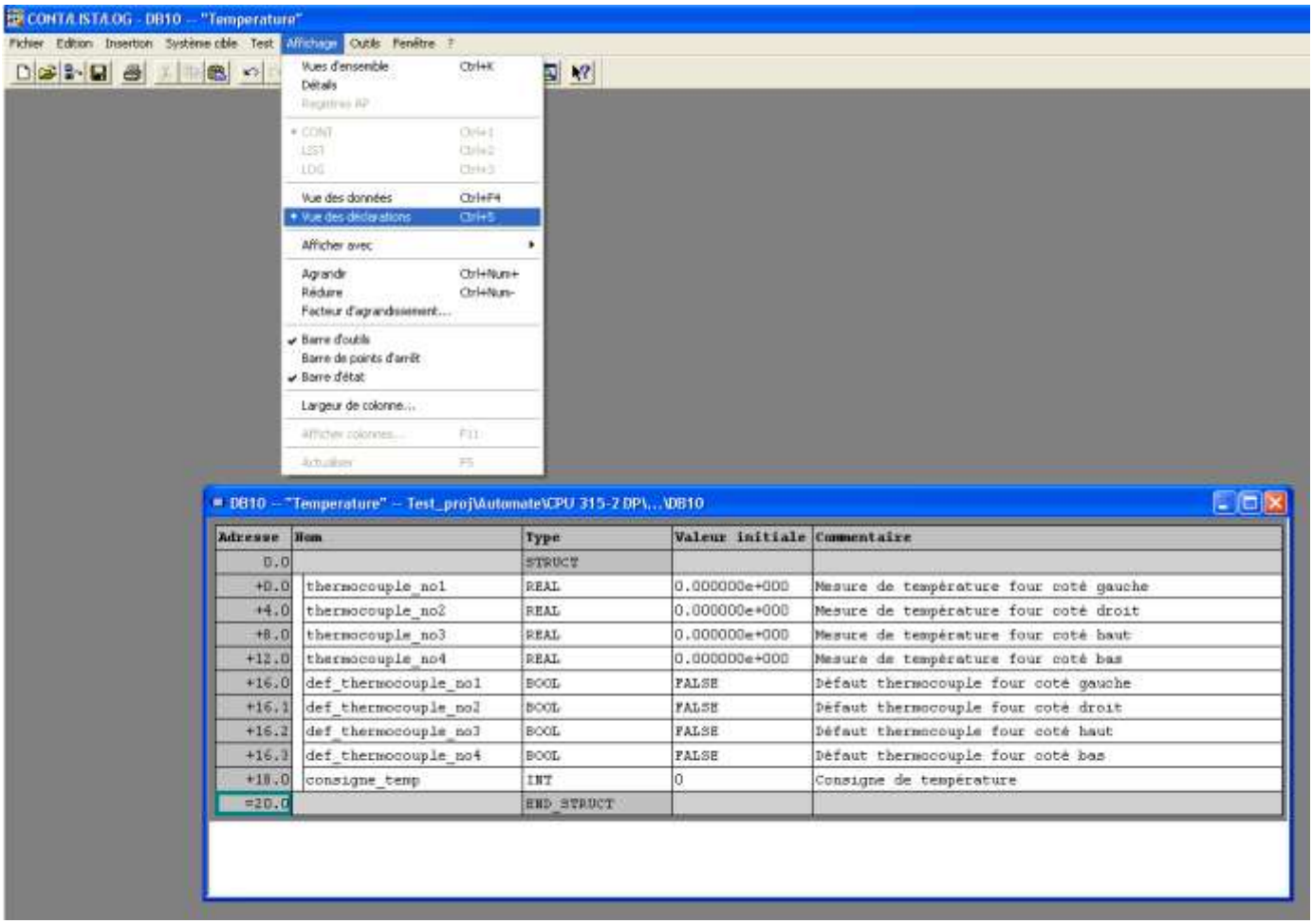


Il suffit de choisir un numéro de DB (dans notre exemple DB10), de lui donner un nom symbolique si nécessaire et un commentaire et surtout définir son type (global ou d'instance).
Remarque : le choix "instance" apparaît que si au moins un FB ou un SFB est présent dans le projet.

La syntaxe pour appeler une variable dans une DB est la suivante:
DB10.DBX10.0 → interroge le bit 0 de l'octet 10 de la DB10.
DB10.DBB10 → interroge l'octet 10 de la DB10.
DB10.DBW10 → interroge le mot 10 de la DB10.
DB10.DBD10 → interroge le double mot 10 de la DB10.

*Remarque : comme on peut le voir sur la photo ci-dessous, si l'on a que des variables de type REAL alors les adresses sont des multiples de "**4.0**" (0.0; 4.0; 8.0; 12.0; …) car une variable de type REAL est composée de **4 octets.***

Contenu d'une DB :



Adresse	Nom	Type	Valeur initiale	Commentaire
0.0		STRUCT		
+0.0	thermocouple_no1	REAL	0.000000e+000	Mesure de température four coté gauche
+4.0	thermocouple_no2	REAL	0.000000e+000	Mesure de température four coté droit
+8.0	thermocouple_no3	REAL	0.000000e+000	Mesure de température four coté haut
+12.0	thermocouple_no4	REAL	0.000000e+000	Mesure de température four coté bas
+16.0	def_thermocouple_no1	BOOL	FALSE	Défaut thermocouple four coté gauche
+16.1	def_thermocouple_no2	BOOL	FALSE	Défaut thermocouple four coté droit
+16.2	def_thermocouple_no3	BOOL	FALSE	Défaut thermocouple four coté haut
+16.3	def_thermocouple_no4	BOOL	FALSE	Défaut thermocouple four coté bas
+18.0	consigne_temp	INT	0	Consigne de température
=20.0		END_STRUCT		

On obtient le type de fenêtre ci-dessus à l'ouverture d'une DB. Dans la barre d'outils on peut choisir dans "Affichage" soit d'être en "Vue des données" ou en "Vue des déclarations". La première option permet de voir et de modifier les données en état actuel et la seconde permet de modifier les valeurs à l'état initial.
Remarque: la "Vue des données" est particulièrement utile pour visualiser une structure de variables (type UDT, voir chapitre suivant) ou un tableau.

Le chapitre suivant est important, il est un complément du chapitre sur les DBs.

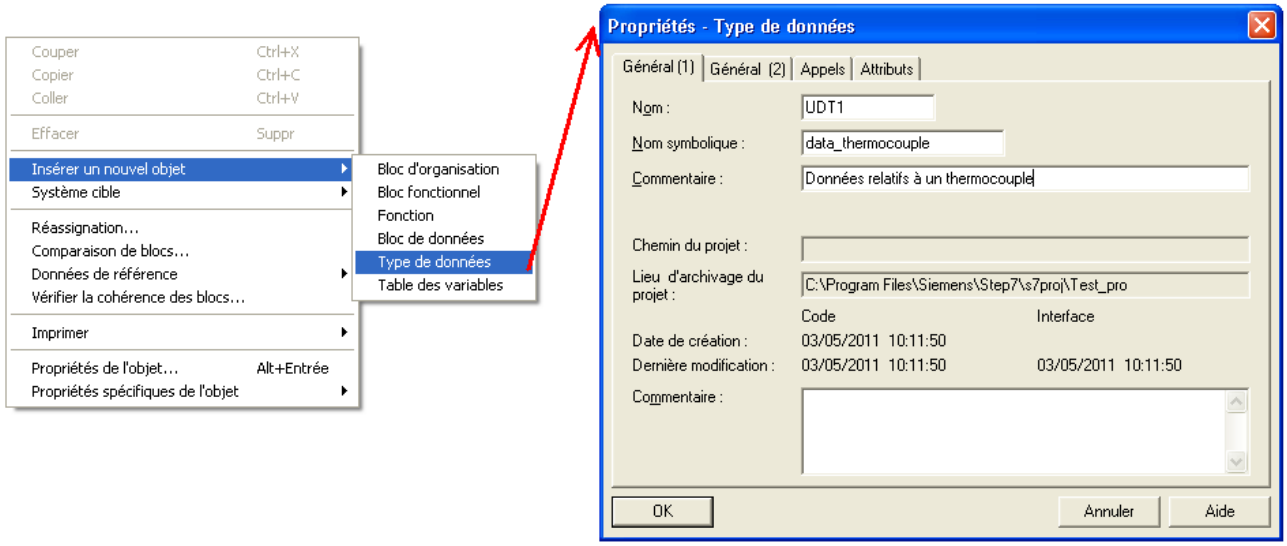
5.6 LES UDT

Il est possible dans STEP7 de créer une structure de données qui pourra être utilisé plusieurs fois dans un même DB ou dans un DB différent.
Reprenons notre exemple précédent (DB10: Température).
L'UDT sera composé seulement de 2 variables:

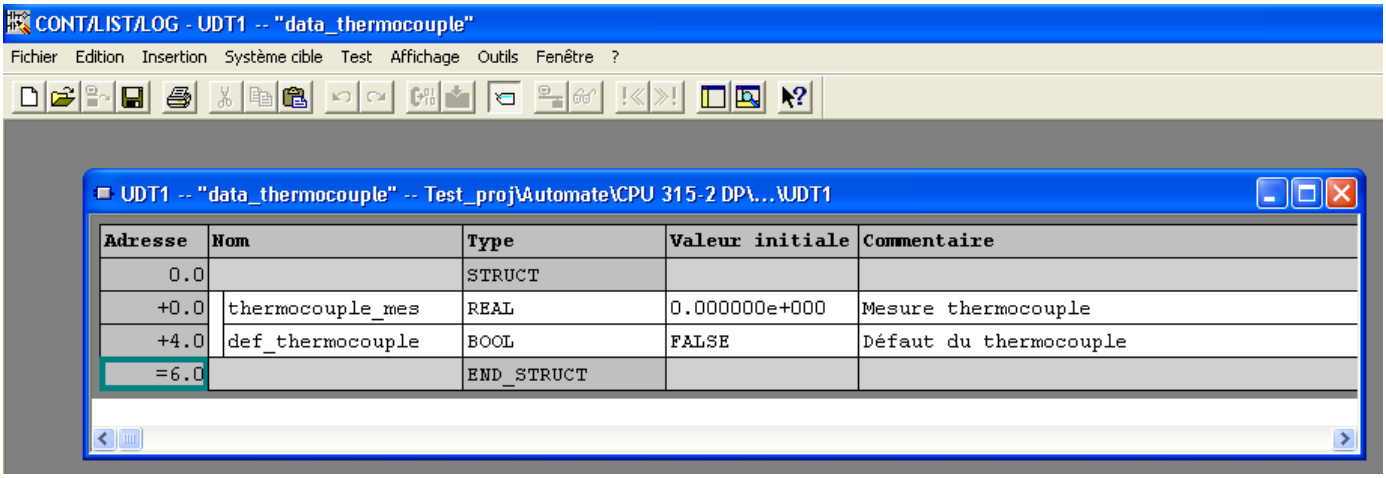
- Température de la zone mesurée par le thermocouple (type REAL).
- Défaut du thermocouple (type BOOL).

Voici la marche à suivre:
a) Dans la liste des objets du programme, faire un clic droit et choisir "*Insérer un objet > Type de données*" via le menu contextuel.

Nom de l'objet	Nom symbolique	Commentaire	Langage de création	Taille dans la mémoir...	Type	Version (en-tête)	Nom (en-tête)
Données système	---	---	---	---	SDB	---	---
OB1		"Main Program Sweep (Cy...	CONT	38	Bloc d'organisation	0.1	
DB10	Temperature		DB	56	Bloc de données	0.1	

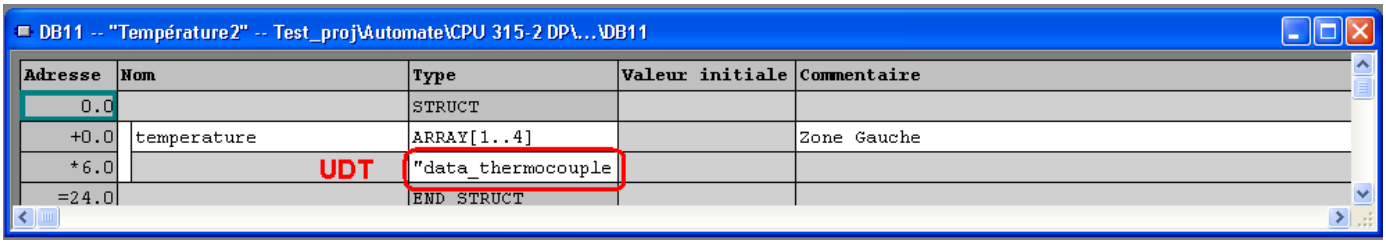


b) Double cliquer sur l'objet UDT que l'on vient de créer (UDT1 dans notre exemple) pour éditer l'UDT.



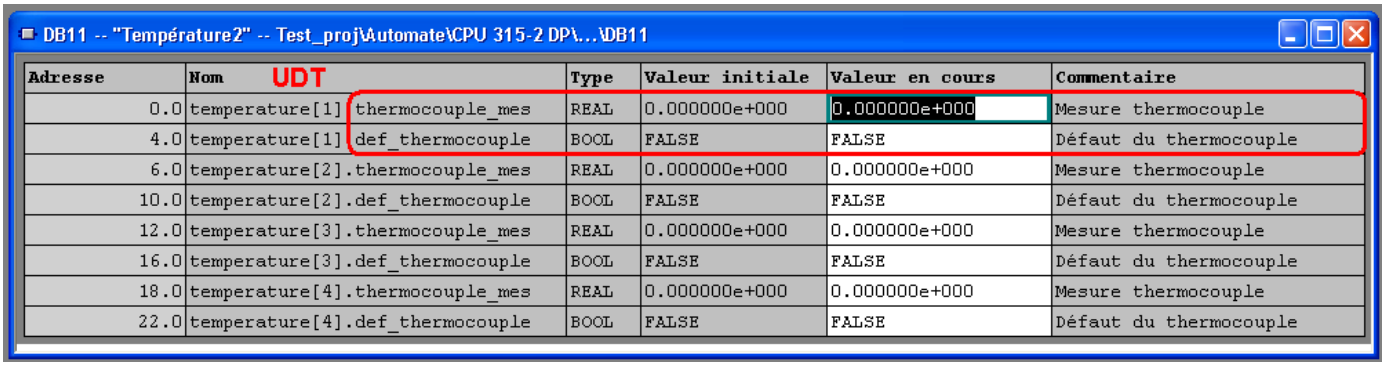
Enregistrez l'UDT après l'avoir rempli.

c) Ouvrir la DB où l'on va appeler notre UDT (dans notre exemple on crée un nouveau DB, DB11 par exemple), dans cette DB on appel 4 fois l'UDT créer précédemment pour nos 4 zones de four (comme dans le DB10), pour cela on crée un tableau (ARRAY) de 4 UDT1.


DB11 -- "Température2" -- Test_proj\Automate\CPU 315-2 DP\...\DB11

Adresse	Nom	Type	Valeur initiale	Commentaire
0.0		STRUCT		
+0.0	temperature	ARRAY[1..4]		Zone Gauche
*6.0	UDT	"data_thermocouple		
=24.0		END_STRUCT		

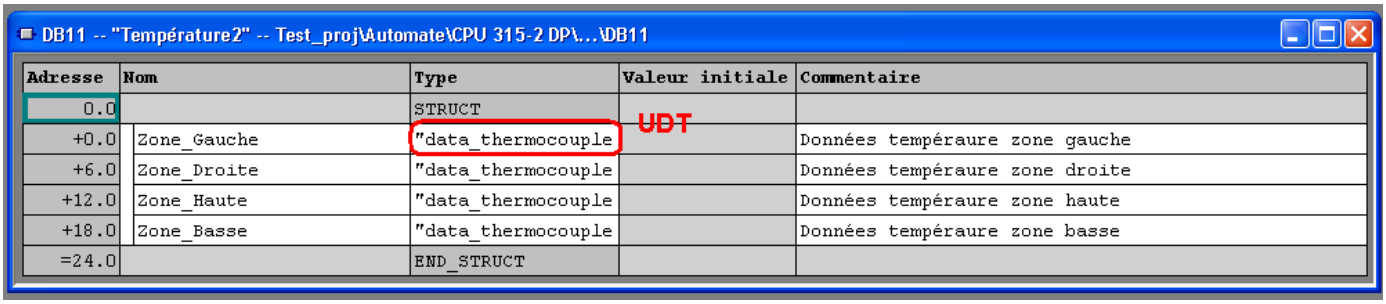
vue des déclarations


DB11 -- "Température2" -- Test_proj\Automate\CPU 315-2 DP\...\DB11

Adresse	Nom UDT	Type	Valeur initiale	Valeur en cours	Commentaire
0.0	temperature[1].thermocouple_mes	REAL	0.000000e+000	0.000000e+000	Mesure thermocouple
4.0	temperature[1].def_thermocouple	BOOL	FALSE	FALSE	Défaut du thermocouple
6.0	temperature[2].thermocouple_mes	REAL	0.000000e+000	0.000000e+000	Mesure thermocouple
10.0	temperature[2].def_thermocouple	BOOL	FALSE	FALSE	Défaut du thermocouple
12.0	temperature[3].thermocouple_mes	REAL	0.000000e+000	0.000000e+000	Mesure thermocouple
16.0	temperature[3].def_thermocouple	BOOL	FALSE	FALSE	Défaut du thermocouple
18.0	temperature[4].thermocouple_mes	REAL	0.000000e+000	0.000000e+000	Mesure thermocouple
22.0	temperature[4].def_thermocouple	BOOL	FALSE	FALSE	Défaut du thermocouple

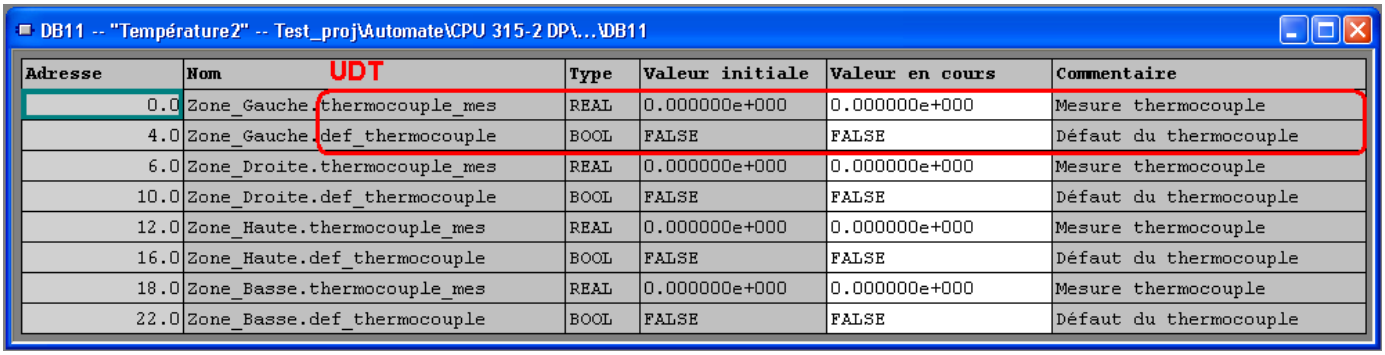
vues des données

On peut aussi appeler 4 fois à la suite l'UDT au lieu de créer un tableau afin de pouvoir donnée un nom pour chaque zone de notre four pour une meilleure lecture du programme: au lieu de "thermocouple [1]" on aura "Zone_Gauche", "thermocouple [2]" on aura "Zone_Droite", …


DB11 -- "Température2" -- Test_proj\Automate\CPU 315-2 DP\...\DB11

Adresse	Nom	Type	Valeur initiale	Commentaire
0.0		STRUCT		
+0.0	Zone_Gauche	"data_thermocouple	UDT	Données température zone gauche
+6.0	Zone_Droite	"data_thermocouple		Données température zone droite
+12.0	Zone_Haute	"data_thermocouple		Données température zone haute
+18.0	Zone_Basse	"data_thermocouple		Données température zone basse
=24.0		END_STRUCT		

vue des déclarations


DB11 -- "Température2" -- Test_proj\Automate\CPU 315-2 DP\...\DB11

Adresse	Nom UDT	Type	Valeur initiale	Valeur en cours	Commentaire
0.0	Zone_Gauche.thermocouple_mes	REAL	0.000000e+000	0.000000e+000	Mesure thermocouple
4.0	Zone_Gauche.def_thermocouple	BOOL	FALSE	FALSE	Défaut du thermocouple
6.0	Zone_Droite.thermocouple_mes	REAL	0.000000e+000	0.000000e+000	Mesure thermocouple
10.0	Zone_Droite.def_thermocouple	BOOL	FALSE	FALSE	Défaut du thermocouple
12.0	Zone_Haute.thermocouple_mes	REAL	0.000000e+000	0.000000e+000	Mesure thermocouple
16.0	Zone_Haute.def_thermocouple	BOOL	FALSE	FALSE	Défaut du thermocouple
18.0	Zone_Basse.thermocouple_mes	REAL	0.000000e+000	0.000000e+000	Mesure thermocouple
22.0	Zone_Basse.def_thermocouple	BOOL	FALSE	FALSE	Défaut du thermocouple

vues des données

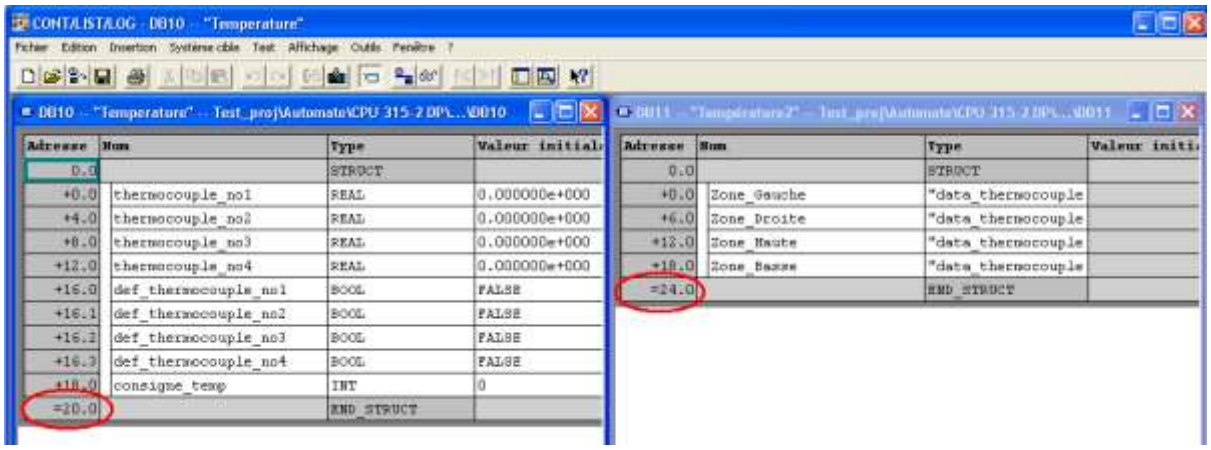
5.6.1 UTILISATION D'UNE VARIABLE ISSU D'UNE UDT DANS LE PROGRAMME

Constitution d'une la variable issu d'une structure (UDT) :

Exemple: DB11.DBX10.0
"Température2".Zone_Droite.def_thermocouple
avec:
"Température2" = DB11
Zone_Droite = variable type UDT
def_thermocouple = élément de structure UDT

5.6.2 ORGANISATION DE LA MEMOIRE DU DB

Dans notre exemple, si on compare la DB10 à la DB11, on constate que la DB11 utilise plus de mémoire car notre structure (UDT) est composée d'un REAL et d'un BOOL. Or si un BOOL est suivi d'une variable qui n'est pas un BOOL, alors celle-ci redémarre au bit 0 de l'octet suivant paire. D'où des zones de mémoire inutilisées et donc gaspillées.



Exemple:
Ceci est visible dans la vue des données du DB11, il y a une variable type BOOL en adresse 4.0 puis la variable suivante passe directement en adresse 6.0 car c'est une variable type REAL.

4.0	Zone_Gauche.def_thermocouple	BOOL	F
6.0	Zone_Droite.thermocouple_mes	REAL	0

Alors que dans le DB10 on a arrange nos variables de type BOOL les unes à la suite des autres. On perd aussi des variables mais moins que dans la DB11.

+12.0	thermocouple_no4	REAL	0.000000e+000	
+16.0	def_thermocouple_no1	BOOL	FALSE	
+16.1	def_thermocouple_no2	BOOL	FALSE	
+16.2	def_thermocouple_no3	BOOL	FALSE	
+16.3	def_thermocouple_no4	BOOL	FALSE	
+18.0	consigne_temp	INT	0	

6 PROGRAMMATION

Dans ce chapitre nous traiterons les bases pour programmer, pour des points plus spécifiques (langage LIST, …), jetez un coup dans le chapitre "Annexe".

6.1 LES DIFFERENTS LANGAGES

Voici les langages les plus répandues dans l'utilisation de STEP7, aucun langage n'est meilleur qu'un autre, cela dépendra de ce que l'on veux programmer (voir colonne "**Application**" du tableau suivant).

Langage de programmation	Groupe d'utilisateurs	Application
LIST (**List**e d'instructions)	Utilisateurs voulant une programmation proche de la machine.	Programmes optimisés en temps d'exécution et en espace mémoire.
CONT (Schéma à **con**tacts)	Utilisateurs habitués aux schémas de circuits.	Programmation de commandes combinatoires.
LOG (**Log**igramme)	Utilisateurs habitués aux boîtes logiques de l'algèbre booléenne.	Programmation de commandes combinatoires.
SCL (**S**tructured **C**ontrol **L**anguage) *(Progiciel optionnel)*	Utilisateurs ayant programmé en langages évolués comme Pascal ou C.	Programmation de tâches de programmation de données.
GRAPH (**Graph**cet) *(Progiciel optionnel)*	Utilisateurs se basant sur la technologie, ayant peu de connaissances approfondies de la programmation ou des automates programmables.	Description souple de processus séquentiels (graphcets).

6.2 SELECTION DU LANGAGE

La sélection du langage se fait à l'intérieur de l'utilitaire CONT/LIST/LOG via la barre d'outils: "*Affichage*".

On utilisera dans nos exemples le langage CONT et/ou LIST (pas le langage LOG pour ne pas surcharger les explications)

6.3 CREATION D'UN RESEAU ET INSERTION D'ELEMENT DE PROGRAMME

Ce chapitre sera traité dans l'utilitaire CONT/LIST/LOG

6.3.1 CREATION D'UN RESEAU

Que ce soit pour le langage CONT, LIST ou LOG, la programmation se fait par l'insertion de réseau en cliquant sur l'icône  dans la barre d'outils.

6.3.2 INSERTION D'ELEMENT DE PROGRAMME DANS UN RESEAU

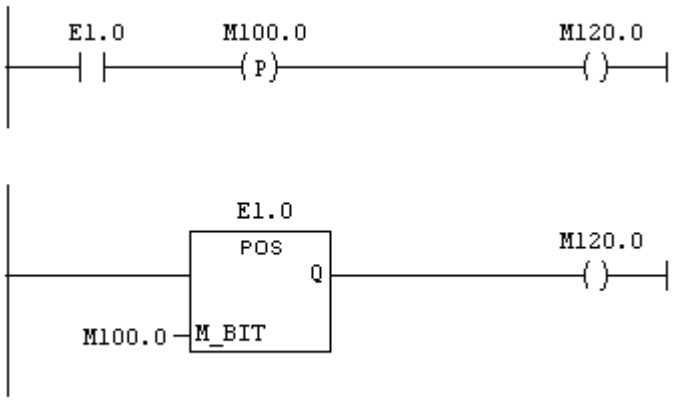
Il faut utiliser la bibliothèque …

6.4 QUELQUES "ELEMENTS DE PROGRAMME" (LANGAGES CONT ET LIST)

….

6.4.1 FRONT MONTANT ET DESCENDANT

Voici les 2 manières différentes pour créer un front montant (P), le même principe est possible avec le front descendant (N) :



E1.0 : variable sur laquelle on veut détecter le front.
M100.0 : sert uniquement à gérer le front.
M120.0 : **bit front montant.**

6.4.2 TEMPORISATIONS

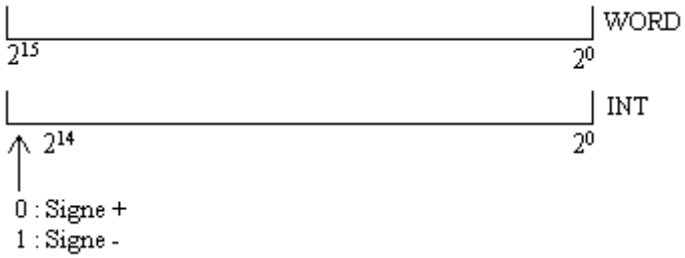
Nom	Description	Fonctionnement
SI	Impulsion	La sortie Q passe à 1 lorsque l'entrée S passe à 1 pendant un temps TW. Si S retombe à 0 alors Q retombe à 0.
SV	Impulsion maintenue	Idem ci-dessus sauf que Q reste à 1 m^me si S retombe à 0.
SE	Retard à la montée	La sortie Q passe à 1 quand le temps TW est écoulé et si S reste à 1. Si S retombe à 0 alors Q passe à 0.
SS	Retard à la montée	La sortie Q passe à 1 au bout du temps TW à condition d'avoir S à 1. Q reste à 1 tant que l'entrée RAZ reste à 0.
SA	Retard à la retombée	La sortie Q passe à 1 tant que S passe à 1. La sortie passe à 0 quand S est à 0 plus l'attente du temps TW.

6.4.3 AJOUTER UN WORD A UN INT

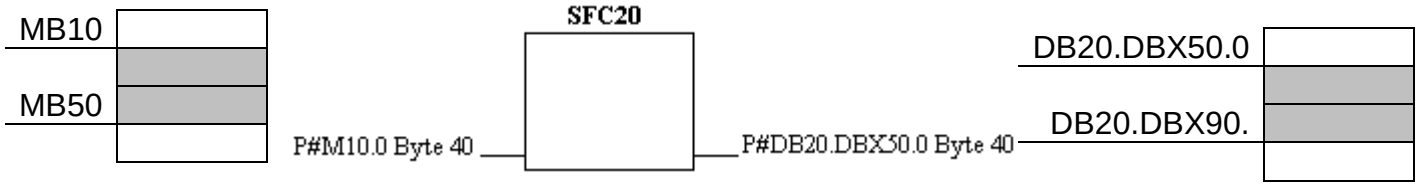
Il est possible d'ajouter un WORD avec un INT :

- <u>En LIST</u>
  L MW100 // type WORD
  L MW200 // type INT
  +I
  T MW300 // transfert de la somme dans le mot MW300

- En CONT/LOG
  Utilisation d'un bloc addition d'entier (ADD_I) et décocher "Vérification du type d'opérande" (dans barre d'outil "Outil" onglet Paramètre. pour ne pas avoir de conflit de type.



6.4.4 UTILISATION DU SFC20 (BLK_MOVE)

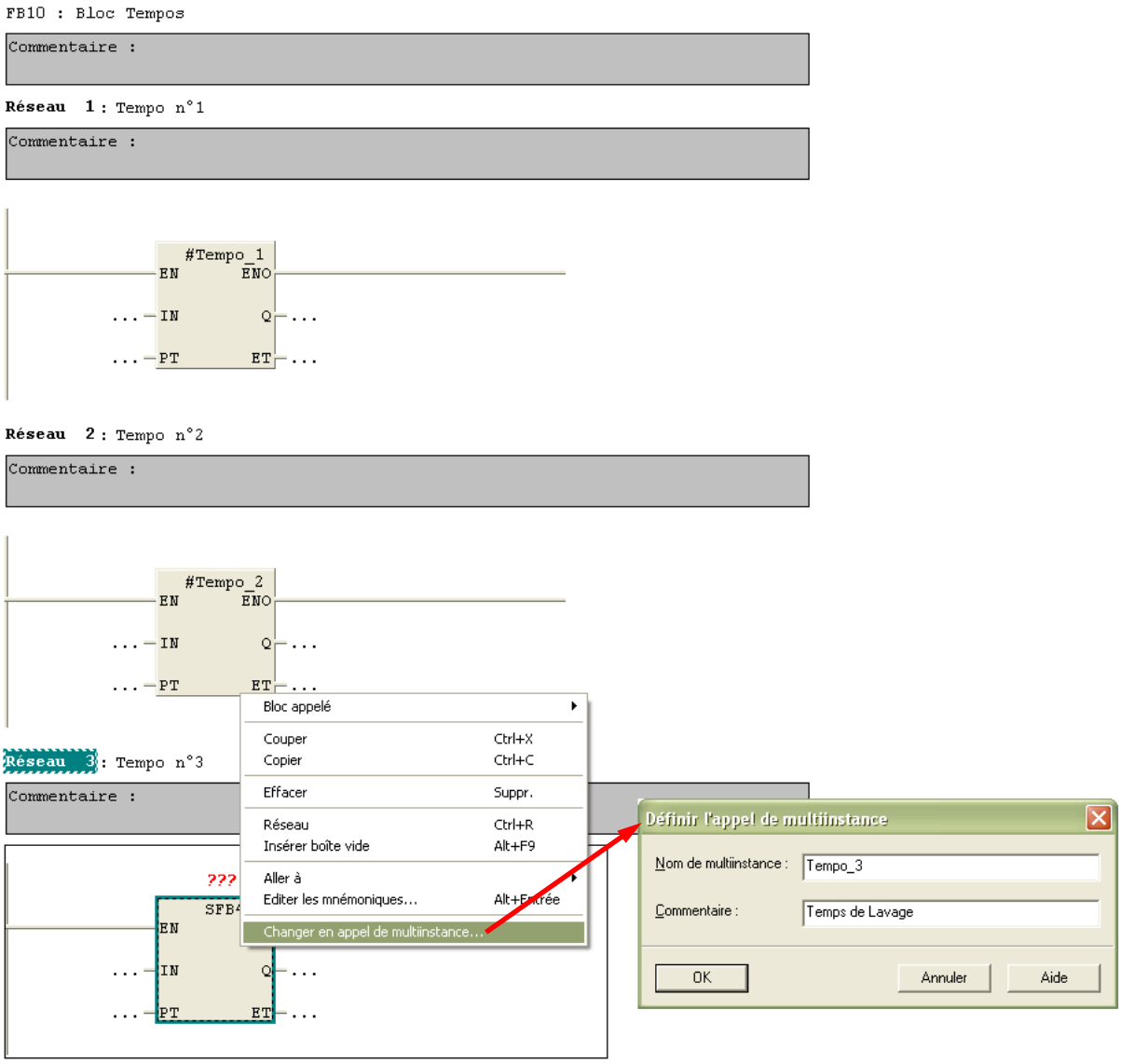


6.4.5 UTILISATION DU SFC21 (FILL)

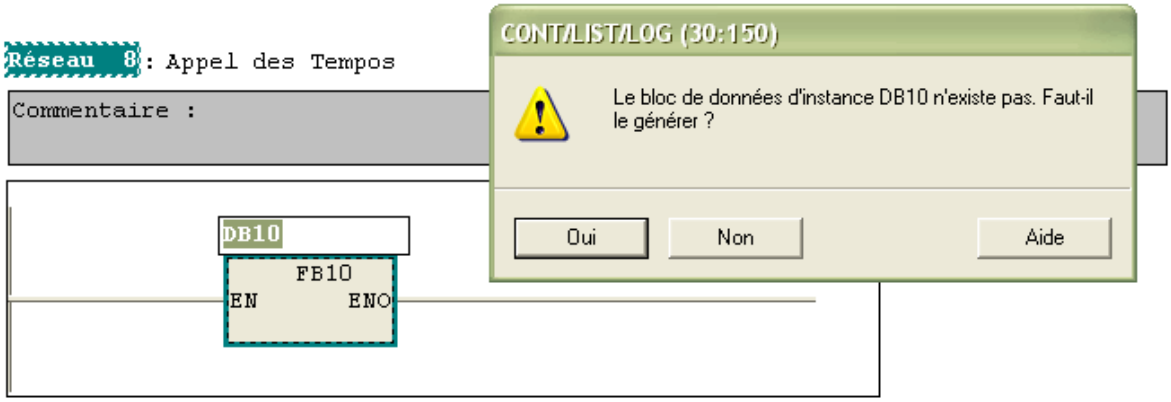
Ce bloc permet …

6.4.6 MULTINSTANCE

Pour commencer, créer un FB (dans notre exemple FB10). Insérer un réseau et appelé dans celui-ci un SFB4 (Fonction TON). Faites 2 autres réseaux semblables à celui-ci.
En temps normal, un SFB (ou un FB) à besoin d'un DB d'Instance différent à chaque appel, dans notre cas il aurait fallu 3 DB différent car la fonction SFB4 est appelé 3 fois. Pour faire du Multi-Instance, il ne faut pas attribuer un DB à la fonction mais il suffit de cliquer sur la fonction appelé et de choisir l'option "changer un appel de multinstance" (voir photo ci-dessous). Puis donner un nom spécifique à l'appel de cette fonction (dans notre exemple nous avons décider de donner un nom de tempo différent à chaque appel de la fonction SFB4.



Ceci fait, il faut maintenant appelé la fonction parent (dans notre cas FB10) dans l'OB1 par exemple (voir photo ci-dessous) et lui attribuer un DB d'instance (dans notre cas DB10) qui sera finalement un regroupement des DB d'instances de nos trois SFB4.



Architecture du DB d'instance

DB10 -- Sediver_BrossageTraction\SIMATIC 300\CPU 315-2 DP

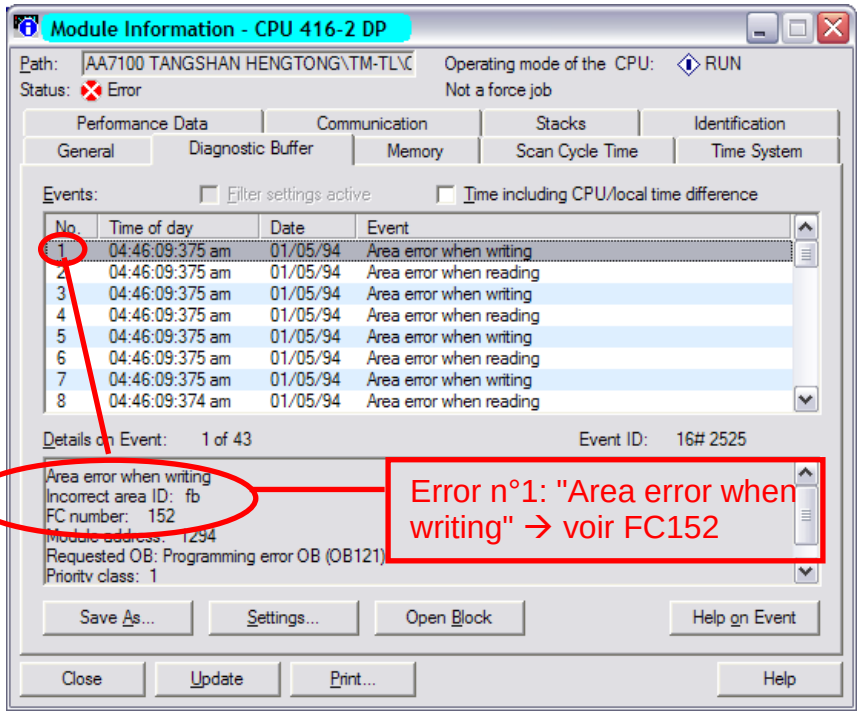
	Adresse	Décl.	Nom	Type	Valeur initiale	Valeur en cours	Commentaire
1	0.0	stat:in	Tempo_3.IN	BOOL	FALSE	FALSE	
2	2.0	stat:in	Tempo_3.PT	TIME	T#0MS	T#0MS	
3	6.0	stat:out	Tempo_3.Q	BOOL	FALSE	FALSE	
4	8.0	stat:out	Tempo_3.ET	TIME	T#0MS	T#0MS	
5	12.0	stat	Tempo_3.STATE	BYTE	B#16#0	B#16#0	
6	14.0	stat	Tempo_3.STIME	TIME	T#0MS	T#0MS	
7	18.0	stat	Tempo_3.ATIME	TIME	T#0MS	T#0MS	
8	22.0	stat:in	Tempo_2.IN	BOOL	FALSE	FALSE	
9	24.0	stat:in	Tempo_2.PT	TIME	T#0MS	T#0MS	
10	28.0	stat:out	Tempo_2.Q	BOOL	FALSE	FALSE	
11	30.0	stat:out	Tempo_2.ET	TIME	T#0MS	T#0MS	
12	34.0	stat	Tempo_2.STATE	BYTE	B#16#0	B#16#0	
13	36.0	stat	Tempo_2.STIME	TIME	T#0MS	T#0MS	
14	40.0	stat	Tempo_2.ATIME	TIME	T#0MS	T#0MS	
15	44.0	stat:in	Tempo_1.IN	BOOL	FALSE	FALSE	
16	46.0	stat:in	Tempo_1.PT	TIME	T#0MS	T#0MS	
17	50.0	stat:out	Tempo_1.Q	BOOL	FALSE	FALSE	
18	52.0	stat:out	Tempo_1.ET	TIME	T#0MS	T#0MS	
19	56.0	stat	Tempo_1.STATE	BYTE	B#16#0	B#16#0	
20	58.0	stat	Tempo_1.STIME	TIME	T#0MS	T#0MS	
21	62.0	stat	Tempo_1.ATIME	TIME	T#0MS	T#0MS	

7 DEBUG

7.1 MODULE INFORMATION

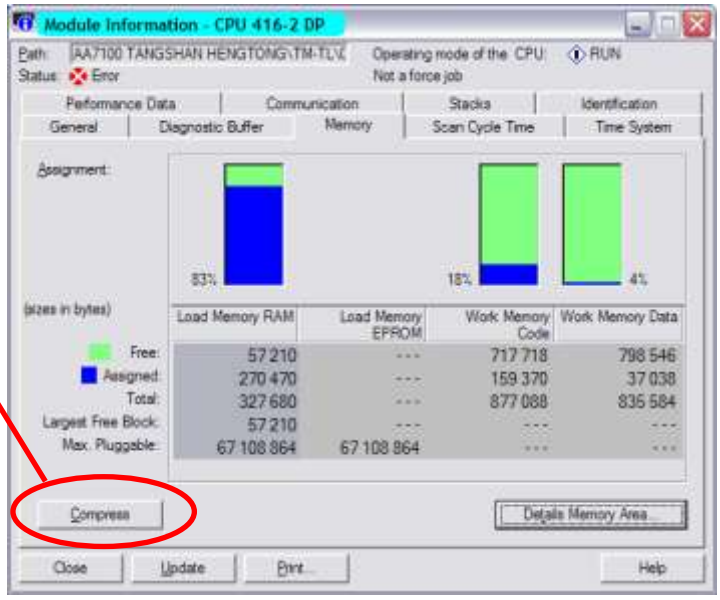
Dans la barre d'outil de la fenêtre principale du project, cliquer sur une des CPU et aller dans : PLC→ Diagnostic/Settings …→Module Information [Ctrl+D]

7.1.1 DIAGNOSTIC BUFFER



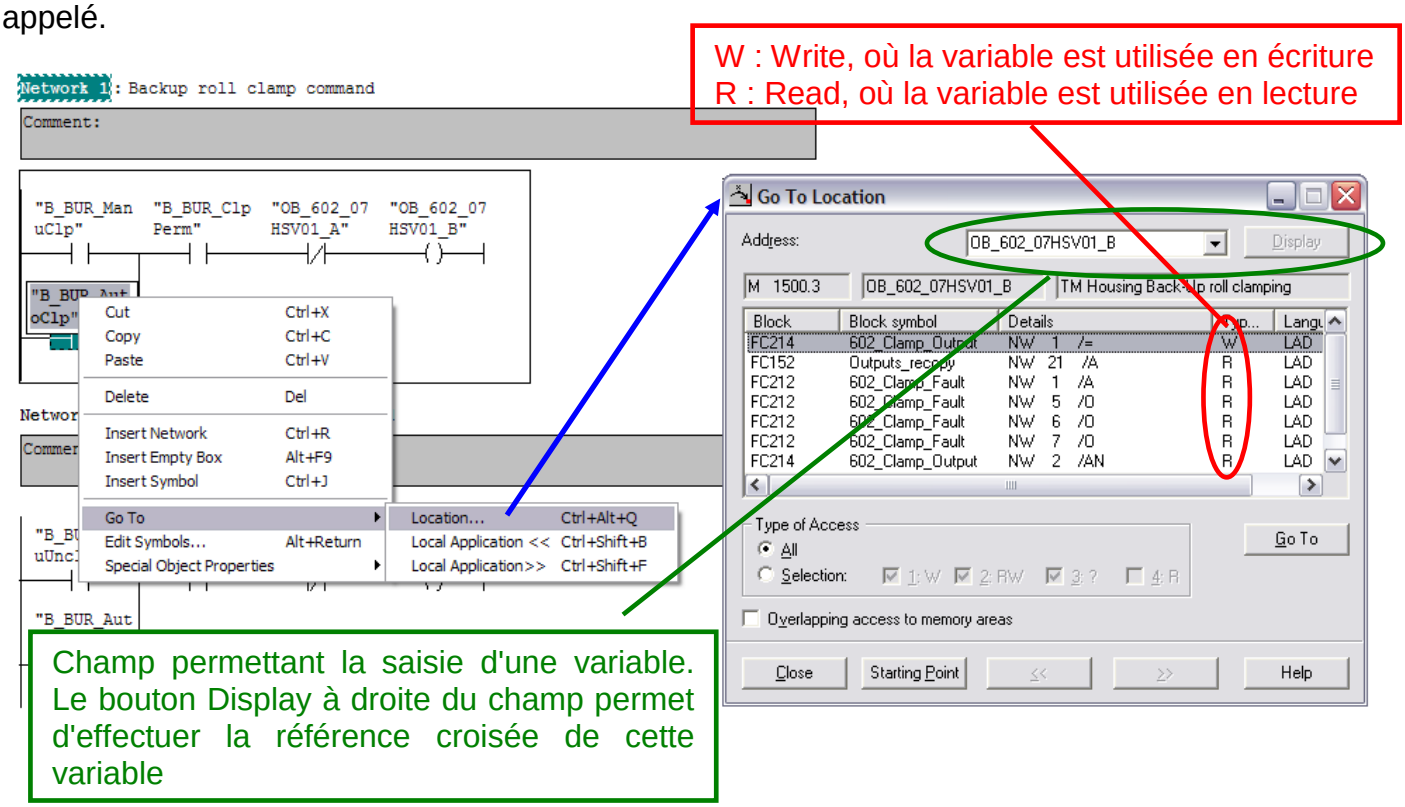
7.1.2 MEMORY

Permet de libérer de la mémoire. N.B.: limiter l'ouverture de FC, DB, VAT pendant la compression de la



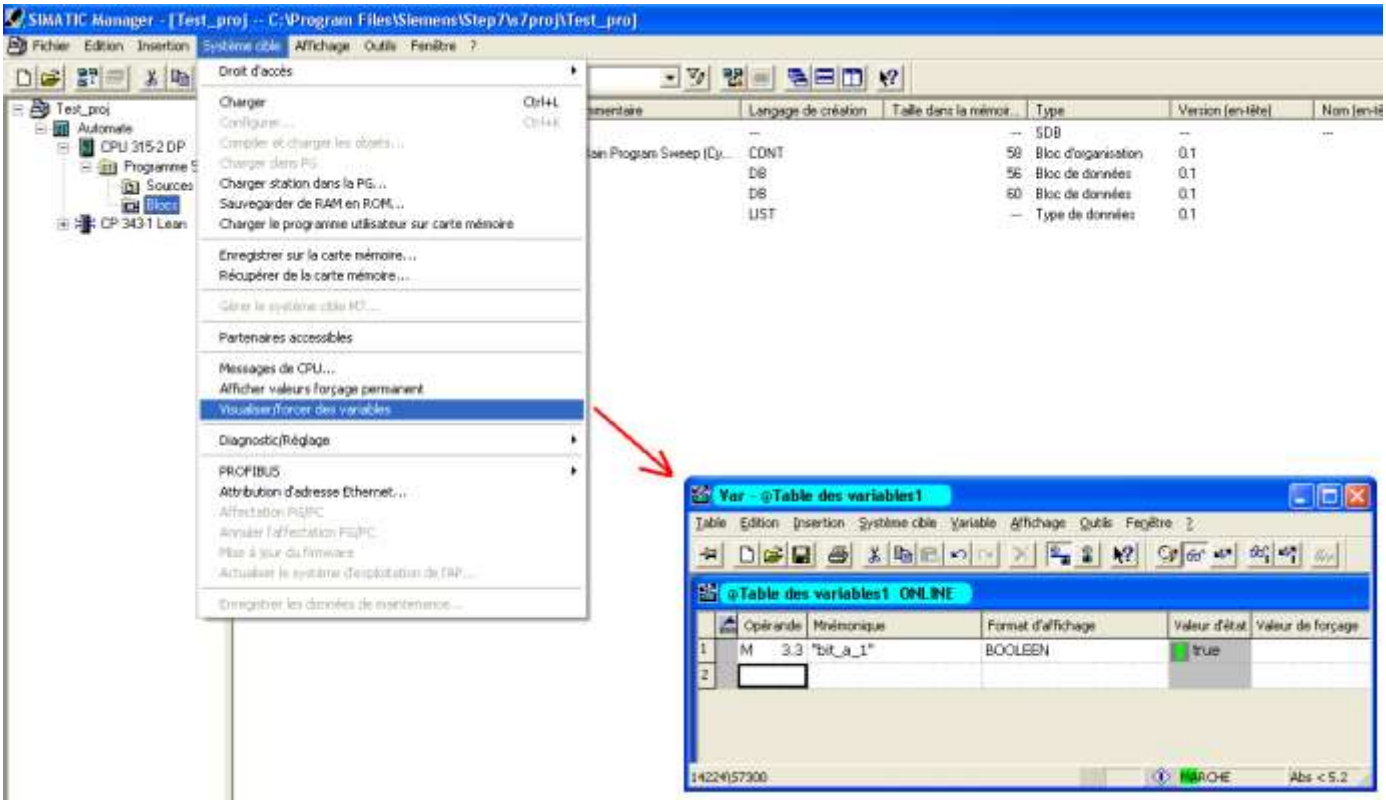
7.2 REFERENCE CROISEE

Dans le programme, cliquer droit sur une variable et choisir:
Go To→ Location … [Ctrl+Alt+Q] pour retrouver où (dans quel(s) fonction(s)) la variable est appelé.



7.3 VISUALISATION ET FORÇAGE DE VARIABLE

Il est possible de visualiser/ forcer des variables, pour cela il faut se rendre dans la barre d'outils et sélectionner : "*Système cible > Visualiser/forcer des variables*".



Cette table peut être enregistrée via la barre d'outils de la table de variables : "*Table > Enregistrer sous …*".

Il est possible de créer une table de variables de la même manière que l'on crée un FC, FB, OB ou UDT: dans la liste des objets du programme, faire un clic droit et choisir "*Insérer un objet > Table des variables*" via le menu contextuel.

7.3.1 UTILISATION

Voir l'aide

7.3.2 DECLENCHEMENT

Voir l'aide

7.4 FORÇAGE PERMANENT

Vous pouvez affecter des valeurs fixes à des variables individuelles d'un programme utilisateur afin qu'elles ne puissent ni être modifiées ni être écrasées, même par le programme utilisateur exécuté dans la CPU. Il faut évidemment que la CPU possède cette fonction (comme, par exemple, la CPU S7-400).

Le forçage permanent de variables à des valeurs fixes permet de simuler des situations précises pour votre programme utilisateur et de tester ainsi les fonctions programmées.

Pour afficher cette fenêtre, choisissez la commande "*Variable > Afficher valeurs de forçage permanent*" via la barre d'outils d'une Table des variables. Ou sinon via la barre d'outils du projet, sélectionner : "*Système cible > Afficher valeur forçage permanent*".

Remarque lorsque au moins une seule variable est forcée, la LED orange "FRCE" s'allume sur la CPU.

7.4.1 UTILISATION

Voir aide

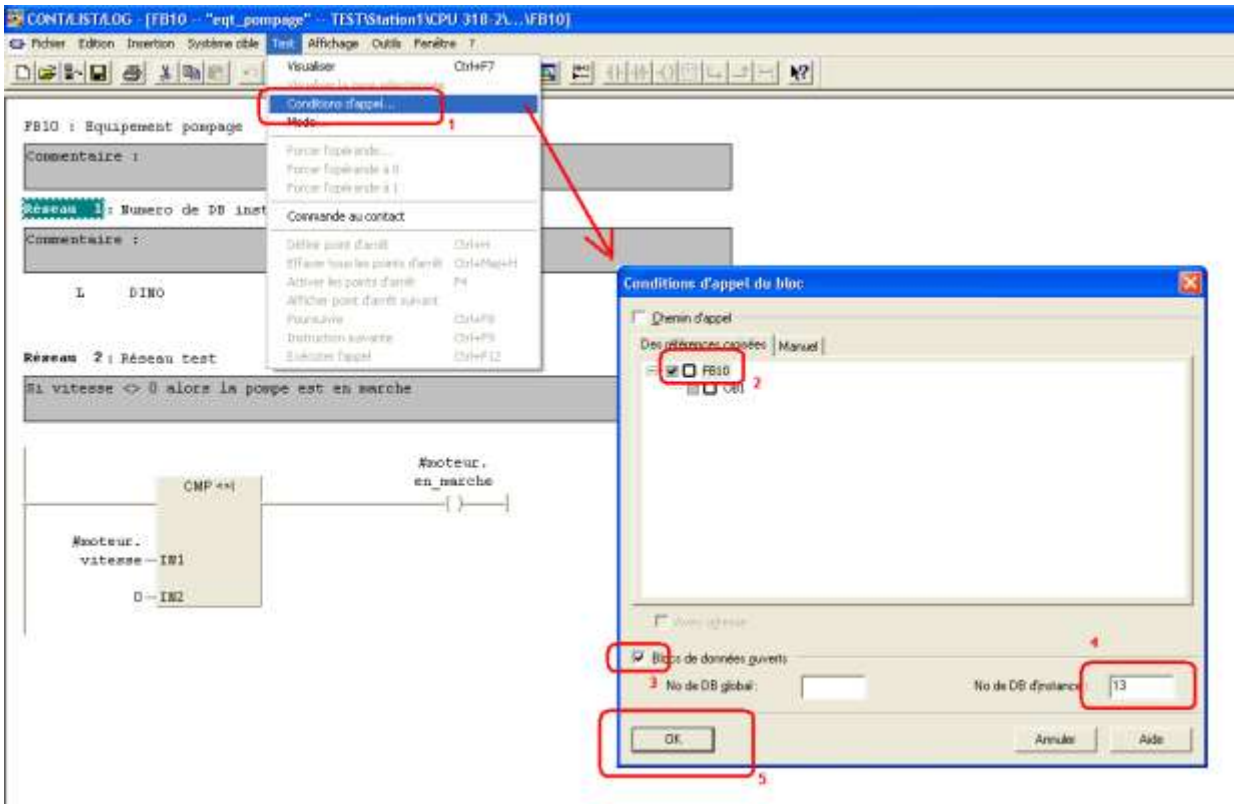
7.4.2 DIFFERENCES ENTRE FORÇAGE DE VARIABLES ET FORÇAGE PERMANENT DE VARIABLES

Voir l'aide

7.5 VISUALISATION D'UN FB EN FONCTION D'UNE DB D'INSTANCE

Lors d'appels multiples d'un bloc, il existe la possibilité d'observer le bloc avec un appel tout à fait déterminé. Pour observer un bloc avec l'environnement d'appel souhaité, procédez de la manière suivante:

1) Aller dans le menu **« Test / Mode … »** via la barre d'outils puis valider par **OK** la fenêtre en ayant contrôle que le "Mode test" était bien coché.
2) Aller dans le menu **« Test / Conditions d'appel … »** via la barre d'outils :



3) Passez en mode visu dynamique.

Pour des informations complémentaires :
http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&lang=fr&objid=21671563&caller=view

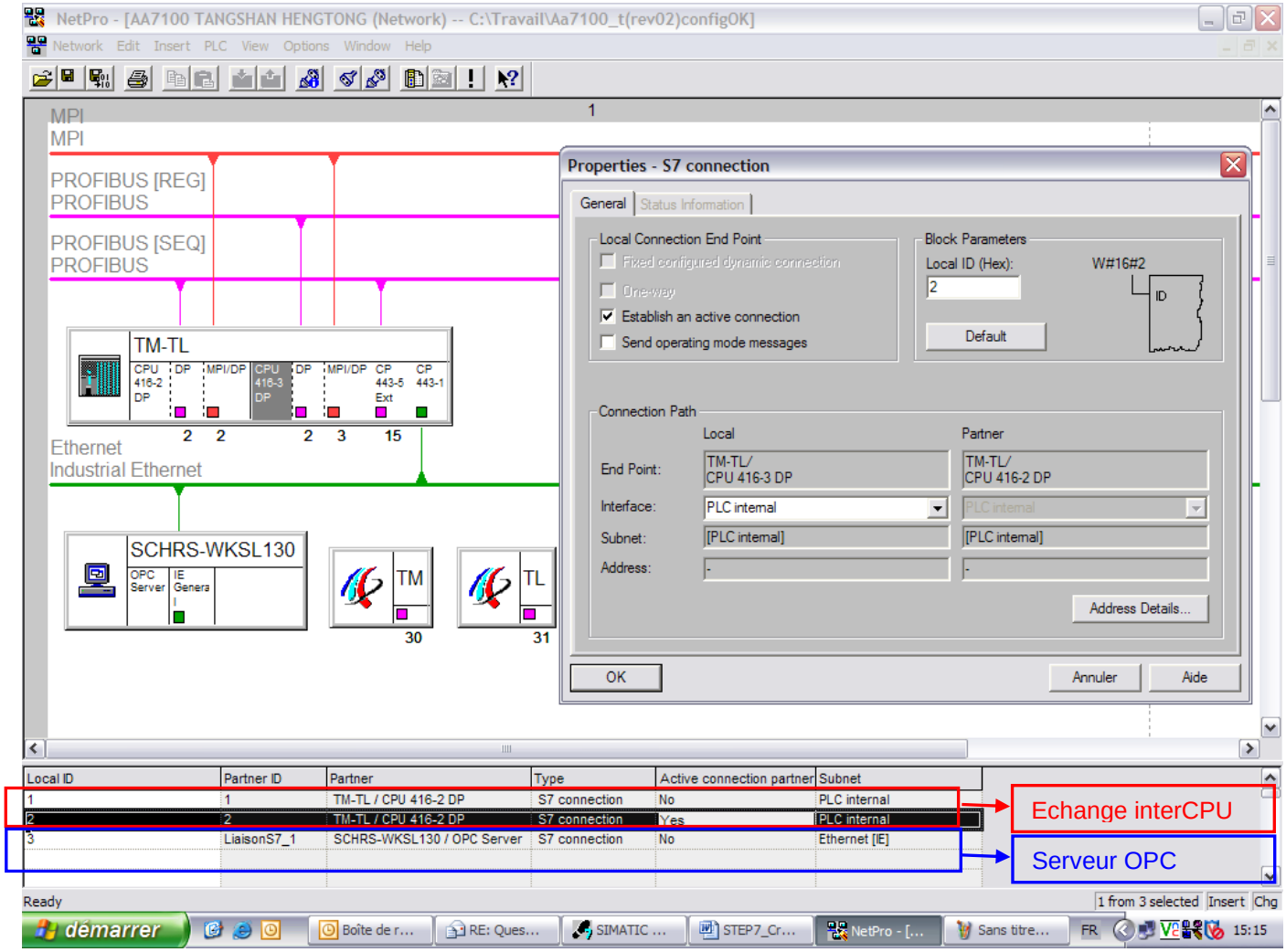
8 UTILITAIRES DE STEP7

8.1 ARCHIVER ET DESARCHIVER UN PROJET

Dans la fenêtre projet il est possible d'archiver son projet (au format .zip) en allant dans la barre d'outils : "*Fichier > Archiver…*" ("*Fichier > Désarchiver…*" permet lui d'aller chercher un projet archiver pour le désarchiver). Une boîte de dialogue nous demande le projet à archiver, une fois sélectionner valider par OK. Une seconde boîte de dialogue vous demande où voulez vous sauvegarder le fichier d'archivage, indiquez-lui l'emplacement et validez (bouton "enregistrer"). Le nom du fichier ne doit pas comporte plus de 8 caractères.

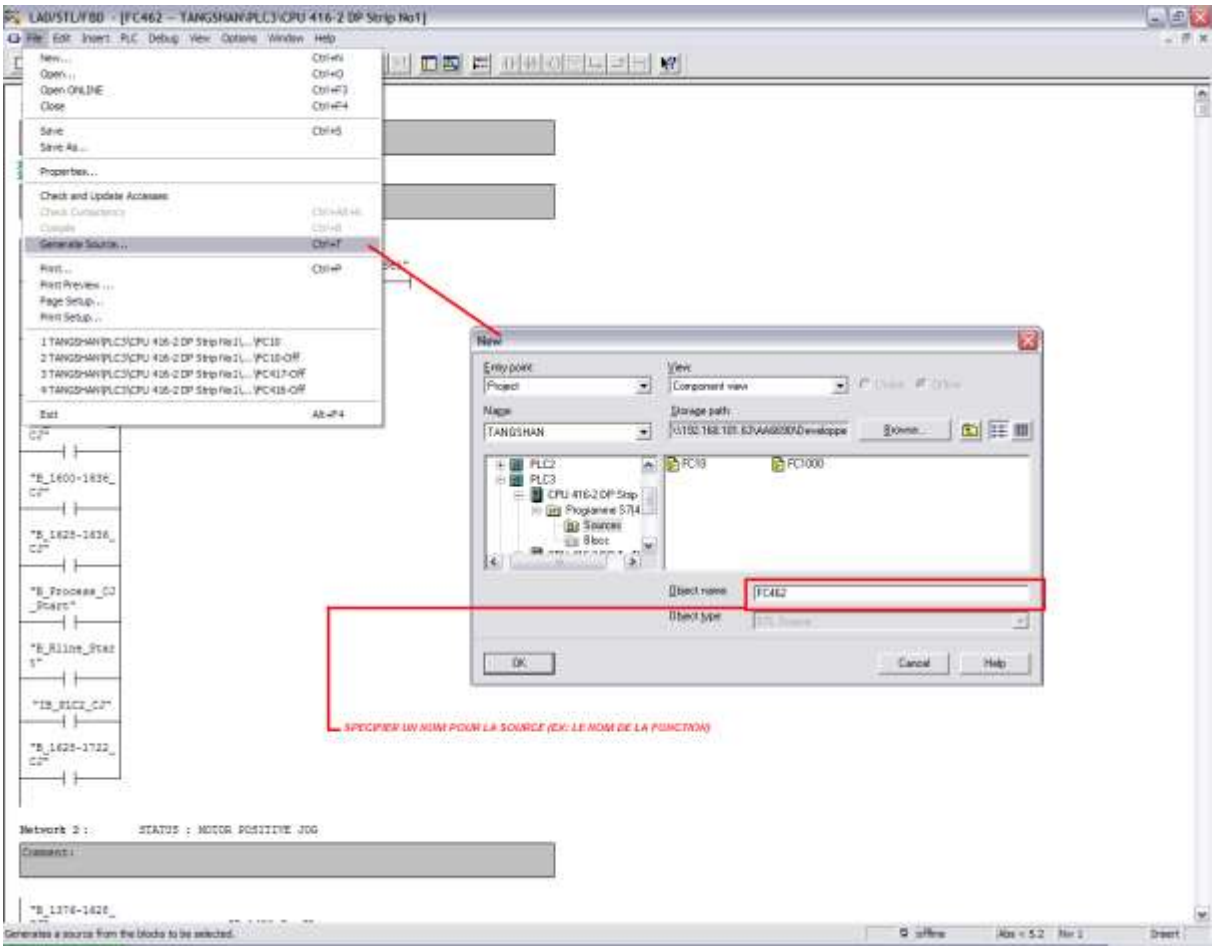
8.2 NETPRO

Dans la fenêtre projet, choisissez dans la barre d'outil : Options\Configure Network, pour lancer NetPro. Ceci est utile pour visualiser l'ensemble des connexions réseaux. Il permet aussi de créer des liaisons entre CPU dans un PLC où entre CPU de différent PLC et aussi de créer une liaison OPC.



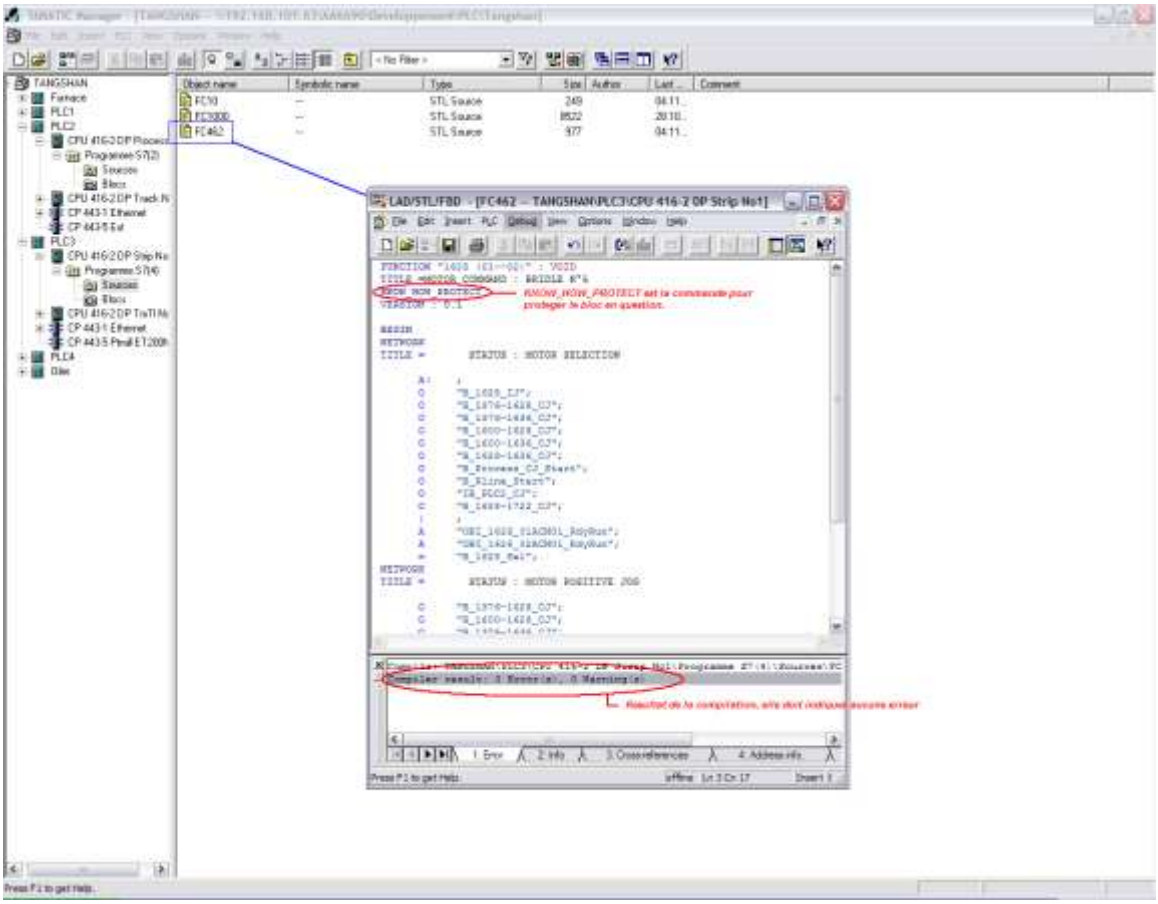
8.3 GENERER UN SOURCE A PARTIR D'UN BLOCK ET LE PROTEGER

Pour commencer il faut tout d'abord ouvrir un block dans votre programme (l'OB1 par exemple). Ensuite dans la barre d'outil choisir "File\Generate Source …" et donner un nom à votre fichier source (voir image ci-dessous)



En validant le nom de votre source une autre fenêtre apparaît et vous demande de sélectionner les blocks à générer en code dans votre fichier source (voir image ci-dessous).

Une fois ceci validé, allez dans votre arborescence projet, sous votre CPU → Programme\Source et ouvrir le fichier source que vous venez de créer à l'étape précédente.

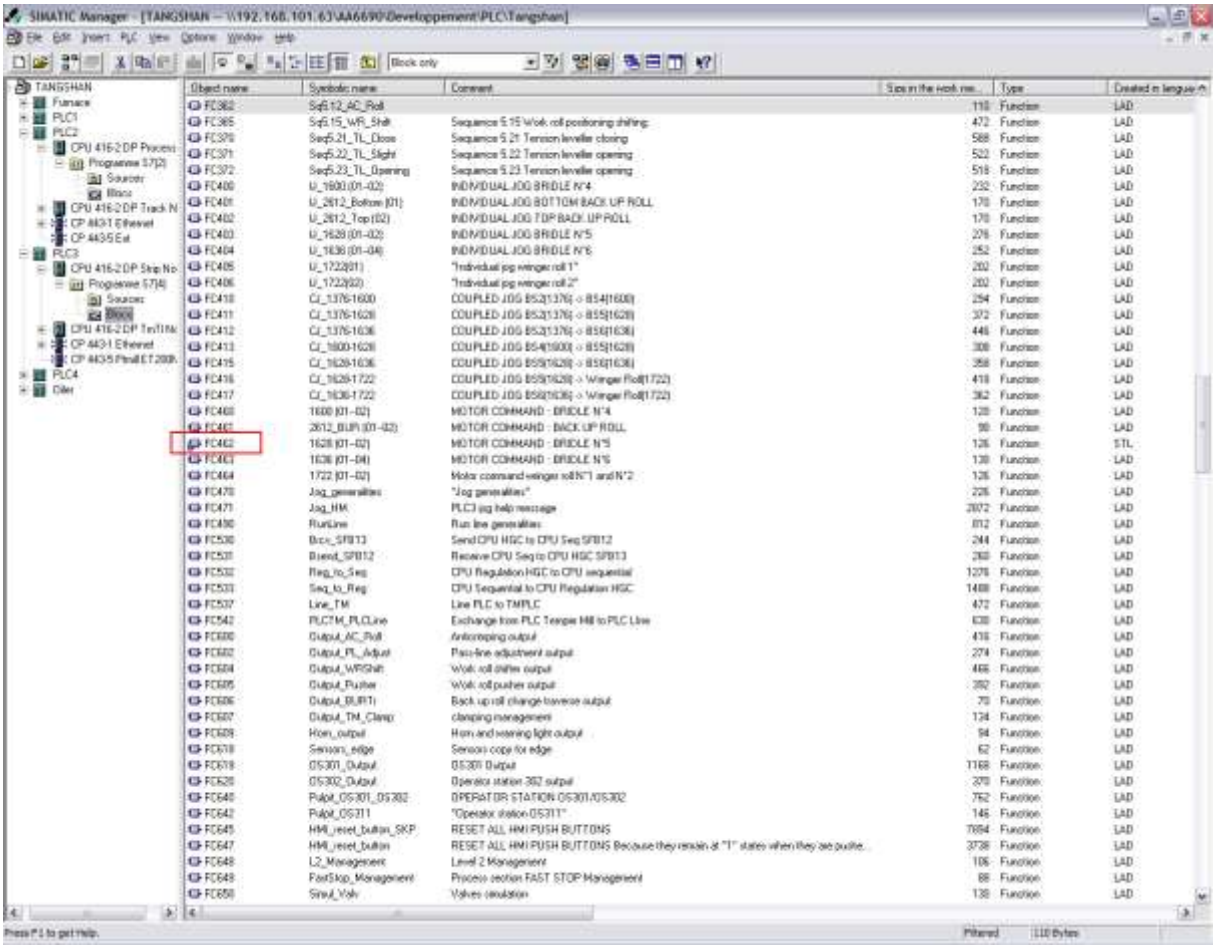


Comme sur l'image ci-dessus, rajouter la commande KNOW_HOW_PROTECT en dessous de la propriété TITLE de votre Block (s'il y a plusieurs Block dans la source répéter ceci pour chacun d'entre eux).

Attention: ne pas oublier de fermer les blocks concernés (seul le fichier source doit rester ouvert)

Enfin, il suffit de compiler (dans la barre de menu: File\Compile…) pour créer le(s) Block(s) en version protégée.

Sur la photo ci-dessous le bloc FC462 est bien protégé (on peut voir un cadenas sur l'icône de la fonction)



9 ANNEXE

9.1 COMMUNICATION PROFIBUS ENTRE 2 AUTOMATES S7-300

9.1.1 MISE EN SITUATION

Nous disposons de deux automates: Station1 et Station2, chaque automate dispose d'un projet STEP7 respectivement TEST et TEST2. Le but est d'établir une zone d'échange entre les deux automates. Chaque automate est à la fois Maître (quand il envoi) et Esclave (quand il reçoit).

	TEST	TEST2
Station1	Maître	Esclave
Station2	Esclave	Maître

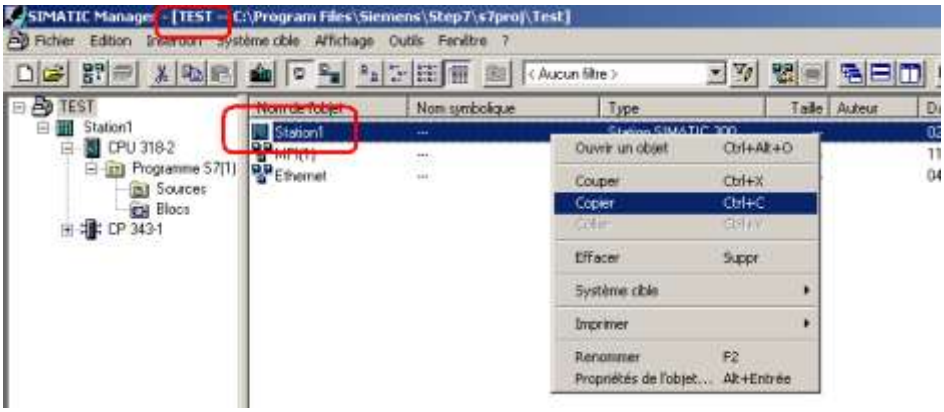
9.1.2 MISE EN ŒUVRE

En Profibus, seul des Entrées et Sorties peuvent être lu avec cette méthode. Pour avoir accès aux autres variables, il faut ajouter des coupleurs (CP) aux les automates afin de pouvoir configurer une liaison S7 entre automates. Par contre, les ports Ethernet intégrés des CPU permettent de d'établir une liaison S7.

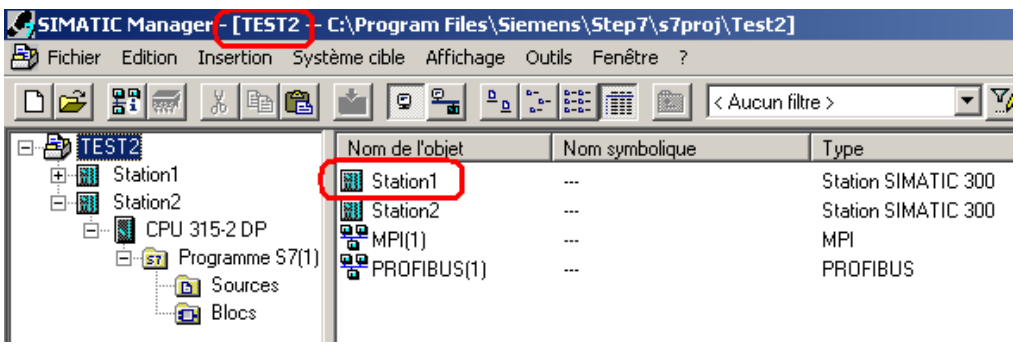
1) Copie de la station dans l'arborescence projet

Il faut copier la station de chaque projet et la coller dans l'autre projet:

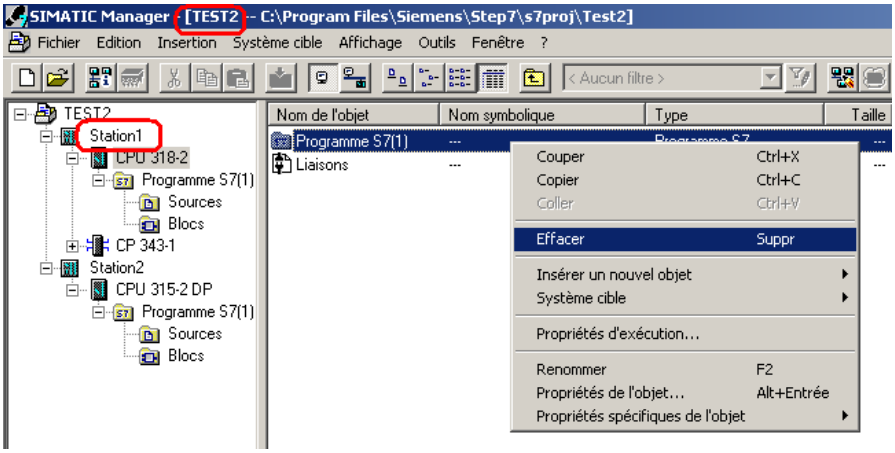
a) dans le premier projet (dans l'exemple le projet est : TEST), cliquer gauche sur la station puis faire "copier".



b) dans l'autre projet (dans l'exemple le projet est : TEST2), "coller" la station.



c) effacer le programme de la station qui vient d'être "coller" (on a besoin que de sa configuration matérielle pas du programme).



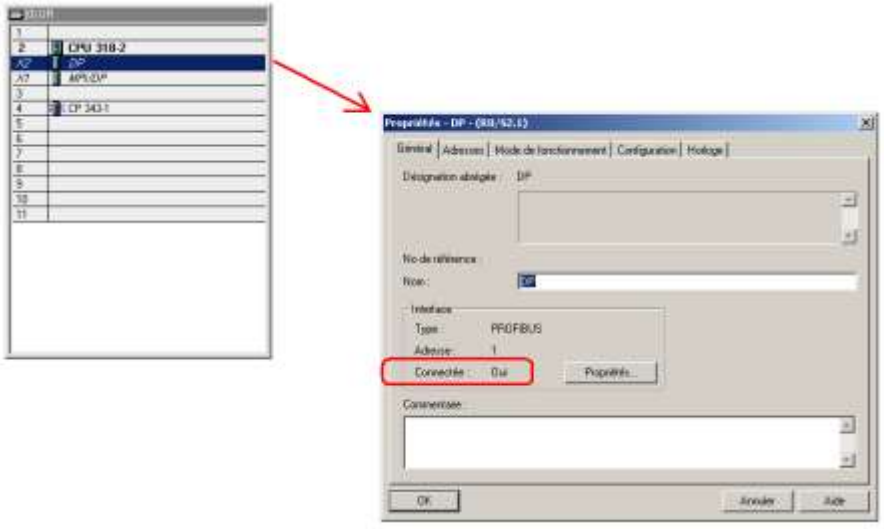
d) Répéter les étapes a,b et c pour l'autre projet (dans l'exemple le projet est : TEST).

2) Configuration matérielle pour les 2 projets

Marche à suivre pour le projet TEST2 par exemple.

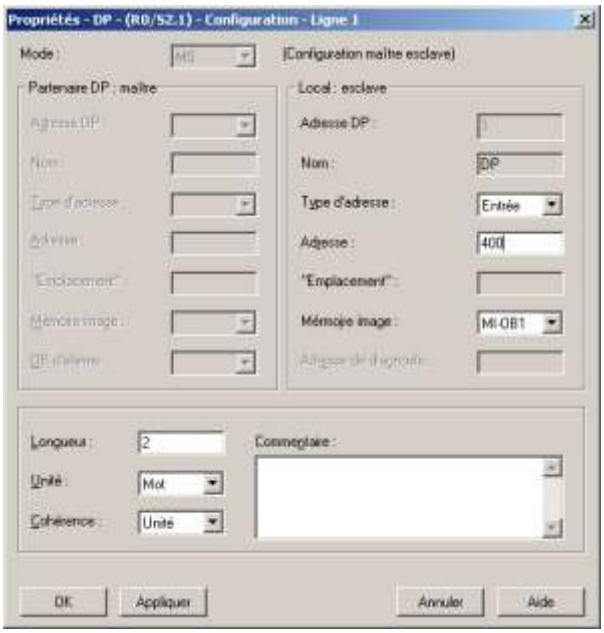
*** Ouvrir la configuration matérielle de l'automate "Esclave" (Station1)**

a) dans la configuration matérielle, activer le réseau PROFIBUS:



b) dans la même fenêtre, dans l'onglet "Mode de fonctionnement, cocher la case "Esclave DP".

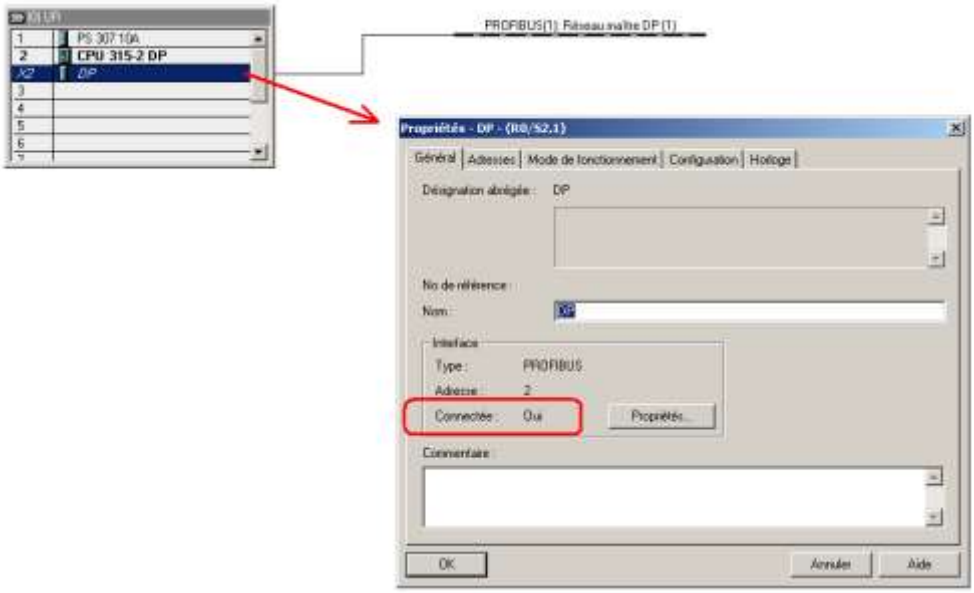
c) dans la même fenêtre, dans l'onglet "Configuration", faire "Nouveau" afin de définir la plage d'échange (en réception).



d) compiler puis enregistrer pour vérifier que la configuration soit correct.

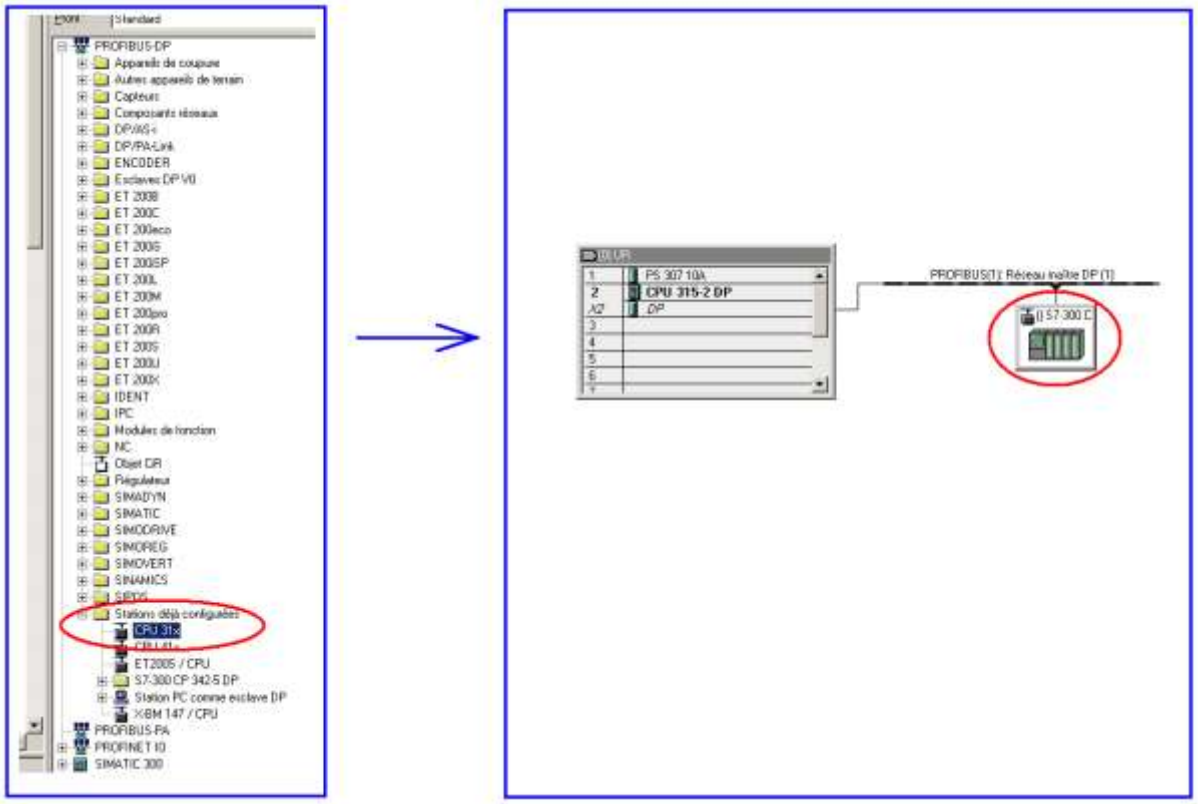
*** Ouvrir la configuration matérielle de l'automate "Maître"**

a) dans la configuration matérielle, activer le réseau PROFIBUS:

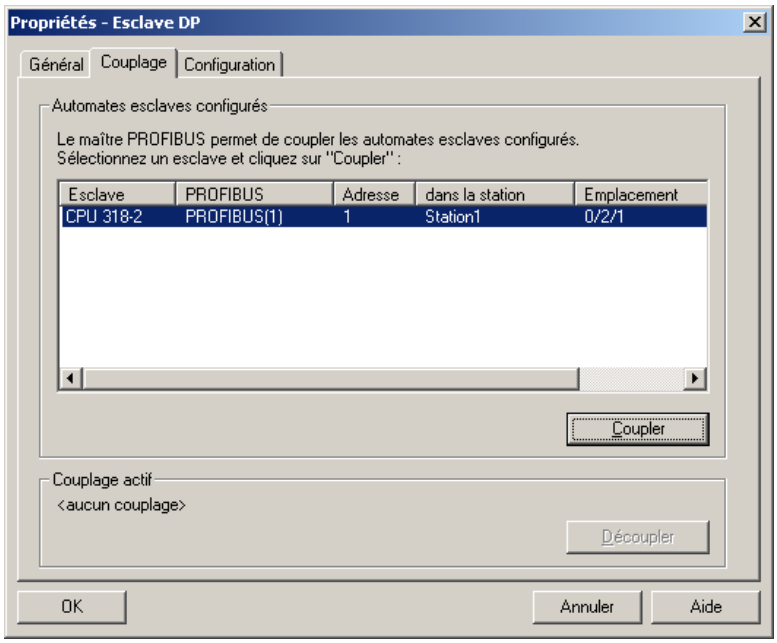


b) Dans la même fenêtre, dans l'onglet "Mode de fonctionnement, vérifier que la case "Maître DP" soit cochée.

c) Rajouter sur le réseau Profibus un automate S7-300 en tant que esclave DP:

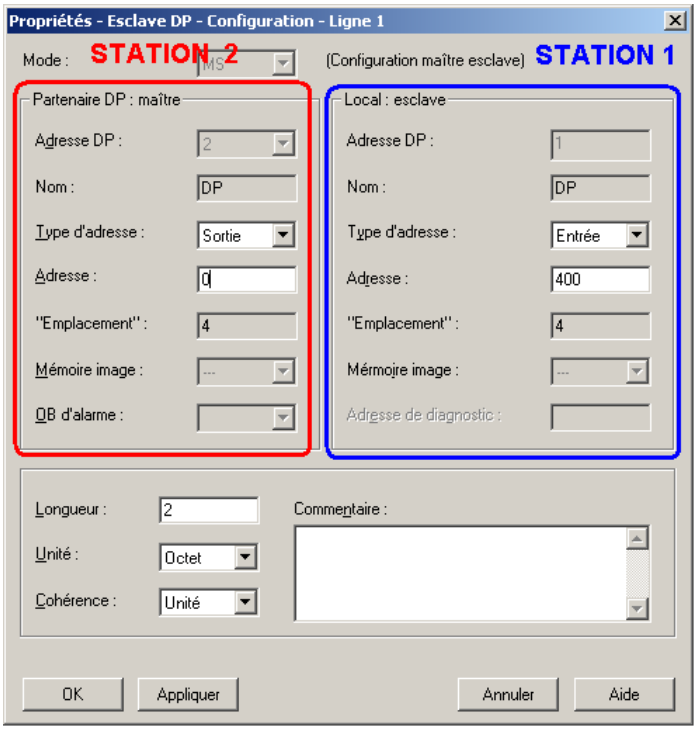


d) Double-cliquer sur la station, dans l'onglet "Couplage", faire "coupler" afin de définir avec quel esclave on va dialoguer.



Remarque: On reconnaît bien l'autre CPU configurée précédemment en Esclave (Station1).

e) Dans la même fenêtre, dans l'onglet "Configuration", faire "Nouveau" afin de définir la plage d'échange côté maître (en émission), la station 1 (esclave) est déjà configuré car cela a déjà été fait dans la configuration matérielle de la **station 1 du projet TEST2.**



f) compiler puis enregistrer pour vérifier que la configuration soit correct. Puis charger dans l'automate.

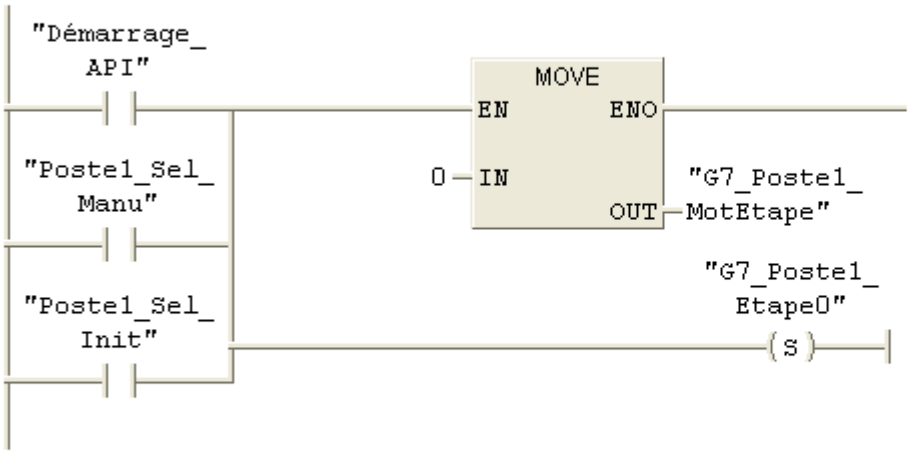
Ces étapes permettaient à la Station2 d'envoyer des données vers la Station1, pour pouvoir envoyer des données de la Station1 vers la Station2 il suffit de répéter les étapes comprises dans 2.1 et 2.2 dans l'autre projet en définissant la Station1 en Maître et la Station2 en esclave et en réglant correctement les plages d'adresses.

9.2 GRAPHCET EN LANGAGE CONT

Voici une manière de créer un Graphcet en langage CONT

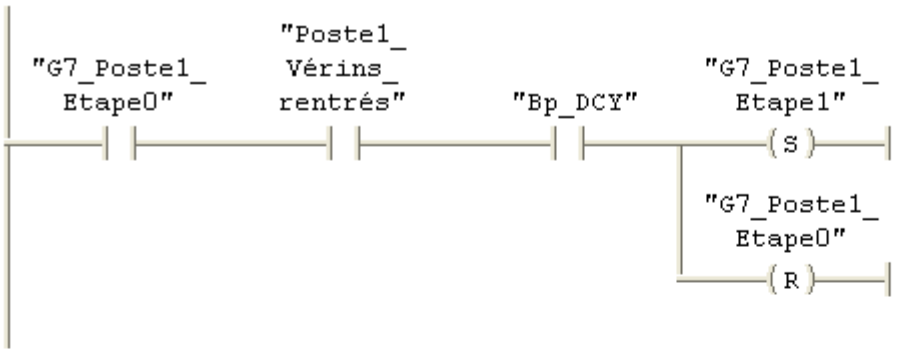
Réseau 1 : RAZ Graphcet Poste 1

Commentaire :



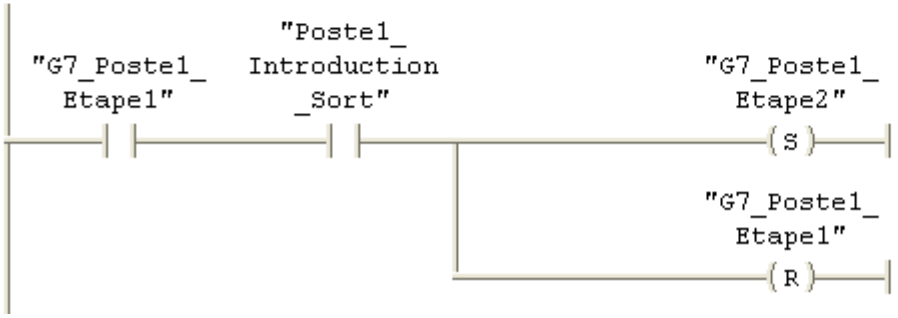
Réseau 2 : Graphcet Poste 1 : Activation Etape 1

Sortie Vérn Introduction



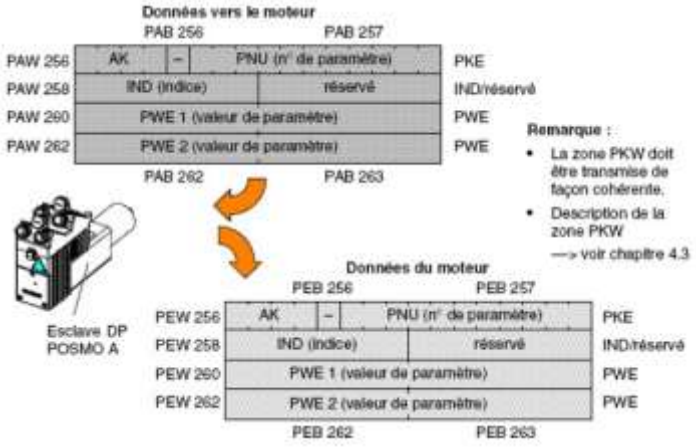
Réseau 3 : Graphcet Poste 1 : Activation Etape 2

Rentrée Vérin Introduction

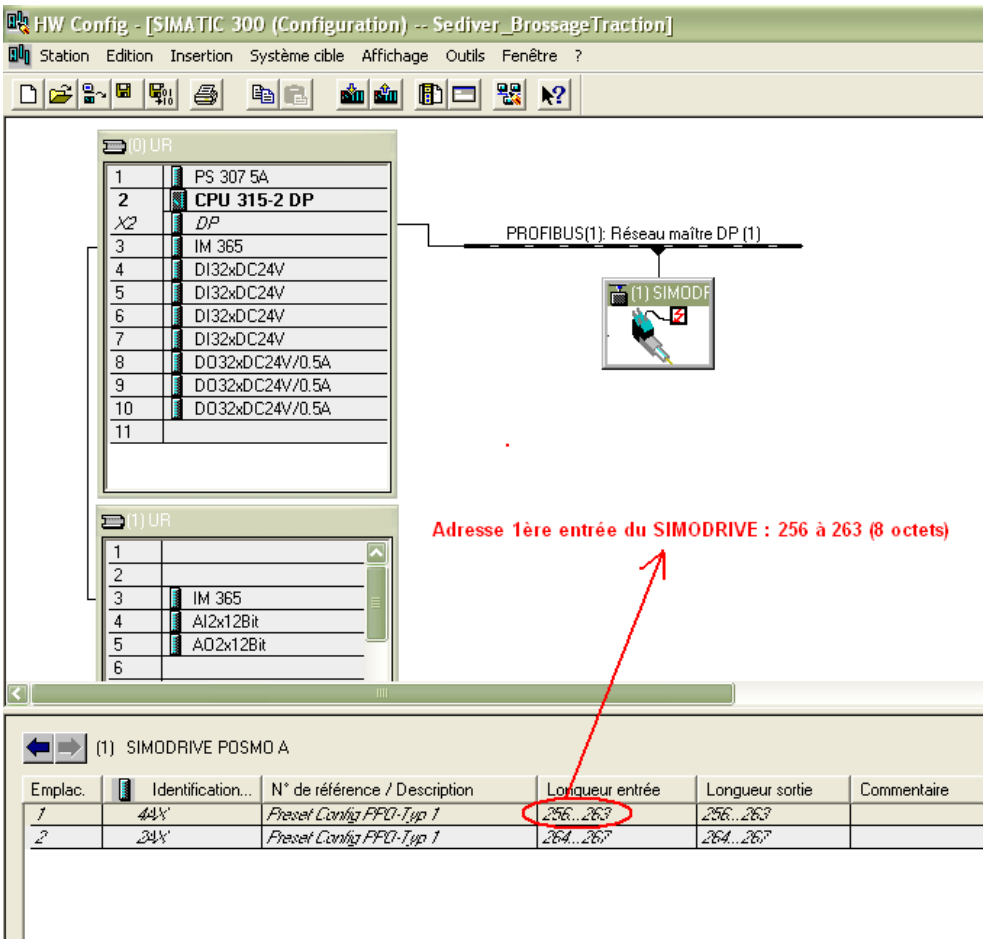


9.3 COMMUNICATION DP AVEC MOTEUR BRUSHLESS

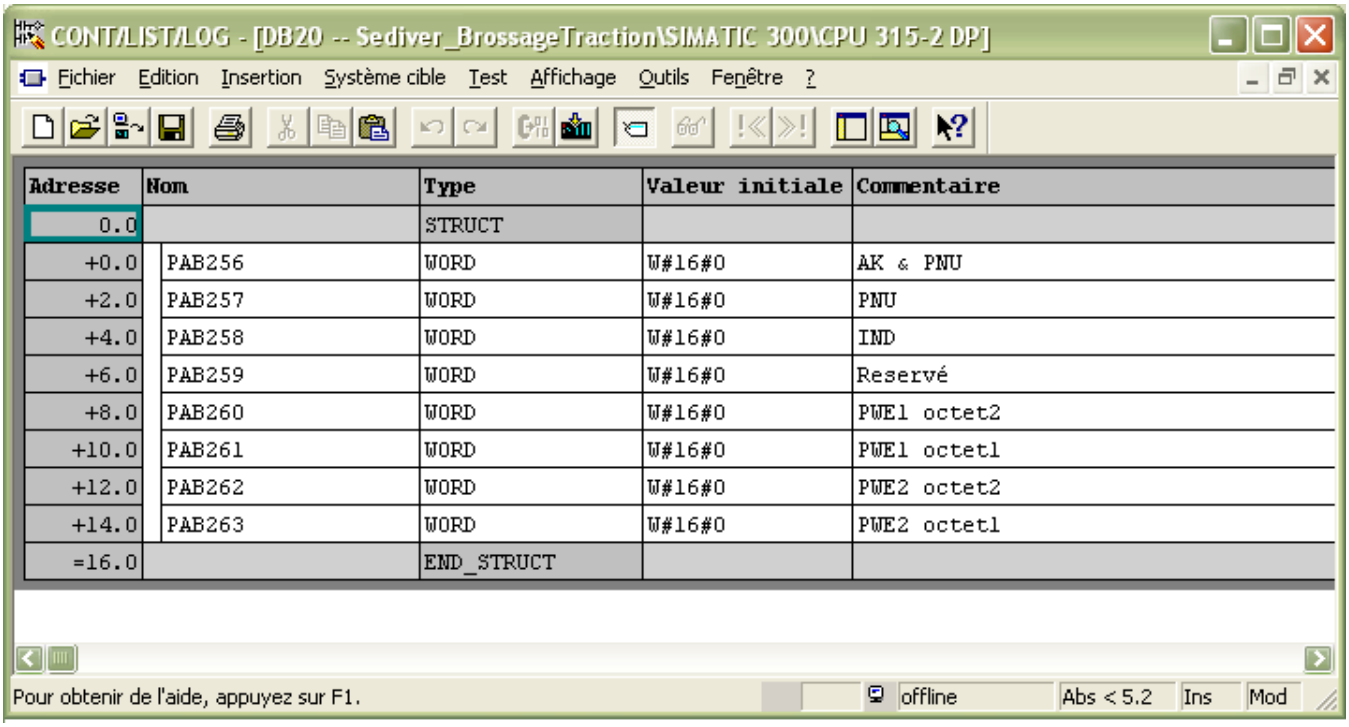
Voici la trame qui permet de communiquer avec un SIMODRIVE POSMO A



Identification de l'adressage dans configuration matériel



Création du DB contenant octets d'échange pour l'écriture



CONT/LIST/LOG - [DB20 -- Sediver_BrossageTraction\SIMATIC 300\CPU 315-2 DP]

Fichier Edition Insertion Système cible Test Affichage Outils Fenêtre ?

Adresse	Nom	Type	Valeur initiale	Commentaire
0.0		STRUCT		
+0.0	PAB256	WORD	W#16#0	AK & PNU
+2.0	PAB257	WORD	W#16#0	PNU
+4.0	PAB258	WORD	W#16#0	IND
+6.0	PAB259	WORD	W#16#0	Reservé
+8.0	PAB260	WORD	W#16#0	PWE1 octet2
+10.0	PAB261	WORD	W#16#0	PWE1 octet1
+12.0	PAB262	WORD	W#16#0	PWE2 octet2
+14.0	PAB263	WORD	W#16#0	PWE2 octet1
=16.0		END_STRUCT		

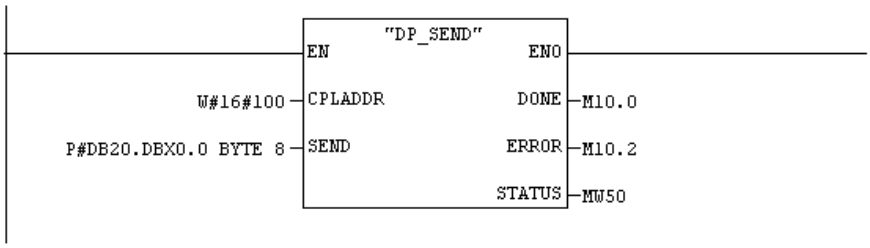
Pour obtenir de l'aide, appuyez sur F1. offline Abs < 5.2 Ins Mod

##erreur sur cette photo, ce ne sont pas des WORD mais des BYTE

Programmation

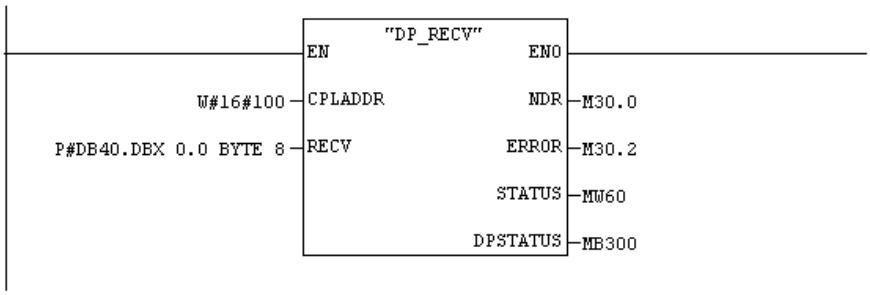
Réseau 1 : Envoi de 8 octets à l'esclave DP adresse 256 (PAW256-PAW262)

```
CPLADDR = 100 -> 256 en HEXA = 100
```



Réseau 2 : Réception de 8 octets de l'esclave DP adresse 256(PEW256-PEW262)

```
CPLADDR = 100 -> 256 en HEXA = 100
```



9.4 LANGAGE SCL

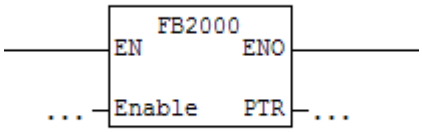
Le langage SCL s'apparente au C et permet sous STEP7 de fabriquer des FBs ou des FCs. On l'utilise surtout pour les calculs et non pas pour le séquentiel qui utilise plutôt les langages List, Ladder ou Logigramme.

Exemple :

```
FUNCTION_BLOCK FB2000     // déclaration du FB2000

//Déclaration des variables

VAR_INPUT                 // variable(s) d'entrée du block
   Enable      :  BOOL;   // Block enable
END_VAR

VAR_OUTPUT                // variable(s) de sortie du block
   PTR         :  INT;    // pointer sur adresse
END_VAR

VAR                       // variable(s) interne du block
   TimeTemp    :  REAL;   // base de temps
END_VAR

//Programme principal

BEGIN

IF  ….
   THEN ….
   ELSE ….
   END_IF;
END_FUNCTION_BLOCK        // fin du FB2000
```

Écrire le programme

Voici le résultat après l'appel du FB dans un autre bloc :



La variable interne TimeTemp n'est ni en entrée ni en sortie elle est uniquement utilisée à l'intérieur du programme comme zone de mémoire et de calcul.

On retrouve comme on peut le voir dans le programme en SCL ci-dessus la fonction BEGIN à la manière d'une fonction *main ()* en C qui est donc le point de départ du programme à exécuté à l'intérieur du FB (ou FC). De la même manière qu'en C, on doit déclarer ses variables avant l'exécution BEGIN, on remarque aussi la fonction IF qui nous est familière.

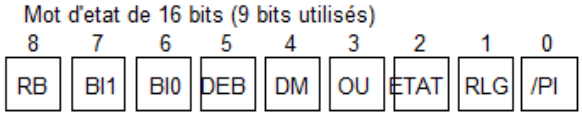
9.5 LANGAGE LIST

C'est un langage textuel, qui est le plus proche du comportement interne de l'automate. Le programme se compose d'une suite de lignes, chacune spécifiant un code opération suivi d'un seul opérande.

9.5.1 TRAITEMENT SUR BIT

9.5.1.1 Notion importante : Le Mot d'état

Les bits du ***mot d'état*** sont lus et écrits par la CPU et directement par vos instructions de programmation.
Lors du traitement des différentes instructions, les bits du mot d'état de la CPU sont utilisés en fonction de l'opération. Les instructions de votre programme sont combinées aux bits du mot d'état, qui indiquent également les informations d'erreur et les résultats directs. Les instructions peuvent alors lire les bits dans le mot d'état et réagir en conséquence.

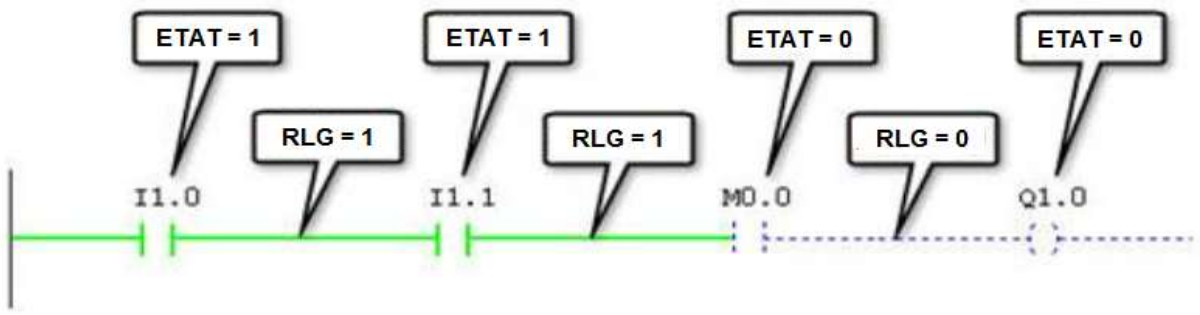


N°	bit	Fonction
0	/PI	**Première interrogation** L'état de signal du bit /PI gère l'exécution d'une séquence combinatoire. Chaque opération de combinaison interroge l'état de signal du bit /PI ainsi que l'état de signal du contact auquel l'opération accède. - Si l'état de signal du bit /PI est 1, une opération combine le résultat de l'interrogation de l'état de signal du contact auquel elle accède au résultat logique (RLG) formé depuis la première interrogation, et sauvegarde ensuite le résultat dans le bit RLG. - Si l'état de signal du bit /PI est 0, la séquence combinatoire commence avec une première interrogation. Elle s'achève avec l'affectation d'une valeur (S, R, =) ou avec une opération de saut dépendante du RLG et le bit /PI est mis à 0.
1	RLG	**Résultat logique** Le bit RLG mémorise le résultat d'une séquence d'opérations de combinaison ou de comparaison. La première opération dans un réseau interroge l'état de signal d'un contact. Le bit RLG est mis à 1 si l'interrogation est satisfaite. La deuxième opération dans le réseau interroge également l'état de signal d'un contact. Le résultat de cette interrogation est alors combiné à la valeur sauvegardée dans le bit RLG selon les règles de l'algèbre de Boole, le résultat étant mémorisé dans le bit RLG. Cette séquence combinatoire s'achève après une affectation ou un saut conditionnel ; l'exécution d'une affectation ou d'un saut conditionnel dépend de la valeur figurant dans le bit RLG.
2	ETAT	**Bit d'état** Le bit ETAT contient la valeur d'un bit en accès. L'état de signal d'une opération de combinaison qui a accès en lecture à la mémoire (U, UN, O, ON, X ou XN) est toujours le même que la valeur du bit en accès. L'état d'une opération de combinaison qui a accès en écriture à la mémoire (R, S, ou =) est identique à la valeur du bit dans lequel a lieu l'écriture. En cas d'absence d'écriture, l'état de signal est égal à la valeur du bit en accès. Le bit d'état est sans signification pour les opérations de combinaison qui n'accèdent pas à la mémoire. Ces opérations mettent le bit d'état à 1. Le bit d'état n'est pas interrogé par des opérations, mais est uniquement évalué pour l'affichage de l'état en ligne de variables de programme.
3	OU	Le bit OU est utilisé lorsque vous exécutez une opération ET avant une opération OU. Le bit OU est mis à 1 lorsque le RLG de la combinaison ET égale 1, anticipant ainsi le résultat de la combinaison OU. Toute autre opération de traitement de bits remet le bit OU à 0.

4	DM	**Débordement mémorisé** Le bit DM mémorise le bit DEB (débordement) en cas d'erreur lors d'opérations arithmétiques ou d'opérations de comparaison avec nombres à virgule flottante. Si une erreur apparaît, le bit de débordement mémorisé DM est mis à 1 en même temps que le bit DEB (débordement). Son état de signal est conservé, même après élimination de l'erreur. Le bit DM mémorise donc l'état de signal du bit DEB et indique si une erreur s'est produite dans l'une des opérations exécutées précédemment. Les opérations suivantes remettent le bit DM à zéro : - SPS (saut si DM égale 1), - appels de bloc et - fin de bloc.
5	DEB	**Débordement** Le bit DEB signale des erreurs lors d'opérations arithmétiques ou d'opérations de comparaison avec nombres à virgule flottante. Il est mis à 1 par une opération arithmétique ou une opération de comparaison de nombres à virgule flottante lorsqu'apparaît une erreur telle que débordement, opération illicite, comparaison illicite. Le bit DEB est remis à 0 une fois l'erreur éliminée.
6	BI0	**Bits indicateurs** Les bits indicateurs BI1 et BI0 mettent à disposition les résultats des opérations suivantes : - Opérations combinatoires sur bits - Opérations de comparaison - Opérations arithmétiques - Opérations de décalage et de rotation - Opérations combinatoires sur mots *Nota : BI1 et BI0 peuvent être lus par des opérations de saut conditionnel.*
7	BI1	
8	RB	Le bit RB sert à transmettre le résultat du traitement d'instructions LIST aux instructions suivantes à traiter. Si vous écrivez un bloc fonctionnel ou une fonction en LIST et si vous désirez l'appeler en CONT/LOG, vous devez ranger le résultat logique RLG dans le bit RB juste avant de quitter le bloc afin de disposer de la sortie de validation (ENO) pour le pavé de représentation CONT/LOG. Pour sauvegarder un RLG dans le bit RB, utilisez les opérations SAVE, SPBB et SPBNB. Si vous appelez dans votre programme un bloc fonctionnel système (SFB) ou une fonction système (SFC), le SFB ou la SFC indique par l'état de signal du bit RB si la CPU a exécuté la fonction avec ou sans erreur : - En cas d'apparition d'une erreur au cours de l'exécution, le bit RB égale 0. - Si la fonction a été exécutée sans erreur, le bit RB égale 1.

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

Exemple sur un réseau en langage à contact (LAD) :



En langage LIST vous disposez des opérations de base suivantes :

- **U** ET
- **UN** ET NON
- **O** OU
- **ON** OU NON
- **X** OU exclusif
- **XN** OU NON exclusif

Les opérations suivantes permettent de combiner des parties de séquence combinatoire figurant entre parenthèses :

- **U(** ET d'une expression
- **UN(** ET NON d'une expression
- **O(** OU d'une expression
- **ON(** OU NON d'une expression
- **X(** OU exclusif d'une expression
- **XN(** OU NON exclusif d'une expression
- **)** Fermer la parenthèse d'une expression

Les opérations suivantes mettent fin à une séquence combinatoire :

- **=** Affectation (Cette opération sauvegarde le RLG dans le bit en accès)
- **R** Mettre à 0 (Cette opération écrit 0 dans le bit en accès si le RLG est égale à 1)
- **S** Mettre à 1 (Cette opération écrit 1 dans le bit en accès si le RLG est égale à 1)

Les opérations suivantes vous permettent de modifier le résultat logique RLG :

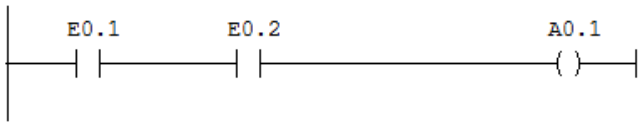
- **NOT** Négation du RLG
- **SET** Mettre RLG à 1
- **CLR** Mettre RLG à 0
- **SAVE** Sauvegarder RLG dans le bit RB

Les opérations suivantes détectent les transitions dans le résultat logique RLG et y réagissent :

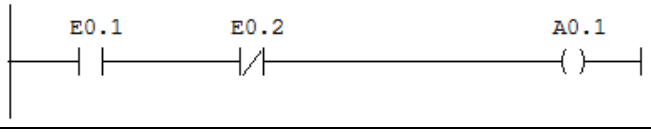
- **FN** Front descendant
- **FP** Front montant

9.5.1.2 Opérations Combinatoires sur bits

« ET » : **U** <bit>

List	Ladder
U E 0.1 U E 0.2 = A 0.1	

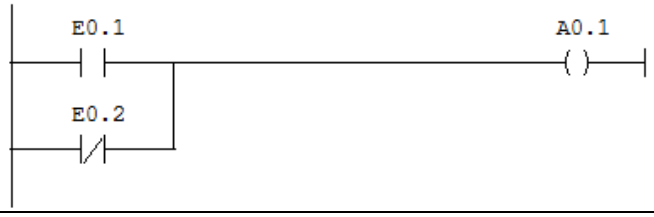
« ET NON » : **UN** <bit>

List	Ladder
U E 0.1 UN E 0.2 = A 0.1	

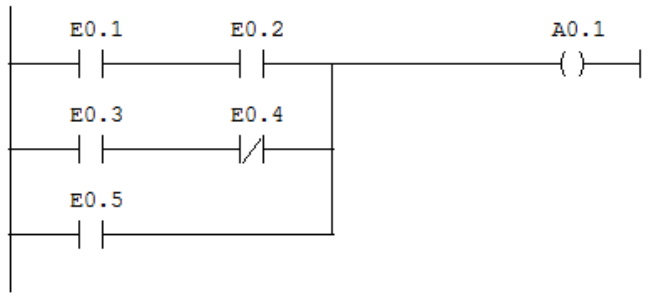
« OU » : **O** <bit>

List	Ladder
O E 0.1 O E 0.2 = A 0.1	

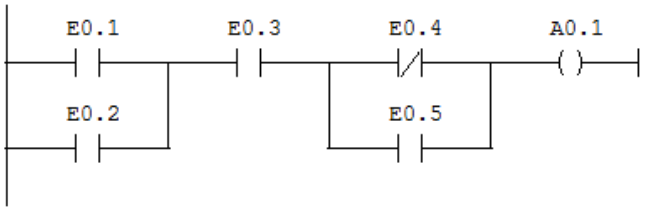
« OU NON » : **ON** <bit>

List	Ladder
O E 0.1 ON E 0.2 = A 0.1	

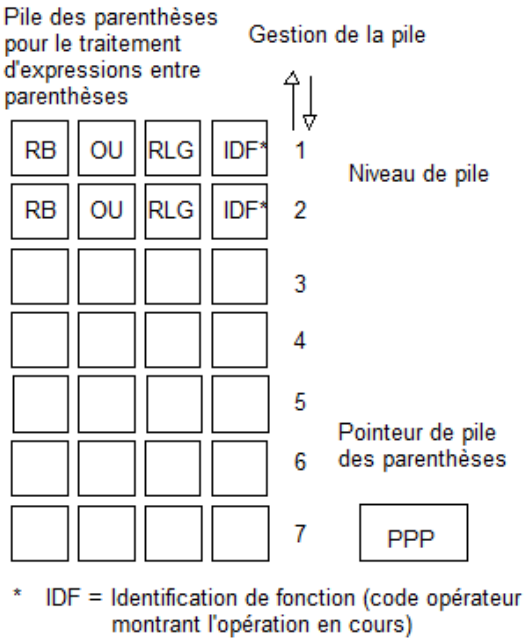
« ET avant OU » (la combinaison OU sur des combinaisons ET) : **O**

List	Ladder
U E 0.1 U E 0.2 O U E 0.3 UN E 0.4 O E 0.5 = A 0.1	

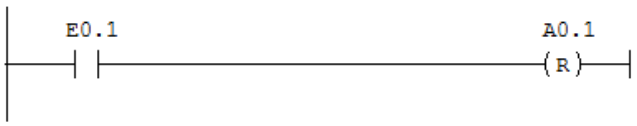
« ET d'une expression » : **U(**

List	Ladder
U(O E 0.1 O E 0.2) U E 0.3 U(ON E 0.4 O E 0.5) = A 0.1	

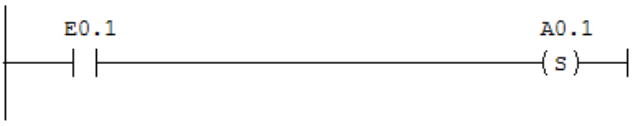
La pile des parenthèses peut contenir jusqu'à 7 entrées.
Il est possible aussi d'avoir **UN(**, **O(** et **ON(**.



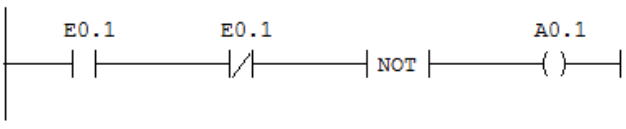
« Mettre à 0 » : **R** <bit>

List	Ladder
U E 0.1 R A 0.1	

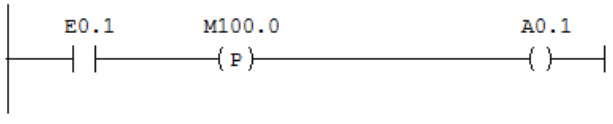
« Mettre à 1 » : **S** <bit>

List	Ladder
U E 0.1 S A 0.1	

« Négation du RLG » : **NOT**

List	Ladder
U E 0.1 UN E 0.1 NOT = A 0.1	

« Front montant » : **FP** <bit> / « Front descendant » : **FN**

List	Ladder
U E 0.1 FP M 100.0 = A 0.1	

La commande **FN** permet de traiter les fronts descendant, la programmation est basé sur le même principe

9.5.2 TRAITEMENT SUR MOTS

En LIST, on peut aussi traiter les octets, les mots, les entiers, les réels.
Les opérations sur mots ne peuvent se faire que par l'intermédiaire des accumulateurs, les automates disposent de 2 à 4 accumulateurs de 16 à 64 bits suivant le modèle d'UC.

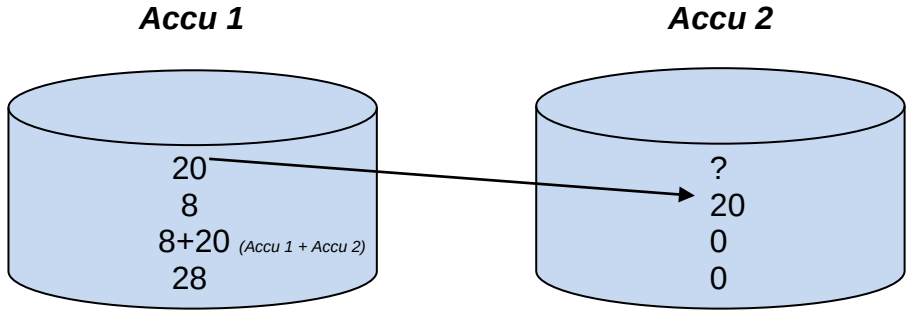
9.5.2.1 Principe des ACCU

Les deux accumulateurs de 32 bits sont des registres universels permettant de traiter octets, mots et doubles mots. Vous pouvez charger dans l'accumulateur 1 des constantes ou des valeurs de la mémoire, opérer des combinaisons ou même transférer le résultat d'une opération de l'accumulateur 1 dans une adresse d'opérande.

Le mécanisme de pile pour la gestion des accumulateurs fonctionne de la manière suivante :
- Une opération de chargement agit uniquement sur l'accumulateur 1 et sauvegarde l'ancien contenu dans l'accumulateur 2.
- Une opération de transfert (opération de copie) ne modifie pas les accumulateurs.
- L'opération TAK permute les contenus de l'accumulateur 1 et de l'accumulateur 2.
- Le résultat d'opérations de combinaison entre l'accumulateur 1 et l'accumulateur 2 (opérations arithmétiques, opérations de comparaison, ET, OU, ...) est toujours rangé dans l'accumulateur 1.

Exemple : on souhaite ajouter la valeur « 8 » au mot MW10 (avec MW10 = 20) et transférer le résultat dans MW12.

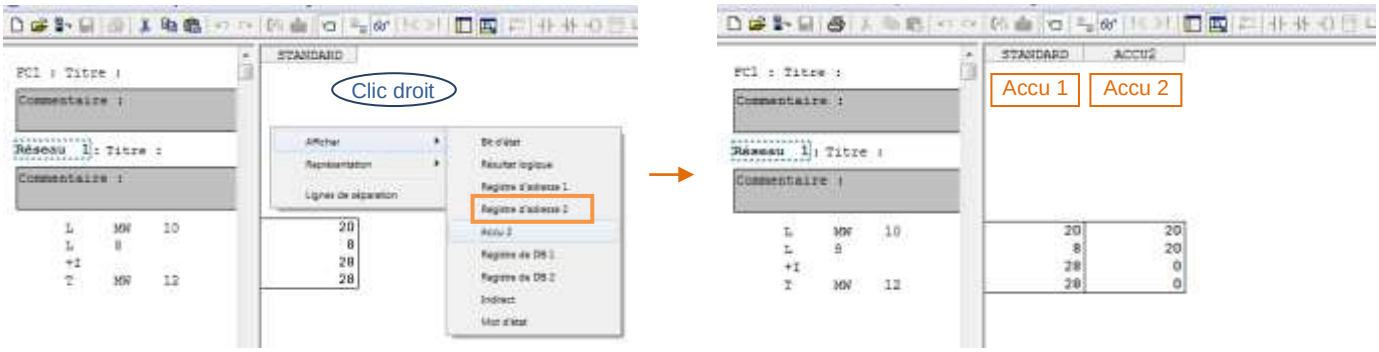
Code :
L MW10
L 8
+I
T MW12



Explication :
- On charge le mot MW10 dans l'Accu n°1 par la commande « Chargement » L (**L**oad).
- On charge la valeur « 8 » dans l'Accu n°1.
- On additionne l'Accu n°1 avec l'Accu n°2 avec la commande « +I » (addition d'**I**nteger).
- On transfère le résultat de l'addition dans le mot MW12 par la commande « Transfert » T (**T**ransfert).

Remarque :
- *On considère la valeur de l'Accu n°2 inconnu à la première instruction (« ? ») car le programme qui précède la première instruction nous est inconnu.*
- *Après une addition, l'Accu n°2 prends la valeur 0 (car sa valeur est passé dans l'addition qui s'est effectué dans l'Accu n°1).*
- *On charge les deux opérandes puis seulement on dit l'opération à effectuer.*

Visualisation des Accu en ligne :



9.5.2.2 Opérations de comparaison

Les opérations de comparaison comparent le contenu de l'accumulateur 2 à celui de l'accumulateur 1 selon les types de comparaison suivants :

==	ACCU 2 **est égal à** ACCU 1
<>	ACCU 2 **est différent de** ACCU 1
>	ACCU 2 **est supérieur à** ACCU 1
<	ACCU 2 **est inférieur à** ACCU 1
>=	ACCU 2 **est supérieur ou égal à** ACCU 1
<=	ACCU 2 **est inférieur ou égal à** ACCU 1

Si le RLG égale 1, le résultat de comparaison est vrai.
Si le RLG égale 0, le résultat de comparaison est faux.

Les bits BI1 et BI0 indiquent la relation "inférieur à", "égal à" ou "supérieur à".

Type disponible pour la comparaison :

< comparateur > **I** : Comparer **entiers de 16 bits**
< comparateur > **D** : Comparer **entiers de 32 bits**
< comparateur > **R** : Comparer **réels de 32 bits**

Exemple : Si MW10 est supérieur à 25 alors le RLG est à 1 donc le bit M20.0 passe à 1.

```
L    MW10
L    25
>I
=    M20.0
```

9.5.2.3 Fonctions sur nombres entiers et réels

Les opérations arithmétiques combinent le contenu des accumulateurs 1 et 2. Le résultat est rangé dans l'accumulateur 1. Le contenu de l'accumulateur 2 reste inchangé. L'opération s'exécute sans tenir compte du RLG ni influer sur lui.

Pour les CPU à quatre accumulateurs, le contenu de l'accumulateur 3 est ensuite copié dans l'accumulateur 2 et le contenu de l'accumulateur 4 est copié dans l'accumulateur 3. Le contenu de l'accumulateur 4 reste inchangé.

Liste des opérations arithmétiques :

+	**Additionner** Accu 1 avec Accu 2	INT, DINT,REAL
-	**Soustraire** Accu 1 de Accu 2	INT, DINT,REAL
*****	**Multiplier** Accu 1 par Accu 2	INT, DINT,REAL
/	**Diviser** Accu 2 par Accu 1	INT, DINT,REAL
MOD	**Reste de division entière** Accu 2 par Accu 1	DINT,REAL

Type disponible pour les opérations arithmétiques :

< opérateur > **I** : Opération arithmétique sur **entiers de 16 bits**
< opérateur > **D** : Opération arithmétique sur **entiers de 32 bits**
< opérateur > **R** : Opération arithmétique sur **réels de 32 bits**

Exemple : Soustraire la valeur 15.0 à MD100 et transférer le résultat dans DB10.DBD320

```
L    15.0
L    MD100
-R
T    DB10.DBD320
```

Liste des fonctions arithmétiques :

ABS	Valeur absolue	REAL
SQR	Carré d'un nombre	REAL
SQRT	Racine carrée	REAL
EXP	Valeur exponentielle	REAL
LN	Logarithme naturel	REAL
SIN	Sinus d'un angle	REAL
COS	Cosinus d'un angle	REAL
TAN	Tangente d'un angle	REAL
ASIN	Arc sinus d'un angle	REAL
ACOS	Arc cosinus d'un angle	REAL
ATAN	Arc tangente d'un angle	REAL

Exemple : Calculer le cosinus de MD10 et transférer le résultat dans MD20

```
L    MD10
COS
T    MD20
```

9.5.2.4 Opérations combinatoires sur mots

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

Lors de la combinaison de mots, le contenu du mot poids faible de l'accumulateur 2 est combiné au contenu du mot de poids faible de l'accumulateur 1. Le résultat de la combinaison se substitue à l'ancien contenu du mot de poids faible de l'accumulateur 1.

Lors de la combinaison de doubles mots, le contenu de l'accumulateur 2 est combiné au contenu de l'accumulateur 1. Le résultat de la combinaison se substitue à l'ancien contenu de l'accumulateur 1.

Le bit du mot d'état BI1 est mis à 1 (BI1 égale 1 si le résultat est différent de zéro) comme résultat de l'opération.

Vous disposez des opérations combinatoires sur mots les suivants :

- **UW** ET mot (16 bits)
- **OW** OU mot (16 bits)
- **XOW** OU exclusif mot (16 bits)
- **UD** ET double mot (32 bits)
- **OD** OU double mot (32 bits)
- **XOD** OU exclusif double mot (32 bits)

Il y a deux façons d'utiliser ces opérations :

- < opérateur > (voir Exemple n°1)
- < opérateur > < constante > (voir Exemple n°2)

Exemple n°1 : Combiner, bit par bit, le contenu de EW20 au contenu de EW22 selon la table de vérité OU puis transférer le résultat dans le mot de mémento MW8.

```
L    EW20
L    EW22
OW
T    MW8
```

Exemple n°2 : Combiner les bits de ED 20 au profil binaire de la constante de 32 bits (0000_1111_1111_1111_1110_1111_0010_0001) selon la table de vérité ET puis transférer le résultat dans le mot de mémento MW8

```
L    ED 20
UD   DW#16#0FFF_EF21
T    MW8
```

9.5.2.5 Opérations de décalage et de rotation

Les opérations de décalage sont inconditionnelles : leur exécution ne dépend d'aucune condition spéciale. Elles n'affectent pas le résultat logique RLG.

Vous disposez des opérations de décalage suivantes :

- SSI Décalage vers la droite d'un entier avec signe (16 bits)
- SSD Décalage vers la droite d'un entier avec signe (32 bits)
- SLW Décalage vers la gauche d'un mot (16 bits)
- SRW Décalage vers la droite d'un mot (16 bits)
- SLD Décalage vers la gauche d'un double mot (32 bits)
- SRD Décalage vers la droite d'un double mot (32 bits)

Il y a deux façons d'utiliser ces opérations :

- < opérateur > (voir Exemple n°1)
- < opérateur > < nombre de bits à décaler > (voir Exemple n°2)

Exemple 1 : Décaler les bits dans MW20, signe inclus, de 3 positions vers la droite, mettre les positions libérées à l'état de signal du bit de signe et transférer le résultat dans le mot de mémento MW8.

```
L    +3
L    MW20
SSI
T    MW8
```

Exemple 2 : Décaler les bits dans MW4, signe inclus, de 6 positions vers la droite et transférer le résultat dans le mot de mémento MW8.

```
L    MW4
SSI  6
T    MW8
```

9.5.2.6 Opérations de conversion

A venir …

9.5.3 OPERATIONS DE SAUT

Les opérations de saut (« **SP**rung » en Allemand) et de boucle (Loop) permettent de gérer le déroulement de votre programme.
Elles interrompent le déroulement linéaire de votre programme afin de reprendre son exécution à un endroit différent. L'opérande d'une opération de saut ou de boucle est un repère de saut (comme **LABEL** en langage CONT et LOG)

Un saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'instruction de saut et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique.

*Remarque : Dans les programmes pour les CPU S7–300, veiller à ce que la destination du saut soit toujours le **début** d'une séquence d'instructions combinatoires (pas obligatoire pour 318-2). La destination du saut ne doit pas se trouver à l'intérieur de la séquence d'instructions combinatoires.*

9.5.3.1 Les différents types de sauts :

Les opérations de saut suivantes permettent d'interrompre la séquence normale de votre programme de manière inconditionnelle :

- SPA Saut inconditionnel
- SPL Saut vers liste (équivalent à un **CASE** en langage SCL)

Les opérations de saut suivantes interrompent la séquence normale dans votre programme selon le résultat logique RLG généré par l'instruction précédente :

- SPB Saut si RLG est 1 ← *Equivalent du **JMP** en langage CONT et LOG*
- SPBN Saut si RLG est 0 ← *Equivalent du **JMPN** en langage CONT et LOG*
- SPBB Saut si RLG est 1 avec RB
- SPBNB Saut si RLG est 0 avec RB

Les opérations de saut suivantes interrompent la séquence normale dans votre programme selon l'état de signal d'un bit du mot d'état :

- SPBI Saut si RB est 1
- SPBIN Saut si RB est 0
- SPO Saut si DEB est 1
- SPS Saut si DM est 1

Les opérations de sauts suivantes interrompent la séquence normale dans votre programme selon le résultat d'un calcul (voir bit BI0 et BI1 du mot d'état):

- SPZ Saut si égale à 0 (Saut si BI0 = 0 et BI1 = 0)
- SPN Saut si différent de 0 (Saut si BI1 = 0/BI0 = 1 ou BI1= 1/BI0 = 0)
- SPP Saut si plus (Saut si BI0 = 0 et BI1 = 1)
- SPM Saut si moins (Saut si BI0 = 1 et BI1 = 0)
- SPPZ Saut si supérieur ou égal à 0 (Saut si BI1 = 0/BI0= 0 ou BI1 = 1/BI0 = 0)
- SPMZ Saut si inférieur ou égal à 0 (Saut si BI1 = 0/BI0= 0 ou BI1 = 0/BI0 = 1)
- SPU Saut si illicite (Saut si BI0 = 1 et BI1 = 1)

On utilise les sauts de la manière suivante : <type de saut > < repère de saut >

Exemple : Si E1.0 et E1.2 sont à 1 on saute vers le repère SAUT sinon on ne saute pas et on charge EW8 dans MW22.

```
            U       E 1.0
            U       E 1.2
            SPB     SAUT
            L       EW8
            T       MW22
SAUT:       U E 2.1
```

9.5.3.2 Boucle de programme LOOP

LOOP : Décrémenter l'accumulateur 1-L et sauter si accumulateur 1-L différent de 0

Cette opération simplifie la programmation de boucles. Le compteur de boucles est un nombre entier non signé de 16 bits qui se trouve dans l'accumulateur 1-L. L'instruction saute au repère de saut indiqué tant que le contenu de l'accumulateur 1-L est différent de 0. Le traitement du programme se poursuit à la destination de saut précisée par un repère. Le saut peut s'exécuter aussi bien vers l'avant que vers l'arrière, mais absolument à l'intérieur d'un bloc (l'opération « Boucle de programme » et le repère de saut doivent se trouver à l'intérieur du même bloc). La destination de saut à l'intérieur de ce bloc doit être unique. La portée de saut maximale est de -32768 ou +32767 mots du code de programme. Le nombre maximal réel d'instructions pouvant être sautées dépend de la combinaison des instructions à l'intérieur du programme (instructions à un, deux ou trois mots).

Exemple :

```
            L       5       //Charger le nombre de boucles dans l'accumulateur 1-L.
SUIV:       T       MW10    //Repère de saut = début de la boucle
                            //Transférer l'accumulateur 1-L dans le compteur de boucles.
            < instructions qui seront exécutés 5 fois >
            L       MW10    //Charger le contenu du compteur de boucles dans l'accumulateur 1.
            LOOP    SUIV    //Décrémenter le contenu de l'accumulateur 1 et sauter au repère SUIV si
                            //l'accumulateur 1-L est supérieur à 0.
            < La séquence de programme se poursuit ici après la fin de la boucle >
```

9.5.4 OPERATIONS SUR BLOCS DE DONNEES

Vous disposez des opérations sur bloc de données suivantes :

- AUF Ouvrir bloc de données
- TDB Permuter DB global et DB d'instance
- L DBLG Charger longueur de DB global dans l'accumulateur 1
- L DBNO Charger numéro de DB global dans l'accumulateur 1
- L DILG Charger longueur de DB d'instance dans l'accumulateur 1
- L DINO Charger numéro de DB d'instance dans l'accumulateur 1

9.5.4.1 AUF Ouvrir bloc de données

AUF <*bloc de données*>

L'opération **AUF** (ouvrir bloc de données) permet d'ouvrir un bloc de données global ou un bloc de données d'instance. Un bloc de données global et un bloc de données d'instance peuvent être ouverts simultanément dans le programme.

AUF DBxxx : ouverture d'un bloc de données global
AUF DIxxx : ouverture d'un bloc de données d'instance

Exemple :

```
AUF     DB10    //Ouvrir DB10 comme bloc de données global.
L       DBW35   //Charger dans l'accumulateur 1-L le mot DBW35 de la DB globale ouverte.
T       MW22    //Transférer le contenu de l'accumulateur 1-L dans MW22
AUF     DI20    //Ouvrir DB20 comme bloc de données d'instance.
L       DIB12   //Charger dans l'accumulateur 1-L-L. l'octet DIB12 de la DB d'instance ouverte
T       DBB37   //Transférer le contenu de l'accumulateur 1-L-L dans l'octet DBB37 de la DB
                //global ouvert.
```

9.5.5 LES TEMPOS

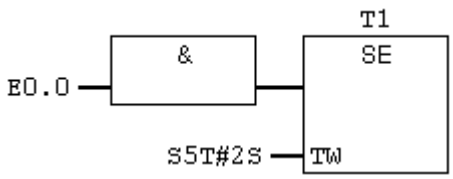
A venir …

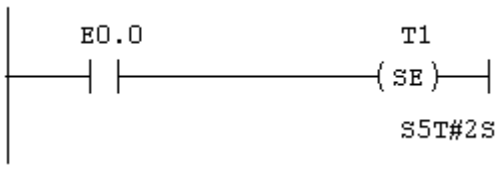
9.5.6 LES COMPTEURS

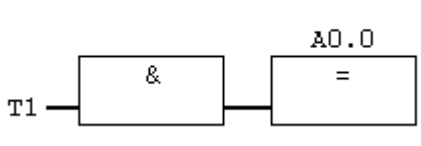
A venir …

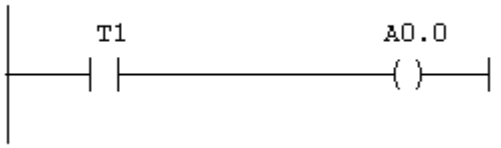
Les tempos et compteurs
il y a dans l'automate 256 mémoires (16 bits) réservées aux tempos, nommées T0 à T255. La durée y est notée d'une manière un peu spéciale : les deux bits de poids fort sont à 0. Les deux suivants définissent la base de temps : 00=10ms, 01=100ms, 10=1s, 11=10s. Les 12 bits restants correspondent à la durée, en BCD (donc maxi 999). Si on veut simplifier, on peut utiliser le format S5T par ex S5T#1H_2M_20S_100MS
Il y a diverses tempos possibles : SE (normale : déclenche au bout de la tempo, s'éteint dès que l'entrée repasse à 0), SS (comme SE, mais ne s'éteint que par un reset), SV (dure exactement le temps donné, quelle que soit la durée de l'entrée), SI (comme SV, mais s'arrête dès que l'entrée passe à 0), SA (s'allume dès que l'entrée est à 1, s'éteind avec un temps de retard par rapport au passage à 0 de l'entrée).
A la rigueur on peut forcer l'état d'une tempo (si son entrée est à 1, quand elle est à 0 de toute façon rien ne se passe) : FR redémarre le comptage du temps même si on avait déjà commencé à compter, R termine une tempo (même si le temps n'est pas fini) et remet immédiatement la sortie à 0
Exemple : A0.0 passe à 1 2s après l'activation de E0.0, s'éteint en même temps que E0.0 :

```
U  E0.0
L  S5T#2S
SE T1

U  T1
=  A0.0
```









9.5.7 OPERATIONS DE GESTION D'EXECUTION DE PROGRAMME

A venir …

9.5.8 ADRESSAGE INDIRECT EN MEMOIRE

9.5.8.1 Avec un numéro (Mot de 16 bits)

Ceci concerne l'adressage des tempos, compteurs et blocs.
Rappel : Valeur maximum pour un mot de 16 bits : 65535.
Exemple :

```
L     10        //Charge la valeur 10 dans l'Accu n°1
T     MW33      //Transfert la valeur de l'Accu n°1 dans  MW33
AUF   DB[MW33]  //Ouvre le DB10
U     E 0.0     //Teste l'entrée E0.0
SE    T[MW33]   //Démarre la tempo T10
```

L'**adresse de l'opérande**, dans lequel le numéro est mémorisé (ici le MW33), peut se trouver dans les zones de mémentos, données globales (**DBX**), données d'instances (**DIX**) et données locales. L'utilisation des **zones de données d'instances** (**DIX**) pour les adresses d'opérande est seulement possible dans le CODE_VERSION1 (pas dans les FB multi-instances) des blocs fonctionnel (FBs).

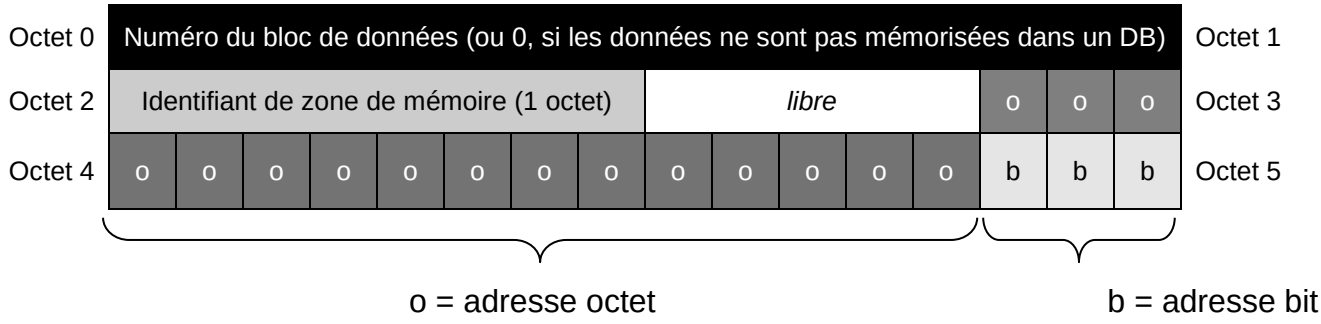
9.5.8.2 Avec pointeur de zone (32 bits)

Ceci concerne l'adressage des périphéries, entrées, sorties, mémentos, données globales (DBX), données d'instances (DIX) et données locales.

Rappel sur le type POINTER :

- Pointeur : Un pointeur est utilisé pour adresser un opérande. L'avantage de ce type d'adressage est que vous pouvez modifier de manière dynamique l'opérande de l'instruction durant l'exécution du programme.
  Notation d'un pointeur : **P#**< adressage >
- Zone de mémoire : E (Zone d'Entrée), A (Zone d'Sortie), M (Zone de Mémento), DB (Zone de Données gloables), …
- Adressage intrazone : Le pointeur ne contient aucune indication de zone de mémoire.
  Exemple : P#**8.7**
- Adressage interzone : la zone de mémoire est précisée dans le pointeur.
  Exemple : P#**E8.7**

- Voici la structure de type POINTER (Il occupe 6 octets) :



L'Identifiant de zone est à 0 pour l'adressage intrazone.
L'identifiant de zone prend l'une des valeurs dans le tableau ci-dessous pour l'adressage interzone.

Identifiant de zone	Code		Zone
	binaire	hexa	
P	**1**000 0**000**	80	Zone de périphérie
E	**1**000 0**001**	81	Zone d'entrée
A	**1**000 0**010**	82	Zone de sortie
M	**1**000 0**011**	83	Zone de mémentos
DB	**1**000 0**100**	84	Zone de données
DI	**1**000 0**101**	85	Zone de données d'instance
L	**1**000 0**110**	86	Zone de données locales
VL	**1**000 0**111**	87	Zone de données locales précédente (accès aux données locales du bloc appelant)

Utilisation d'adressage indirect en mémoire avec des pointeurs

L'identification de zone **doit être à 0** dans le pointeur (pointeur intrazone) pour ce type d'adressage.

Exemple n°1 :

```
L     P#8.7      //Chargement de la valeur du pointer dans l'Accu n°1
T     MD2        //Transfert du pointeur dans MD2
U     E[MD2]     //Test de l'état de signal de l'entrée E8.7
=     A[MD2]     //Affectation de l'état de signal à la sortie A8.7
```

Le code suivant fait la même chose que le code précédent :

```
U     E8.7
=     A8.7
```

Exemple n°2 On souhaite transférer MW200 dans DBW40 de la DB50.

```
L     P#40.0     //Chargement de la valeur de pointeur P#40.0 dans l'Accu n°1
T     MD100      //Transfert du pointeur dans MD100
AUF   DB50       //Ouverture du DB50
L     MW200      //Chargement de MW200 dans l'Accu n°1
T     DBW[MD100] //Transfert de l'Accu n°1 dans le mot DBW40
```

Remarque : Lors de l'accès indirect sur un octet, mot ou double mot, l'adresse du bit doit être 0 dans le pointeur.

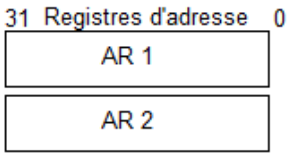
Le code suivant fait la même chose que le code précédent :

```
L     MW200
T     DB50.DBW40
```

9.5.9 ADRESSAGE INDIRECT AVEC REGISTRE ET POINTEUR DE ZONE

9.5.9.1 Les registres d'adresse AR1 et AR2

Les registres d'adresses renferment les pointeurs intrazones ou interzones pour les opérations utilisant l'adressage indirect par registre. Les registres d'adresses ont une longueur de 32 bits.



Vous disposez des opérations suivantes :

- **LAR1** Charger contenu de l'accumulateur 1 dans registre d'adresse 1
- **LAR1** <*D*> Charger pointeur de 32 bits dans registre d'adresse 1
- **LAR1 AR2** Charger contenu du registre d'adresse 2 dans registre d'adresse 1
- **LAR2** Charger contenu de l'accumulateur 1 dans registre d'adresse 2
- **LAR2** <*D*> Charger pointeur de 32 bits dans registre d'adresse 2

- **TAR** Permuter registre d'adresse 1 avec registre d'adresse 2
- **TAR1** Transférer registre d'adresse 1 dans l'accumulateur 1
- **TAR1** <*D*> Transférer registre d'adresse 1 à l'adresse de destination (32 bits)
- **TAR1 AR2** Transférer registre d'adresse 1 dans registre d'adresse 2
- **TAR2** Transférer registre d'adresse 2 dans l'accumulateur 1
- **TAR**2 <*D*> Transférer registre d'adresse 2 à l'adresse de destination (32 bits)

- **+AR1** Additionner accumulateur 1 au registre d'adresse 1
- **+AR2** Additionner accumulateur 1 au registre d'adresse 2

Exemple avec **LAR1**

```
L       P#M100.0        //charger la valeur du pointeur dans l'Accu n°1
LAR1                    //charger le pointeur de l'Accu n°1 dans AR1
```

Exemple avec **LAR1** <*D*>

```
L       P#M100.0
T       MD 20
LAR1    MD 20           //Charger dans AR1 le pointeur figurant dans MD20.
```

Autre exemple avec **LAR1** <*D*>

```
LAR1    P#M100.0        //Charger une constante de pointeur de 32 bits dans AR1.
```

Remarque : les 3 exemples précédents permettent de charger le pointeur P#M100.0 dans AR1.

Exemple n°1 avec **+AR1**

```
L       +300            //Charger la valeur dans l'accumulateur 1-L.
+AR1
```

Exemple n°2 avec **+AR1**

```
+AR1    P#300           //Additionner le décalage 300.0 au registre d'adresse 1.
```

Remarque : Les deux exemples précédents permettent d'ajouter 300 à AR1.

9.5.9.2 Exemples d'utilisation des registres pour l'adressage indirect

Les instructions de programme utilisant ce type d'adressage sont composées d'une opération et des éléments suivants : identificateur d'opérande, identificateur de registre d'adresse, décalage. Le registre d'adresse (AR1/2) et le décalage doivent être indiqués ensemble entre crochets.

Exemple d'adressage intrazone

```
L       P#8.7              //charger la valeur du pointeur dans l'ACCU 1
LAR1                       //charger le pointeur de l'ACCU 1 dans AR1
U       E[AR1, P#0.0]      //interroger l'état de signal de l'entrée E 8.7
=       A[AR1, P#1.1]      //affecter l'état de signal à la sortie A 10.0
```

Le décalage 0.0 n'a pas d'effet. La sortie 10.0 se calcule à partir de 8.7 (AR1) plus le décalage 1.1. Le résultat est 10.0 et non pas 9.8, voir le format du pointeur.

Exemple d'adressage interzone

```
L       P# E8.7            //charger la valeur du pointeur et l'identification de zone dans l'ACCU 1
LAR1                       //charger la zone de mémoire E et l'adresse 8.7 dans AR1
L       P# A8.7            //charger la valeur du pointeur et l'identification de zone dans l'ACCU 1
LAR2                       //charger la zone de mémoire A et l'adresse 8.7 dans AR2
U       [AR1, P#0.0]       //interroger l'état de signal de l'entrée E 8.7
=       [AR2, P#1.1]       //affecter l'état de signal à la sortie A 10.0.
```

Le décalage 0.0 n'a pas d'effet. La sortie 10.0 se calcule à partir de 8.7 (AR2) plus 1.1 (décalage). Le résultat est 10.0 et non pas 9.8, voir format du pointeur.

9.5.9.3 Utilisation des registres avec le type ANY

Type ANY

Voici la structure de type ANY (Il occupe 10 octets) :

Octet 0	10h (pour S7)								Type de données							Octet 1
Octet 2	Facteur de répétition															Octet 3
Octet 4	Numéro du bloc de données (ou 0, si les données ne sont pas mémorisées dans une DB)															Octet 5
Octet 6	Identifiant de zone de mémoire (1 octet)								*libre*					o	o	o	Octet 7
Octet 8	o	o	o	o	o	o	o	o	o	o	o	o	o	b	b	b	Octet 9

Octet 0 : toujours la valeur 10 (en héxa)
Octet 1 : Type de données à transmettre
Octets 2 et 3 : Quantité du type de données identifié qui est à transmettre (voir tableau)
Octets 4 et 5 : Numéro du bloc de données (ou 0, si les données ne sont pas mémorisées dans une DB)
Octet 6 : identifiant zone de mémoire (voir tableau)
Octets 7 à 9 : adresse au format octet.bit.

Codage des types de données

Code hexadécimal	Type de données	Description
b#16#00	NIL	Pointeur zéro
b#16#01	BOOL	Bits
b#16#02	BYTE	Octets (8 bits)
b#16#03	CHAR	Caractères (8 bits)
b#16#04	WORD	Mots (16 bits)
b#16#05	INT	Entiers (16 bits)
b#16#06	DWORD	Mots (32 bits)
b#16#07	DINT	Entiers (32 bits)
b#16#08	REAL	Nombres à virgule flottante (32 bits)
b#16#09	DATE	Date
b#16#0A	TIME_OF_DAY (TOD)	Heure
b#16#0B	TIME	Temporisation
b#16#0C	S5TIME	Type de données S5TIME
b#16#0E	DATE_AND_TIME (DT)	Date et heure (64 bits)
b#16#13	STRING	Chaîne de caractères

Codage des zones de mémoire

Code hexadécimal	Zone	Description
b#16#80	P	Zone de mémoire de périphérie
b#16#81	E	Zone de mémoire des entrées
b#16#82	A	Zone de mémoire des sorties
b#16#83	M	Zone de mémoire des mémentos
b#16#84	DB	Bloc de données
b#16#85	DI	Bloc de données d'instance
b#16#86	L	Données locales (pile L)
b#16#87	V	Données locales précédentes

Exemple d'utilisation des registres avec le type ANY

Réalisation d'une fonction qui permet de copier une quantité d'octet désiré d'une DB à une autre en précisant les adresses de départs (offset) dans chacune des deux DB

Variables de la fonction :

Nom	Type	Adresse
Input		
num_DB_src	Int	
num_DB_Dest	Int	
offset_src	Int	
offset_dest	Int	
nb_Octet	Int	
Temp		
Source	Any	0.0
Destination	Any	10.0

Réseau 1:

```
LAR1    P##Source
L       B#16#10          //identificateur 10h
T       LB [AR1, P#0.0]
L       B#16#02          //type octet
T       LB [AR1, P#1.0]
L       #nb_Octet        //nb octet à copier
T       LW [AR1, P#2.0]
L       #num_DB_src      //numéro du DB source
T       LW [AR1, P#4.0]
L       P#DBX0.0
L       #offset_src      //point de départ de la copie
SLD     3                //décalage à gauche de 3 bits (voir 4 derniers octets du type ANY)
+D                       //ajout de l'offset à DBX0.0
T       LD [AR1, P#6.0]
LAR1    P##Destination
L       B#16#10          //identificateur 10h
T       LB [AR1, P#0.0]
L       B#16#02          //type octet
T       LB [AR1, P#1.0]
L       #nb_Octet        //nb octet à copier
T       LW [AR1, P#2.0]
L       #num_DB_Dest     //numéro du DB de Destination
T       LW [AR1, P#4.0]
L       P#DBX0.0         //charge le pointeur dans ACCU1
L       #offset_dest     //point de départ de la copie
SLD     3                //décalage à gauche de 3 bits (voir 4 derniers octets du type ANY)
+D                       //ajout de l'offset à DBX0.0
T       LD [AR1, P#6.0]
```

Réseau 2: Appel du SFC20

```
CALL    BLKMOV
        SRCBLK := #Source
        RET_VAL:= #Diag
        DSTBLK := #Destination
NOP 0
```

9.5.10 AUTRES EXEMPLES EN LIST

9.5.10.1 Manipulation de mots (Rotation et masque sur mots)

			Accu n°1	*Accu n°2*
L	5923	//Chargement de la valeur 5923 (1723 en hexa) dans l'Accu n°1	*1723*	*0*
T	MW50	//Transfert de l'Accu n°1 dans MW50	*1723*	*0*
SRW	8	//Décalage de 8 bits vers la droite du contenu de l'Accu n°1	*17*	*0*
T	MW52	//Transfert de l'Accu n°1 dans MW52	*17*	*0*
L	W#16#F	//Chargement de la valeur W#16#F dans l'Accu n°2	*F*	*17*
UW		// « Et » logique bit à bit entre Accu n°1 et Accu n°2	*7*	*17*
T	W54	//Transfert de l'Accu n°1 dans MW54	*7*	*17*

9.5.10.2 Manipulation des pointeurs

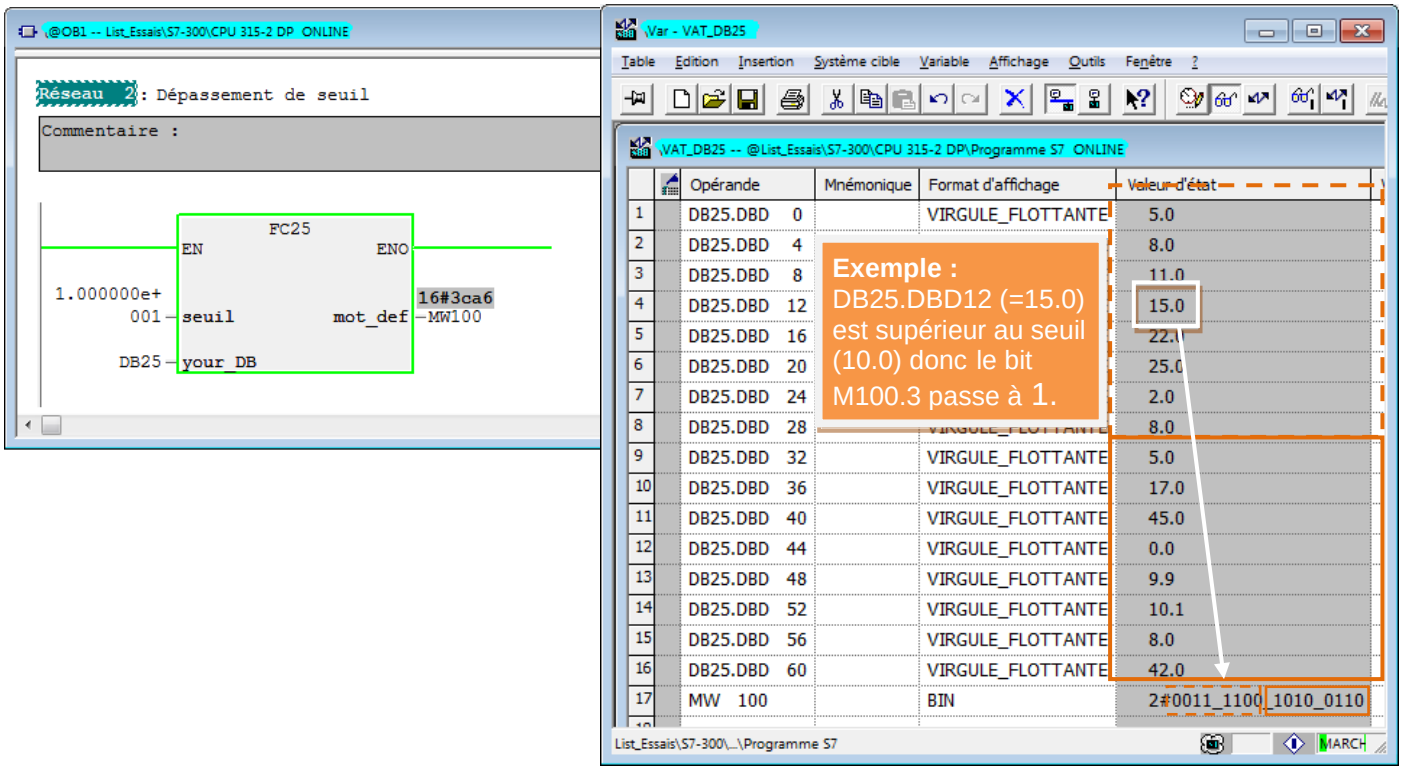
On souhaite détecter si une valeur dans une DB dépasse un seuil, chaque bit d'un mot (16 bits) indiquera si la valeur a dépassé le seuil (si seuil dépassé alors défaut, le bit du mot passe à 1). Le code sera écrit dans une FC dont voici les variables :

Nom	Type	Adresse
Variables IN (Entrées du FC)		
seuil	Real	
Variables OUT (Sorties du FC)		
mot_def	Word	
Variables TEMP		
pointeur	DWord	0.0
index	Int	4.0
def_temp	Word	6.0

```
//Init Mot de défaut (Temp)
        L       W#16#0
        T       # def_temp

//Init du pointeur pour scrutation DB
        L       P#0.0
        T       #pointeur

//Init du pointeur pour mot de défaut (Temp)
        LAR1    P#0.0
```

```
//Boucle sur les 16 valeurs de la DB
            L       16          //Init de la boucle : 16 valeurs à lire (aussi : mot de défaut → WORD)
for:        T       #index      //Transfert de l'Accu 1 pour mise à jour de l'index boucle
            AUF     #your_DB    //Ouverture de la DB

//Comparaison si valeur en cours de lecture dans la DB > à valeur "seuil"
            L       DBD [#pointeur]
            L       #seuil
            >R

//Si valeur en cours de lecture dans la DB > à valeur "seuil" alors bit en cours du mot de défaut (Temp) est mise à 1
            SPBNB   next
            S       L [AR1,P#6.0]

//Incrément des pointeurs (incrément de 1 bit pour le traitement du défaut suivant et incrément de 4 octets car type de données REAL)
next:       +AR1    P#0.1
            L       P#4.0
            L       #pointeur
            +D
            T       #pointeur

            L       #index      //Chargement de  l'index boucle
            LOOP    for         //Boucle : décrément de la valeur de l'Accu n°1 (décrement de l'index boucle) //Boucle : décrement de l'index boucle
//Affectation Sortie du FC → mot de défaut (Temp) dans mot de défaut (Out)
            L       #def_temp
            T       #mot_def
```

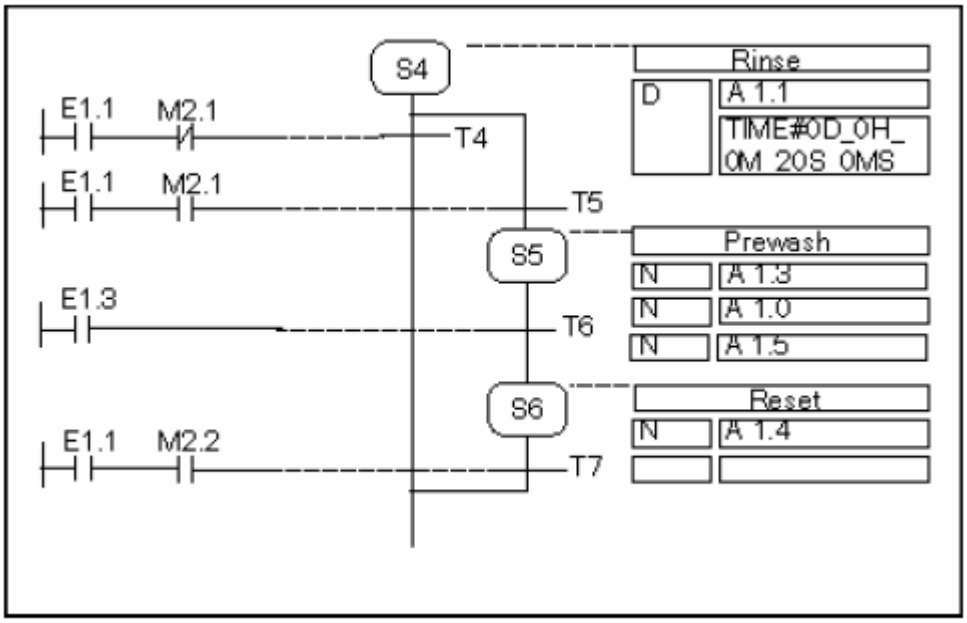
Utilisation du bloc :
Exemple pour un seuil à « 10.0 », les valeurs sont dans la DB25. MW100 est le mot de défaut.



LANGAGE S7-GRAPH

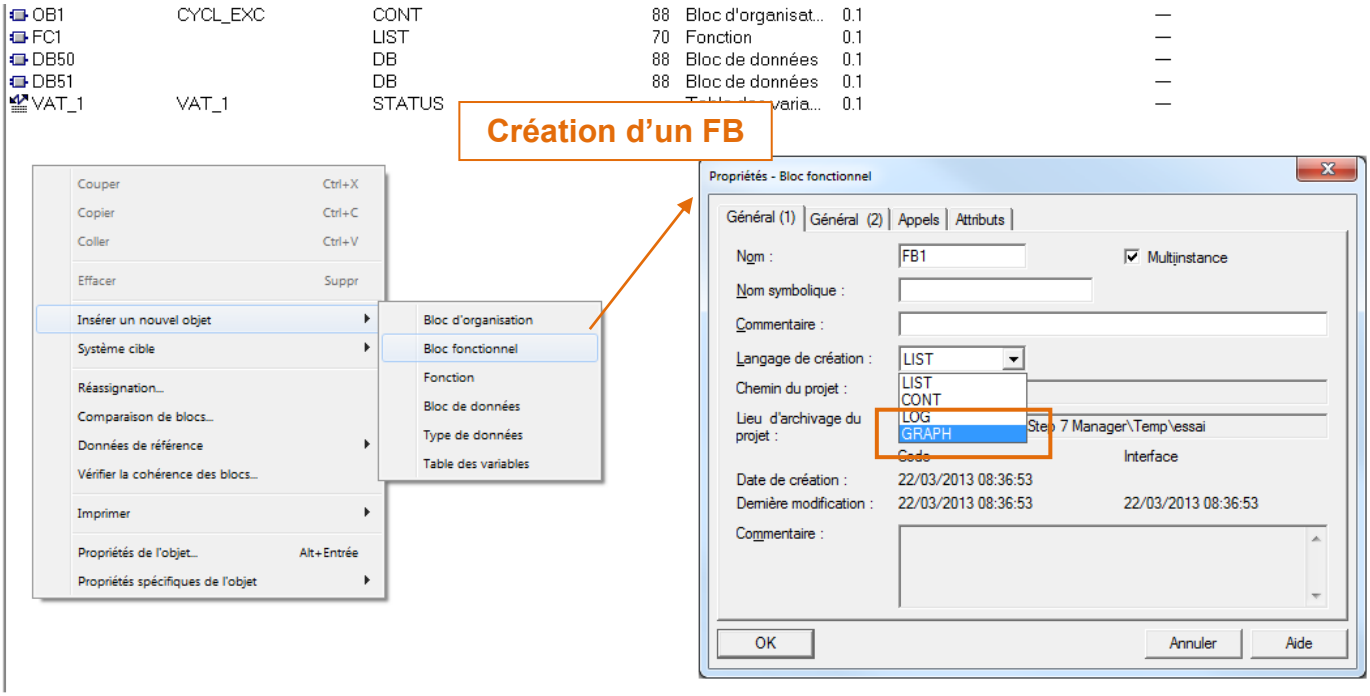
Le S7-Graph (langage proche du Grafcet mais moins puissant) n'est pas un langage de base dans S7 (une licence est nécessaire). Il permet de de créer des séquences de grafcets incluant des étapes et des transitions.
Les actions d'étapes sont programmées…
Les transitions sont programmées soit en langage CONT ou LOG.
L'avantage du langage S7-Graph est qu'il permet d'avoir une …

Exemple de séquence en langage S7-Graph



Blocks Created

With the S7-GRAPH editor you program the function block that contains the sequencer. A corresponding instance DB contains the data for the sequencer, e.g. the FB parameters, step and transition conditions. You can generate this instance DB automatically in the S7-GRAPH editor.



Source File

A text-based source file (GRAPH source file) can be generated from a function block created in S7-GRAPH which can be interpreted by OPs or text-based displays for displaying the sequencer.

Il suffit de créer un bloc fonctionnel (FB) en Graph : on se met au niveau des blocs et on insère (clic droit) un FB (bloc fonctionnel). Il nous ouvre une fenêtre de propriétés, où l'on choisit le nom (FB1 est très bien) mais surtout le langage (choisir GRAPH). Le système crée automatiquement un DB (les données associées, DB1), un FC72 et un SFC64 (fonctions système nécessaires). Il n'y a plus qu'à rentrer le programme, le sauver. Vous aviez également le droit de préciser des mnémoniques. Le système me semble nécessiter des convergences en ET symétriques aux divergences (mais accepte les étapes initiales multiples). Par contre pour les OU il m'a l'air un peu plus souple. Il ne respecte pas la norme, en particulier les OU sont exclusifs (si deux voies sont possibles, seule la plus à gauche est empruntée), la règle 5 est bafouée (si une étape doit être activée et désactivée en même temps, il la désactive !), les simultanéités sont farfelues.
Une étape est définie par un identificateur Sx (x numéro unique), un nom (par défaut StepX) qui servira pour les synchronisations, et une extension (commentaire) noté à droite. On clique avec le bouton droit sur cette extension pour demander l'insertion d'un objet (une action). Celle-ci est définie par un code sur une lettre (N normal : sortie allumée au début de l'activation et éteinte à la désactivation, S set : sortie mise à 1, R Reset : sortie éteinte, D délai : allumage au bout d'un certain délai après l'activation, extinction à la désactivation, il y a d'autres options comme compteurs...).
Pour les transitions, on peut choisir le langage CONT ou LOG (j'utilise CONT). On peut insérer très facilement une tempo (Step12.T<10s), vérifier l'activation d'une étape (Step43.X). Par contre les fronts ne sont pas proposés de base. Pour y remédier, on peut par exemple rajouter une étape, on attend d'abord l'état 0 puis l'état 1.
Avant de terminer, il faut définir l'OB1 (programme principal), qui doit appeler notre FB. Je conseille d'entrer dans l'OB1 en langage CONT, d'insérer (voir fenêtre à gauche) le bloc FB en double cliquant sur FB1, donnez un nom de DB associée sur la boite (DB1), et marquer M0.0 devant l'entrée INIT_SQ du bloc.
Il ne reste plus qu'à sauver, charger l'automate (voir ci-dessous) et tester.
Transfert vers l'automate

Après avoir enregistré votre projet (vous aviez le droit de le faire plus tôt, si vous êtes prudent), il faut transférer le projet dans l'automate (il vaut mieux que l'automate soit en mode STOP, mais en RUN-P l'automate peut être arrêté temporairement à distance). Il suffit de choisir « système cible -> charger ». On peut regarder le programme actuellement dans l'automate (s'il est en mode RUN ou RUN-P) par « affichage -> en ligne » (hors ligne correspond au projet que l'on est en train de créer sur le PC). On peut même directement modifier un programme dans la fenêtre « en ligne » (si l'automate est au repos), voire faire du copier-coller ou glisser entre la fenêtre en ligne et hors ligne.
Dans la fenêtre « en ligne », en entrant dans le programme (OB1 ou autres blocs), on peut directement visualiser l'état des variables dans le programme. On choisit pour cela « test -> visualiser ». En CONT, les schémas deviennent en pointillés au endroits où « le courant n'arrive pas ». En LIST, un tableau est affiché à coté du programme, spécifiant les valeurs (0 ou 1) des opérandes, en LOG des 0 ou 1 sont écrits sur les liaisons. En Grafcet, les étapes actives sont en vert, les transitions validées sont montrées comme dans le langage correspondant, les valeurs des tempos, compteurs... sont notées à côté du schéma.
On peut également lister l'état de toutes les variables, voire les modifier. Pour cela, se placer sur les blocs (fenêtre gauche du projet), puis dans la fenêtre droite (il s'y trouve au moins OB1) cliquer avec le bouton droit et insérer une table des variables (VAT).
PLCSIM est un logiciel de simulation d'automates livré avec STEP7. On peut donc tester un programme sur un PC non relié aux automates (mais avec STEP7 installé, évidemment). Pour tester un programme, il n'est pas nécessaire d'avoir défini de matériel. Sinon, enregistrez-sous, puis supprimez la description du matériel (cliquez dessus avec le bouton de droite par exemple), et répondez NON quand il vous demande s'il faut également supprimer le programme. Démarrez le simulateur (outils -> simulation de modules ou l'icône représentant l'automate virtuel dans un nuage), affichez les E/S (insertion ->entrées ou sorties). Transférez le programme (par exemple par « système cible ->partenaires accessibles » et un copier-coller). Vous pouvez désormais tester (en mode RUN).
Pour en savoir plus
La documentation se trouve dans le logiciel (menu ? ou F1), mais aussi dans le menu démarrer sous Simatic -> documentation -> français. Même les experts se servent fréquemment de l'aide en ligne, (F1 sur un composant s'il a oublié le détail de ses entrées par exemple). Dans « STEP7, getting started » vous trouverez à peu près la même chose que dans ce document (mais on y parle aussi d'autres choses, par exemple des sous-programmes FB).
Dans « Step7, configuration matérielle » on détaille la configuration du matériel. Pour une seule valise c'est inutile de le lire, mais on y détaille la périphérie décentralisée (DP), les projets multi-cpu, la communication par données globales (GD)...

9.6 ADRESSES ET TYPES DE DONNEES AUTORISES DANS LA TABLE DES MNEMONIQUES

Les **mnémoniques** globaux (tous les blocs d'un même CPU) sont entre guillemets, et acceptent tous caractères. Les mnémoniques locaux débutent par # et comportent lettres, chiffres et souligné.
Ci-dessous tous les types de variables pouvant être associées à un mnémonique :

Anglais	Allemand	Désignation	Type de données	Plage d'adresses
I	E	Bit d'entrée	BOOL	0.0..65535.7
IB	EB	Octet d'entrée	BYTE, CHAR	0..65535
IW	EW	Mot d'entrée	WORD, INT, S5TIME, DATE	0..65534
ID	ED	Double mot d'entrée	DWORD, DINT, REAL, TOD, TIME	0..65532
Q	A	Bit de sortie	BOOL	0.0..65535.7
QB	AB	Octet de sortie	BYTE, CHAR	0..65535
QW	AW	Mot de sortie	WORD, INT, S5TIME, DATE	0..65534
QD	AD	Double mot de sortie	DWORD, DINT, REAL, TOD, TIME	0..65532
M	M	Bit de mémento	BOOL	0.0..65535.7
MB	MB	Octet de mémento	BYTE, CHAR	0..65535
MW	MW	Mot de mémento	WORD, INT, S5TIME, DATE	0..65534
MD	MD	Double mot de mémento	DWORD, DINT, REAL, TOD, TIME	0..65532
PIB	PEB	Octet de périphérie d'entrée	BYTE, CHAR	0..65535
PQB	PAB	Octet de périphérie de sortie	BYTE, CHAR	0..65535
PIW	PEW	Mot de périphérie d'entrée	WORD, INT, S5TIME, DATE	0..65534
PQW	PAW	Mot de périphérie de sortie	WORD, INT, S5TIME, DATE	0..65534
PID	PED	Double mot de périphérie d'entrée	DWORD, DINT, REAL, TOD, TIME	0..65532
PQD	PAD	Double mot de périphérie de sortie	DWORD, DINT, REAL, TOD, TIME	0..65532
T	T	Temporisation	TIMER	0..65535
C	Z	Compteur	COUNTER	0..65535

Anglais	Allemand	Désignation	Type de données	Plage d'adresses
FB	FB	Bloc fonctionnel	FB	0..65535
OB	OB	Bloc d'organisation	OB	1..65535
DB	DB	Bloc de données	DB, FB, SFB, UDT	1..65535
FC	FC	Fonction	FC	0..65535
SFB	SFB	Bloc fonctionnel système	SFB	0..65535
SFC	SFC	Fonction système	SFC	0..65535
VAT	VAT	Table des variables		0..65535
UDT	UDT	Type de données utilisateur	UDT	0..65535

10 GLOSSAIRE

Terme	Description	Explication
C7	Combo PLC/HMI system	A PLC and screen in one package
CFC	Continuous Function Chart	Optional programming language
CP	Communication Processor	Modules used for special communication protocols
DB	Data Block	Memory storage areas for user data
FB	Function Block	A function with it's own data block
FBD	Function Block Diagram	Standard programming language
FC	Function Call	Called progammed blocks
FM	Function Module	Modules with special functions (e.g. positioning)
GSD	Generic Station Description	Files used for Profibus descriptions
HiGraph		Optional programming language
IM	Interface Module	Modules to connect remote racks
LAD	Ladder Logic Diagram	Standard programming language
M7	Programmable modules	A module with processing capabilities
MMC	Micro Memory Card	Compact plug-in memory card
MPI	Multi Point Interface	Standard communication protocol
OB	Organization Block	Blocks for user programs based on different operating system events.
OP	Operator Panel	Simple display with or without buttons
PCS	Process Control System	Software for the entire process chain
PG	Programming Terminal	Dedicated Siemens device - basically a PC
PPI	Point to Point Interface	Serial RS-232 communication
Profibus DP	Profibus Decentral Peripherals	Networking protocol used for factory automation
Profibus PA	Profibus Process Automation	Networking protocol used for process automation
S7		SIMATIC Step 7 product line
SCL	Structured Control Language	Optional programming language
SFB	System Function Block	Integrated FB for CPU information
SFC	System Function Call	Integrated FC for CPU information
SM	Signal Module	Standard Input/Output modules
STL	Statement List	Text based programming language
TP	Touch Panel	Touch screen display
UDT	User-Definded Data Type	Special data structures defined by the user
VAT	Variable Access Table	Tables used to monitor/modify values in the PLC