Utilisation de Step 7

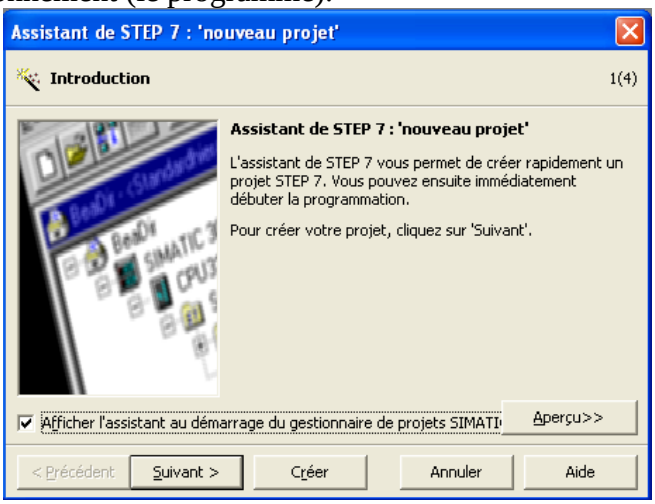
Introduction

Step 7 permet l'accès "de base" aux automates Siemens. Il permet de programmer individuellement un automate (en différents langages). Il prend également en compte le réseau, ce qui permet d'accéder à tout automate du réseau (pour le programmer), et éventuellement aux automates de s'envoyer des messages entre eux. Mais il ne permet pas de faire participer les ordinateurs à l'automatisme (possible sous PCS7 ou TIA Portal). Le logiciel s'appelle « Simatic Manager », disponible en général dans le menu démarrer sous « Siemens »

Créer son projet

Un projet contient la description complète de votre automatisme. Il comporte donc deux grandes parties : la description du matériel, et la description du fonctionnement (le programme).

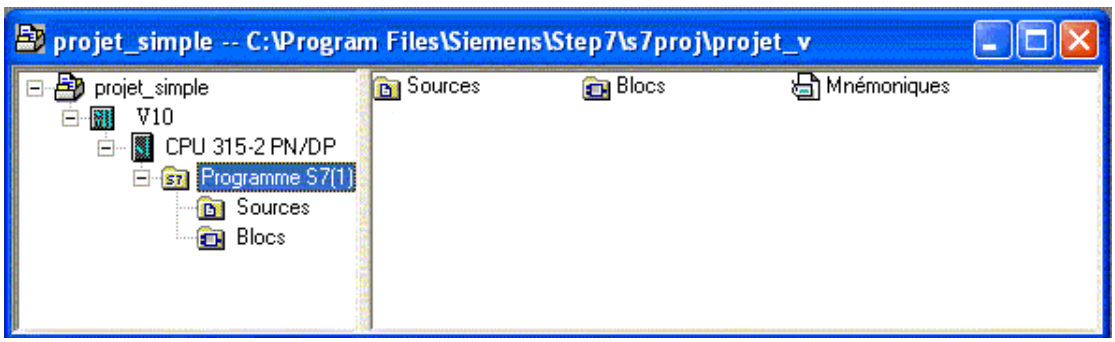
En entrant dans Step7, il peut y avoir un assistant qui vous propose de créer un nouveau projet, il vaut mieux l'**annuler** car par défaut il configure mal la liaison avec l'automate. On choisira donc plutôt « fichier -> nouveau » (mettez votre nom dans le nom de projet). Ou alors « fichier -> ouvrir » : pour retrouver un projet existant sur ce poste, cliquez « parcourir » puis « lancer la recherche » sans rien changer ; mais s'il était sur votre répertoire perso, il sera sur « U: » N'utilisez pas un projet existant autre qu'un des vôtres, suivant les filières les projets peuvent être incompatibles (et quelquefois planter le réseau) ! Si, en lançant Step7, d'autres projets sont ouverts, fermez les, pour éviter toute interférence avec le votre.



Le matériel

La première chose à faire est de décrire le matériel. Nous disposons de 10 PC appelés ESx (x entre 0 et 9), et de 10 valises (comportant un automate, ses modules d'entrées/sorties, ainsi qu'une console de simulation) nommées Vxx (V10 à V19). Ils sont tous reliés par un réseau industriel (PROFIBUS, câble violet) et par Ethernet (redondant, inutilisé ici). Repérez votre PC et votre valise, leurs numéros sont indiqués sur une étiquette rouge

Pour commencer, insérez dans le projet une station SIMATIC 300 (renommez la du nom de votre valise, par exemple V10).



On définit ensuite le **matériel** : un rail support (à trouver dans la liste du matériel, pour la gamme Simatic 300, dans les racks). Il vous prévoit plusieurs lignes : la première (dans l'ordre de leur implantation physique, de gauche à droite) l'alimentation (repérez sur le matériel, en haut son type : PS 307 5A, en bas son numéro de référence : 307-1EA-0AA0). Insérez en ligne 2 l'automate (son numéro IP 192.168.0.1xx est noté sur la valise). La ligne 3 est réservée aux coupleurs (en particulier pour connecter un second rail), puis le module 32entrées ToR, le module 32 sorties ToR, le module 8E/8S (sauf sur certains postes, mais le mettre ne pose pas de pb), et enfin le module analogique. Il est judicieux de préparer un projet contenant la description de l'automate, sans y mettre de programme, et l'ouvrir à chaque nouveau programme (enregistrer sous... pour garder le projet initial). Je parle évidemment de ceux qui auraient plusieurs TP successifs à effectuer ici.

Emplacement	Module ...	Référence	Firmware	Adresse MPI	Adresse d'entrée	Adresse de sortie
1	PS 307 5A	6ES7 307-1EA00-0AA0				
2	**CPU 315-2 PN/DP**	**6ES7 315-2EG10-0AB0**	**V2.3**			
X1	*MPI/DP*				*2047**	
X2	*V10*				*2046**	
3						
4	DI32xDC24V	6ES7 321-1BL00-0AA0			0...3	
5	DO32xDC24V/0.5A	6ES7 322-1BL00-0AA0				0...3
6	DI8/DO8xDC24V/0,5A	6ES7 323-1BH01-0AA0			4	4
7	AI2x12Bit	6ES7 331-7KB02-0AB0			6...9	

Vérifiez bien les différentes adresses. Utilisez exactement l'adressage du tableau ci-dessus pour les modules d'E/S Pour le CPU, en ligne X1, vous définissez la connexion PROFIBUS (et non MPI). On se connecte à 500kbits, en format standard : les 10 valises Vxx (V10 à V19), ont xx comme numéro PROFIBUS (et surtout pas 2 qu'il vous propose par défaut). En X2 (seconde interface, ici Ethernet), définissez son numéro IP (192.168.0.1xx) et donnez le nom de votre valise (par exemple V10), il ne doit plus s'appeler PN-IO (activez toujours les deux liaisons, même si vous en utiliserez qu'une, c'est utile en cas de problème réseau).

Dans certains projets très spécifiques (PCS7 par exemple), il faudra également définir le PC, nommé ESx (x entre 20 et 29), de numéro PROFIBUS x (le numéro est sur l'interface Profibus), voire plusieurs PC.

Dans certains cas, nous utiliserons plusieurs valises, comme elles se ressemblent un copier-coller suivi de quelques modifications (nom, adresses IP et Profibus au minimum) est envisageable. Nous avons en plus un gros automate (série S400) nommé AS400, d'adresse PROFIBUS 7, et 192.168.0.107 comme IP. Ci dessous sa description actuelle (à définir uniquement si vous l'utilisez !). Le rack est de type UR2ALU (6ES7 400-1JA11-0AA0), Il est associé à une périphérie décentralisée (IM153, dans Profibus_DP→ ET200M, d'adresse Profibus 5).

Emplacement	Module	Référence	Firmware	Adresse MPI	Adresse d'entrée	Adresse de sortie
1	PS 407 10A	6ES7 407-0KA01-0AA0				
3	**CPU 414-3 DP**	**6ES7 414-3XJ04-0AB0**	**V4.0**			
X2	*DP*				*8191**	
X1	*MPI/DP*				*8190**	
IF1						
5	CP 443-1	6GK7 443-1EX11-0XE0	V2.5		8189	

Les mnémoniques

On peut (devrait, sauf dans les problèmes suffisamment simples comme le premier TP) donner des noms explicites aux différentes E/S, en choisissant "mnémoniques" dans le dossier "programme" de l'automate. De base, les entrées ToR se notent E a.b (E=ein) avec a l'adresse du module (ou la partie d'adresse, on regroupe par octet, donc dans un module 32 E/S il y a 4 adresses a), b étant le numéro du bit dans l'octet (entre 0 et 7). Exemple : E0.4 est la cinquième entrée du premier bloc d'entrées) les sorties se notent A a.b (A=Aus). Les entrées et les sorties peuvent utiliser les mêmes adresses, les 32 entrées du premier bloc

s'appellent E0.0 à E3.7, les 32 sorties du second bloc s'appellent A0.0 à A3.7. On peut également accéder directement à un octet complet (B, par ex EB0), un mot (W) de deux octets, un double mot (D) de 4 octets. Pour stocker des résultats intermédiaire, on dispose de mémoires internes (mémento) nommés en ToR M0.0 à M65535.7 (si on a assez de mémoire), ou MB, MW, MD. Pour les valeurs numériques et l'arithmétique on dispose aussi des types int, dint, real, char, date, time… (que nous traiterons en particulier en LIST).

Les mnémoniques sont des variables globales (pour tous les blocs ou sous-programmes). Mais elles sont définies pour un matériel donné uniquement (une valise). Voici un exemple de table de mnémoniques :

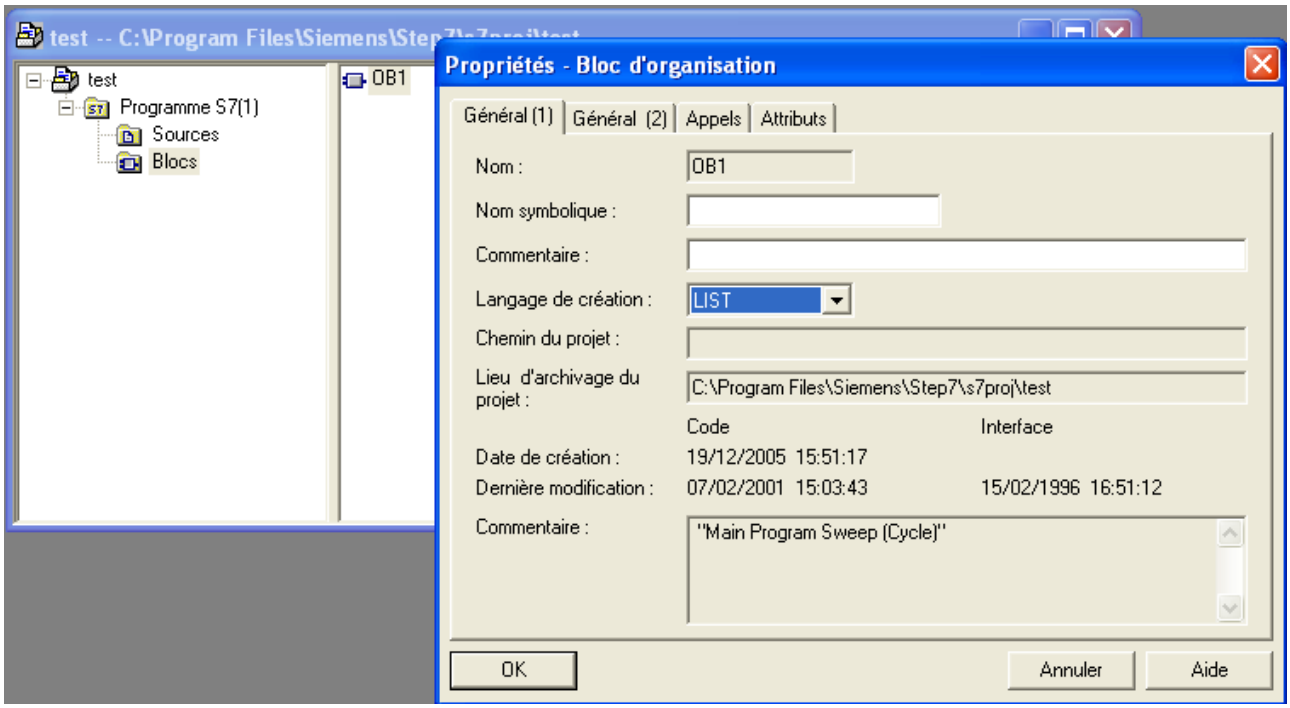
Sur nos consoles de simulation, EB0 et EB1 sont reliés à 16 interrupteurs, EB2 et EB3 à 4 roues codeuses. AB0 et AB1 à 16 LEDs, AB2 et AB3 (ou AW2 pour 16 bits) à l'afficheur numérique, EW6 aux potentiomètres (et voltmètre).

Programme S7(1) (Mnémoniques) -- projet_simple\...

	Etat	Mnémonique	Opéran		Type de d	Commen
15		led15	A	1.6	BOOL	
16		led16	A	1.7	BOOL	
17		octet s1	AB	0	BYTE	
18		octet s2	AB	1	BYTE	
19		octet s3	AB	2	BYTE	
20		octet s4	AB	3	BYTE	
21		double s1	AD	0	DWORD	
22		afficheur	AD	2	DWORD	
23		mot s1	AW	0	WORD	
24		mot s2	AW	2	WORD	
25		bouton1	E	0.0	BOOL	
26		bouton2	E	0.1	BOOL	
27		bouton3	E	0.2	BOOL	

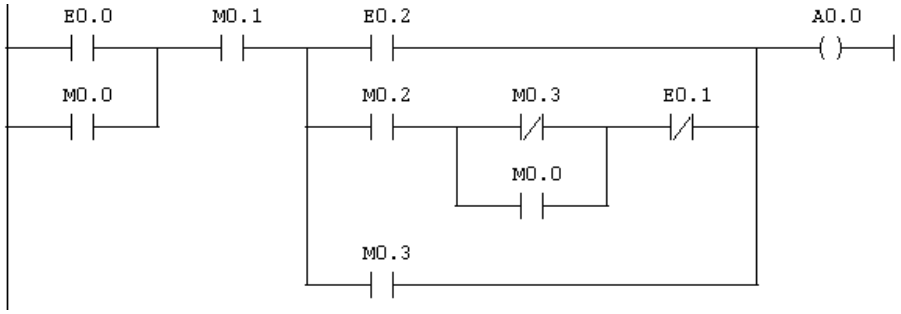
Le programme

Le programme sera placé **dans l'automate** (->programme->blocs). Le "programme principal" s'appelle obligatoirement OB1 (OB= Bloc d'Organisation, contient un bout de programme, on pourrait aussi appeler cela un sous-programme). On double clique sur OB1 pour entrer le programme. Il faut avant tout choisir son langage préféré (dans "affichage" s'il ne le propose pas automatiquement) : CONT (langage à contacts), LIST (langage textuel), ou LOG (portes logiques). D'autres langages (optionnels) existent, les trois qui me semblent les plus intéressants sont SCL (langage proche du Pascal, permettant des algorithmes et calculs complexes), GRAPH (proche du Grafcet), CFC (blocs fonctionnels)...

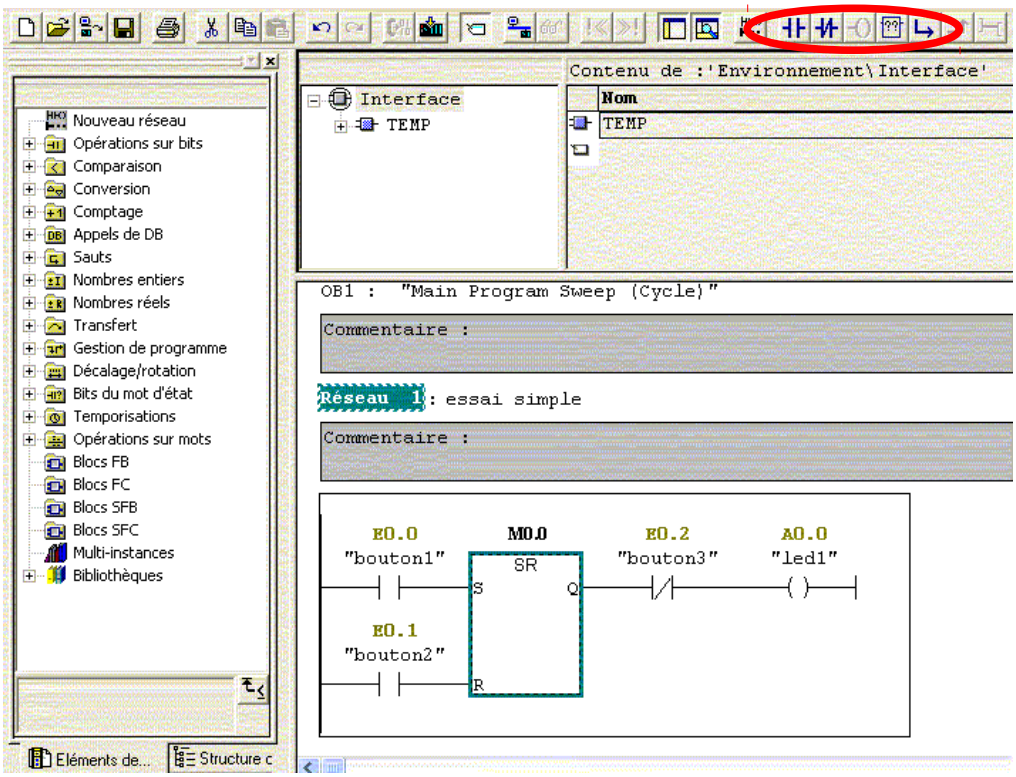


Le langage CONT

C'est une suite de réseaux (en ToR combinatoire, je conseille un réseau par sortie). Les entrées sont



représentées par des interrupteurs -| |- (ou -|/|- si entrée inversée) ; les sorties par des bobines -( ) ou des bascules -(S) -(R). Il y a également des opérations unaires (une entrée une sortie) : l'inverseur -|NOT|-, l'attente d'un front montant -(P)- ou descendant -(N)- ces opérations ont souvent besoin qu'un leur donne un bit de mémoire, par ex M0.0 s'il ne sert pas déjà). Les sorties sont obligatoirement à droite du réseau. On doit évidemment définir nos E/S, soit directement par leur code (E a.b / A a.b), ou avec leur nom en clair défini dans la table des mnémoniques (entrez le nom entre guillemets). On relie les éléments en série pour la fonction ET, en parallèle pour le OU. On peut utiliser des bits internes (peuvent servir en bobines et interrupteurs), comme on utilise dans une calculatrice une mémoire pour stocker un résultat intermédiaire (M a.b). On peut aussi introduire des éléments plus complexes, en particulier les opérations sur bits comme ci-contre une bascule SR. Toutes ces fonctionnalités sont disponibles dans le menu de gauche (et celles de base : en haut à droite), sauf si on l'a fermée, sinon choisir « affichage -> vues d'ensemble ». Mais si vous voulez insérer des blocs complexes, utilisez plutôt le langage LOG.Dans le document en ligne « CONT pour S7 » on trouvera d'autres fonctions utiles, les compteurs, les tempos, à la rigueur le registre à décalage qui permettrait de gérer du séquentiel sans Grafcet. On peut également utiliser des fonctions plus complexes (calculs sur mots par exemple), mais là je pense qu'il vaut mieux travailler en langage LIST.

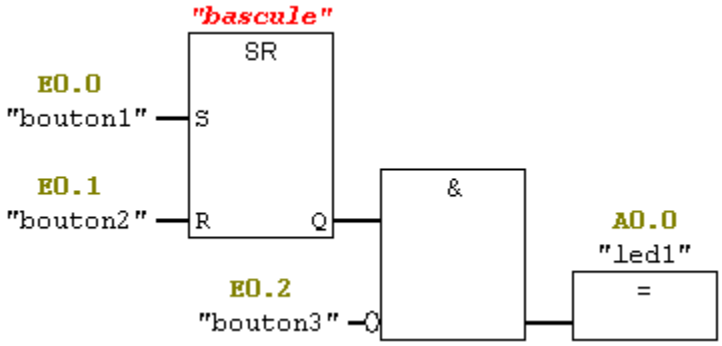


Le programme est en général décomposé en plusieurs réseaux, par exemple un réseau par sortie (2 parties du schéma non reliées entre elles doivent être dans deux réseaux différents). Les réseaux sont exécutés séquentiellement.

Le langage LOG

C'est un langage graphique, utilisant les symboles de l'électronique numérique (portes logiques). Il n'y a rien de spécial à dire, c'est très intuitif. On peut utiliser de nombreuses entrées pour une même porte, placer des inverseurs sur les entrées.... Ici, on découpe son programme en plusieurs réseaux (en général quand un ensemble de blocs n'est pas relié au reste, ou un réseau par sortie...) Voici l'exemple correspondant au programme CONT montré plus haut :

Une bascule nécessite un bit mémo (par ex M0.0)



Le langage LIST

C'est un langage textuel, qui est le plus proche du comportement interne de l'automate (correspond à peu près à l'assembleur dans un ordinateur). Le système sait **toujours** traduire du CONT ou du LOG en LIST, mais pas obligatoirement l'inverse. Le programme se compose d'une suite de lignes, chacune spécifiant un code opération suivi d'un opérande (et un seul). L'opérande peut être une adresse absolue (E0.0) ou un mnémonique entre guillemets (si les mnémoniques ont été définis, bien sûr). Comme on ne peut pas utiliser deux opérandes dans une même ligne, pour faire « x=a et b » on écrit :

```
U "a"
U "b"
= "x"
```

On utilise U pour ET (und), O pour Ou (oder), X pour Ou Exclusif), UN, ON et même XN pour les entrées inversées, = pour stocker le résultat. L'opération (O ou U) pour le premier opérande n'a pas grande importance (si l'opération précédente était =, R ou S, il recommence un nouveau calcul).

Pour une bascule on utilisera S et R :

```
U "a"
S "x"
U "b"
R "x"
```

NOT inverse le résultat précédent, FP indique si le résultat précédent vient de passer de 0 à 1 (front montant), FN pour le front descendant (voyez l'aide, il faut lui donner un bit de mémoire disponible, par exemple M0.0). Ici aussi, on peut décomposer le programme en plusieurs réseaux, mais peut-être est-ce moins courant qu'en CONT.

On peut, comme en programmation classique, faire des sauts (goto) : SPA label (inconditionnel), SPB (si dernier calcul = 1), SPBN (si 0). On saute à une ligne précédée de « label : » (4 caractères maxi je crois), qui peut être placé avant (boucle) ou après (pourtant appelé saut avant). Attention les boucles ne doivent pas durer plus de quelques milisecondes.

En LIST, on peut aussi traiter des mots (peut-être plus facilement que dans les autres langages) : octets (B), entiers de 16 bits (W) ou 32 bits (D), réels 32 bits. On utilise un accumulateur sous forme d'une pile (à 2 niveaux). On empile une valeur par L (load). L charge une valeur dans l'accumulateur, tout en poussant l'ancienne valeur qu'il contenait dans la pile. Les calculs cherchent leurs deux valeurs dans (1) l'accumulateur (dernier résultat ou dernier load) et (2) la pile (calcul ou load précédent). Le résultat est mis dans l'accumulateur (son ancienne valeur est perdue, car la pile n'est pas modifiée). Les calculs possibles se notent, pour les entiers, +I, -I, /I, *I (idem +D,... sur 32 bits, +R réels). On peut comparer les deux derniers niveaux de la pile par <I, >I, <=I, >=I, = =I, <>I (idem <D pour les doubles ou <R pour les réels). On sauve l'accumulateur par T. Voici par exemple comment on incrémente

```
L   MW0      //charger le mot double
L   1        //charger la constante 1
+I           //les ajouter (format entier)
T   MW0      //enregistrer le résultat
```

Attention, l'écriture est plus proche de la notation polonaise que d'une calculatrice : on charge les deux opérandes puis seulement on dit l'opération à effectuer.

Autre exemple (sur des entiers 16 bits) :

```
L   MW0      //charger le mot double
L   130      //charger la constante
>I           //tester
=   A0.0     //s'allume si MW0 était supérieur à 130
```

Sur les réels on peut faire de nombreux calculs : ABS, SQR, SQRT, LN, EXP, SIN, COS, TAN, ASIN, ACOS, ATAN.

Les conversions de formats de mots sont possibles : ITD ou DTI (16 à 32 bits), DTR (entier à réel) ou RND, TRUNC (réel à entier, arrondi ou troncqué). On note B les nombres en BCD : sur 16 bits, ils sont entre -999 et +999, les 3 quartets de droite pour les 3 chiffres, celui de gauche pour le signe (0 = positif, F = négatif). On traduit par ITB ou BTI (12 bits + signe), DTB ou BTD (16 à 28 bits + signe). C'est avec cela que vous gérerez l'afficheur numérique de la valise :

```
L   EW6        //acquérir l'entrée CNA
L   W#16#7FF8  //masque pour ne garder que 12bits, la capacité du convertisseur
UW             //masquage
SRW 3          //décallage de 3 bits à droite
DTB            //traduire en BCD
T   AW2        //transférer sur l'afficheur
```

DTB trouve le nombre binaire qui s'affichera comme si c'était du décimal. Par ex. si dans l'accu il y a 100d (1100100b), BTD le remplacera par 256d (1 0000 0000b) qui s'affichera 100 (par 4 bits : centaine, dizaine, unité). DTB traduit un BCD en sa valeur binaire, ce qui nous permettra de faire des calculs.

Les constantes sont par défaut en décimal. On peut insérer des « _ » non significatifs mais pas d'espaces (1234 ou 1_234 mais pas 1 234). On peut utiliser une autre base X en commençant par X# (2#0110_1100 16#1ABA6). Les réels contiennent un « . », et peuvent être suivis d'un exposant (puissance de 10) : 1.2E-5. Les caractères sont entre quotes (simple apostrophe), le « $ » est un caractère d'échappement, c'est à dire qu'il permet d'entrer un caractère spécial qu'un ne pourrait pas entrer autrement ($', $$, $N pour nouvelle ligne...) ou directement son code en hexa ($4E pour 'N').

Le système gère automatiquement le type de la constante, si on veut imposer son type il suffit de la précéder par TYPE# : BOOL (bool#true, Bool#FALSE, BOOL#1), BYTE ou CHAR (BYTE#16#FF, BYTE#'a', BYTE#127), INT ou WORD ou W (W#16#FFFF), DINT ou DWORD ou DW voire REAL (car les réels sont stockés sur 32 bits).

Les constantes de dates débutent par DATE# (ou D#), TIME# (T#) DATE_AND_TIME# (DT#) ou TIME-OF_DAY# (TOD#). La date est sous la forme année-mois-jour (D#2021-06-28), l'heure h:mn:s.ms (les derniers sont optionnels) (DT#2019-02-02-11:11:11) Pour les durées, on utilise TIME dans le format définit plus haut, ou on précise les unités (on peut alors prendre des nombres réels pour le dernier) : D (jour)H (heure) M S MS (T#1d3h2m, T#2.5s, T#150ms). Il y a aussi des formats hérités de S5 (pour les tempos : S5T#300ms).

Pour interagir entre numérique et ToR : faire ou non certains calculs en fonction de ToR :

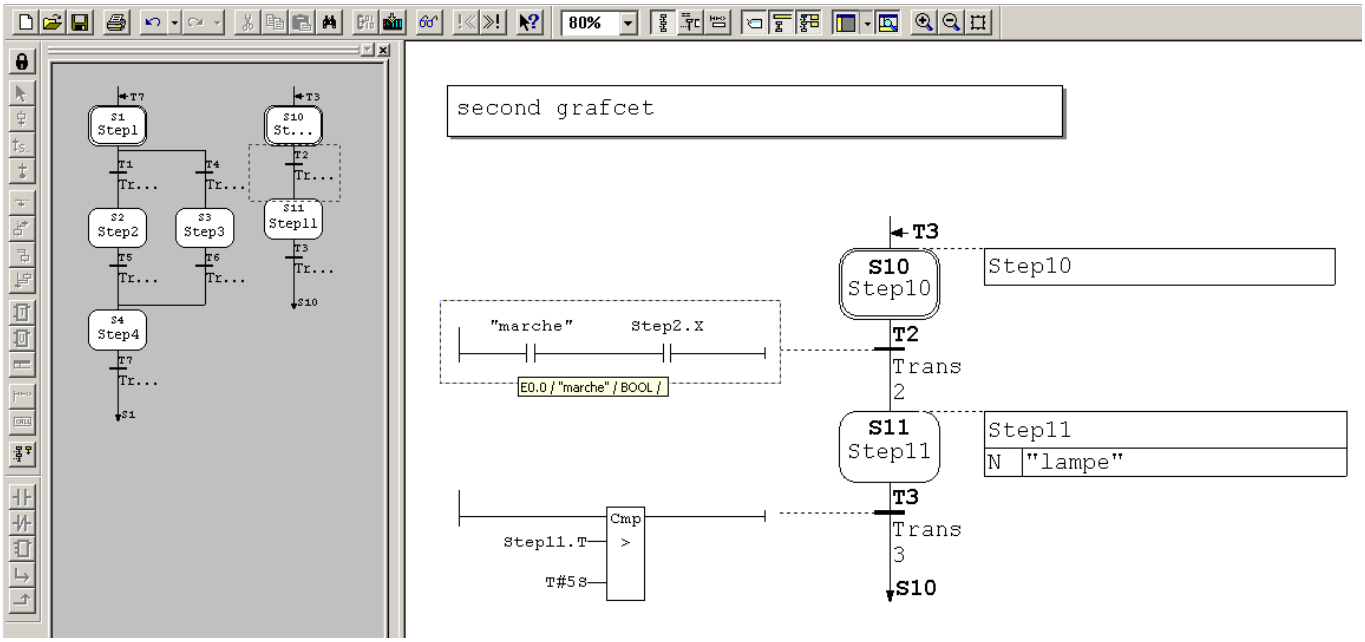
```
      U   E0.0
      UN  E0.1  //condition en fonction d'E/S ToR
      SPBN Lbl1 //si vrai on fait la suite, sinon on saute à « Lbl1 »
          //ici mettre vos calculs numériques
Lbl1 : //ici mettre la suite du programme (fait dans les 2 cas)
```

à l'inverse : faire du ToR en fonction de calculs numériques :

```
      L   EW2
      L   W#16#0A3F //faire des calculs numériques
      <I            //test qui a pour résultat vrai ou faux
      U   E0.0      //éventuellement continuer du calcul combinatoire
      =   A0.0      //affecter une sortie ToR en fonction des calculs précédents
```

Le langage GRAPH

Le Graph (langage proche du Grafcet mais moins puissant) n'est pas un langage de base dans S7. Mais 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 créera automatiquement un DB (les données associées, DB1), un FC72 et un SFC64 (fonctions système nécessaires) quand le Grafcet sera fini (et sans erreur, seuls les warnings sont autorisés). Il n'y a plus qu'à rentrer le programme, le sauver. Vous aviez également le droit de préciser des mnémoniques. Le système préfère 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 (rajouter des tempos de quelques millisecondes).



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...). Dans la plupart des cas, choisissez N ! et dans la seconde moitié, notez la sortie (mnémonique ou Ax.y)

Pour les **transitions**, on les définit graphiquement, à gauche de la transition. On peut choisir le langage CONT ou LOG (j'utilise CONT). On peut insérer très facilement une tempo voir bandeau de gauche, objet T (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 y a bien d'autres options, par exemple si vous voulez un mode auto et un mode manu, faire un Grafcet normal et un grafcet d'initialisation : voir l'aide du logiciel

Il ne reste plus qu'à sauver (s'il dit qu'il n'y est pas arrivé, c'est qu'il y a une erreur), charger l'automate (voir ci-dessous) et tester.

Erreurs fréquentes : Grafcet non fini, ou fini par une « fin » plutôt qu'un « saut » à la première étape, action mise dans le commentaire (StepX). Grafcet sans étape initiale (en général si vous avez inséré un second Grafcet, comme sur l'exemple ci-dessus). Oubli de l'OB1. Par contre les avertissements ne sont pas gênants.

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 pour ne pas démarrer intempestivement des actions). Il suffit de choisir « système cible -> charger ». Au premier chargement, chargez TOUTE votre valise, car les étudiants précédents ont peut-être utilisé une configuration matérielle différente, ou lancé des programmes mis à d'autres endroits comme l'OB100… dans la suite du TP, vous pouvez ne charger que ce que vous venez de changer. Attention : il charge ce qui est sur le disque dur, donc pas ce que vous voyez si vous avez oublié de sauvegarder. Moi je conseille souvent de fermer toutes les fenêtres sauf le projet, quand on a des problèmes de chargement.

On peut regarder le programme actuellement dans l'automate (s'il est en mode RUN) par « affichage -> en ligne », ou en cliquant sur les « lunettes » (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)…

Patrick TRAU