IUT ANNECY UNIVERSITÉ SAVOIE MONT BLANC

Les fonctions

Cours

v1

IUT d'Annecy, 9 rue de l'Arc en Ciel, 74940 Annecy

Exercices complémentaires

Ce cours présente des rappels de l'utilisation du logiciel CodeSys dans une approche dite "Classique". Nous verrons dans un prochain cours une approche orientée objet appliquée à ce logiciel.

1 Architecture d'un système automatisé

1.1 Architecture matérielle

Un système automatisé centralisé consiste en une unique unité centrale qui gère l'ensemble des entrées et des sorties. Cette unité centrale est donc reliée à l'ensemble des capteurs et des actionneurs. Dans le domaine qui nous intéresse, l'unité centrale est un automate programmable industriel (API) et pourrait gérer une série d'étapes d'une chaîne de production.

Avantages
- Un automate unique.
- Architecture de données simple.

Inconvénients
- Nombre d'entrées/sorties important.
- Traitement d'automatisation complexe.
- Maintenance et évolutions difficiles.

A l'inverse, un système automatisé modulaire et réparti consiste en plusieurs unités centrales qui gèrent chacune une partie des entrées et des sorties. Chaque unité centrale est reliée à un ensemble de capteurs et d'actionneurs. Un protocole de communication est mis en place entre les unités centrales.

Avantages
- Nombre d'entrées/sorties limité par unité centrale.
- Maintenance et évolutions propres à chaque unité.

Inconvénients
- Architecture de données complexe.
  - ◇ Orientation des données.
  - ◇ Perte de la sémantique des données qu'il faut reconstruire (union, chevauchement, ...).
- Protocole de communication à mettre en place.

Critères de choix Le choix entre une architecture centralisée ou modulaire ainsi que les matériels et logiciels à utiliser se feront en fonction de différents critères :
- Nombre d'entrées/sorties TOR.
- Nombre d'entrées/sorties analogiques.
- Communications à mettre en place entre les unités centrales (protocoles, nombres).
- Maintenance et évolution.
- Interconnexion avec le Cloud (4.0).
- Puissance de calcul nécessaire.
- Boucle de régulation intégrée ou non.
- Coût (matériel, chaîne de développement, ...).
- Langages IEC 61131-3 supportés.

- Les extensions de ces langages pouvant conduire à une programmation orientée objet.

1.2 Le système d'exploitation

L'automate industriel programmable embarque un système d'exploitation temps réel. Ce système d'exploitation est un logiciel qui permet de gérer les ressources matérielles de l'automate (processeur, mémoire, entrées/sorties, ...) et de fournir des services aux applications (gestion des tâches, gestion des communications, ...).

Pour programmer efficacement et correctement un automate, il est nécessaire de comprendre, au moins sommairement, le fonctionnement du système d'exploitation.

Système temps réel

Un système d'exploitation temps réel garantit un temps de réponse maximum à toute requête ou événement matériel.

Ce type de système d'exploitation est utilisé dans les systèmes embarqués, les automates programmables, les systèmes de contrôle-commande, ...En résumé, dans tous les domaines où la réactivité à un événement externe est primordiale.

1.3 Propriétés du système d'exploitation d'un API

Un système d'exploitation d'un API possède les propriétés suivantes :

Temps réel : par la mise en place d'un chien de garde (watchdog).

Interruptible : les tâches peuvent être interrompues par une tâche de **priorité supérieure** parce que l'événement qu'elle traite est **fugitif** ou d'**importance** pour l'application.

Mono/Multi-tâche : Plusieurs tâches peuvent être exécutées de façon apparemment simultanée.

Accès facilité aux entrées/sorties.

Accès facilité à la mémoire : Permet de partager des données entre les tâches, de gérer les accès concurrents à la mémoire, ...

Interaction avec le programme utilisateur .

Les événements sont capturés par des fonctions appelées **event handlers**. Ces fonctions sont automatiquement appelées par le système d'exploitation lorsqu'un événement défini se produit.

Système Multitâches

Un système multitache permet d'éxécuter plusieurs tâches "en même temps". En réalité, une stratégie de priorité des tâches est mis en place. Selon le nombre de processeur présents et le nombre de tâches, elles seront exécutées en parallèle ou en alternance.

2 Environnement de programmation : CoDeSys

Dans ce module, nous programmerons sous l'environnement de développement CoDeSys.

2.1 Présentation

CoDeSys est un environnement de développement intégré (IDE) permettant de programmer des automates programmables industriels (API). Il est basé sur la norme IEC 61131-3. Il permet de programmer des automates de différentes marques (Beckhoff, Wago, Schneider, ...). Il est disponible sous Windows et sous Linux. Il est gratuit pour une utilisation non commerciale.

2.2 Langages

CoDeSys permet de programmer des automates en utilisant les langages suivants :
- Instruction List (IL).
- Ladder Diagram (LD).
- Function Block Diagram (FBD).
- Structured Text (ST).
- Sequential Function Chart (SFC).

2.3 Architecture d'un programme

CoDeSys permet de programmer en Monotâche et en Multitâche :
- 15 tâches périodiques au maximum.
- Priorité des taches de 1 à 15, de la plus prioritaire à la moins prioritaire.
- Temps de cycle compris entre 100 µs et 10 s.

2.4 Unité de Programme – Program Organisation Unit (POU)

Une unité de programme est un ensemble d'instructions qui peuvent être appelées depuis un autre programme. Il existe 3 types de POU :

Programme (Program – PRG) : Un programme consiste en une série d'instruction pouvant produire une ou plusieurs valeurs en sortie. Il peut être appelé depuis un autre programme ou depuis un bloc fonction. Toutes les valeurs des variables du programme restent inchangées entre deux appels, quelque soit sa provenance.

Bloc fonction (Function Block – FB) : Un bloc fonction est un ensemble d'instructions pouvant produire une ou plusieurs valeurs en sortie. Il peut être appelé depuis un autre programme ou depuis un bloc fonction. Un bloc fonction sera toujours appelé au travers d'une instance de bloc fonction, copie du bloc fonction. Par conséquent, les valeurs des variables du bloc fonction seront inchangées entre deux appels d'une même instance uniquement. Ce type de POU sera particulièrement adapté à la programmation orientée objet que nous développerons dans ce module.

Fonction (Function – FUN) : Une fonction ne peut produire qu'une seule valeur en sortie. Elle peut être appelée par n'importe quel POU et les valeurs des variables ne persistent pas d'un appel à l'autre.

Structure d'un POU

Un POU est divisé en 2 parties :
1. Déclaration des variables.
   - Variables d'entrées/sorties : **VAR_INPUT**, **VAR_OUTPUT**, **VAR_IN_OUT**.
   - Variables Locales : **VAR**, **CONSTANT**.
2. Implémentation du POU

METHOD et ACTION

Au sein d'une POU, il est possible de définir des méthodes et des actions.

ACTION : Portion de code toujours accessible qui ne peut utiliser que le **contexte** de l'entité à laquelle elle est associée.

Exemple : Une fonction RESET qui remet à zéro toutes les variables d'entrées/sorties d'un bloc fonction.

METHOD : Portion de code accessible qui utilise le contexte de l'entité à laquelle elle est associée mais qui peut également créer son propre contexte. On peut définir **l'accessibilité** d'une méthode (PRIVATE,

PROTECTED, PUBLIC, INTERNAL, ...).

2.5 Les types de données

2.5.1 Les types de données natifs

CodeSys propose un ensemble de types de données natifs. Ces types de données sont définis par la norme IEC 61131-3. Pour chaque type de données, le tableau suivant précise son empreinte mémoire ainsi que les règles de nommage (préfixe) que nous utiliserons dans ce module.

Type de données	Empreinte mémoire	Règles de nommage
BOOL	1 bit	x
BYTE	8 bits	by
SINT	8 bits	si
USINT	8 bits	usi
WORD	16 bits	w
INT	16 bits	i
UINT	16 bits	ui
DWORD	32 bits	dw
DINT	32 bits	d
UDINT	32 bits	udi
REAL	32 bits	r
LWORD	64 bits	lw
LINT	64 bits	li
ULINT	64 bits	uli
LREAL	64 bits	lr
STRING	Variable	s

TABLE 1 – Empreinte mémoire et règles de nommage des types de données natifs

2.5.2 Les types de données de datation

Type de données	Empreinte mémoire	Règles de nommage
TIME (T#)	32 bits	tim
TIME_OF_DAY (TOD#)	32 bits	tod
DATE (D#)	32 bits	date
DATE_AND_TIME (DT#)	32 bits	dt

TABLE 2 – Empreinte mémoire et règles de nommage des types de données de datation