PC 3000

Langages

CE

EUROTHERM
AUTOMATION

HA 174 517 - Indice 1

Chapitre 1

INTRODUCTION

Edition 1

Sommaire

PRESENTATION

Le PC3000 est la tête de série d'une nouvelle génération de régulateurs programmables qui peuvent servir à traiter à la fois les processus de production et les processus prototypes. Pour permettre le développement de programmes de régulation pour le PC3000, Eurotherm Automation offre deux stations de programmation évoluées : la station de programmation DOS qui est décrite de manière détaillée dans le présent guide utilisateur et la station de programmation Microcell qui autorise une programmation graphique ainsi que des écrans de simulation et une gestion des recettes intégrés.

Les deux stations de programmation offrent un ensemble complet de fonctions accessibles par menus et par écrans conviviaux.

Elles fournissent au concepteur une station de travail conçue spécialement pour faciliter le développement de programmes de régulation et ce pour toutes les phases, de la conception initiale à l'exploitation en ligne en passant par le développement et la mise en service. Chaque système offre aussi des informations d'aide intégrées permettant l'accès aux descriptions de nombreuses fonctions et de nombreux éditeurs directement depuis l'écran.

Afin de vous permettre d'exploiter à fond les multiples fonctions du PC3000, vous trouverez des informations supplémentaires dans une série de guides utilisateur et de manuels de référence.

GUIDES UTILISATEUR PC3000

Pour vous aider à utiliser la station de programmation PC3000, les guides utilisateur sont présentés sous la forme de quatre volumes. Chacun d'entre eux contient un ou plusieurs manuels et est structuré de manière à répondre à des questions élémentaires bien précises :

Volume 1 - Programmation de PC3000

"Comment utiliser le PC3000 ?"

Le manuel 1 indique la manière d'utiliser toutes les fonctions de la station de programmation pour développer et mettre en route les programmes de régulation.

Volume 2 - Langages PC3000

"Quels sont les utilitaires offerts par le PC3000 ?"

Ce guide utilisateur présente les langages de programmation et les concepts utilisés par le PC3000. Il donne également quelques exemples simples de langages.

Volume 3 - Applications PC3000

"Pourquoi dispose-t-on de certains utilitaires et comment les utilise-t-on ?"

Ce volume contient un certain nombre de présentations portant sur des sujets précis, en particulier :

Présentation des communications PC3000

- présente les utilitaires et les blocs fonctions spéciaux qui permettent au PC3000 d'échanger des informations de régulation et des données en temps réel avec d'autres PC3000 et d'autres matériels de marque comme des régulateurs programmables, les SCADA et les systèmes de supervision.

Présentation de la régulation PC3000

- décrit une partie des méthodes et techniques standard utilisées pour la régulation de process à l'aide des blocs fonctions PC3000 intégrés et programmables par l'utilisateur, en particulier avec les PID.

BASE DU PC3000

Outre les guides utilisateur, il existe un certain nombre de manuels de référence qui donnent des informations détaillées sur une large gamme de sujets. Ces manuels ne sont pas destinés à être lus dans leur intégralité mais sont structurés de manière à permettre de localiser rapidement des informations bien précises.

Les manuels de référence PC3000 comprennent :

Manuel du système d'exploitation temps réel PC3000 :

- donne une description détaillée du système d'exploitation temps réel PC3000 et des sujets qui y sont liés comme les multi-tâches, les performances, l'implantation mémoire et la détection des défauts.

Manuel d'Installation PC3000

- fournit des informations détaillées sur l'ensemble des modules matériels PC3000, en particulier des détails sur l'étalonnage, le câblage et la configuration physique.

Manuel des fonctions PC3000

- décrit toutes les fonctions qui peuvent être appelées dans le langage Texte structuré (ST).

Manuel des blocs fonctions PC3000

- décrit les multiples blocs fonctions disponibles incorporables dans votre programme de régulation pour la régulation PID, les rampes, les compteurs, les filtres, les timers, etc.

N.B. : en raison de notre politique d'amélioration permanente de nos produits et de l'information à leur sujet, la liste et la description des manuels concernant votre système peuvent différer légèrement des indications fournies.

INFORMATIONS RELATIVES A CE GUIDE

Nous vous invitons à lire ce guide avant de développer un programme de régulation PC3000. Il est possible d'optimiser les performances du PC3000 et de garantir l'intégrité du programme en respectant quelques principes simples. Ce guide est destiné à compléter le Volume 1 du Guide utilisateur du PC3000 (Programmation). Vous pouvez avoir besoin de vous reporter aux deux guides utilisateur lorsque vous développerez des programmes pour la première fois.

Le **chapitre 2** fournit une présentation des langages et des concepts du PC3000 et est utile lorsqu'on souhaite comprendre rapidement la manière dont on peut programmer le PC3000 pour résoudre les problèmes de régulation. Un exemple de programme simple est fourni et décrit une bonne partie de ces concepts.

Le **chapitre 3** donne une description détaillée du langage Texte structuré avec des exemples.

Le **chapitre 4** décrit la manière d'utiliser les GRAFCET PC3000 pour programmer tous les besoins de séquencement des systèmes de régulation simples et des systèmes de régulation complexes.

Le **chapitre 5** est particulièrement important. Nous vous conseillons de lire les parties suivantes de ce chapitre : Considérations sur la conception des programmes et Phases de développement des programmes avant de vous lancer dans un projet de programmation important.

TERMINOLOGIE ET ABREVIATIONS

Les abréviations suivantes sont employées dans ce guide utilisateur :

Actions	Instructions en Texte structuré, y compris les affectations qui sont évaluées lorsqu'un pas donné est actif.
Affectation	Instruction en Texte structuré qui produit une valeur écrite dans un paramètre de bloc fonction spécifié.
Expression boolénne	Expression en Texte structuré qui produit un résultat "vrai" ou "faux".
GRAFCET	Un GRAFCET est un ensemble de pas et de transitions reliés graphiquement (câblés) qui forment une ou plusieurs séquence(s). Il possède toujours un pas à début unique.
Expression	Elément en Texte structuré qui produit une valeur possédant un type de données spécifique.
Déclaration de bloc fonction	Bloc fonction qui a été créé pour être d'un type particulier et qui possède un nom unique.
Déclaration	Opération de création de blocs fonctions d'un type donné.
PID	Algorithme proportionnel, intégral et dérivé.
SFC	Désigne le langage et la construction des GRAFCET IEC.
ST	Désigne le langage évolué Texte structuré IEC
Instruction	Une instruction en langage Texte structuré est un ensemble de mots-clés, d'opérateurs et de fonctions de langage qui effectue une opération spécifique et se termine par un point virgule.
Pas	Pas dans le GRAFCET qui définit un état ou une phase d'un process particulier et qui est représenté par un encadré rectangulaire.
Transition	Ligne horizontale courte qui représente le point où se produit un changement (c'est-à-dire une transition) d'un ou plusieurs pas à un ou plusieurs autres pas. Représente un point de décision à un certain stade du process. Si la condition est remplie, la régulation passe du(des) pas actuel(s) au(x) pas suivant(s).
Condition de transition	Condition qui, lorsqu'elle est "vraie", provoque la transition entre les pas actifs. Dans le cas de l'utilisation de la Station de programmation PC3000, est définie comme une expression booléenne en Texte structuré. Lorsque la condition de transition est remplie, le traitement passe de l'ensemble de pas actuel à un nouvel ensemble de pas.

Chapitre 2

CONCEPTS DE PROGRAMMATION

Edition 1

Sommaire

PRESENTATION

Les concepts suivants de programmation du PC3000 sont décrits dans ce chapitre.

1. Conception des langages de programmation PC3000 et manière dont ils sont réalisés à partir de normes internationales.

2. Manière dont des programmes de pilotage complexes peuvent être réalisés par interconnexion d'"instruments logiciels" ou de blocs fonctions au moyen du logiciel.

3. But des blocs fonctions PC3000 et manière dont ils sont décrits.

4. Avantages présentés par l'utilisation du système multi-tâches du PC3000 qui permet à différentes parties du programme de pilotage de tourner à des fréquences de scrutation différentes.

5. Possibilités d'utiliser les blocs fonctions et les GRAFCET (SFC) pour qu'ils fonctionnent ensemble lors de la création d'un programme de régulation.

Eléments de base

Le PC3000 est un régulateur programmable de nouvelle génération, facile à configurer, qui convient à l'automatisation d'une vaste gamme de process industriels. Etant donné que le PC3000 est facile à configurer car il possède à la fois un matériel et un logiciel modulaires, il peut être appliqué aux process de production prototypes et principaux.

Les langages de programmation du PC3000 ont été conçus de manière à répondre à la norme IEC 1131-3 applicable aux régulateurs programmables. Ces langages conviennent pour programmer la large gamme d'applications PC3000. Ils vont de la régulation de fours pour le traitement thermique, la fabrication des câbles, le traitement des eaux et la fermentation jusqu'à des applications évoluées de "haute technologie" comme les presses de formage des super-plastiques pour les aubes de turbines d'avions et les systèmes épitaxiaux à rayons moléculaires. En outre, ces langages conviennent parfaitement pour créer des programmes de régulation avec des petites stations de programmation de type PC.

Un système type de régulation de process de production doit traiter toute une gamme de problèmes de pilotage différents qui comprennent :

Des **verrouillages** qui contrôlent les conditions dans lesquelles certaines activités ou certains process peuvent fonctionner. Par exemple, une soupape à vapeur ne peut pas être activée si des capteurs n'indiquent pas que la pression de vapeur est supérieure à un minimum donné et que le process nécessite de la vapeur.

Des **alarmes** qui sont déclenchées lorsque certaines conditions limites sont dépassées. Par exemple, un signal d'alarme est déclenché lorsque la température d'une cuve de process est supérieure à la température normale de service.

Une **régulation en boucle fermée** garantissant que les process tournent dans les conditions optimales. Par exemple, les boucles de PID peuvent servir à garantir le maintien de la température d'un four dans une bande acceptable, c'est-à-dire dans des limites haute et basse données au cours du chargement du four.

Un **séquencement** pour faciliter le démarrage et la fin des phases déterminantes d'un process dans des conditions bien définies. Par exemple, il faut retirer un lingot d'un four de traitement thermique lorsque le four a atteint ou dépassé une température fixée pendant une durée donnée.

Une **régulation en boucle longue** où les conditions à long terme sont suivies et des mesures sont prises pour optimiser le rendement du process. Par exemple, le rendement d'une pompe peut se dégrader progressivement à cause de l'usure de la roue, ce qui peut être détecté par la surveillance à long terme de la consommation moyenne de courant de la pompe avec des techniques de contrôle statistique. Le débit de la pompe peut être ensuite ajusté pour compenser l'usure.

Les langages adoptés pour le PC3000 vous permettent de décrire la totalité de ces aspects du programme de régulation d'une manière homogène et naturelle. Ces langages sont faciles à lire et à retenir et ont une forme compréhensible par des personnes ayant des degrés divers de connaissances informatiques.

NORME IEC 1131-3 PLC

La figure 2-1 décrit les composants principaux d'un programme PC3000.

La Commission Electrotechnique Internationale (IEC) a élaboré un ensemble de normes pour les régulateurs programmables, dénommé IEC 1131. La partie 3 de cette norme, appelée IEC 1131-3, traite des langages des régulateurs programmables et de la manière dont ces derniers font tourner les programmes de régulation. Bon nombre des concepts de cette norme sont utilisés à la fois dans le PC3000 et dans d'autres produits d'Eurotherm Automation, dont le "cell controller" du gestionnaire de production.

Il est possible de programmer le PC3000 à l'aide des langages suivants :

Texte structuré (ST) - langage évolué qui peut servir à rédiger des expressions analogiques et numériques complexes. Ce langage comprend une prise en charge des opérations artithmétiques complexes, des calculs portant sur des dates et des heures et des expressions conditionnelles utilisant des éléments comme SI, ALORS et OU (exclusif). Il existe aussi une bibliothèque complète qui offre des fonctions comme SQRT(), SIN(), MAX()

GRAFCET ou SFC - langage graphique qui offre une représentation schématique des séquences présentées sous forme d'une série de pas enchaînés et de transitions. Le GRAFCET répond à la norme, désormais bien acceptée, Grafcet mais avec quelques fonctions supplémentaires. Il offre une méthode très claire de définition et, en cours de fonctionnement, d'analyse du comportement du système de pilotage en montrant les états actifs du système dans le contexte de toutes les séquences possibles de remplacement et parallèles.

Diagramme de bloc fonction ou FBD - langage graphique qui permet l'interconnexion d'éléments de programme, appelés blocs fonctions, par simple traçage de traits sur l'écran, c'est-à-dire sous une forme analogue à la conception d'un schéma de circuits à l'écran à l'aide d'un système de CAO.

N.B. : la programmation FBD est uniquement possible sur la station de programmation Microcell.

Multi-tâches

Conformément à IEC 1131-3, le PC3000 permet l'organisation du programme de régulation en tâches qui fonctionnent à des fréquences d'exécution différentes. Normalement, le PC3000 possède deux tâches qui fonctionnent toutes les 10ms et toutes les 100ms mais il est possible de créer des tâches supplémentaires ou de modifier les fréquences d'exécution des tâches pour s'adapter aux exigences du process considéré. Une tâche est une partie d'un programme qui s'exécute périodiquement.

De nombreux régulateurs programmables possèdent une stratégie très simple dans laquelle tous les programmes de l'automate (normalement des expressions de schémas contacts) sont scrutés, c'est-à-dire exécutés, à une fréquence de scrutation fixe. Toutefois, avec le PC3000, un programme peut utiliser plus efficacement les ressources de régulation si différentes parties du programme s'exécutent à des fréquences de scrutation différentes, selon ce qui est dicté par les capacités de réponse de l'installation associée.

Par exemple, il peut être nécessaire de scruter très rapidement certaines entrées numériques liées à des verrouillages mécaniques très rapides alors que d'autres entrées numériques liées à des capteurs de température, par exemple, peuvent être scrutées à des fréquences inférieures. Normalement, les entrées analogiques sont scrutées plus lentement que les entrées numériques. Certaines parties de l'application peuvent également être nécessaires pour analyser les tendances à long terme. Dans ces cas, les tâches peuvent être configurées pour s'exécuter à des fréquences bien inférieures (toutes les 5 minutes ou plus, par exemple).

Fonctionnement déterministe

Contrairement à de nombreux régulateurs programmables, le PC3000 offre des tâches à fréquences de scrutation fixes. Par exemple, à condition d'observer certaines caractéristiques de fonctionnement, il est possible de construire un système de régulation dans lequel les blocs fonctions prévus pour tourner dans une tâche 100 ms s'exécutent toutes les 100 ms. Le fonctionnement déterministe est important pour que le comportement du système de régulation soit stable et prévisible.

Si un trop grand nombre de blocs fonctions sont affectés à une tâche, le PC3000 peut être débordé, c'est-à-dire que la durée de scrutation de la tâche doit être augmentée pour garantir le bon achèvement de la tâche. Cela est considéré comme une situation exceptionnelle qui nécessite la modification des affectations de tâches des blocs fonctions.

La Base du système d'exploitation temps réel PC3000 fournit des informations supplémentaires sur l'utilisation des tâches.

BLOCS FONCTIONS

En règle générale, les systèmes de régulation étaient formés par le câblage physique d'un certain nombre d'instruments discrets comme des régulateurs de température, des timers, des afficheurs, etc. Toutefois, les systèmes de ce type sont d'une installation coûteuse et d'une très grande rigidité.

Par opposition, le PC3000 permet de construire des programmes de régulation à partir d'"instruments logiciels" appelés blocs fonctions. Ce concept puissant, formalisé par la norme IEC 1131-3, permet de relier entre eux des composants logiciels éprouvés possédant des fonctions identiques à celles d'instruments réels pour former des systèmes de régulation complexes appelés **"câblage par soft"**.

Un bloc fonction contient un programme ou un algorithme "encapsulé" qui est accessible et pilotable extérieurement par un ensemble de paramètres. Il est "encapsulé" de telle manière que le concepteur n'a pas besoin de connaître la constitution interne du bloc fonction. En fait, le seul accès au bloc fonction peut s'effectuer par un ensemble de paramètres formels fournis par le concepteur initial.

Normalement, un bloc fonction possède un ensemble de paramètres d'entrée qui peuvent servir à le connecter (ou le câbler par soft) à d'autres blocs fonctions. Les paramètres d'entrée peuvent être régulés à partir de signaux de régulation réels émanant de l'usine ou peuvent se faire affecter des valeurs par des périphériques extérieurs comme des stations opérateur ou des systèmes de supervision.

Les paramètres d'entrée peuvent servir à modifier le comportement du bloc fonction. A chaque fois qu'un bloc fonction s'exécute, ce qui dépend de la tâche à laquelle il est associé, il est activé à l'aide des paramètres d'entrée en cours et d'autres données emmagasinées dans la mémoire interne. L'algorithme interne met ensuite à jour les paramètres de sortie.

Les blocs fonctions sont utilisés dans la régulation sous une certaine forme depuis des années. Par exemple, les algorithmes PID qui font tourner des instruments discrets comme les EPC 818 et 900 d'Eurotherm Automation se comportent comme des blocs fonctions. Le PC3000 possède une importante bibliothèque de blocs fonctions à partir de laquelle le concepteur peut élaborer des applications complexes. Cette bibliothèque comprend des blocs fonctions PID et Positionneurs de vannes avec réglage automatique ou adaptatif, des timers, des compteurs, des filtres et des bistables. Il existe aussi des blocs réservés à des applications spécifiques, par exemple pour construire des systèmes de gestion de recettes, pour communiquer avec d'autres périphériques de marques comme les automates programmables et pour le contrôle statistique en temps réel (SPC). Les blocs fonctions offrent également des possibilités d'utilisation immédiate et de développement de solutions inédites à des problèmes de process.

Exemple de bloc fonction

La figure 2-2 montre un exemple de bloc fonction Rampe utilisable toutes les fois qu'il faut produire une valeur qui augmente à vitesse constante. Le bloc fonction **Rampe** amène le paramètre de sortie (appelé **Sortie**) à la valeur du paramètre d'entrée **Point de consigne** à une vitesse déterminée par le paramètre d'entrée **Vitesse**.

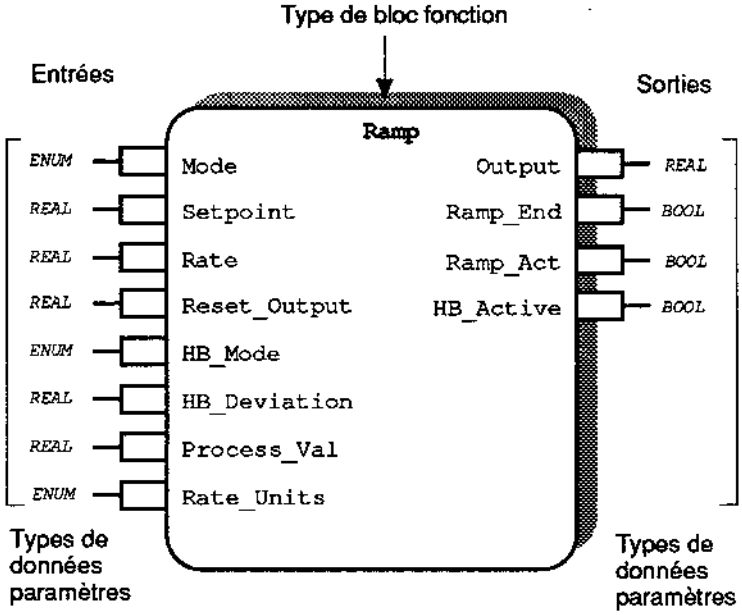


Figure 2-2 Exemple de bloc fonction

Le bloc fonction prend aussi en charge un maintien sur écart par lequel la Sortie peut suivre la valeur de process, c'est-à-dire le paramètre **Process_Val**, dans un écart donné. Cela peut par exemple être utile lorsque la température d'un four suit une rampe, quand la différence entre le point de consigne de la température du four et la température réelle doit se maintenir dans des limites données.

Chaque paramètre de bloc fonction est associé à un **type de données** particulier qui définit le type et la plage de valeurs que ce paramètre peut mémoriser. Une large gamme de types de données est utilisable, dont les paramètres REAL utilisés pour les valeurs décimales (à virgule flottante) comme 12,54, 0,0541, - 1220,1, les paramètres BOOL qui ont deux états comme MARCHE/ARRET, VRAI/FAUX, HAUT/BAS, les paramètres ENUM qui peuvent avoir un certain nombre d'états dénommés ou énumérés, par exemple le paramètre Mode du bloc fonction Rampe possède les états Reset, Run et Hold. D'autres types de données sont fournis pour conserver les comptes, les messages et les dates et heures.

Le chapitre 3 donne des détails complets sur les types de données.

Dans le PC3000, il est possible de créer de nombreuses copies du même type de bloc fonction. Par exemple, vous pouvez souhaiter trois blocs fonctions Rampe pour réguler trois variables de régulation différentes comme débit de la pompe, température et pression. Les copies peuvent recevoir des noms uniques, par exemple RampPump, RampTemp, RampPres . Les copies d'un type de bloc fonction donné sont appelées **Déclarations**. Lors de la création d'une copie nouvelle d'un bloc fonction, tous les paramètres d'entrée reçoivent un ensemble standard de valeurs par défaut. Dans de nombreux cas, ces valeurs peuvent rester inchangées lorsque les valeurs par défaut conviennent pour l'application ou si une certaine caractéristique du bloc fonction n'est pas nécessaire.

Toutes les valeurs affectées comme constantes aux paramètres d'entrée, par exemple l'attribution de la valeur 6,0 % à un paramètre PID Prop_Band, servent à remettre le bloc fonction à zéro lorsqu'il est initialisé, c'est-à-dire utilisé pour les valeurs **Démarrage à froid.**

Lorsque le PC3000 tourne, chaque copie ou déclaration de bloc fonction tourne de façon totalement indépendante de toute autre déclaration. Par exemple, RampPump peut être dans l'état Hold alors que RampTemp et RampPres sont en rampe vers des points de consigne différents.

Lors de l'exécution d'un bloc fonction, les valeurs changent, dans de nombreux cas, pour chaque exécution, même lorsque la valeur de tous les paramètres d'entrée du bloc fonction reste inchangée. Cela est dû au fait que le bloc fonction possède des variables mémorisées en interne qui servent à accumuler des valeurs pour les compteurs, les valeurs intégrées, etc. Par exemple, lorsque le bloc fonction Rampe est en mode Run, la Sortie augmente à chaque exécution ultérieure du bloc fonction, jusqu'à ce que le point de consigne de la rampe soit atteint bien que tous les paramètres d'entrée restent inchangés.

Blocs fonctions d'E/S

Pour des raisons d'homogénéité, la collecte des valeurs d'entrée provenant des capteurs et l'envoi des signaux de sortie aux actionneurs, radiateurs, etc. sont gérés par des blocs fonctions d'E/S (E/S). Par exemple, le bloc fonction d'entrée analogique **Analog_In** offre une gamme de paramètres d'entrée qui configurent le type de capteur et changent l'échelle de la valeur d'entrée pour qu'elle traite une large gamme de types de capteurs d'entrée incluant la plupart des types de thermocouples. La valeur du capteur d'entrée est fournie par le paramètre de sortie **Process_Val** du bloc fonction en unités de process comme les degrés Celsius, les millibars, etc. Il est à noter que ce bloc fonction offre également un utilitaire de test pour remplacer la valeur d'entrée actuelle par une valeur de test.

Chaque bloc fonction d'E/S est associé à un canal matériel d'E/S donné défini par un paramètre d'entrée appelé **IO_Address.** Ce paramètre définit l'emplacement physique du canal d'E/S par numéro de rack, numéro de module et canal.

Par exemple, un bloc fonction de canal d'E/S pour le deuxième canal d'un module matériel à l'emplacement 3 du rack 1 possède l'adresse d'E/S suivante : 1:03:02

Les adresses d'E/S sont automatiquement attribuées par la station de programmation PC3000 lorsqu'un canal est affecté. Il est impossible de modifier directement la valeur du paramètre de l'adresse d'E/S mais le bloc fonction du canal d'E/S peut être transféré vers un module matériel différent.

Pour avoir des informations supplémentaires sur les blocs fonctions Rampe et Analog_In, consulter le Manuel des blocs fonctions du PC3000.

PRINCIPES D'EXECUTION DES PROGRAMMES

Le programme utilisateur PC3000 pour une application de régulation donnée est composé d'un certain nombre de blocs fonctions interconnectés fournissant les possibilités de régulation continue du système et un ou plusieurs GRAFCET (SFC) pour définir les possibilités séquentielles. Les GRAFCET contiennent des actions qui peuvent modifier les valeurs des paramètres d'entrée des blocs fonctions en réponse à certains événements. Les GRAFCET peuvent par conséquent modifier le comportement du système en fonction de variations bien définies de conditions de process particulières comme le fait d'atteindre la température de service ou la vidange d'une cuve de réacteur.

Les blocs fonctions sont découpés en tâches qui s'exécutent à des fréquences de scrutation fixes. La configuration par défaut des tâches qui convient pour un grand nombre d'applications PC3000 possède deux tâches appelées Task_1 et Task_2, qui s'exécutent respectivement toutes les 10 ms et 100 ms. La plupart des blocs fonctions associés à une E/S analogique et à une régulation analogique sont activés toutes les 100 ms (blocs fonctions PID en particulier). Les autres, c'est-à-dire les blocs fonctions associés à des signaux numériques et nécessitant par conséquent une réponse système plus rapide, sont activés toutes les 10 ms. Les GRAFCET (SFC) par défaut sont activés dans la tâche la plus lente de 100 ms .

Nom de la tâche	Scrutation de la tâche	But
Task_2	toutes les 100 ms	Tous les blocs fonctions analogiques La plupart des blocs fonctions analogiques d'E/S Tous les GRAFCET
Task_1	toutes les 10 ms	Tous les blocs fonctions numériques Tous les blocs fonctions numériques d'E/S Les blocs fonctions analogiques rapides d'E/S

Table 2-1 Configuration par défaut des tâches

Toutefois, il est possible de modifier la configuration par défaut des tâches si le problème de régulation est assorti d'exigences spéciales. Il est possible d'affecter à tous les blocs fonctions, y compris ceux pour l'E/S, un fonctionnement à une autre fréquence de scrutation en modifiant les fréquences de scrutation des tâches ou en créant des tâches supplémentaires. Cela peut par exemple être nécessaire lorsqu'il faut augmenter la rapidité de réaction de certaines entrées en diminuant les fréquences de scrutation des tâches ou lorsqu'il est nécessaire de réduire les temps système en augmentant la fréquence de scrutation de certains blocs fonctions.

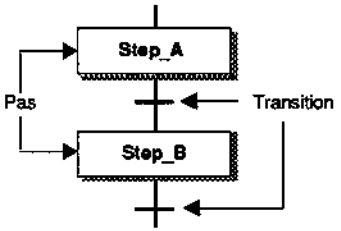
Attention :

Il faut prendre des précautions lorsqu'on modifie les paramètres de configuration des tâches ou les affectations des tâches car cela peut avoir des répercussions sur la rapidité de réaction du système ou entraîner des effets secondaires imprévus. La station de programmation PC3000 affecte automatiquement les blocs fonctions aux deux tâches par défaut. Par conséquent, la modification de la configuration des tâches n'est normalement pas nécessaire.

Se reporter à la Base du système d'exploitation temps réel PC3000, chapitre Programmateur de tâches temps réel, pour avoir des détails sur la configuration des tâches.

GRAFCET

Les GRAFCET (SFC) sont composés de deux éléments, les pas et les transitions, décrits sur la figure 2.3.



Un **pas** définit un ensemble d'actions effectuées lorsque le programme de régulation est dans un **état donné**. Un pas reste actif jusqu'à ce qu'une **transition** suivante devienne vraie. Une transition est définie par une condition de test qui doit donner un résultat vrai ou faux (exemple: une entrée numérique donnée provenant d'un microrupteur est sur "marche").

Figure 2-3 Eléments des GRAFCET

Sur la figure 2-3 , si Step_A est actif, Step_B deviendra actif et Step_A inactif lorsque la condition pour la Transition de Step_A à Step_B deviendra vraie.

Lorsque les GRAFCET sont évalués (normalement toutes les 100 ms par défaut), seules les conditions des transitions suivant les pas actifs sont testées. Dans un programme bien conçu, seul un petit nombre de pas sont simultanément actifs à tout moment. Par conséquent, le PC3000 peut exécuter des programmes complexes avec des GRAFCET possédant un grand nombre de pas et de transitions, sans que le temps système soit exagérément long.

Les actions dans les pas peuvent être définies en Texte structuré (ST) ou en termes de GRAFCET supplémentaires. Nous donnons ci-dessous des exemples d'instructions en texte structuré servant à définir des actions dans les pas :

```
loop1.Setpoint := 400.0;
tempRamp.Mode := 1 (* Run *);
vacpump.Process_Val := vacpump.Process_Val
          + 12.50 ;
```

Chaque transition est définie par une condition exprimée en Texte structuré, comme par exemple :

```
pumpswt.Process_Val = 1 (* On *) AND
      O2valve.Process_Val = 1 (* Open *);
```

Dans cet exemple, deux entrées numériques pumpswt et O2valve fournissent l'état de deux variables process numériques. Ainsi, la condition de transition peut se lire de la manière suivante : "interrupteur de la pompe en position marche et vanne d'oxygène ouverte".

Pour avoir des détails supplémentaires, cf. le chapitre 3 Texte structuré et le chapitre 4 Programmation des GRAFCET.

> **N.B. :** sur la station de programmation Microcell, on dispose de méthodes de programmation graphique de remplacement, en particulier des tableurs et des diagrammes de blocs fonctions.

PRESENTATION DU LOGICIEL

Un programme d'application pour PC3000 est composé de deux parties principales : 1) les blocs fonctions et leur câblage associé pour fournir les fonctions continues logiques et de régulation et 2) les GRAFCET pour traiter le séquencement.

La figure 2.1 montre la manière dont ces parties de programme interagissent avec le matériel du PC3000 qui fournit les interfaces avec les capteurs et actionneurs d'E/S et les communications avec les périphériques externes. Cette figure montre quelques exemples des multiples variétés de types de blocs fonctions fournis avec le PC3000.

Les interfaces avec l'I/O sont assurées par les blocs fonctions d'E/S comme Digital_In et Analog_Out. Le bus d'E/S PC3000 offre un échange d'informations bidirectionnel entre ces blocs fonctions dans le programme d'application PC3000 et les canaux d'E/S des modules matériels d'E/S.

La station de programmation PC3000 (ainsi que MicroCell) a accès à tous les paramètres de blocs fonctions lorsque le PC3000 tourne avec le protocole de communications Eurotherm Bisync. Cet utilitaire est appelé "communications par défaut" et est disponible en permanence pour les besoins de diagnostic et de développement de programmes.

Les communications avec les autres périphériques externes sont assurées par des blocs fonctions de drivers de communications comme Bisync_M et Euro_Panel. Sur la figure 2-1, le bloc fonction Bisync_M est associé à un port de communications relié à un certain nombre d'instruments à liaisons multipoints ; dans ce cas, c'est le protocole Eurotherm Bisync qui est utilisé. Cette figure montre aussi une console opérateur reliée à l'aide du bloc fonction Afficheur Eurotherm. On dispose également d'une vaste gamme de blocs fonctions pour d'autres protocoles comme JBus/Modbus. On peut ainsi relier au PC3000 un grand nombre de différents types de périphériques en communication.

Consulter le document Présentation des communications PC3000 pour avoir des informations supplémentaires sur les communications avec des périphériques extérieurs.

EXEMPLE DE PROGRAMME

La figure 2.4 montre un exemple simple de programme qui présente de nombreux concepts de programmation du PC3000. Ce programme utilise une boucle de régulation simple PID pour piloter un process thermique et modifie le point de consigne destiné à la boucle de régulation à l'aide d'un petit programme séquentiel.

Ce programme sert à présenter les techniques de programmation du PC3000 et est par conséquent délibérément simpliste mais il fonctionne.

Régulation continue

La boucle de régulation à canal unique est construite par "câblage par soft" d'un bloc fonction de canal d'entrée analogique (Temp1) à un bloc PID (Loop1) qui envoie un signal de sortie à un bloc fonction de canal de sortie analogique (Heat1).

Les paramètres de configuration nécessaires pour personnaliser les blocs de canaux d'entrée et de sortie afin qu'ils coïncident avec les caractéristiques des capteurs et des actionneurs et les paramètres de réglage du PID ne sont pas présentés sur la figure 2.4 pour plus de clarté.

Pour construire la partie "pilotage continu" du programme, il faut effectuer les principales actions suivantes :

1. Affecter les modules et canaux matériels pour l'entrée et la sortie analogiques à l'aide de l'écran de configuration matérielle (par exemple à l'aide des modules comme AI4 et AO4).

Dans cet exemple, l'entrée analogique est pilotée par le canal 1 du module 2 et la sortie pilote le canal 2 du module 4, tous les deux étant situés dans le rack 1.

2 Attribuer des valeurs aux paramètres de configuration pour personnaliser l'entrée et la sortie. Le canal d'entrée analogique a normalement besoin de valeurs pour les paramètres suivants :

Input_type, par exemple Range_3. Vérifier sur la fiche technique la plage acceptée par un module matériel donné. Dans le cas présent, Range_3 sélectionne le fonctionnement -10 à 50mV.

Lin_Type pour fixer le type de linéarisation, par exemple J pour un thermocouple de type J.

CJC_Type pour fixer le type de Compensation de soudure froide, par exemple intern pour interne.

Pre_Scaler, Pre_Offset , Post_Scaler, Post_Offset pour configurer les coefficients de telle manière qu'ils changent l'échelle du signal d'entrée et le décalent avant et après linéarisation. Ces paramètres sont utilisés en liaison avec **PV_Max** et **PV_Min** pour donner les unités physiques nécessaires (c'est-à-dire 0 à 100 degrés Celsius). Les valeurs 1,0 et 0,0 pour les coefficients et les décalages devraient convenir pour cet exemple. Par exemple, des valeurs de 9/5 pour Post_Scaler et 32 pour Post_Offset donneraient une valeur en degrés Fahrenheit. Des valeurs de 1 pour Post_Scaler et 0 pour Post_Offset donnent une valeur en degrés Celsius.

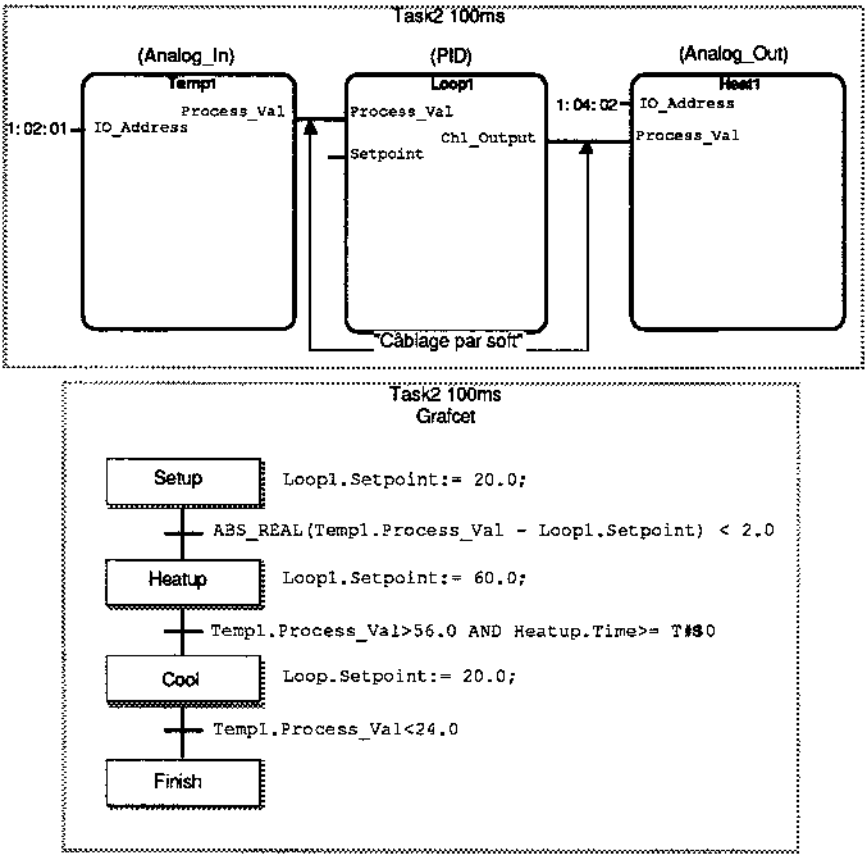


Figure 2-4 Exemple de programme simple

PV_Max, PV_Min permettent de définir la plage de valeurs acceptables pour l'entrée, par exemple 0 à 100.

> **N.B.** : si PV_Max est maintenu égal à 0, l'état de l'entrée analogique **Act_Status** peut être fixé sur NOGO parce que la valeur d'entrée est hors plage.

Le bloc fonction du canal de sortie analogique a besoin d'une valeur pour **Output_type** pour définir la plage de l'actionneur ou de l'entraînement de sortie, par exemple mA4_20 pour 4 à 20 mA.

3. Le bloc fonction PID a également besoin des paramètres de configuration suivants :

Span_High, Span_low : configurer ces paramètres pour qu'ils coïncident avec la plage de l'entrée analogique.

Output_High,Output_Low : veiller à ce que ces paramètres soient positionnés sur 100 et 0 pour cent. Si Output_Low est sur -100 pour cent, le PID fonctionne en mode canal double (chauffage/refroidissement).

Prop_Band, Integral et **Derivative** : configurer ces paramètres conformément à ce qu'exige le process régulé.

Manual : positionner ce paramètre sur Auto pour faire passer le PID en mode automatique.

4. Construire le "câblage par soft" pour relier les trois blocs fonctions, c'est-à-dire :

Loop1.Process_Val := Temp1.Process_Val;

et

Heat1.Process_Val := Loop1.Ch1_Output;

Lors de l'utilisation de la station de programmation PC3000, ces instructions sont reliées au paramètre d'entrée du bloc fonction pour recevoir la valeur (dans le cas présent, Loop1 et Heat1).

> **N.B.** : les instructions de câblage sont créées dans le paramètre DESTINATION.

On trouvera des détails supplémentaires sur la configuration matérielle dans le Manuel du matériel PC3000.

La partie "régulation continue" de cet exemple simple est maintenant terminée et, si l'on compile, construit et télécharge ce programme dans le PC3000, les trois blocs fonctions vont se comporter comme un simple régulateur à boucle unique. Il est possible de fixer le point de consigne de la boucle en modifiant le paramètre d'entrée du PID Loop1.Setpoint .

Séquencement

La boucle de régulation est séquencée par un petit GRAFCET comportant les pas Setup, Heatup, Cool et Finish. Le texte structuré pour chaque pas et chaque transition est représenté à côté du GRAFCET sur la figure 2-4. Le pas Setup établit un point de consigne initial de 20,0 et attend que la valeur de process se stabilise. Au cours du pas Heatup, le process est amené à un point de consigne supérieur de 60,0. Le pas Heatup continue jusqu'à ce que la valeur de process ait atteint une valeur seuil de 56,0 et ait été active pendant plus de 30 minutes. Le pas Cool est ensuite activé, ce qui ramène le point de consigne à 20,0. Lorsque la valeur de process est revenue à moins de 24,0, le GRAFCET prend fin en passant au pas Finish.

Pour programmer le GRAFCET à l'aide de la station de programmation, il faut effectuer les actions suivantes :

1. Utiliser l'éditeur de GRAFCET pour créer un GRAFCET à quatre pas dans le graphique principal. Le premier pas doit être défini comme un pas "début" et doit recevoir le nom Setup. Donner aussi les noms Heatup, Cool et Finish aux autres pas. Le pas Finish doit être défini comme un pas "fin".

2. Les trois pas doivent être définis comme des "pas dont les actions sont exécutées une fois", ce qui implique qu'ils s'exécuteront une seule fois lorsqu'ils seront entrés pour la première fois.

3. Saisir le texte structuré pour chaque pas et chaque transition comme le montre la figure 2-4. Il faut noter que la transition entre Start et Heat utilise la fonction ABS_REAL qui sert à calculer la valeur absolue du paramètre d'erreur PID.

La condition pour la deuxième transition utilise l'opérateur ET pour vérifier que :

a) la valeur de process est supérieure à 56,0 et

b) le pas Heatup a été actif pendant plus de 30 minutes.

La durée de n'importe quel pas actif est fournie par le paramètre de sortie **Time** (abrégé en .T) du pas.

Pour tester le programme, le compiler, le construire et le télécharger dans le PC3000. Avec le PC3000 commuté en mode RUNNING, le programme est exécuté par défilement séquentiel des pas du début à la fin qui fixe le point de consigne du PID. Il faut noter que, lorsque le pas Finish est atteint, le GRAFCET prend fin sans pas actif supplémentaire. Toutefois, l'exécution des blocs fonctions de la boucle de régulation se poursuit. Avec cet exemple simple, il est nécessaire de REINITIALISER le PC3000 puis de le ramener en mode RUNNING (MARCHE) pour relancer le GRAFCET. Il est possible de construire le GRAFCET de manière à ce qu'il redémarre en permanence, ce qui est traité dans le chapitre 4 Programmation des GRAFCET.

Le mode PC3000 est affiché sur le menu principal de la station de programmation PC3000 lorsqu'il est en mode en ligne. Utiliser la commande RUN (MARCHE) pour faire passer le PC3000 en mode RUNNING.

Tous les blocs fonctions de cet exemple sont associés à la régulation analogique et s'exécutent dans Task2 qui, par défaut, est scrutée toutes les 100 ms. Le GRAFCET s'exécute aussi dans la même tâche mais il ne faut pas oublier que seul le pas actif et ses conditions de transition sont évalués toutes les 100 ms.

Pour avoir des détails système supplémentaires sur la synchronisation et l'exécution des GRAFCET et des blocs fonctions, se reporter au Manuel du système d'exploitation temps réel PC3000.

Logique numérique

Pour compléter cet exemple, la figure 2-5 montre la manière dont une condition d'alarme simple peut être ajoutée par l'utilisation des blocs fonctions numériques. Ces blocs sont interconnectés par câblage par soft de la même manière que les blocs analogiques. L'exemple suppose qu'un signal d'alarme est nécessaire pour couper l'alimentation électrique du process lorsqu'une des deux portes (Door1 et Door2) est ouverte et lorsque la température du process est supérieure à 50,0.

Les blocs fonctions des canaux d'entrée pour les entrées numériques Door1 et Door2 déterminent l'état des portes, par exemple, en reliant les entrées matérielles numériques aux microrupteurs. Le signal d'alarme de sortie est piloté par un bloc fonction de sortie numérique, Alarm.

Le paramètre d'entrée Process_Val d'Alarm est "câblé par soft" à l'aide d'une expression en texte structuré qui implique à la fois la valeur de process du bloc fonction d'entrée numérique et le bloc fonction Temp1. Le câblage par soft peut soit fournir des liaisons directes point à point utilisées pour construire la boucle de régulation soit comporter des expressions complexes avec des paramètres de blocs fonctions aussi bien numériques qu'analogiques.

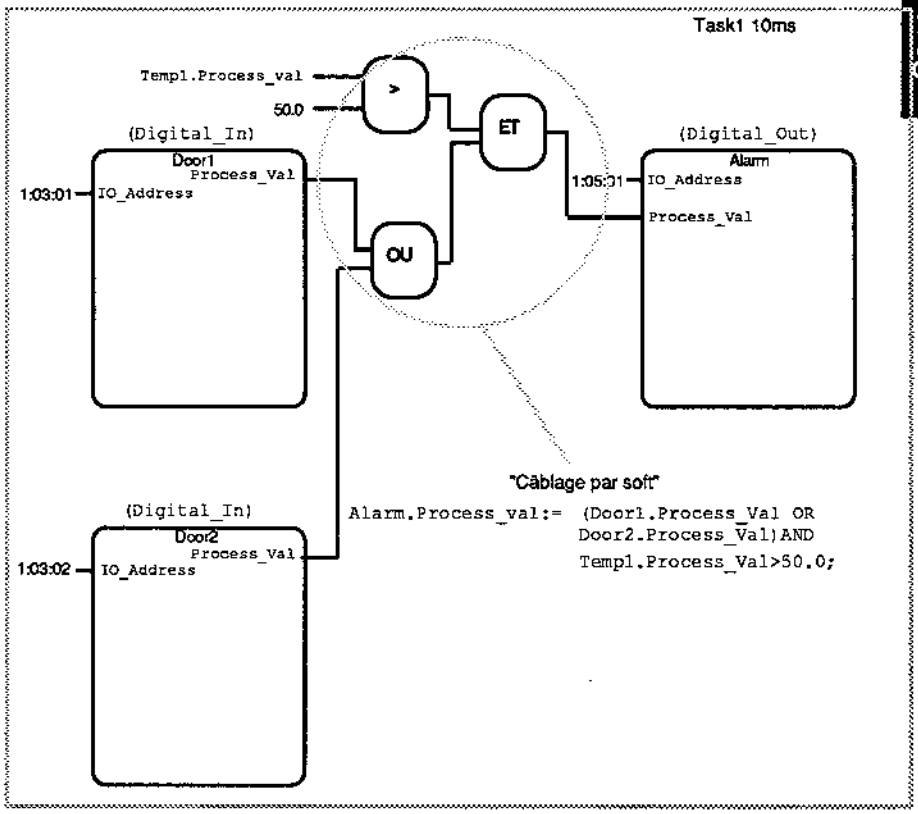


Figure 2-5 Exemple de logique numérique

Pour créer ces blocs fonctions et le câblage par soft, il faut effectuer les actions suivantes :

1. Créer deux blocs fonctions de canaux d'entrée numérique d'E/S appelés Door1 et Door2 à l'aide d'un module matériel comme DI14_CON. Aucun paramètre de configuration de canal n'est à configurer. S'assurer que le paramètre **Test_Enable** de chaque bloc est sur Off (valeur par défaut).

2. Créer un bloc fonction de canal de sortie numérique appelé Alarm à l'aide d'un module matériel comme DO12_RLY. S'assurer que le paramètre **Test_Enable** est sur Off (valeur par défaut).

3. Sélectionner le paramètre **Alarm.Process_Val** et lui annexer l'instruction en texte structuré de câblage par soft, comme l'indique la figure 2-5 .

Les blocs fonctions numériques sont affectés à Task 1 qui est scrutée toutes les 10 ms. Le câblage par soft relié à la valeur de process du bloc fonction de sortie Alarm est lui aussi évalué toutes les 10 ms, c'est-à-dire comme le bloc fonction Alarm.

L'exemple de programme est maintenant complet et peut être compilé, construit et téléchargé dans le PC3000. La boucle de régulation et la logique numérique d'alarme sont exécutées en continu lorsque le PC3000 est en mode RUNNING (MARCHE). Le séquencement défile les quatre pas en commençant par Setup. Le tableau 2-2 montre l'ordre d'exécution qui est créé automatiquement par la station de programmation pour faire exécuter le programme. Dans la plupart des cas, il n'est pas nécessaire de connaître l'ordre d'exécution dans le détail.

Toutes les 10 millisecondes -
Lecture des canaux matériels d'entrée numérique Door1 et Door2 Exécution des blocs fonctions des canaux d'E/S et du câblage par soft de la logique numérique de l'alarme. Ecriture de la valeur de sortie d'Alarm vers le canal matériel numérique.
Toutes les 100 millisecondes -
Lecture du canal matériel d'entrée analogique Temp1 Exécution des blocs fonctions Temp1, Loop1 et Heat1 et du câblage par soft formant la boucle de régulation. Exécution des pas acifs des GRAFCET et évaluation des conditions de transition suivant les pas actifs. Ecriture de la valeur de sortie vers le canal matériel analogique.

Table 2-2 Ordre d'exécution de l'exemple de programme

Remarques sur les performances du câblage par soft

Lors de la conception de gros programmes PC3000, il faut voir qu'un grand nombre d'instructions de câblage par soft peut provoquer une augmentation significative du temps système.

> **N.B. :** chaque instruction en texte structuré de câblage par soft est exécutée à la même fréquence et dans la même tâche que le bloc fonction qui reçoit la valeur produite par le câblage par soft.

Il faut par conséquent prendre soin de minimiser le nombre et la complexité des instructions de câblage par soft affectées aux blocs fonctions possédant une fréquence de scrutation élevée, en particulier les blocs fonctions numériques. Le câblage par soft qui contient un grand nombre d'opérations comportant des valeurs à virgule flottante, comme la comparaison de Temp1.Process_val avec 50,0 utilisée dans l'exemple de programme précédent, entraîne une augmentation importante du temps système. Lorsque les programmes sont bien conçus, il est possible de diminuer le nombre de calculs avec virgule flottante.

Par contraste, un grand nombre d'opérations comportant des paramètres numériques, comme l'OR de door1.Process_val et door2.Process_val dans cet exemple, peuvent être exécutés sans augmentation significative du temps système.

Le chapitre 5 Développement des programmes donne des indications supplémentaires sur la conception des programmes. L'annexe C de la base du système d'exploitation temps réel PC3000 donne également des informations détaillées sur les performances.

Chapitre 3

TEXTE STRUCTURE

Edition 1

Sommaire

PRESENTATION

Ce chapitre traite les points suivants :

- Fonctions du langage Texte structuré
- Endroits où il est possible d'utiliser le Texte structuré dans les programmes PC3000
- Manière dont il est possible de traiter des données de types différents en Texte structuré
- Manière d'utiliser les fonctions en Texte structuré
- Style de programmation adapté au Texte structuré

Introduction au texte structuré

Le Texte structuré (abrégé en ST) est un langage texte évolué qui a été formalisé par la norme IEC 1131-3 PLC relative aux langages de programmation et qui est destiné à être utilisé dans les régulateurs programmables pour les process de fabrication industrielle. Le Texte structuré offre une vaste gamme de fonctions qui le rendent particulièrement bien adapté à la programmation de la régulation des process industriels :

- Toutes les variables utilisées dans ce langage peuvent recevoir des noms ayant une signification, comme :

  pumpRate.Setpoint

  heat1.Output_Type

- Il est possible d'exprimer de manière claire des expressions simples ou complexes pour les calculs mathématiques, comme par exemple :

  FlowRate.Val := FanSpeed.Process_Val * 3.5;

- Les expressions conditionnelles, c'est-àdire pour les valeurs qui sont dérivées des tests sous une forme ou une autre, sont bien acceptées. Par exemple, pour faire face à des impératifs comme "La vitesse du moteur est de 200 tr/min lorsque la pièce est à moins de 5 mètres de la tête de l'outil, dans le cas contraire, la vitesse est de 20 tr/min" peut s'exprimer de la manière suivante :

```
IF Position.Val < 5.0 THEN
        Motor.Process_Val := 200;
ELSE
        Motor.Process_Val := 20.0;
END_IF;
```

- Il est possible d'utiliser une vaste gamme de différents types de données, dont les valeurs à virgule flottante et les valeurs numériques, les durées, la date et l'heure. Il existe aussi des possibilités de traiter les messages texte.
- Il est possible d'avoir des expressions comportant des types différents de données, comme par exemple :

```
gas.Process_Val := Zone1.Process_Val > 310.4
```

- Il existe une vaste gamme d'opérateurs intégrés standard :

```
+,-,*,>,<,AND,OR,NOT
```

- Le langage possède des sécurités intégrées pour empêcher les opérations illogiques entre des types de données différents. Par exemple, il est impossible d'utiliser les opérations mathématiques avec les valeurs d'entrée numériques ou les opérateurs logiques comme ET avec les valeurs analogiques.
- Les fonctions peuvent servir à simplifier des expressions de programmes complexes comme par exemple :

```
position.Val := SQRT(X.Val*X.Val + Y.Val*Y.Val);
```

position.Val := SQRT(X.Val*X.Val + Y.Val*Y.Val);

En règle générale, les instructions relativement simples en Texte structuré conviennent pour programmer la majorité des applications de régulation PC3000. Toutefois, la station de programmation PC3000 ne permet pas la création de blocs volumineux et complexes en Texte structuré. La taille maximale d'un bloc isolé de Texte structuré est limitée mais cette taille est suffisamment élevée pour que cette restriction n'ait aucune signification pratique.

Pour la programmation du PC3000, le Texte structuré sert à décrire :

a) le "câblage par soft" qui assure l'interconnexion des blocs fonctions

b) les actions dans les pas des GRAFCET

c) les conditions des transitions des GRAFCET.

Certaines restrictions s'appliquent au Texte structuré lorsqu'il est destiné à des utilisations différentes ; ces restrictions sont traitées plus en détail dans la suite de ce chapitre.

Commentaires

Pour faciliter la lecture, il est possible d'ajouter des commentaires à volonté aux instructions en Texte structuré en plaçant le texte entre étoiles (* et *). Exemples :

(* Ce pas provoque le démarrage du réacteur 2 *)

(* Modifié pour optimiser l'utilisation de l'énergie *)

TYPES DE DONNEES

Le PC3000 prend en charge toute une série de types de données dans le Texte structuré qui couvrent la plupart des besoins de traitement pour la régulation de process de production. L'écriture d'expressions en Texte structuré qui utilisent ces différents types de données est traitée dans des parties ultérieures de ce chapitre.

N.B. : dans tout ce guide utilisateur, où une description d'un type de données quelconque est fournie, le nom du type de données normalisé IEC est parfois indiqué entre parenthèses.

Virgule flottante (REAL)

Plage	Taille en bits	But
$\pm 10^{\pm 38}$	32	Type de données universel pour toutes les valeurs à virgule flottante (décimales)

Ce type de données sert à mémoriser toutes les valeurs analogiques à virgule flottante, à la fois négatives et positives, et les valeurs fractionnelles très grandes ou très petites, par exemple 100,56, 100245,21, -0,000233

Entier (DINT)

Plage	Taille en bits	But
-2147483648 à +2147483647	32	Type de données universel pour stocker les valeurs entières, par exemple pour les compteurs.

Sert à stocker les valeurs entières, c'est-à-dire les nombres entiers, et est utilisé pour les comptages, les numéros de lots, etc. Il est possible de stocker une gamme étendue de valeurs positives et négatives, par exemple 12, 1235687, - 100040.

Entier énuméré (DINT)

Plage	Taille en bits	But
-2147483648 à +2147483647	32	Type de données universel pour mémoriser les valeurs entières qui possèdent un ensemble défini de valeurs ayant un nom, par exemple pour les modes ou l'état.

Ce type de données (parfois abrégé en ENUM) utilise la même taille mémoire que l'entier (DINT) mais avec des noms associés à un ensemble défini de valeurs. Ces noms sont affichés sur la station de programmation PC3000 et sont fournis pour faciliter la lecture.

Par exemple, le bloc fonction PcsSTATE qui définit l'état actuel du système de régulation PC3000 possède un paramètre de sortie Battery_Cond (état de la batterie) qui est un entier énuméré. Ce dernier possède des états qui portent un nom :

Bon (0)
Faible (1)
Défectueux (2)

Le nom qui convient est affiché sur la station de programmation PC3000 en même temps que la valeur du paramètre.

Les entiers énumérés peuvent être utilisés à volonté avec les entiers normaux dans les expressions en Texte structuré.

Booléen (BOOL)

Plage	Taille en bits	But
0 ou 1	1	Type de données universel pour stocker les valeurs booléennes, c'est-à-dire 0 ou 1.

Ce type de données sert à mémoriser les valeurs booléennes du genre de celles qui sont associées aux entrées numériques pour les manocontacts, qui peuvent avoir deux états : 0 ou 1. En règle générale, 0 est associé à "Off" (arrêt) ou "False" (faux) et 1 est associé à "On" (marche) ou "True" (vrai). Toutefois, la station de programmation PC3000 permet l'attribution de noms ayant un sens différent si besoin est. Les noms ayant un sens servent à améliorer la lisibilité du programme.

Exemples :

Down (bas) (0)
Up (haut) (1)

In (intérieur) (0)
Out (extérieur) (1)

Durée (TIME)

Plage	Taille en bits	But
Jusqu'à 49 jours	32	Utilisé spécifiquement pour mémoriser les durées temporelles comme la durée des tâches ou les constantes temporelles PID

Ce type de données sert à mémoriser des durées temporelles avec une précision de l'ordre de la milliseconde. Il est possible de stocker n'importe quelle durée jusqu'à 49 jours. L'affichage des durées sur la station de programmation se fait dans la présentation IEC. Exemples :

T#2d_01h_30m (* 2 jours, 1 heure, 30 minutes *)

T#3s200ms (* 3 secondes et 200 millisecondes *)

L'emploi du préfixe T# en Texte structuré indique qu'il s'agit de constantes de type TIME (DUREE). Ce préfixe est automatiquement inséré par la station de programmation.

Heure (TIME_OF_DAY)

Plage	Taille en bits	But
00:00:00 à 23:59:59	32	Utilisé spécifiquement pour mémoriser les valeurs heure du jour (sur 24 heures).

Ce type de données est utilisé toutes les fois qu'il faut mémoriser l'heure, par exemple pour définir l'heure du jour pour démarrer une tâche donnée. Les valeurs sont mémorisées avec une précision de l'ordre de la seconde et affichées sur la station de programmation sous la forme "24 heures".

Exemples :

TOD#09:30 (* 9:30 du matin *)

TOD#13:45 (* 1.45 de l'après-midi *)

L'utilisation du préfixe TOD# dans le texte structuré indique qu'il s'agit de constantes TIME_OF_DAY. Ce préfixe est automatiquement inséré par la station de programmation.

Date calendaire (DATE)

Plage	Taille en bits	But
01-jan-1970 au 01-jan-2136	32	Utilisé spécifiquement pour mémoriser les valeurs pour les dates calendaires.

Ce type de données sert à mémoriser les dates calendaires. Une large plage de dates est acceptée.

Exemples :

D#28-Jul-1992

D#01-Jan-1993

L'utilisation du préfixe D# dans le texte structuré indique qu'il s'agit de constantes de date. Ce préfixe est automatiquement inséré par la station de programmation.

Date calendaire et heure du jour (DATE_AND_TIME)

Plage	Taille en bits	But
01-jan-1970-00:00:00 au 01-jan-2136-23:59:59	32	Utilisé spécifiquement pour mémoriser les valeurs de dates calendaires combinées à l'heure du jour.

Ce type de données sert à mémoriser les dates calendaires avec l'heure du jour ; on l'utilise généralement pour mémoriser les "horodatages" des événements clés comme le moment du démarrage des tâches, du déclenchement des alarmes, des changements de service des opérateurs, etc.

Exemples :

DT#02-Sep-1992-20:30:00

DT#25-Jan-1991-23:00:30

L'utilisation du préfixe DT# dans le texte structuré indique qu'il s'agit de constantes de date DATE_AND_TIME. Ce préfixe est automatiquement inséré par la station de programmation.

Chaînes texte (STRING)

Plage	Taille	But
Chaque caractère peut contenir n'importe quel caractère du jeu de codes ASCII. Les codes autres qu'ASCII (valeurs hexadécimales $80 à $FF) sont également pris en charge.	La longueur d'une chaîne texte est variable - cf. la description	Type de données universel utilisé pour mémoriser les informations texte composées d'une chaîne de caractères.

Ce type de données sert à contenir les informations texte comme les messages opérateur, les messages de rapports d'imprimante, les adresses de communication, les descriptions de lots, etc. La longueur de la chaîne, c'est-à-dire le nombre maximal de lettres et de chiffres (caractères), qui peut être mémorisée dépend de l'utilisation.

Par exemple, le bloc fonction String User Variable peut mémoriser des chaînes texte d'une longueur maximale de 80 caractères, alors que le bloc fonction Long_String User Variable est utilisable pour les chaînes d'une longueur maximale de 255 caractères.

Pour avoir des informations supplémentaires sur les blocs fonctions Variables utilisateur, se reporter au Manuel des blocs fonctions PC3000.

Exemples de chaînes texte :

'Batch 723XA' (* Current Batch Identity *)

'0A01 PV' (* Address Port 0A, instrument 1 PV *)

'ADD REAGENT X1' (* Operator Message *)

Les chaînes en Texte structuré figurent entre apostrophes. Ces caractères sont insérés automatiquement par la station de programmation lorsqu'une constante chaîne est créée.

Il est possible d'insérer un caractère non imprimable dans une chaîne en tapant un signe dollar ($) suivi du code ASCII en hexadécimal ; par exemple, pour insérer le caractère d'appel qui produira un son sur certaines stations opérateur, taper $07. Le code ASCII complet est donné dans le Manuel des blocs fonctions PC3000. Des caractères non imprimables peuvent être nécessaires pour structurer les messages texte afin qu'ils puissent être sortis sur imprimante.

Types de données entières supplémentaires

La majorité des entiers utilisés dans les blocs fonctions et fonctions PC3000 sont de type DINT. Toutefois, dans quelques rares cas, il est possible de définir les paramètres à l'aide des types de données entières énumérées dans le tableau 3-1.

Type de données IEC	Description	Taille en bits	Plage
SINT	Entier court avec signe	8	-128 à 127
USINT	Entier court sans signe	8	0 à 255
INT	Entier avec signe	16	32768 à -32767
UDINT	Entier double sans signe	32	0 à 42944967296

Tableau 3-1 Types de données entières supplémentaires pour les paramètres des blocs fonctions

Dans le tableau 3-1, les types de données décrits comme "sans signe" impliquent qu'il est uniquement possible de mémoriser des valeurs positives.

Attention

Il faut prendre des précautions lorsqu'on affecte des valeurs entières créées par des expressions en Texte stucturé à des paramètres entiers qui ne sont pas du type normal DINT. La plage de la valeur entière créée par le Texte structuré doit se situer dans la plage du type de données du paramètre qui doit recevoir la valeur. Si cette valeur est hors plage, une valeur incorrecte risque d'être affectée, par exemple le fait d'affecter la valeur DINT 300 à un USINT échouera car la valeur maximale pour USINT est 255.

N.B. : ces types de données entières supplémentaires ne sont utilisés dans aucun des blocs fonctions et fonctions standard PC3000, à l'exception des fonctions de chaîne LEN, LEFT, RIGHT,MID, INSERT, DELETE, REPLACE, FIND, JUSTIFY_LEFT, JUSTIFY_RIGHT, JUSTIFY_CENTRE qui possèdent certains paramètres entiers USINT.

La description des fonctions et des blocs fonctions dans les Manuels des fonctions et blocs fonctions PC3000 inclut le type de données de chaque paramètre.

Adresse d'E/S

Le type de données Adresse d'E/S sert à mémoriser la position physique d'un bloc fonction de canal d'E/S. Il s'agit d'un type de données PC3000 spécial qui n'est pas défini par la norme IEC. Il est impossible de manipuler les paramètres de ce type en Texte structuré car cela n'aurait aucun intérêt pratique.

N.B. : les blocs fonctions de canaux d'E/S peuvent être affectés à différentes adresses physiques à l'aide des utilitaires offerts par la station de programmation sur l'écran Définition matérielle.

Les paramètres d'adresses d'E/S font l'objet d'une discussion dans le chapitre 2, Concepts de programmation PC3000.

Conversion de types de données

Il existe une vaste gamme de fonctions qui permettent de convertir les différents types de données entre eux ; cf. le Manuel des fonctions du PC3000, chapitre Fonctions de conversion des types, pour avoir la liste complète.

Par exemple, un comptage mémorisé sous la forme d'un entier peut être nécessaire dans une expression pour le calcul de la valeur d'une sortie analogique. Dans ce cas, il est possible d'utiliser la fonction DINT_TO_REAL pour convertir la valeur entière en valeur à virgule flottante (REAL), c'est-à-dire

```
height.Process_val :=
        DINT_TO_REAL(count) * 100.5;
```

ENDROITS OU IL FAUT UTILISER LE TEXTE STRUCTURE

Il est possible d'utiliser le Texte structuré pour trois emplois différents dans un programme PC3000 mais, dans chaque cas, la structure de base du langage est identique.

Emploi	But	Eléments de langage
Câblage par soft	Interconnexion des blocs fonctions	Affectations Expressions
Actions de pas	Affectation de valeurs nouvelles aux paramètres des blocs fonctions	Affectations Expressions Instructions Instructions conditionnelles
Conditions de transition	Définition d'une condition qui, lorsqu'elle est "vraie", provoque l'activation d'un ou plusieurs pas nouveau(x)	Expressions

Table 3-2 Utilisation du texte structuré

Le Texte structuré offre quelques éléments simples et faciles à retenir (affectations, instructions, expressions et instructions conditionnelles) qui, lorsqu'ils sont utilisés ensemble, donnent un langage souple et expressif. Pour réaliser des programmes PC3000 efficaces, vous avez intérêt à vous familiariser avec les éléments de langage suivants.

AFFECTATIONS

Les affectations en Texte structuré permettent l'écriture de nouvelles valeurs dans les paramètres d'entrée des blocs fonctions. Ces valeurs peuvent être des constantes ou la valeur d'autres paramètres ou bien peuvent être dérivées d'autres paramètres à l'aide d'expressions. Les affectations servent à interconnecter les blocs fonctions, c'est-à-dire à les câbler par soft, et à définir de nouvelles valeurs de paramètres dans les pas des GRAFCET.

Exemples :

```
loop1.Setpoint := 30.5;
count1.Process_Val := 300;
pulse3.Prog_Time := T#1s;
heat1.Process_Val := loop1.Output;
speed.Process_Val := rate.Process_Val * 20;
```

Une affectation commence toujours par le nom du paramètre d'entrée du bloc fonction à gauche suivi du symbole ":=".

La valeur à droite (c'est-à-dire située après le symbole ":=") doit donner une valeur possédant le même type de données que le paramètre d'entrée du bloc fonction. Le texte de l'affectation se termine toujours par un point-virgule ";".

Lors de la création d'affectations par "câblage par soft", la station de programmation insère automatiquement les symboles ":=" et ";".

> **N.B.** : la station de programmation indique "Erreur Texte structuré incorrect" si l'on effectue une affectation à un paramètre de sortie de bloc fonction. Les affectations aux paramètres d'entrée/sortie sont autorisées.

Lorsqu'on utilise une affectation pour inter-connecter ou "câbler par soft" des blocs fonctions comme élément de la stratégie de régulation continue, il faut penser que l'affectation sera évaluée en continu, c'est-à-dire à la fréquence de scrutation de tâches du bloc fonction auquel l'affectation est effectuée.

Expressions

Il est possible de créer une vaste gamme d'expressions à l'aide des opérateurs Texte structuré standard, afin que les valeurs dérivées puissent être calculées à partir des paramètres des différents types de données ST. Les expressions sont utilisées à droite des affectations et dans les expressions conditionnelles. Une expression simple type comporte un ou deux paramètres ou constantes et un opérateur.

Exemples d'expressions simples :

```
123 + 34.6
10000 * count.Process_Val - loop1.Setpoint
NOT switch.Process_Val
```

Exemples d'expressions simples dans les affectations :

```
(* Add 120.0 degrees to the current soak temperature *)
soakTemp.Setpoint := soakTemp.Setpoint + 120.0;

(* The line rate is 2 M/S plus 50% the conveyor speed *)
lineRate.Process_Val := 2.0 +
                conveyor.Process_Val * 0.5;

(*Valve is ON if O2 supply is OFF and H2 is ON*)
valve.Process_Val := NOT O2.Process_Val AND
                                        H2.Process_Val;
```


Expressions complexes

Dans certains cas, il peut être nécessaire de construire des expressions complexes comportant de nombreux paramètres et opérateurs. Une expression complexe est composée de nombreuses sous-expressions qui peuvent elles-mêmes être composées d'autres sous-expressions ou d'expressions simples. Pour clarifier l'ordre d'exécution des expressions complexes, il est possible d'utiliser des parenthèses "(", ")" pour identifier les sous-expressions.

Exemples :

```
(( airPress.Val * 13.54 )
            + ( gasPress.Val * 34.32 )
            - ( vapPress.Val * 0.5 ))

(switch.Process_Val AND dig3.Process_Val)
        OR
        (overpress.Process_Val AND          NOT Alarm.Val)
```

INSTRUCTIONS

Une section de Texte structuré est composée d'un certain nombre d'instructions. Une affectation est un exemple d'instruction simple en Texte structuré. Les instructions conditionnelles sont plus complexes et permettent l'exécution sélective d'ensembles d'instructions en Texte structuré, de la manière décrite dans la section suivante.

Dans un pas de GRAFCET, il est possible d'utiliser une grande variété d'instructions pour configurer les valeurs des paramètres d'entrée des blocs fonctions selon les besoins de la phase ou de l'état du process considéré.

Exemple :

```
BEGIN
        heater1.Process_Val := 123.0;
        heater2.Process_Val := 130.0;
        fan.Process_Val := 1 (* ON *);
END
```

N.B. : les mots-clés BEGIN et END sont ajoutés automatiquement par la station de programmation lors de la création de Texte structuré pour un pas. Ils indiquent que le Texte structuré définit les actions des pas.

Instructions conditionnelles

Il est possible d'évaluer un ensemble d'instructions de manière conditionnelle en utilisant une structure d'instructions conditionnelles. Le Texte structuré PC3000 fournit les mots-clés IF, THEN, ELSIF, ELSE et END_IF pour permettre la sélection conditionnelle des instructions en fonction de certains critères de régulation.

Dans la forme la plus simple, il est possible d'utiliser IF, THEN et END_IF pour sélectionner un ensemble d'instructions de manière conditionnelle.

Exemple:

```
IF switch.Process_Val = 1 (* COOL *) THEN
        fan.Process_Val := 1 (* ON *);
        fanSpeed.Process_Val := 230.0;
END_IF;
```

L'expression entre les mots-clés IF et THEN peut être soit simple soit complexe mais doit donner un résultat booléen (BOOL), c'est-à-dire que l'expression doit donner un résultat qui ne peut être que "vrai" (1) ou "faux" (0). Cette expression est appelée expression booléenne.

Dans l'exemple, les affectations à fan.Process_Val et fanSpeed.Process_Val ne peuvent être effectuées que lorsque switch.Process_Val a la valeur 1, dans le cas contraire les affectations ne sont pas prises en compte. Tout le Texte structuré entre IF et END_IF est appelé instruction conditionnelle IF et, comme les autres instructions, il doit se terminer par un point-virgule ";".

Il est possible d'évaluer des instructions de remplacement à l'aide du mot-clé ELSE.

Exemple:

```
IF switch.Process_Val THEN
        gasFlow.Process_Val := 100.0;
        almEnable.Val := 1 (* ON *);
ELSE
        gasFlow.Process_Val := 30.0;
        almEnable.Val := 0 (* OFF *);
        report.Val := 'STANDBY';
END_IF;
```

Dans cet exemple, les affectations à gasFlow.Process_Val, almEnable.Val et report.Val sont uniquement évaluées si switch.Process_Val est sur off (arrêt).

Etant donné que switch.Process_Val produit une valeur booléenne, elle peut être utilisée comme expression conditionnelle dans l'élément IF...THEN.

L'élément ELSIF ... THEN peut servir à sélectionner un ou plusieurs autres ensembles d'instructions de remplacement.

Exemple:

```
IF count = 0 THEN
        speed.Process_Val := 20.0;
        door1.Process_Val := 0 (* SHUT *);
ELSIF count < 10 THEN
        speed.Process_Val := 40.0;
        door2.Process_Val := 0 (* SHUT *);
ELSE
        speed.Process_Val := 60.0;
        door3.Process_Val := 0 (* SHUT *);
END_IF;
```

Dans ce cas, si le comptage est égal à zéro, le premier ensemble d'affectations est effectué, si le comptage est inférieur à 10 mais n'est pas nul, le deuxième ensemble d'affectations est effectué. Dans le cas contraire, le dernier ensemble d'affectations est appliqué.

Lorsqu'un élément IF... THEN est suivi d'éléments ELSIF ... THEN multiples, seul l'ensemble d'instructions associé à la première expression conditionnelle vraie est évalué.

> **N.B. :** il est bon de s'assurer que les éléments ELSIF multiples ont des conditions qui s'excluent mutuellement pour faciliter la lecture des programmes.

Il est possible d'utiliser d'autres instructions conditionnelles dans n'importe quel ensemble d'instructions d'une instruction conditionnelle. En d'autres termes, les instructions utilisant les éléments IF peuvent être utilisées dans d'autres éléments IF. Cela permet de construire des instructions extrêmement complexes. Chaque instruction IF "emboîtée" doit se terminer par "END_IF ;"

Exemple:

```
IF zone1.Process_Val > 300 THEN
        timer.Prog_Time := T#4s;
        cool.Output.High := 90.0;
        IF zone1.Process_Val > 310 THEN
          boost.Process_Val := 400.0;
        ELSE
          boost.Process_Val := 350.0;
        END_IF;
END_IF;
```

OPERATEURS

Les opérateurs indiqués dans les tableaux 3-3, 3-4 et 3-5 peuvent être utilisés dans les expressions en Texte structuré PC3000.

Opération arithmétique

Opération	Symbole	Types de données
Addition	+	Virgule flottante(REAL), Entier(DINT)
Soustraction	−	Virgule flottante(REAL), Entier(DINT) Remarque 1
Multiplication	*	Vigule flottante(REAL), Entier(DINT)
Division	/	Virgule flottante(REAL) Entier(DINT) Remarque 2
Modulo Remarque 3	MOD	Entier(DINT)

Tableau 3-3 Opérateurs arithmétiques en Texte structuré

Remarque 1. Le symbole "-" est aussi utilisable pour la négation, c'est-à-dire pour convertir une valeur en lui donnant le signe contraire.

Exemple :

```
loop.Setpoint := - SPlow.Val;
```

Remarque 2. L'opérateur de division peut être utilisé avec deux paramètres ou constantes de type entier. Le résultat est toujours un entier et toute partie décimale du résultat est rejetée.

Exemple :

```
12 / 5
(* This will yield 2 as the integer result.*)
```

Remarque 3. L'opérateur modulo donne le reste d'une division entière. Il peut uniquement être utilisé avec des données de type "entier" et est normalement utilisé pour compter selon un certain modulo.

Exemples :

```
12 MOD 4  (* Result is 0 *)
14 MOD 4  (* Result is 2 *)
-4 MOD 3  (* Result is -1 *)
gearPos.Val := (gearPos.Val + pulses.Val) MOD 7;
(* The value of gearPos.Val is always between 0 and 6 for any
value of pulses.Val *)
```

Opérateurs de comparaison

Opération	Symbole	Types de données
Supérieur à	>	Virgule flottante(REAL), Entier(DINT), Tous les types de données date et heure
Inférieur à	<	Virgule flottante(REAL), Entier(DINT), Tous les types de données date et heure
Egal	=	Virgule flottante(REAL) Remarque 1, Entier(DINT), Tous les types de données date et heure
Différent de	<>	Virgule flottante(REAL) Remarque 1, Entier(DINT), Tous les types de données date et heure
Supérieur ou égal à	>=	Virgule flottante(REAL) Entier(DINT), Tous les types de données date et heure
Inférieur ou égal à	<=	Virgule flottante(REAL) Entier(DINT), Tous les types de données date et heure

Tableau 3-4 Opérateurs de comparaison en texte structuré

Remarque 1. Il est normalement déconseillé de tester l'égalité des valeurs à virgule flottante car de petites erreurs d'approximation peuvent provoquer un échec imprévisible du test, par exemple (3.43212 * 2 = 6.86424) pourrait être faux.

Ces opérateurs peuvent servir à comparer les valeurs des données de type à virgule flottante, entier et date et heure, c'est-à-dire les données de type IEC REAL, DINT, TIME, TIME_OF_DAY, DATE, DATE_AND_TIME. Les deux valeurs comparées doivent avoir le même type de données. Ces opérateurs restituent toujours un résultat de type booléen (BOOL) et servent normalement à définir les conditions limites des process.

Exemples:

```
purge.Time >= T#4m  (* Purge step duration
    greater than or equal to 4 minutes ? *)
loop1.Setpoint > 300.0 (* Loop Setpoint
                                over 300.0 ? *)
```

Opérateurs booléens

Opération	Symbole	Types de données
ET booléen	AND	Booléen (BOOL)
OU booléen	OR	Booléen (BOOL)
OU exclusif booléen	XOR	Booléen (BOOL) Remarque 1
Sens booléen inverse ou contraire.	NOT	Booléen (BOOL) Remarque 2

Tableau 3-5 Opérateurs booléens en Texte structuré

Remarque 1. L'opérateur OU exclusif (XOR) est utile lorsqu'il est nécessaire d'évaluer si, sur deux paramètres, un seul est "actif" ou "vrai" mais pas les deux. Cet opérateur donne un résultat faux si les deux paramètres sont vrais.

Remarque 2. L'opérateur NOT (NON) est utilisé avec une valeur ou expression booléenne unique pour nier le sens booléen. Cet opérateur peut être utilisé après et en liaison avec d'autres opérateurs booléens (cf. les exemples antérieurs d'opérateur booléen).

Ces opérateurs servent à créer des expressions comportant des données de type booléen (BOOL). Bien qu'ils soient essentiellement utilisés avec les entrées et sorties numériques, ils sont également utiles lorsqu'ils sont utilisés avec des opérateurs de comparaison pour créer des expressions qui décrivent des conditions complexes.

Exemples:

```
vent.Process_Val := switch1.Process_Val AND
                heatFlag.Val OR overPres.Val;
alarm.Val := NOT pressure.Val AND
                  NOT airValve.Process_Val AND
                  ( zone1.Process_Val > 500.0 ) ;
```

Exemple :

```
out1.Process_Val := (( in1.Val XOR in2.Val ) AND
                                    NOT in3.Val ) OR in3.Val;
(* out1.Process_Val is set to 1 "ON" when only one of the
parameters in1.Val, in2.Val or in3.Val are 1 "ON" *)
```

Priorité des opérateurs

Les opérateurs ont des priorités différentes pour garantir que, lorsqu'il y a des expressions complexes portant sur de nombreux opérateurs, l'ordre d'évaluation est cohérent.

Exemple:

```
total.Val := a.Val * 10 + b.Val * 20;
```

Lorsque cette expression est évaluée, a.Val est multiplié par 10, b.Val par 20 et les deux valeurs sont ajoutées. En d'autres termes, la multiplication a lieu en premier et a priorité sur l'opérateur d'addition. Les opérateurs de priorité identique sont évalués de la gauche vers la droite.

Il est possible de modifier la priorité en insérant des parenthèses "(",")" autour des expressions qui doivent être évaluées en premier lieu.

Exemple

```
total.Val := a.Val * ( 10 + b.Val * 20 );
```

Dans ce cas, b.Val est multiplié par 20, 10 est ensuite ajouté et le résultat est multiplé par a.Val.

En cas de doute sur l'ordre d'évaluation, insérer des parenthèses autour des expressions qui doivent être évaluées ensemble.

Opérateur	Symbole	Priorité
Mise entre parenthèses	(...)	MAXIMALE
Négation Complément	- NOT	
Multiplication Division Modulo	* / MOD	
Addition Soustraction	+ -	
Comparaison	<,>,<=,>=	
Egalité Inégalité	= <>	
ET booléen	AND	
OU exclusif booléen	XOR	
OU booléen	OR	MINIMALE

Tableau 3-6 Priorité des opérateurs Texte structuré

EXPRESSIONS BOOLEENNES

Les expressions booléennes produisent toujours un résultat booléen (BOOL), c'est-à-dire que la valeur est soit 1 pour "vrai" soit 0 pour "faux". Ce type d'expression peut servir à décrire les conditions limites d'un process ou des événements dans des instructions conditionnelles ou des transitions de GRAFCET. Une expression boolénne peut inclure n'importe quels opérateurs et fonctions en Texte structuré mais doit donner une valeur finale booléenne.

Exemples

```
gasFlow.Process_Val >= (valvePos.Process_val
                                        + 300.0 ) * flow.Val

digin1.Process_Val AND (heat1.Process_Val<250.0)
                                AND ( soak.Time>= T#40m)

valve.Process_Val > SQRT ( IN :=
                                loadSize.Process_val * 1.234 )
```

FONCTIONS

Le PC3000 offre une vaste gamme de fonctions standard qui peuvent être utilisées dans les expressions en Texte structuré pour simplifier les opérations complexes. Il est conseillé de se familiariser avec les types de fonctions disponibles ; ne pas oublier que les fonctions peuvent souvent simplifier et minimiser le Texte structuré nécessaire pour résoudre un problème donné. Une fonction possède un ou plusieurs paramètres et restitue toujours un résultat unique ayant des données d'un type particulier.

Les catégories sont énumérées dans les tableaux 3-7 et 3-8.

Catégorie	But	Exemples
Numérique	Offre une vaste gamme de fonctions mathématiques courantes, dont les fonctions trigonométriques et logarithmiques	ABS_REAL, SQRT, LOG, EXP, SIN, ACOS
Sélection	Sert à sélectionner des valeurs en fonction d'une condition	SEL_BOOL, SEL_REAL, SEL_DATE, MAX_REAL
Chaîne	Ensemble complet de fonctions pour manipuler les chaînes texte, dont la fusion de chaînes et l'insertion de texte	EQUAL, LEFT, CONCAT, REPLACE, JUSTIFY_RIGHT

Tableau 3-7 Catégories de fonctions en Texte structuré

Catégorie	But	Exemples
Conversion de type	Conversion entre les types de données	DINT_TO_REAL, REAL_TO_TIME, TIME_TO_UDINT
Conversion de chaîne	Conversion de valeurs dont les données sont de types différents vers et en provenance du format de chaîne texte	STRING_TO_DINT, STRING_TO_REAL, DINT_TO_STRING, DATE_TO_US_STRING
Arithmétique temporelle	Sert à ajouter et à soustraire les durées, les dates et les dates et heures	ADD_DATE_AND_TIME_T, SUB_DATE_AND_TIME_T, ADD_TOD_TIME
Compact	Les fonctions permettent le compactage de valeurs multiples vers et en provenance de chaînes longues pour les communications série.	EXT_REAL_FROM_STR, REP_REAL_IN_STR, EXT_TIME_FROM_STR, REP_TIME_IN_STR

Tableau 3-8 Catégories de fonctions en Texte structuré (suite)

De nombreuses fonctions mathématiques comportent une arithmétique complexe avec virgules flottantes ; par conséquent, pour minimiser l'allongement du temps système, il ne faut utiliser ces fonctions qu'avec parcimonie dans le "câblage par soft" associé à ces blocs fonctions qui tournent à des fréquences de scrutation élevées, par exemple dans le "câblage par soft" des blocs fonctions numériques.

Le Manuel des fonctions du PC3000 donne une liste complète des fonctions.

Exemples de texte structuré utilisant des fonctions :

```
light.Process_Val := EXPT ( BASE := 2.0,
                                        POWER := rate.Val );
(* light is given by raising two to the power of "rate" *)

upTime.Val := UDINT_TO_TIME (
                                        IN := pulses.Val * spaces.Val);
(* pulses scaled by spaces gives milliseconds upTime *)
```

Les noms des paramètres de fonctions sont insérés automatiquement par la station de programmation lors de l'insertion d'une fonction.

Les valeurs fournies aux paramètres de fonctions peuvent comporter des expressions complexes en Texte structuré à condition que le résultat de l'expression ait le même type de données que le paramètre de fonction considéré. La station de programmation vérifie que les expressions produisent le bon type de données pour les paramètres de fonctions.

SELECTION DE VALEURS

Deux méthodes permettent de sélectionner des valeurs de remplacement dans les affectations et expressions en Texte structuré.

Fonctions de sélection

Ce type de fonction permet la sélection d'une valeur de remplacement parmi deux, IN0 et IN1, en fonction de la valeur d'une entrée booléenne (BOOL) G. Le résultat est la valeur d'IN0 si la valeur de G est 0 "fausse", dans le cas contraire, le résultat est la valeur d'IN1 si la valeur de G est 1 "vraie". Des fonctions de sélection sont prévues pour tous les types de données PC3000. L'entrée de sélection G peut prendre une valeur dérivée de n'importe quelle expression booléenne.

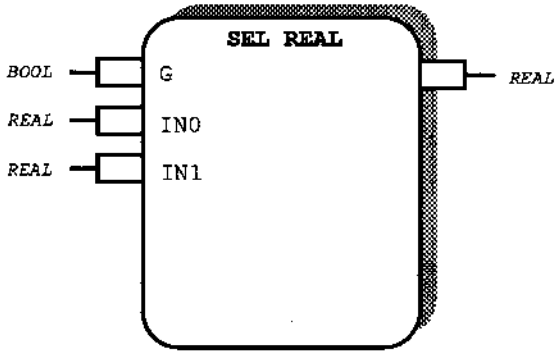


Figure 3-1 Exemple de fonction de sélection

Exemples:

```
speed.Val := SEL_REAL( G:= sensor.Val,
                                        IN0 := -20.0  ,
                                        IN1 := 30.0  );
            (* If sensor.Val is true value is 30.0,
            otherwise the value it is -20.0 *)

pulse.Prog_Time := SEL_TIME(G := rate.Val>100.0,
                                        IN0 := T#5s      ,
                                      IN1 := T#2s500ms);
(* pulse is 2.5 seconds for rate greater than
            100.0 otherwise pulse is 5 seconds *)
```

Les fonctions de sélection sont définies dans la norme IEC 1131-3 et sont particulièrement utiles dans le "câblage par soft" où il est nécessaire de changer la valeur affectée en fonction d'une condition spéciale.

Exemple:

```
loop1.Process_Val := SEL_REAL(G:= input1.Status,
                              IN0:= input2.Process_Val ,
                              IN1:= input1.Process_Val);
```

Dans cet exemple, loop1 du PID est relié à input1 analogique alors que l'état d'input1 est 1, c'est-à-dire "MARCHE". Toutefois, si input1 subit un incident comme une rupture de capteur, l'état passe à 0, c'est-à-dire "NOGO" et loop1 continue avec la valeur de process d'input2.

Sélection de valeurs à l'aide de l'élément IF (SI)

Une deuxième méthode, qui ne répond pas à la norme IEC 1131-3, offre la possibilité de sélectionner des valeurs à l'aide de l'élément IF...THEN (SI...ALORS) pour faciliter la lecture des programmes. L'utilisation de cet élément est déconseillée si une partie quelconque du programme doit être par la suite utilisée sur un système répondant entièrement à la norme IEC 1131-3.

L'exemple précédent peut être exprimé de la manière suivante :

```
loop1.Process_Val := IF input1.Status THEN
                         input1.Process_Val
                     ELSE
                         input2.Process_Val
                     END_IF;
```

IF...THEN nécessite une expression booléenne qui, lorsqu'elle est égale à 1, c'est-à-dire "vraie", provoque l'affectation de la première valeur au paramètre situé à gauche de l'affectation et qui, dans le cas contraire, provoque l'affectation de la deuxième valeur à ce paramètre. Comme pour la fonction de sélection, l'expression booléenne peut être complexe. Les valeurs de remplacement peuvent être fournies par des expressions complexes à condition qu'elles donnent une valeur qui possède le même type de données que le paramètre affecté.

> **N.B.** En Texte structuré, la valeur affectée lorsque l'expression booléenne est 1 "vraie" vient après la valeur affectée lorsque l'expression booléenne est 0 "fausse", lorsqu'on utilise la fonction de sélection, mais cet ordre est inversé lorsqu'on utilise l'élément IF...THEN.

REGLES D'ECRITURE CORRECTE EN TEXTE STRUCTURE

La station de programmation vérifie que l'ensemble des instructions en Texte structuré ont la structure (c'est-à-dire une syntaxe) correcte et, dans certains cas, que les types de données utilisés dans les expressions sont cohérents. Toutefois, il est bon de toujours écrire un Texte structuré possédant une structure correcte et d'utiliser les opérateurs possédant des types de données corrects. Les principaux points à prendre en considération sont les suivants :

1. S'assurer que chaque instruction se termine par un point-virgule, même chaque instruction conditionnelle.
2. Vérifier que les éléments comme IF ... THEN sont correctement structurés et se terminent par END_IF.
3. Les expressions doivent traiter des types de données homogènes.
4. Les opérateurs doivent être utilisés correctement. Par exemple, tous les opérateurs sauf "-" et "NOT" doivent être situés entre deux paramètres, constantes ou expressions. Exemples d'expressions interdites :

switch.Val := a.Val AND OR b.Val ; heat.Val := * 100.0 / O2flow.Val ;	✗

5. Utiliser des opérateurs avec des types de données appariés. Cela peut nécessiter l'utilisation des fonctions TYPE_CONVERSION pour garantir que les paramètres ou les expressions sont convertis dans le type de données qui convient.

Chapitre 4

GRAFCET

Edition 1

Sommaire

PRESENTATION

Ce chapitre décrit ce qui suit :

- Motifs de l'utilisation des GRAFCET (SFC)
- Principaux concepts des GRAFCET
- Utilisation du Texte structuré pour définir les pas et les transitions des GRAFCET
- Relation entre l'exécution des GRAFCET et l'exécution des blocs fonctions pour la régulation continue
- Conception sûre ou dangereuse des GRAFCET.

Introduction aux GRAFCET

Le GRAFCET (abrégé en SFC) est un langage de programmation graphique formalisé par la norme IEC 1131-3 et qui sert à décrire les aspects séquentiels d'un programme de régulation en termes de pas et de transitions discrets.

Pour construire une séquence, on relie graphiquement deux éléments de base d'un programme, les pas et les transitions, comme le montre la figure 4-1. Les liaisons graphiques sont analogues au "câblage par soft" utilisé pour connecter les blocs fonctions .

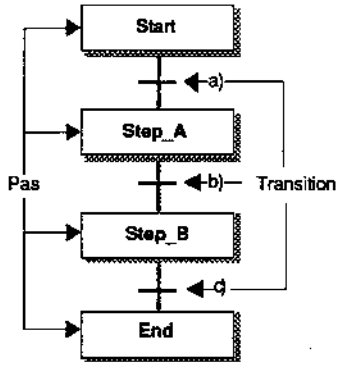


Figure 4-1 Eléments de base des GRAFCET

Un **pas**, qui est décrit comme un cadre autour d'un nom de pas, définit un ensemble d'actions qui sont effectuées lorsque l'installation ou la machine régulée est dans un état défini. Le pas reste actif jusqu'à ce que la condition associée à une **transition** suivante, décrite par une courte ligne horizontale, devienne "vraie".

Par exemple, dans le cas de la régulation d'un four, un pas pourrait établir des températures de points de consigne pour les boucles de régulation PID et la transition suivante pourrait avoir une condition qui attendrait que les boucles de régulation soient stables.

Une série de pas et de transitions est reliée par des lignes, comme le montre la figure 1, pour former une séquence. Lors de l'exécution d'une séquence, le premier pas (dans le cas présent, le pas "Start") est activé. Il reste actif jusqu'à ce que la condition pour la transition suivante soit vraie, c'est-à-dire la transition a). Lorsque la condition pour la transition a) est vraie, le pas suivant "Step_A" est activé et le pas "Start" est désactivé. De la même manière, la séquence défile tous les pas reliés. Dans cet exemple, un seul pas à la fois est actif.

Normalement, un pas contient des actions qui changent les valeurs des paramètres d'entrée des blocs fonctions pour modifier le comportement de la partie "régulation continue" du programme. Un pas défini habituellement les actions en Texte structuré mais peut représenter un autre GRAFCET en utilisant le concept "étape macro" traité dans la suite de ce chapitre.

La condition associée à une transition est définie à l'aide d'une expression booléenne écrite en langage Texte structuré (cf. chapitre 3 Texte structuré). Une condition peut définir n'importe quel événement temporel ou condition limite. La figure 4.2 présente un exemple de séquence simple et ses actions et transitions de pas associées définies à l'aide du Texte structuré ; cet exemple est tiré de l'exemple de programme présenté dans le chapitre 2 .

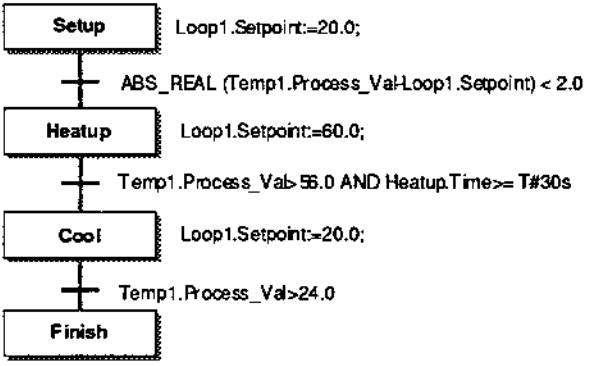


Figure 4-2 Exemple de séquence en Texte structuré

Sur la figure 4-2, si l'on suppose que le pas Setup est actif, le pas Heatup est activé lorsque la condition de transition "`ABS_REAL(Templ.Process_Val - Loopl.Setpoint) < 2.0`" devient vraie, c'est-à-dire lorsque la valeur absolue de l'erreur de la boucle de régulation du PID est inférieure à 2,0. Dès que Heatup est activé, le pas Setup est désactivé et sa transition associée n'est plus évaluée.

N.B. : il suffit que la condition d'une transition suivant un pas actif soit évaluée une fois comme étant "vraie" pour provoquer l'activation du pas suivant.

SELECTION DE SEQUENCES ALTERNATIVES

Il est possible de sélectionner les parties alternatives d'une séquence avec des transitions multiples en provenance d'un pas, comme le montre l'exemple a) de la figure 4-3. A partir du pas "Load", il existe trois pas alternatifs qui peuvent être activés. La sélection effectuée dépend de la transition qui est "vraie" la première lorsque "Load" est actif.

Cas où les séquences alternatives sont utiles :

- Lorsque les phases du process doivent varier en fonction du type de produit en cours de fabrication, comme le montre la figure 4-3.
- Lorsqu'un ensemble spécial d'actions est nécessaire à la suite d'une défaillance du process, c'est-à-dire une reprise après défaut.
- Lorsqu'il est nécessaire de répéter un ensemble de pas un certain nombre de fois puis de saisir une séquence alternative une fois que la condition finale a été atteinte.

Le système d'exploitation PC3000 évalue les transitions de la gauche vers la droite. Dans cet exemple, si les trois conditions de transition sont "vraies", seul le pas "Heat" sera sélectionné

N.B. : lorsqu'il y a des transitions alternatives, un seul des pas successifs est activé, même si plusieurs conditions de transition sont "vraies".

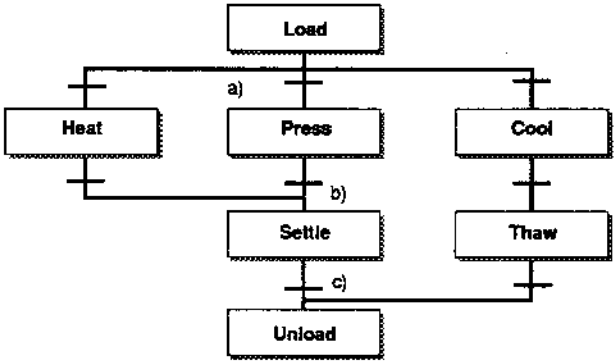


Figure 4-3 Sélection de séquences alternatives

Les séquences alternatives peuvent rejoindre d'autres séquences en reliant la transition du dernier pas à un pas d'une séquence différente, cf. les figures 4-3b) et c).

SEQUENCES PARALLELES

Il est possible de définir des séquences qui doivent tourner en parallèle en les reliant par un trait horizontal double suivant une transition simple (cf. figure 4-4 a).

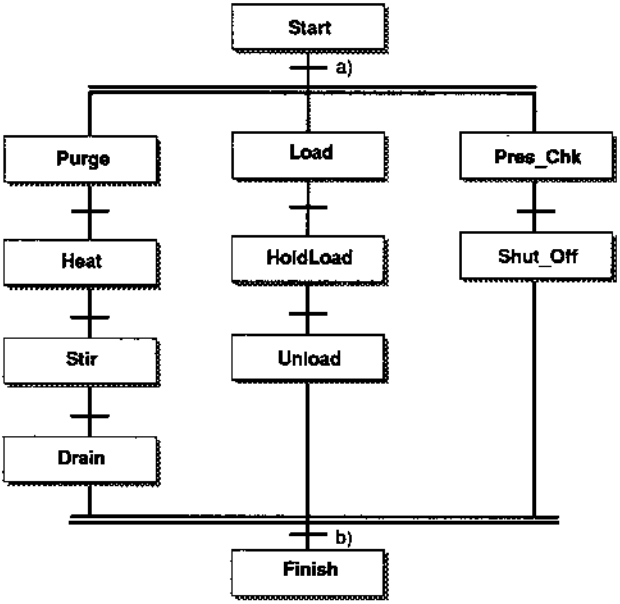


Figure 4-4 Séquences parallèles

Sur la figure 4-4, lorsque le pas "Start" est actif et que la condition de transition en a) est "vraie", les pas "Purge", "Load" et "Pres_Chk" sont activés simultanément, ce qui fait tourner les trois séquences en parallèle.

Des séquences parallèles peuvent être nécessaires lorsqu'un certain nombre d'opérations de process peuvent tourner indépendamment les unes des autres. La figure 4-4 montre trois séquences de régulation d'une cuve de réacteur, d'un mécanisme de chargement de tâches et de régulateurs de pression.

La station de programmation permet de lancer un maximum de 12 séquences parallèles à partir d'une seule transition. D'autres transitions peuvent lancer des séquences parallèles supplémentaires sur le même grafcet si besoin est.

Dans de nombreuses applications de régulation, il est souvent nécessaire d'attendre qu'un certain nombre de séquences parallèles se termine avant de passer à la partie suivante de la séquence principale. On peut y parvenir en utilisant une séquence parallèle **rendez-vous** représentée sur la figure 4-4b) par un trait horizontal double joignant un certain nombre de séquences parallèles. Un rendez-vous est toujours suivi d'une transition simple.

> **N.B. :** la condition pour la transition rendez-vous est uniquement évaluée lorsque tous les pas précédents connectés sont actifs. Lorsqu'elle est "vraie", tous les pas précédents sont désactivés et le ou les pas suivant(s) est(sont) activé(s).

Un rendez-vous peut être nécessaire lorsque plusieurs séquences doivent être finies avant qu'il soit possible de continuer. Par exemple, vous pouvez avoir besoin d'être sûr qu'une cuve de réacteur est vidangée, que le produit a été déchargé et que les régulateurs de pression ont été fermés avant d'ouvrir une porte de la cuve.

ETAPES MACROS ET MACROS

La station de programmation permet de construire des séquences complexes à partir d'une hiérarchie de **GRAFCET**. Le graphique supérieur est affiché lorsque l'éditeur de GRAFCET apparaît pour la première fois sur la station de programmation et il est appelé graphique **principal**. Il est possible de créer une étape comme étape **macro** en lui affectant l'attribut "Macro". Cela implique que l'étape représente une macro de niveau inférieur.

Chaque macro doit avoir une seule étape identifiée comme l'étape de départ. Lorsqu'une étape macro est activée, elle provoque l'activation de l'étape de départ de l'étape macro de niveau inférieur.

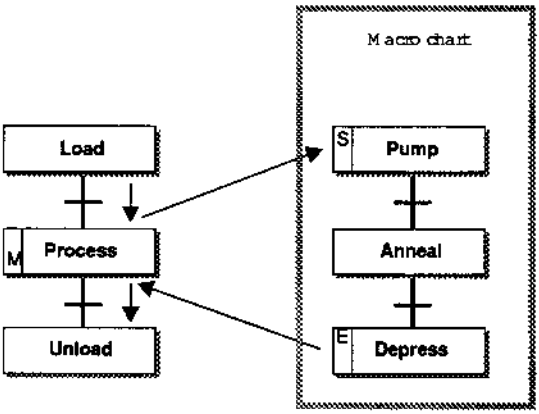


Figure 4-5 Appel d'une macro à partir d'une étape macro

La figure 4-5 montre un exemple simple d'étape macro "Process" qui représente une macro de niveau élémentaire. La séquence est transférée de l'étape "Load" sur le graphique de niveau supérieur à l'étape de départ "Pump" sur la macro lorsque la transition de "Load" à "Process" devient "vraie". La transition de "Process" à "Unload" est uniquement évaluée lorsque l'étape finale de la macro "Depress" est atteinte. Lorsque cette transition est "vraie" et l'étape "Depress" est active, la séquence revient au graphique de niveau supérieur, à l'étape "Unload".

Les macros peuvent appeler des macros supplémentaires sur un maximum de 20 niveaux, ce qui permet de créer des hiérarchies complexes de séquences.

> **N.B. :** il est bon de s'assurer que le graphique principal décrit uniquement les étapes de process primaires. Normalement, les étapes primaires sont des étapes macros qui représentent des graphiques de niveau élémentaire.

Etapes macros interruptibles

Une **étape macro interruptible** offre un mécanisme permettant de désactiver toutes les étapes dans les macros de niveau inférieur. Lorsqu'une étape macro interruptible est active, la macro de niveau inférieur est activée à partir de l'étape de départ (c'est-à-dire comme pour une étape macro normale). Toutefois, dans ce cas, la transition suivant l'étape macro est évaluée en continu. Si la condition de transition devient "vraie", toutes les étapes de la macro de niveau inférieur (y compris celles des macros situées à un niveau encore inférieur et qui proviennent d'étapes macros) sont désactivées.

Cela peut être utile lorsqu'une séquence complexe doit être rapidement supprimée, par exemple pour arrêter une séquence de chargement automatique lorsqu'on appuie sur un interrupteur manuel.

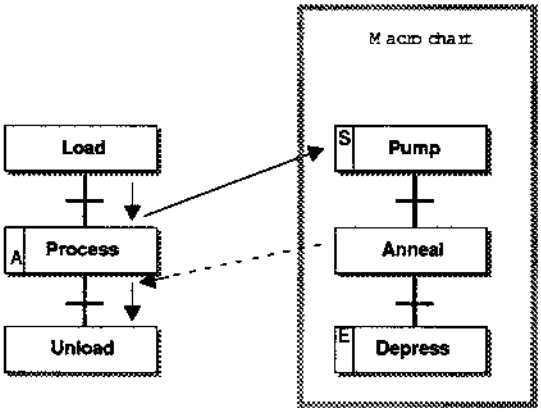


Figure 4-6 Utilisation d'une étape macro interruptible

Sur la figure 4-6, "Process" est une étape macro interruptible. Lorsque "Process" est activé, on entre dans la macro à partir de l'étape de départ "Pump" . La condition de la transition à partir de "Process" est évaluée en continu pendant que la séquence de la macro est active. La ligne en pointillés montre le cas où l'étape "Anneal" est interrompue parce que la transition provenant de "Process" est devenue "vraie". La séquence de la macro de niveau supérieur continue à partir de l'étape "Unload" qui suit "Process".

Il est possible de faire d'une étape finale une macro interruptible. Dans ce cas, c'est la(les) transition(s) de l'étape de la macro de niveau immédiatement supérieur qui est(sont) testée(s) et qui, lorsqu'elle(s) est(sont) vraie(s), met(mettent) fin à la macro de niveau inférieur.

Désactivation des étapes

Lorsque des étapes sont désactivées à la suite de l'interruption d'une macro, les actions d'une étape sont toujours exécutées en totalité. Cela signifie qu'il n'est jamais possible qu'une étape soit désactivée alors que des actions sont en cours d'exécution.

Une étape dont les actions sont exécutées une fois est désactivée après que toutes les actions ont été exécutées une fois.

Une étape continue est désactivée après que toutes les actions ont été exécutées au moins une fois. Les actions sont exécutées encore une fois après la désactivation de l'étape. Au cours de cette dernière exécution de l'étape continue, le paramètre **.Executing** de l'étape est faux. Le texte ci-après donne une description plus détaillée à ce sujet.

EXECUTION DES ACTIONS DE PAS

Les actions de tout pas décrit en Texte structuré peuvent être exécutées dans deux modes : pas à pas ou continu.

Pas dont les actions sont exécutées une fois

Il s'agit du mode d'exécution normal par défaut dans lequel toutes les actions du pas sont exécutées une fois lorsque le pas est activé pour la première fois.

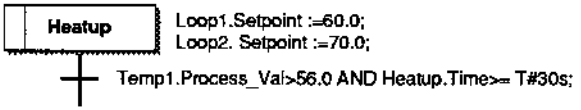


Figure 4-7 Pas dont les actions sont exécutées une fois

La figure 4-7 décrit un pas dont les actions sont exécutées une fois "Heatup". Les affectations à Loop1.Setpoint et Loop2.Setpoint se produisent uniquement lorsque le pas est activé pour la première fois. La condition de transition attend que le pas soit actif pendant au minimum 30 secondes. Toutefois, bien que le pas reste actif, les affectations ne sont effectuées qu'au début et n'ont aucun effet pendant le reste de cette période.

Ce mode d'exécution convient à la majorité des pas où il est nécessaire de simplement configurer les paramètres des blocs fonctions pour un état ou une opération de process donné(e).

> **N.B.** : ce mode est équivalent au qualificateur d'action **P**ulse défini dans la norme **IEC** 1131-3.

Pas continu

Ce mode d'exécution qui peut être fixé par l'attribut pas continu provoque l'exécution répétée des actions d'un pas tant que ce pas est actif. Cela peut être par exemple nécessaire lorsqu'il faut suivre les valeurs d'entrée pendant une période donnée ou modifier en continu les valeurs de sortie. En fait, un pas continu se comporte comme un élément de "câblage par soft" temporaire.

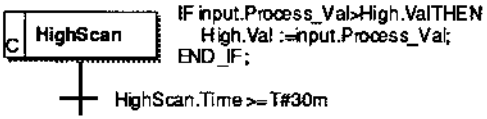


Figure 4-8 Pas continu

Sur la figure 4.8, un pas continu sert à surveiller une entrée analogique pendant une période de 30 minutes et à enregistrer la valeur maximale atteinte dans le paramètre High.Val.

> **N.B.** : les actions d'un pas continu sont exécutées encore une fois après que la transition suivante du pas est devenue "vraie", ce qui fait l'objet de la discussion dans la section suivante.

SYNCHRONISATION DE L'EXECUTION DES PAS

Tous les graphiques sont scrutés de manière répétitive à la fréquence de scrutation fixée par la tâche affectée au GRAFCET. Tous les pas actifs sont évalués et les actions de pas associées sont exécutées en fonction du mode d'exécution de chaque pas.

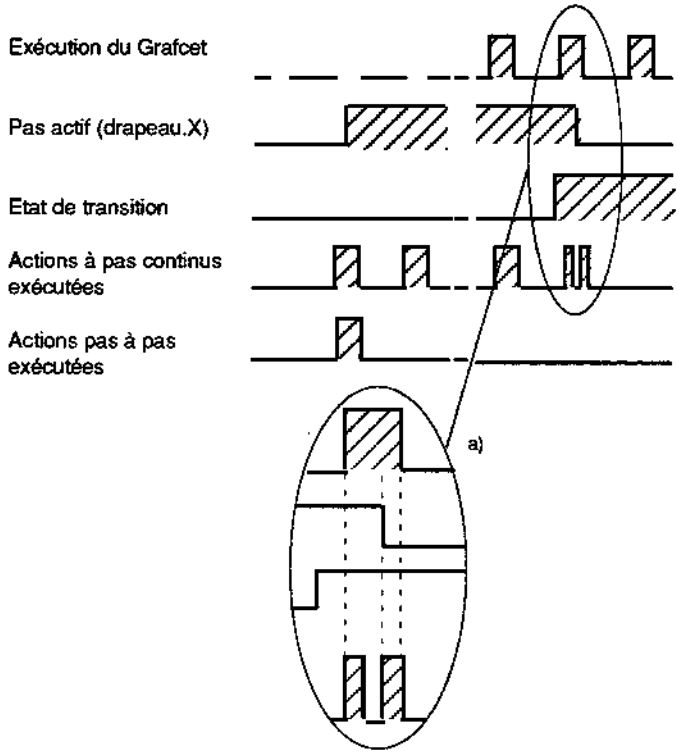


Figure 4-9 Syncrhonisation de l'exécution des pas

La figure 4-9 est un diagramme de synchronisation qui montre les différences entre l'exécution des actions pas à pas et l'exécution des actions à pas continus. Un pas est actif jusqu'à ce qu'une transition suivante devienne "vraie" ; il est ensuite désactivé. Le paramètre **Executing (exécution)** reste "vrai" pendant que le pas est actif (cf. également la section Blocs fonctions de pas et macros).

Les actions d'un pas unique sont exécutées une fois lorsque le premier pas devient actif.

Par opposition, les actions d'un pas continu actif sont exécutées à chaque scrutation du GRAFCET. Il faut toutefois noter que les actions sont également exécutées une fois encore après que le pas a été désactivé, cf. figure 4-10. Cette possibilité est offerte afin que les pas continus puissent avoir des fonctions leur permettant de fermer certaines fonctions lors de la sortie.

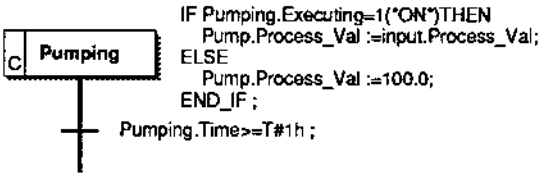


Figure 4-10 Exécution des actions à pas continus lors de la sortie

La figure 4-10 montre un exemple de pas continu dans lequel une affectation est effectuée lorsque le pas est désactivé. Lorsque le pas "Pumping" est actif, `Pump.Process_Val` est mis à jour en continu à partir de `input.Process_Val`. Toutefois, lorsque le pas est désactivé, Pump.Process_val est remis à 100,0. Le paramètre de pas **Executing** (abrégé en X) sert à sélectionner les actions qui doivent se produire lorsque le pas est actif et celles qui ont lieu une fois lorsque le pas est désactivé, c'est-à-dire lorsqu'il est sur off (cf. également les blocs fonctions de pas et macros).

Actions de synchronisation dans les pas continus

Le paramètre **.Time** d'un pas continu est mis à jour en continu lorsque ce pas continu est actif. Cette caractéristique peut être utilisée pour provoquer la réalisation de certaines actions à des moments différents après le démarrage du pas.

Exemple :

```
IF shutdown.Time < T#1m THEN
  pump1.Process_Val := 30;
ELSE
  pump1.Process_Val := 0;
END_IF;

Valve1.Process_Val := (shutdown.Time > T#2m);
```

Dans cet exemple, nous supposons que l'arrêt est un pas continu actif pendant 4 minutes, par exemple, synchronisé par la transition suivante du pas. Les instructions en Texte structuré des actions du pas garantissent que la valeur de process de pump1 est positionnée sur 30 au cours de la première minute puis sur 0. Valve1.`process val` est "vraie", c'est-à-dire que la vanne est positionnée sur "ON" (marche) après deux minutes.

INTERACTION ENTRE LES GRAFCET ET LES BLOCS FONCTIONS

Il faut prendre des précautions particulières dans les cas où les valeurs des paramètres des blocs fonctions sont configurées dans un pas qui est suivi immédiatement d'une transition qui effectue un test pour savoir si le bloc fonction a terminé une opération. Ne pas oublier qu'un bloc fonction donné peut ne pas avoir le temps suffisant pour s'exécuter entre le paramétrage des valeurs du pas et le test du résultat dans la transition.

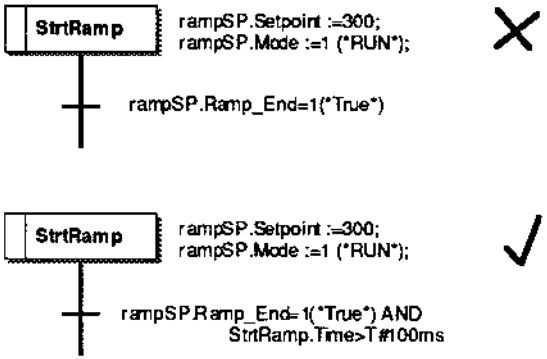


Figure 4-11 Interaction entre les GRAFCET et les blocs fonctions

La figure 4-11 montre un problème qui se produit couramment. Les actions du pas configurent un bloc fonction "rampSP" pour qu'il aille "en rampe" jusqu'à un point de consigne donné. La transition doit attendre que la rampe soit terminée.

Dans le premier exemple, la transition peut ne pas détecter la fin de la rampe car la transition peut être "vraie" immédiatement après que le pas est devenu actif. Cela est dû au fait que le paramètre rampSP.Ramp_End reste "vrai" à la suite d'une exécution précédente du bloc fonction rampSP.

Dans le deuxième cas, la condition de transition a été modifiée pour garantir que rampSP s'exécute au moins une fois avant de tester le paramètre Ramp_End. Cet exemple suppose que le bloc fonction rampSP est dans une tâche 100 ms.

> **N.B. :** toujours s'assurer qu'il y a suffisamment de temps pour l'exécution des blocs fonctions après la configuration des paramètres d'entrée puis le test des valeurs des paramètres de sortie des blocs fonctions.

ATTRIBUTS DES ETAPES

Démarrage	Spécifie que l'étape sera rendue active lorsque la macro qui la contient deviendra active.
Fin	Normalement la dernière étape d'une étape macro. La condition de la transition qui suit une étape macro sera testée lorsque l'étape Fin de la macro associée deviendra active. Si la condition de la transition est "vraie", l'étape Fin devient inactive et la macro qui la contient termine son exécution. Toutefois, l'étape Fin du graphique PRINCIPAL est un cas spécial qui, lorsqu'elle est active, s'exécute une fois et le GRAFCET tout entier est alors terminé. Une macro normale (qui ne peut pas être interrompue) doit contenir une étape finale. Une macro interruptible peut contenir une ou zéro étape finale.
(Valeur par défaut normale)	Attribut par défaut qui n'est pas explicitement défini sur la station de programmation, implique que l'étape n'est ni une étape initiale ni une étape finale.

Table 4-1 Attributs des étapes

Macro	Spécifie que l'étape active une macro qui ne peut être interrompue. La(les) transition(s) qui suit(suivent) une étape macro sera(seront) uniquement testée(s) lorsque la macro associée aura atteint l'étape finale.
Macro interruptible	Spécifie que l'étape active une macro interruptible. La(les) transition(s) qui suit(suivent) une étape macro interruptible est(sont) testée(s) en continu pendant que la macro est active. Si une des transitions a une condition "vraie", la macro tout entière est désactivée.
Pas à pas *(par défaut)*	Spécifie que les actions de l'étape sont uniquement exécutées une fois lorsque l'étape est activée pour la première fois.
Continu	Spécifie que les actions de l'étape sont exécutées en continu pendant que l'étape est active. Les actions sont exécutées encore une fois après que l'étape a été désactivée.

Table 4-2 Attributs des étapes

Lors de la configuration des attributs des étapes, une étape peut posséder un attribut du tableau 4-1 et un attribut du tableau 4-2.

Exemples :

Démarrage + exécution une seule fois

Normal + Continu

Normal+ Macro interruptible

Fin + Macro

BLOCS FONCTIONS PAS ET MACROS

Chaque pas, chaque macro et chaque étape macro interruptible sont représentés dans le système PC3000 sous la forme d'un bloc fonction. Cela permet d'utiliser les paramètres qui donnent l'état et la durée de l'étape dans le Texte structuré, c'est-à-dire dans le "câblage par soft", les actions des étapes et les conditions de transition.

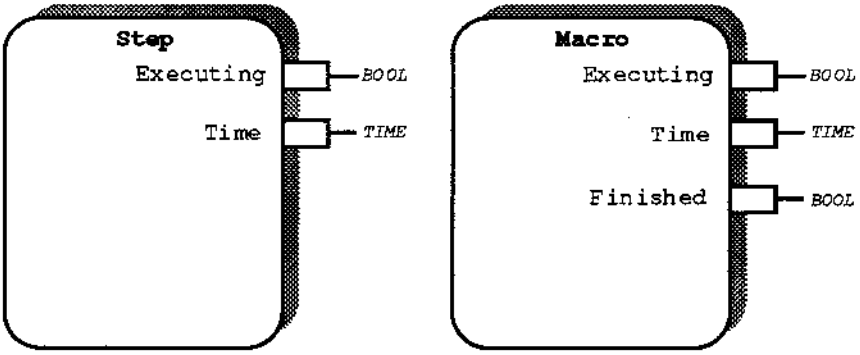


Figure 4-12 Blocs fonctions étapes et macros

Le bloc fonction macro est utilisé pour les étapes macros normales et les étapes macros interruptibles. Ces deux types de blocs possèdent des paramètres Executing et Time.

Le paramètre **Executing** (abrégé en .X) est mis à 1 toutes les fois que l'étape ou la macro est active.

Le paramètre **Time** (abrégé en .T) donne la durée de l'étape ou de la macro. La valeur de ce paramètre est remise à zéro lorsque l'étape est activée pour la première fois puis augmente pendant que l'étape est active. Lorsque l'étape est désactivée, la valeur est bloquée. Ce paramètre peut servir à mesurer la durée d'une étape. Il constitue également une précieuse aide au diagnostic car il enregistre la durée pendant laquelle un programme s'est trouvé dans une étape donnée lors de sa dernière exécution.

Le paramètre **Finished** (abrégé en .F) fourni avec le bloc fonction macro est configuré lorsque la macro atteint l'étape finale. Il est toujours supprimé lorsque l'étape macro n'est pas active. Il peut être utilisé dans les conditions de transition pour tester si une macro donnée est terminée.

REGLES DE PROGRAMMATION DES GRAFCET

Règles d'utilisation des graphiques

Les points suivants s'appliquent toujours aux graphiques :

1. Le graphique supérieur appelé MAIN donne la vue d'ensemble du niveau supérieur du programme séquentiel tout entier.
2. Un graphique de niveau inférieur est associé à une étape macro normale ou à une étape macro interruptible dans le graphique de niveau immédiatement supérieur.
3. Un graphique peut contenir un certain nombre de pas et de transitions reliés par câblage (c'est-à-dire par des lignes graphiques) qui définissent le flux de régulation d'un ensemble d'étapes actives et des transitions qui leur sont associées à l'ensemble suivant.
4. Un graphique doit avoir précisément une étape de démarrage.
5. Un graphique doit avoir une ou zéro étape finale.
6. Un graphique actif est un graphique qui contient une ou plusieurs étape(s) active(s).

Règles d'utilisation des pas

Les points suivants s'appliquent toujours aux pas :

1. Un pas peut être une macro qui représente un autre GRAFCET ou il peut être défini comme un ensemble d'actions utilisant le langage Texte structuré.
2. Un pas est normalement suivi d'une ou plusieurs transitions sauf :

   a) les pas finaux qui ne sont jamais suivis d'une transition et

   b) un pas peut en option ne pas être suivi par des transitions s'il doit toujours rester actif une fois qu'il a été atteint.

   Un pas continu qui n'est suivi par aucune transition reste actif.
3. Un pas ne peut pas être suivi immédiatement d'un autre pas sans transition.
4. Un pas peut posséder un attribut qui définit la manière dont il est exécuté.

Règles d'utilisation des transitions

Les points suivants s'appliquent toujours aux transitions :

1. Chaque transition doit toujours être précédée et suivie d'un ou plusieurs pas.
2. Une transition ne peut jamais être reliée à une autre transition.

3. Une transition représente une condition qui doit être remplie avant que le ou les pas précédent(s) qui lui est(sont) associé(s) soit(soient) désactivé(s).
4. Une condition de transition est décrite par une expression booléenne en Texte structuré.

MAUVAISE CONCEPTION DES GRAFCET

La station de programmation vérifie chaque graphique et s'assure que la construction de base de chaque GRAFCET est correcte avant de permettre la construction d'un programme (par exemple, chaque graphique doit avoir un pas de démarrage). Toutefois, il est possible de construire un graphique dont le comportement est incorrect pour deux principales raisons :

1. Il peut y avoir certains pas qui ne peuvent jamais être activés.
2. Des pas ou des séquences peuvent rester actifs par inadvertence. Cela peut se produire lors d'une nouvelle entrée dans une séquence, en particulier si elle comporte des séquences parallèles qui ne se terminent pas correctement à un rendez-vous.

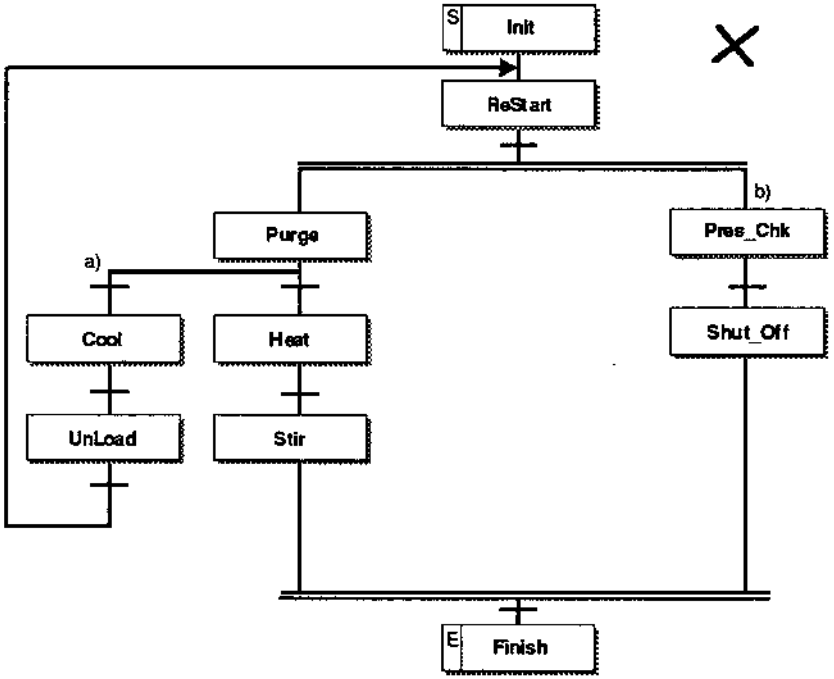


Figure 4-13 Mauvaise construction de GRAFCET

La figure 4-13 montre un GRAFCET mal construit car, si la séquence a) est sélectionnée, il est possible que les pas "Pres_Chk" ou "Shut_off" restent actifs lors de la réactivation du pas "ReStart".

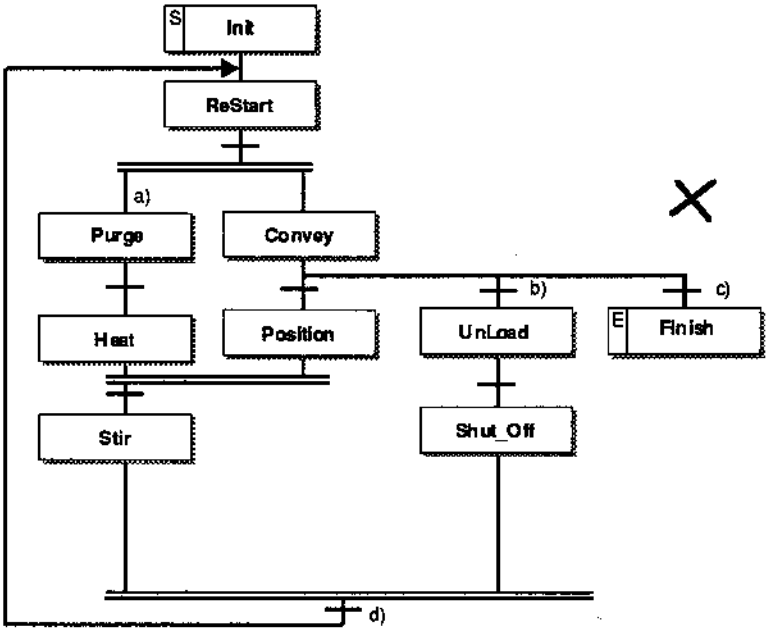


Figure 4-14 Rendez-vous impossible à atteindre

La figure 4-14 donne un autre exemple de GRAFCET erroné ; supposons que ce GRAFCET ait été appelé par une étape macro. Le rendez-vous en d) ne peut jamais être atteint si l'on sélectionne la séquence en b) ou le pas final c). Etant donné que le pas "Position" n'est pas activaté, le pas "Stir" ne peut jamais être atteint.

Il est également possible que le pas final "Finish" soit atteint alors que d'autres pas sont encore actifs (pas "Purge" ou "Heat"). Cela peut entraîner le retour à l'étape macro dans le graphique de niveau supérieur alors que des pas sont encore actifs.

N.B. : sur les macros, toujours s'assurer que toutes les séquences parallèles parviennent à un rendez-vous avant d'atteindre le pas final.

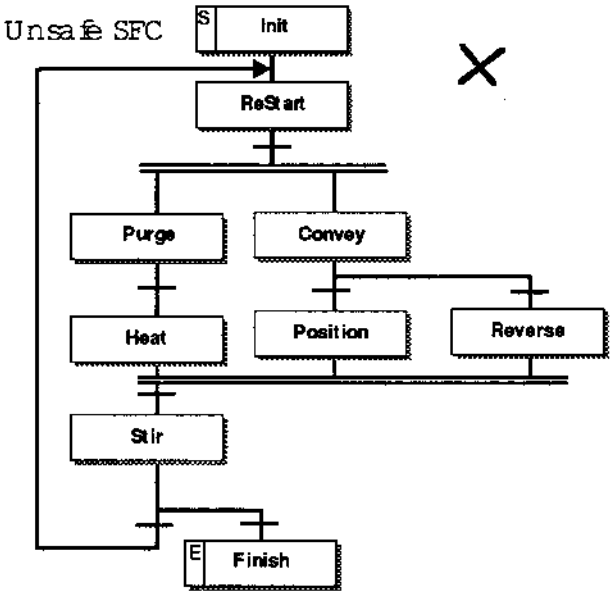


Figure 4-15a Rendez-vous incomplet

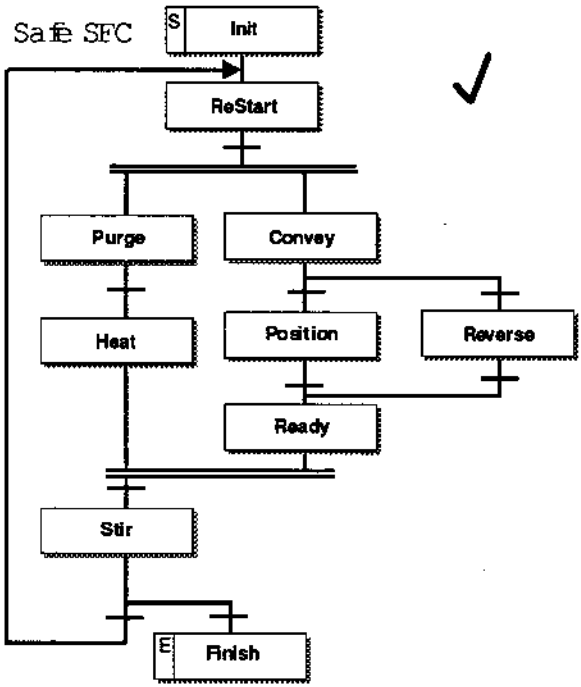


Figure 4-15b Rendez-vous complet

La figure 4-15a montre un GRAFCET erroné et la figure 4-15b montre un GRAFCET correct équivalent. Dans le GRAFCET erroné, la transition rendez-vous avant le pas "Stir" ne peut jamais être évaluée car elle attend que tous les pas "Heat", "Position" et "Reverse" deviennent actifs. Cela ne peut jamais se produire car les pas "Position" et "Reverse" sont des pas alternatifs.

Chapitre 5

DEVELOPPEMENT DES PROGRAMMES

Edition 1

Sommaire

PRESENTATION

Ce chapitre décrit les aspects suivants du développement des programmes pour le PC3000 :

- Manière de structurer un programme PC3000 en analysant les différents besoins d'un système de régulation
- Méthodes correctes de programmation
- Manière d'utiliser efficacement le système de régulation PC3000
- Manière de développer des programmes faciles à lire et à entretenir
- Manière de mettre en service les programmes PC3000
- Manière d'utiliser les blocs fonctions avec les GRAFCET pour résoudre les problèmes de régulation

CONSIDERATIONS RELATIVES A LA CONCEPTION DES PROGRAMMES

Un programme PC3000, comme tout élément logiciel, doit être conçu avec soin pour garantir son bon fonctionnement et une utilisation optimale des ressources et performances du PC3000. Si le programme est utilisé pour réguler un process de production, il est également important que le programme soit facile à lire et à comprendre par les techniciens chargés de la mise en service et de la maintenance.

Bien que le PC3000 soit un système de régulation souple qui permet de construire des programmes par étapes, il est conseillé de connaître et de prendre en compte tous les aspects de l'application du système de régulation avant de commencer le développement du programme.

Il est conseillé de suivre les procédures ci-dessous :

1. Produire un inventaire complet de l'ensemble des entrées et sorties nécessaires au système. Il est conseillé d'ajouter quelques entrées et sorties supplémentaires en cas de besoin et pour les extensions futures.
2. Analyser toutes les entrées et sorties analogiques et identifier celles qui sont associées aux boucles de régulation PID. Identifier les sorties numériques à utiliser pour les sorties proportionnelles au temps.
3. Identifier les temps de réponse, la plage et la précision de toutes les entrées et sorties analogiques .
4. Identifier les temps de réponse nécessaires pour les entrées et sorties numériques.

5. Identifier les aspects de la stratégie de régulation qui s'appliquent en permanence et font par conséquent partie de la stratégie de régulation continue, par exemple les boucles de régulation PID, les verrouillages, le suivi des alarmes.
6. Identifier les principales phases de régulation séquentielles : initialisation de process, étalonnage, arrêt, attente sont des exemples de phases de régulation bien distinctes.
7. Identifier les phases de régulation qui peuvent fonctionner parallèlement et n'exercent pas d'interaction directe les unes sur les autres. Par exemple, la manipulation d'un affichage opérateur et la mise en rampe des points de consigne d'une boucle de régulation peuvent être pilotées par des séquences qui tournent en parallèle.
8. Identifier tous les besoins de communication avec des appareillages externes comme les systèmes SCADA, d'autres PC3000, les régulateurs programmables et les instruments et unités discrets Eurotherm.
9. Identifier les valeurs des paramètres clés du système de régulation qui peuvent nécessiter une modification lors de la mise en service du système de régulation.

Optimisation des performances par le séquencement

.Il est important de prendre en compte tous les aspects de séquencement du programme de régulation. Il faut éviter tout "câblage par soft" superflu dans la stratégie de régulation continue car cela peut entraîner un allongement significatif du temps système. Par opposition, la stratégie de régulation assurée par les actions dans les pas des GRAFCET ne provoque un allongement du temps système que lorsque certains pas sont actifs.

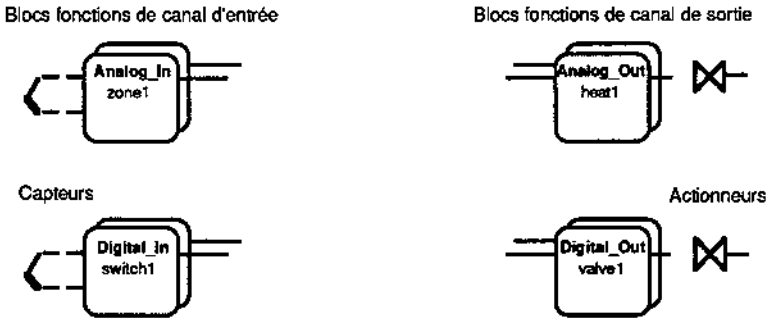
> **N.B. :** contrairement à la programmation en échelle des régulateurs programmables où le défilement séquentiel et la régulation continue sont combinés, le PC3000 permet la séparation entre le séquencement et la régulation continue. Cela diminue les exigences en matière de performances et facilite le développement, la mise en service et la maintenance du programme de régulation.

PHASES DE DEVELOPPEMENT DES PROGRAMMES

Il est conseillé de développer un programme PC3000 dans un certain nombre de phases claires. Toutefois, il ne faut pas oublier que ces phases sont seulement données à titre indicatif et que vous êtes libre de développer n'importe quelle partie du programme à n'importe quel moment. L'exemple de programme donné dans le chapitre 2 répond à ces directives.

Les phases recommandées pour développer un programme PC3000 à l'aide de la Station de programmation PC3000 sont récapitulées ci-dessous:

Phase 1: sélection et configuration des canaux d'E/S

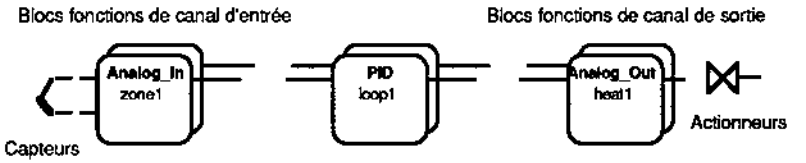


Les principales actions nécessaires sont les suivantes :

1. A l'aide de la station de programmation PC3000, sélectionner les modules matériels nécessaires en fonction de l'inventaire d'E/S. Si plusieurs racks sont nécessaires, il est possible d'optimiser la capacité de traitement des E/S en séparant les racks des entrées numériques et les racks des sorties numériques. De même, lorsque cela est possible, éviter que les modules numériques soient dans le même rack que les modules analogiques.
2. Attribuer des noms à tous les canaux d'E/S. Utiliser des noms qui ont une signification et non pas des noms arbitraires. L'idéal consiste à utiliser les noms qui apparaissent sur les diagrammes de câblage ou de régulation .
3. Configurer les plages des canaux, les décalages et le changement d'échelle. Dans certains cas, cela peut attendre la mise en service du système.

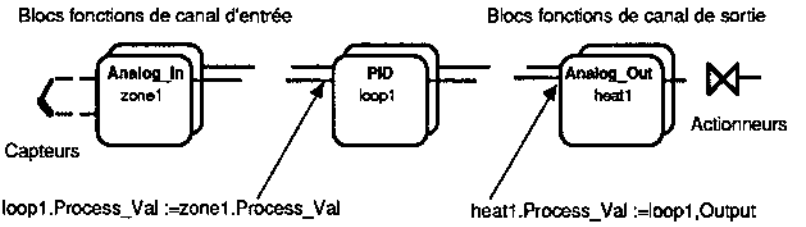
4. Ne pas oublier que les modules spécialisés comme l'ICM et l'AIO8 ne peuvent être installés que dans les 5 premiers emplacements du premier rack. Il faut donc laisser ces emplacements vides si l'un de ces modules doit être ajouté ultérieurement.

Phase 2 : configuration de la régulation analogique continue



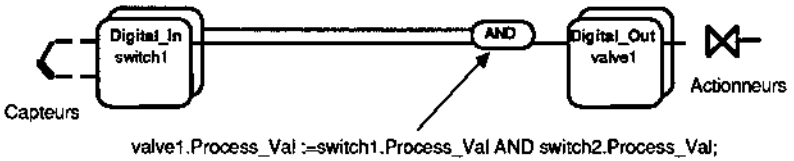
Les principales actions nécessaires sont les suivantes :

1. Création de blocs fonctions PID pour traiter chaque boucle de régulation.
2. Configuration du mode d'exploitation pour chaque PID, par exemple canal double (chauffage/refroidissement), positionneur de vanne, auto-réglage.
3. Utiliser le "câblage par soft" du Texte structuré pour relier la valeur de process des canaux de sortie analogiques (ou des canaux de sortie numériques proportionnels au temps) à la sortie des blocs fonctions PID.



4. "Câbler aussi par soft" la valeur de process de chaque bloc fonction PID avec la valeur de process d'entrée analogique qui convient.
5. Créer d'autres blocs fonctions et utiliser le "câblage par soft" pour créer d'autres aspects de la régulation analogique continue.

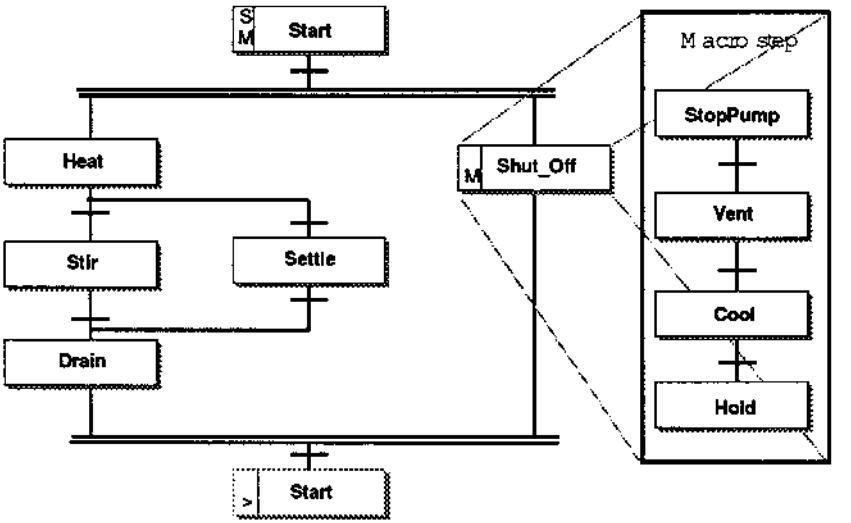
Phase 3 : configuration de la régulation numérique continue



Les principales actions sont les suivantes :

1. Utiliser le "câblage par soft" pour créer la stratégie de régulation numérique continue, c'est-à-dire la logique numérique.
2. Eviter dans la mesure du possible d'utiliser le Texte structuré qui comporte de nombreuses expressions ou fonctions à virgule flottante (REAL) dans le "câblage par soft" vers les blocs fonctions numériques. Cela permettra ainsi de diminuer l'allongement du temps système dans le "câblage par soft" numérique qui tourne normalement dans la tâche qui a la fréquence de scrutation la plus élevée.

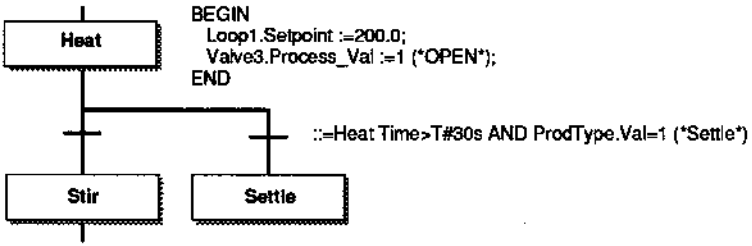
Phase 4 : création d'une stratégie de régulation de séquencement



Les principales actions sont les suivantes :

1. Créer le GRAFCET principal (de niveau supérieur). Chaque pas doit représenter une phase primaire du process. Normalement, les étapes du graphique principal sont définies comme étant des étapes macro. Ainsi, les phases importantes du process sont bien visibles : cela est particulièrement utile lors de la mise en service.
2. Décomposer chaque étape macro en macros de niveau inférieur et enfin en étapes pour terminer le programme séquentiel.

Phase 5 : création de transitions et d'actions de pas



Les principales actions sont les suivantes :

1. Création du Texte structuré pour les conditions de transition.

   N.B. : les transitions qui ne sont pas définies prennent par défaut la valeur 1, c'est-à-dire "vrai" ou "marche".

2. Création des actions de pas.

   N.B. : pour des raisons de synchronisation, il est possible d'avoir des pas sans actions en Texte structuré. Par exemple, un pas nul peut servir d'"attente", il est suivi d'une condition de transition qui attend que le pas soit actif pendant une durée donnée.

CONCEPTION DU GRAFCET PRINCIPAL

Il est bon d'adopter des principes de conception du "haut vers le bas". Cela implique que le GRAFCET principal doit représenter les phases principales du process ou les séquences primaires de régulation.

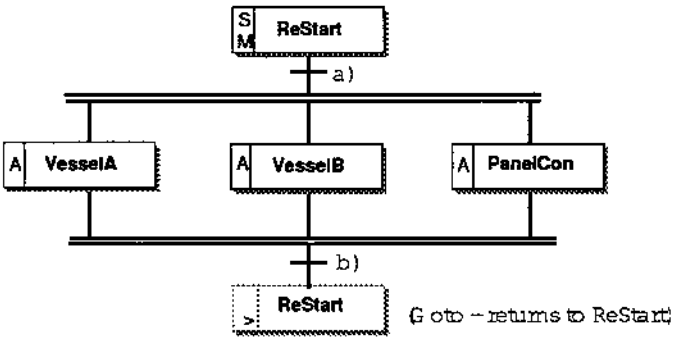


Figure 5-1 Exemple de GRAFCET principal

La figure 5-1 est un exemple de GRAFCET principal (de niveau supérieur) pour un programme de régulation qui agit sur deux cuves de réacteur. Chacune de ces cuves est régulée indépendamment par deux étapes macros interruptibles "VesselA" et "VesselB". Une troisième étape macro interruptible "PanelCon" régule un tableau opérateur qui fournit un affichage en temps réel des paramètres de régulation sélectionnés pour les deux cuves. Ces étapes tournent en parallèle et n'ont aucune interaction directe. Le programme de régulation est initialisé par l'étape macro "ReStart" qui configure les points de consigne de la boucle de régulation, etc.

La transition a) attend que l'opérateur acquitte la possibilité de redémarrage du process en tapant une commande sur le panneau opérateur.

La condition de la transition en b) teste une entrée numérique qui est reliée à une clé d'arrêt manuel. Lorsqu'elle détecte que la clé est "activée", cette transition provoque l'arrêt des trois étapes parallèles. La séquence revient ensuite à l'étape de départ "Restart" par l'intermédiaire d'un "goto".

RETROACTION DANS LE CÂBLAGE PAR SOFT DES BLOCS FONCTIONS

Normalement, l'ordre d'exécution des blocs fonctions dans une scrutation quelconque est tel que les blocs fonctions associés aux paramètres dans les affectations de "câblage par soft" s'exécutent toujours avant le bloc fonction qui doit recevoir la valeur de "câblage par soft".

Il existe certaines situations, comme celle décrite sur la figure 5-2, où il est nécessaire de "câbler par soft" les blocs fonctions entre eux pour former une boucle de rétroaction. Dans ces cas, il est possible de modifier l'ordre d'exécution des blocs fonctions en identifiant une des expressions de câblage par soft formant la boucle comme étant le "câblage par soft de rétroaction".

L'éditeur de câblage de la station de programmation offre une touche marquage de rétroaction qui permet de marquer n'importe quelle affectation de câblage en Texte structuré en insérant la "rétroaction" comme commentaire. Lorsque le programme est compilé et construit, cela provoque l'exécution du bloc fonction auquel est affecté le "câblage par soft" de rétroaction avant les blocs fonctions référencés à droite de l'affectation du "câblage par soft".

> **N.B. :** un bloc fonction qui reçoit une valeur provenant d'une affectation de câblage par soft marquée comme "rétroaction" s'exécute toujours avant les blocs fonctions dont les paramètres sont référencés sur la partie droite de l'affectation.

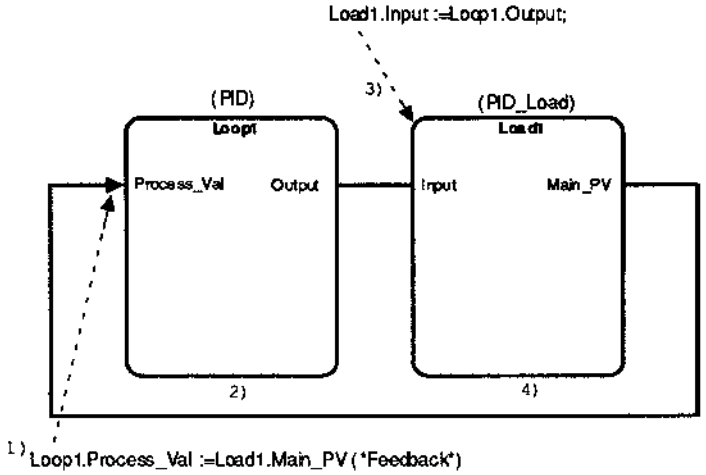


Figure 5-2 Rétroaction dans le câblage par soft des blocs fonctions

La figure 5-2 montre un exemple où une boucle de régulation de PID (Loop1) est reliée à PID_Load (Load1) qui simule différentes charges de régulation d'une installation. Pour terminer la simulation, la valeur de (**Main_PV**) provenant de Load1 est rétroactivée comme valeur de process (**Process_Val**) au bloc Loop1 PID à l'aide du câblage par soft marqué comme "rétroaction".

Cela provoque la création de l'ordre d'exécution suivant lors de la construction du programme :

1. Exécuter le câblage par soft pour créer une nouvelle valeur de `Loop1.Process_Val` à l'aide de la valeur de `Load1.Main_PV` provenant de la dernière scrutation.
2. Exécuter le bloc de régulation Loop1 PID et mettre à jour Loop1.Output.
3. Exécuter le câblage par soft pour créer une nouvelle valeur pour `Load1.Input` à partir de `Loop1.Output` créé dans la scrutation en cours.
4. Exécuter le bloc Load1 et mettre à jour le paramètre Main_PV.

Toute affectation de câblage par soft en Texte structuré qui est marquée comme "rétroaction" alors qu'elle ne fait pas vraiment partie d'une boucle de rétroaction provoquera l'émission d'un message d'erreur "Marque de rétroaction redondante" par la station de programmation lors de la compilation du programme.

Le positionnement de la marque de rétroaction dans le câblage par soft n'a normalement aucun effet significatif lors de la construction de boucles de régulation analogiques, en particulier dans les cas de régulation où les constantes de temps sont significativement plus longues que les durées de scrutation des tâches.

Toutefois, avec la logique numérique, il faut prendre des précautions lors du positionnement du câblage par soft de rétroaction car cela peut parfois avoir un effet significatif.

MODIFICATION DE LA CONFIGURATION DES TACHES

Pour la majorité des programmes PC3000, la configuration par défaut des tâches convient. Elle offre deux tâches : Task_1 qui tourne à une vitesse de scrutation de 10ms pour les blocs fonctions numériques et Task_2 qui tourne à une vitesse de scrutation de 100ms pour les blocs fonctions analogiques et l'exécution des GRAFCET.

Les tâches par défaut offrent des temps de réponse entre les entrées et les sorties numériques (latence du débit) compris entre 15 ms et 25 ms. Il est possible de configurer un maximum de 32 boucles de régulation de PID, chacune tournant à une vitesse de scrutation de 100 ms et pouvant réguler des charges avec des constantes de temps primaires supérieures à 10 secondes.

Il faut envisager une autre configuration des tâches dans les cas suivants :

a) un temps de réponse entre les entrées et les sorties numériques inférieur à 25 ms est impératif,

b) nécessité de faire tourner plus de 32 boucles de régulation PID,

c) nécessité de faire tourner des boucles de régulation PID pour les charges dont les constantes primaires de temps sont inférieures à 10 secondes,

d) le séquencement doit répondre à une cadence inférieure à une fois par 100 ms,

e) le programme nécessite un grand nombre de blocs fonctions ou possède un "câblage par soft" complexe qui provoque un débordement d'une des tâches. Cela signifie qu'une tâche ne peut pas terminer l'exécution de tous les blocs fonctions et le "câblage par soft" au cours de la période de scrutation de la tâche.

Il est conseillé de se reporter au chapitre Programmateur de tâches temps réel et à l'annexe D Exemples de configurations de tâches dans la Base du système d'exploitation temps réel PC3000 pour avoir des détails supplémentaires sur la configuration des tâches.

Attention

Une mauvaise configuration des tâches peut avoir un effet préjudiciable sur les performances et l'intégrité du système de régulation PC3000.

TYPES DE BLOCS FONCTIONS

Il est possible de déclarer et d'interconnecter une vaste gamme de types de blocs fonctions différents pour résoudre les problèmes de régulation simples ou complexes. Dans de nombreux cas, on peut ainsi développer des stratégies de régulation entièrement nouvelles.

Pour avoir une description complète des blocs fonctions standard fournis par la station de programmation, se reporter au Manuel des blocs fonctions PC3000.

Les types de blocs fonctions sont organisés en différentes classes pour faciliter la sélection dans le tableau suivant :

Classe	But	Exemples
SYSTEM	Fournit les blocs fonctions qui assurent l'interface avec le système d'exploitation temps réel PC3000, dont l'horloge temps réel et la gestion des tâches	PcsSTATE, Task, RT_Clock
COMMS	Fournit les blocs fonctions du driver de communications pour les protocoles, dont JBus, EI_Bisync et Euro_Panel	EI_Bisync_M, EI_Bisync_S, JBus_M, JBus_S, Euro_Panel
MODULES	Blocs fonctions destinés à la prise en charge des modules matériels intelligents	PPM, PIM2
INPUTS	Blocs fonctions qui représentent les canaux d'entrée. Remarque 1	Digital_In, Analog_In
OUTPUTS	Blocs fonctions qui représentent les canaux de sortie. Remarque 1	Digital_Out, Analog_Out
CONTROL	Blocs fonctions de boucle de régulation, dont le PID et le positionneur de vanne et les versions avec réglage automatique	PID, VP, PID_Auto, VP_Auto
TIMERS	Fournit des blocs fonctions pour différents objectifs de synchronisation, dont la génération d'impulsions, la temporisation et le chronomètre	Pulse_Timer, On_Delay, Off_Delay, Stopwatch

Classe	But	Exemples
USER_VAR	Des blocs fonctions Variables utilisateur sont fournis pour contenir les valeurs internes : tous les types de données, y compris les virgules flottantes, les entiers, les durées et les chaînes, sont disponibles.	Boolean, Real, Integer, Time, String
REMOTE_VARS	Les variables déportées permettent la lecture de différents types de données à partir d'équipements externes utilisant des drivers de communications spécifiques.	Remote_Bool, Remote_Real, Remote_Str
SLAVE_VARS	Les variables esclaves offrent des valeurs qui sont adressables par des protocoles de communications spécifiques, c'est-à-dire par des équipements externes.	Slave_Bool, Slave_Real, Slave_Int, Slave_Real_8
RECIPE	Les blocs fonctions Recettes fournissent des utilitaires de stockage et de sélection des recettes.	RecipeMan, StageMan, Name_1x128, Real_16x128
STEPS	Les blocs fonctions STEP sont créés automatiquement lorsqu'un GRAFCET est développé. Ils sont de deux types : étapes et macros	Macro, Step
COUNTERS	Fournit des blocs fonctions pour compter vers le haut, vers le bas et à la fois vers le haut et le bas	Up_Counter, Dn_Counter, Up_Dn_Count
SELECT	Les blocs fonctions Select permettent la sélection d'un ensemble de valeurs par l'entrée d'un seul entier. Une grande variété de types de données est fournie.	Select_Real, Select_Int, Select_Time, Select_Bool, Select_Str
FILTERS	Fournit des blocs fonctions pour le filtrage de fréquences des signaux analogiques.	Lag1

Classe	But	Exemples
LOADS	Ces blocs fonctions fournissent des simulations de différents types de charges de régulation. Utilisés avec les blocs PID pour simuler les boucles de régulation.	PID_Load, VP_Load
OTHERS	Offre une grande variété de blocs fonctions différents, dont le registre à décalage, la rampe, le pilotage d'alarmes, le bistable	Shift_16, Rate_Limit, Ramp, Shift_Real, Alarm_Cntrl, Bistable_SD

Table 5-1 Classes de types de blocs fonctions

Remarque 1 : les blocs fonctions d'entrée et de sortie de canaux sont créés à l'aide des écrans de configuration matériels.

Affectation des blocs fonctions aux tâches

Bistable_RD Bistable_SD Boolean Debounce_In Digital_In Digital_Out Dn_Counter EI_Bisync_M EI_Bisync_S JBus_M JBus_S Off_Delay On_Delay Pulse_timer Select_Bool Shift_16 Siemens_M_S Stopwatch Toshiba_M Up_Counter Up_Dn_Count	Le tableau 5-2 énumère les blocs fonctions standard affectés par défaut à Task_1. Le reste des blocs fonctions dans la bibliothèque standard est affecté à Task_2. Par défaut, Task_1 est scruté toutes les 10ms, Task_2 toutes les 100ms. Ne pas modifier ces affectations de tâches sans comprendre les répercussions que cela a sur les performances. Avant de modifier la configuration des tâches, il est bon de se reporter au chapitre Programmateur de tâches temps réel et à l'annexe D Exemple de configurations de tâches dans le système d'exploitation temps réel PC3000. Il est toujours possible de voir la tâche à laquelle est affecté un bloc fonction en visualisant le menu Liste des déclarations de blocs fonctions sur la station de programmation.

Table 5-2 Affectation par défaut des blocs fonctions à Task_1

CONSTRUCTION D'UN PROGRAMMATEUR DE RAMPES ET DE PALIERS

Cette section décrit la manière de construire un programmateur de rampes et de paliers à partir de blocs fonctions standard et de le réguler par quelques pas dans un GRAFCET. L'exemple montre un programmateur de rampes et de paliers qui peut comporter un maximum de 16 segments. Chaque segment peut être soit une rampe soit un palier et il est possible de fixer le nombre de segments du programme entre 1 et 16. Au cours d'un segment en rampe, un bloc fonction Rampe sert à incrémenter une sortie jusqu'à la valeur d'un point de consigne avec une vitesse prescrite.

Un segment en palier contient la sortie actuelle du bloc fonction Rampe pour une durée prescrite.

Les pas d'un GRAFCET utilisent des valeurs produites par des blocs fonctions "select" pour configurer le bloc fonction Rampe de manière à ce qu'il crée les fonctions de rampe et de palier. La sortie de la Rampe (dans l'exemple, le paramètre **ramp1.Output**) fournit la valeur rampe/palier qui peut servir à piloter les sorties physiques ou les points de consigne des PID, etc. Par exemple, en "câblant par soft" un certain nombre de points de consigne PID différents avec ce paramètre, il est possible de construire un programmateur multi-zones.

Les techniques décrites peuvent être étendues pour fournir d'autres fonctions comme un plus grand nombre de segments, des paramètres de segments supplémentaires, etc.

La figure 5-3 décrit les blocs fonctions et le "câblage par soft" nécessaires pour créer le programmateur de rampes et de paliers. Certains des paramètres qui n'ont pas d'importance pour cet exemple ne sont pas indiqués pour des raisons de lisibilité.

Nom de la déclaration	Type	But
setpoint	Select_Real	Fournit un point de consigne pour chaque segment.
rate	Select_Real	Fournit une vitesse pour chaque segment.
dwell	Select_Time	Fournit la durée du palier pour un segment en palier.
segno	Integer	Mémorise le numéro de segment actuel.
segmax	Integer	Mémorise le nombre maximal de segments utilisés.
ramp1	Ramp	Fournit la fonction "rampe" en mode RUN.

Table 5-3 Blocs fonctions du programmateur de rampes et de paliers

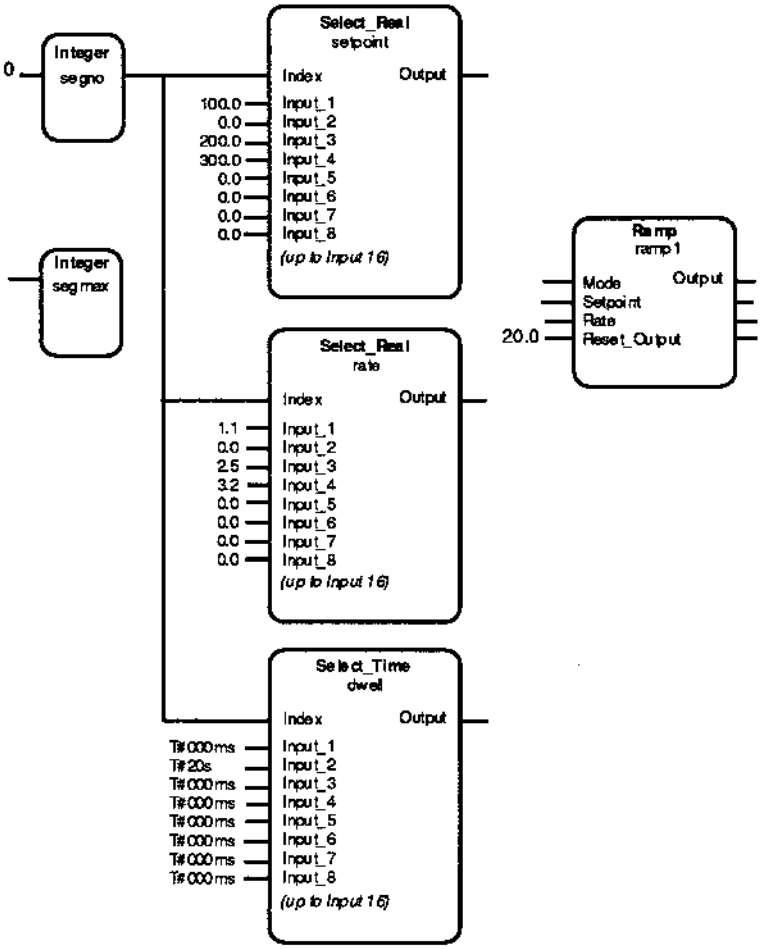


Figure 5-3 Diagramme des blocs fonctions du programmateur de rampes et de paliers

Les affectations de "câblage par soft" en Texte structuré nécessaire sont les suivantes :

```
setpoint.Index := segno.Val;
rate.Index := segno.Val;
dwell.Index := segno.Val;
```

Le paramètre index des fonctions "select" est toujours piloté par la variable utilisateur entière "segno". Si l'on met à 1 un numéro de segment donné dans "segno", les sorties des blocs select fournissent le point de consigne, la vitesse de la rampe et le palier pour un segment donné. Un segment en palier est caractérisé par le bloc fonction de "vitesse" Select_Real qui produit une sortie de 0,0, c'est-à-dire une rampe nulle. Sur la figure 5-3, les blocs fonctions select sont configurés pour avoir 4 segments :

Segment 1 - Rampe jusqu'à 100 à 1,1 unité/seconde

Segment 2 - Palier pendant 20 secondes

Segment 3 - Rampe jusqu'à 200 à 2,5 unités/seconde

Segment 4 - Rampe jusqu'à 300 à 3,2 unités/seconde

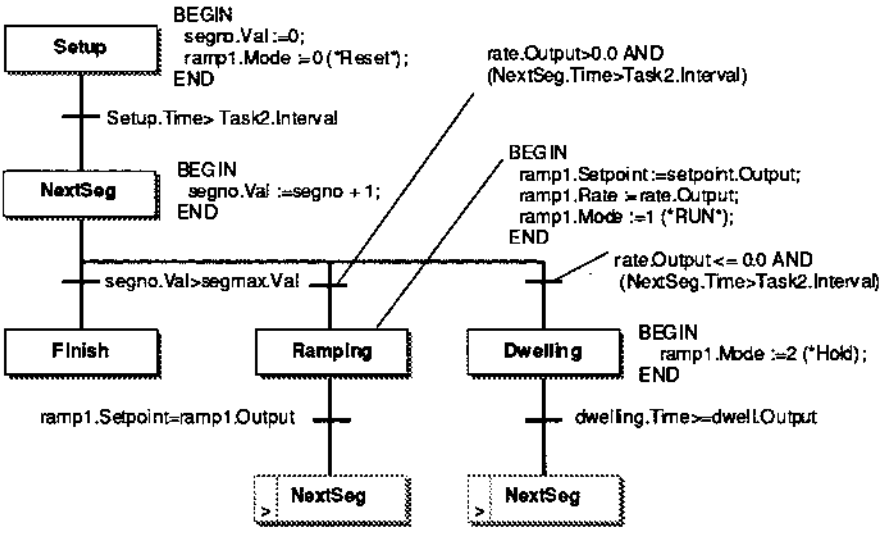


Figure 5-4 GRAFCET de programmateur de rampes et de paliers

La figure 5-4 montre les pas et les transitions nécessaires pour réguler le programmateur de rampes et de paliers et doit être considéré comme faisant partie d'un GRAFCET de plus grande taille.

Le pas "Setup" ramène le numéro de segment à zéro et place le bloc fonction "ramp1" en mode RESET (réinitialisation). A ce stade, le compteur du bloc fonction "ramp1" sera ramené à la valeur du paramètre **Reset_Output** qui, dans cet exemple, a la valeur 20,0.

La transition issue du pas "Setup" attend que le pas soit actif pendant une scrutation de la tâche pour être sûre que les blocs fonctions ont été exécutés avant de continuer. Dans cet exemple, on suppose que le GRAFCET est exécuté dans Task_2 (valeur par défaut normale). Dans ce cas, le pas doit attendre pendant la durée d'une scrutation de Task_2. La durée de scrutation de la tâche est fournie par le paramètre **Task_2.Interval**.

Dans le pas "NextSeg", le numéro de segment contenu dans la variable utilisateur "segno" est incrémenté.

Après "NextSeg", il y a des transitions vers trois pas alternatifs. Ne pas oublier que les transitions sont évaluées de la gauche vers la droite. La transition située la plus à gauche vérifie si le segment final a été atteint en comparant le numéro de segment à la variable utilisateur "segmax".

Les deux autres transitions attendent que le pas "Nextseg" soit actif pendant la durée de scrutation d'une tâche avant de permettre l'exécution des blocs fonctions select pour qu'ils produisent les sorties correctes pour le segment actuel.

La transition médiane vérifie ensuite si le segment sélectionné actuellement possède une vitesse différente de zéro afin de sélectionner le pas "montée en rampe".

La transition située la plus à droite vérifie si la vitesse est différente de zéro afin de sélectionner le pas "palier".

Le pas "montée en rampe" configure le bloc fonction "ramp1" pour incrémenter sa sortie au point de consigne et à la vitesse produits par les blocs fonctions select. Le bloc fonction "ramp1" est ensuite commuté en mode RUN (marche).

La transition issue du pas "Ramping" attend simplement que la sortie du bloc fonction rampe atteigne la valeur du point de consigne puis revienne à "NextSeg" pour le segment suivant.

Le pas "palier" commute le bloc fonction "ramp1" en mode HOLD pour que la sortie reste inchangée.

La transition issue de ce pas attend que le pas "palier" ait une durée égale à la période au cours de laquelle le segment est en palier puis revienne à "NextSeg".

STYLE DE PROGRAMMATION CORRECT

Lors du développement des programmes, il faut prendre en considération les points suivants pour améliorer la lisibilité des programmes et faciliter par conséquent la mise en service et la maintenance du programme :

1. Dans la mesure du possible, utiliser des noms ayant une signification pour les blocs fonctions et les pas.

2. Utiliser des noms ayant une signification pour les valeurs numériques, par exemple "HAUT"/"BAS", "OUVERT/FERME". Cela est particulièrement important si une entrée ou une sortie numérique possède une logique inversée comme par exemple 1 pour "OFF".
3. Insérer des commentaires pour décrire des éléments complexes de Texte structuré.
4. Utiliser une méthode "du haut vers le bas" pour la structure des GRAFCET, le graphique principal décrivant les états primaires du process.
5. Lorsque cela est possible, utiliser des séquences au lieu de machines indiquant l'état à partir de la logique en "câblage par soft" continu. Par exemple, utiliser un pas au lieu de configurer une variable utilisateur booléenne pour définir qu'une machine ou un process se trouve dans un état donné.
6. Eviter d'avoir des constantes en Texte structuré qui peuvent nécessiter une modification lors de la mise en service du système. Utiliser à la place des blocs fonctions Variable utilisateur pour contenir les valeurs constantes. Il sera alors possible de modifier les valeurs à la mise en service du système.
7. Se familiariser avec le potentiel des différents types de blocs fonctions et de fonctions. Dans de nombreux cas, il est possible de résoudre des problèmes complexes avec les capacités existantes.

MISE EN SERVICE DES PROGRAMMES

La station de programmation PC3000 offre un accès en ligne à tous les paramètres des blocs fonctions. Cela permet de contrôler le programme de régulation du PC3000 puisqu'il fonctionne en temps réel. Il est possible de visualiser les différents GRAFCET pour identifier les pas actifs et les étapes macros. On a ainsi une bonne indication de l'état du process ou de la machine.

Suspension de séquence

Si une séquence est suspendue à un pas donné, visualiser le Texte structuré qui définit la condition de chaque transition qui suit le pas. Cela permet de localiser rapidement le motif de la suspension. Identifier l'expression booléenne en Texte structuré qui est censée être "vraie".

Sur la station de programmation PC3000, la valeur réelle de chaque paramètre de bloc fonction utilisé dans l'expression est affichée sur la ligne inférieure de l'écran en Texte structuré Transition.

Si la séquence est suspendue par une étape macro, visualiser la macro de niveau inférieur jusqu'à ce que l'étape incriminée soit localisée. S'il s'agit également d'une étape macro, continuer à descendre dans la hiérarchie du graphique jusqu'à ce que le pas suspendu soit localisé, puis vérifier chacune de ses conditions de transition.

Si une transition de rendez-vous ne provoque pas l'activation de son pas suivant, vérifier que chaque pas qui conduit au rendez-vous, c'est-à-dire qui entre dans les lignes horizontales doubles, est actif.

Ecrans utilisateur

Il est possible de créer des écrans utilisateur sur la station de programmation, lorsque le PC3000 est en marche, pour visualiser les paramètres sélectionnés. Cela permet de visualiser simultanément les paramètres déterminants à des fins de diagnostic et de mise en service. Les écrans utilisateur peuvent également servir à fournir une interface opérateur pour les applications où une interface homme-machine simple convient.

Simulation d'installation

Lors de la mise en service du système, il est peu vraisemblable que toutes les E/S de l'installation soient disponibles pour le test des programmes. Dans certains cas, il peut être peu pratique, voire dangereux, de piloter les sorties physiques. Le PC3000 offre des fonctions permettant de tester les programmes sans qu'ils soient directement reliés à l'installation ou à la machine régulée.

La plupart des canaux d'entrée PC3000 ont un mode test qui permet de remplacer la valeur de l'entrée physique par une valeur de test. Il est également possible de remplacer l'état du canal par un état de test. Par conséquent, il est possible de tester le comportement du programme en simulant différentes valeurs d'entrée de l'installation et les conditions de défaut comme les mesures à l'aide de capteurs de dépassement de gamme. Une partie du programme séquentiel (tournant normalement comme une autre séquence parallèle) peut servir à fournir des valeurs de test multiples aux canaux d'E/S pour former un banc d'essai de simulation de l'installation pour faire fonctionner le programme de régulation. Cette partie du programme peut être éliminée lors de la mise en service définitive du programme.

Test des sorties

En plaçant les blocs fonctions de canaux de sortie en mode test, il est possible de fournir directement des valeurs de test aux sorties physiques, quel que soit l'état du programme normal de régulation PC3000. Cela peut être utile pour tester le câblage physique et la liaison entre les canaux matériels et les actionneurs physiques.

Lorsque les canaux de sortie sont en mode test, les valeurs écrites par le programme dans les blocs fonctions de canaux ne sont pas écrites dans les sorties physiques.

Attention

S'assurer que la totalité des paramètres d'activation de l'ensemble des blocs fonctions de canaux de sortie sont sur "off" (désactivés) dans les programmes PC3000 qui font tourner des process de production.

INDEX

EUROTHERM AUTOMATION SERVICE REGIONAL

SIÈGE SOCIAL
ET USINE
6 chemin des Joncs
BP 55
69572 Dardilly Cedex

Tél. : 04 78 66 45 00
Fax : 04 78 35 24 90

AGENCES

Aix-en-Provence
Tél.: 04 42 39 70 31

Colmar
Tél.: 03 89 23 52 20

Lille
Tél.: 03 20 96 96 39

Lyon
Tél.: 04 78 66 45 10
04 78 66 45 12

Nantes
Tél.: 02 40 30 31 33

Paris
Tél.: 01 69 18 50 60

Toulouse
Tél.: 05 61 71 99 33

BUREAUX

Bordeaux
Clermont-Ferrand
Dijon
Grenoble
Metz
Normandie
Orléans