Langage à contacts (LD)

12

Objet de ce sous-chapitre

Ce chapitre décrit le langage à contacts LD conforme à la norme CEI 611311.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Informations générales sur le langage à contacts (LD)

Présentation

La présente section décrit le langage à contacts (diagramme Ladder) LD selon CEI 61131-3.

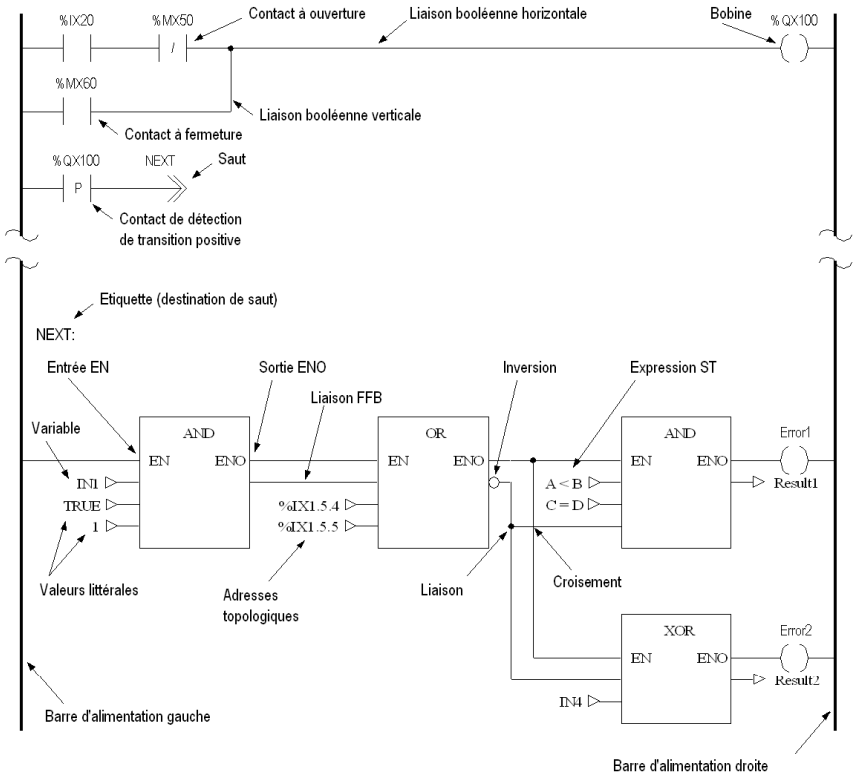
La structure d'une section LD correspond à un rung pour des montages à relais.

Sur le côté gauche de l'éditeur LD, se trouve la barre d'alimentation gauche. Cette barre d'alimentation gauche correspond à la phase (conducteur L) d'un rung. De même que sur un rung, le système ne "traite", lors de la programmation LD, que les objets LD qui sont branchés sur l'alimentation, c'est-à-dire qui sont reliés à la barre d'alimentation gauche. La barre d'alimentation droite correspond au conducteur neutre. Toutes les bobines et sorties FFB y sont reliées directement ou indirectement, ce qui permet d'établir un flux de courant.

Un groupe d'objets reliés les uns aux autres et ne présentant aucune liaison vers d'autres objets (à l'exception de la barre d'alimentation) est appelé réseau ou rung.

Représentation d'une section LD

Représentation :



Objets

Les objets du langage LD offrent des aides permettant de structurer une section en un ensemble de :

- Contacts *(voir page 358)*
- Bobines *(voir page 359)*
- EF et EFB (fonctions élémentaires *(voir page 361)* et blocs fonction élémentaires *(voir page 362)*),
- DFB (blocs fonction dérivés *(voir page 363)*),
- procédures ; *(voir page 363)*
- contrôles *(voir page 372)* et
- blocs d'opération et de comparaison *(voir page 374)* représentant une extension de la norme CEI 61131-3.

Ces objets peuvent être liés les uns aux autres par :

- des liaisons *(voir page 376)* ou
- des paramètres réels *(voir page 364)* (FFB uniquement).

La logique de la section peut être commentée par des objets texte (voir *Objet texte, page 380*).

Taille de la section

Une section LD comprend une fenêtre incluant une seule page.

Cette page est placée sur une grille qui partage la section en lignes et colonnes.

Les sections LD peuvent comporter de 11 à 64 colonnes et de 17 à 2000 lignes.

Le langage LD est basé sur les cellules, c'est-à-dire que seul un objet peut être placé dans chaque cellule.

Ordre d'exécution

L'ordre d'exécution des différents objets dans une section LD est déterminé par le flux de données à l'intérieur de la section. Les réseaux branchés sur la barre d'alimentation gauche sont traités de haut en bas (liaison avec la barre d'alimentation gauche). Les réseaux indépendants les uns des autres à l'intérieur de la section sont traités dans l'ordre de placement (de haut en bas) (voir également *Ordre d'exécution et flux de signaux, page 390*).

Conformité CEI

Pour plus d'informations sur la conformité CEI du langage LD, voir Conformité CEI *(voir page 657)*.

Contacts

Présentation

Un contact est un élément LD permettant de transférer un état de la liaison horizontale vers la droite. Cet état est le résultat d'une opération booléenne AND sur l'état de la liaison horizontale de gauche avec l'état du paramètre booléen réel associé.

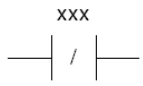
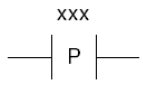
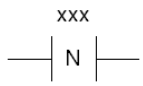
Un contact ne modifie pas la valeur du paramètre réel associé.

Les contacts occupent une cellule.

Sont autorisés comme paramètres réels :
- Variables booléennes
- Constantes booléennes
- Adresses booléennes (adresses topologiques ou symboliques)
- Expression ST *(voir page 509)* avec résultat booléen (par ex. `VarA OR VarB`)
  Les expressions ST comme paramètres réels de contacts représentent une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.

Types de contacts

Les contacts disponibles sont les suivants :

Désignation	Représentation	Description
A fermeture		Dans le cas de contacts à fermeture, l'état de la liaison de gauche est transféré vers la liaison de droite si l'état du paramètre booléen réel associé (indiqué par xxx) est ON. Sinon, l'état de la liaison de droite est OFF.
A ouverture		Dans le cas de contacts à ouverture, l'état de la liaison de gauche est transféré vers la liaison de droite si l'état du paramètre booléen réel approprié (indiqué par xxx) est OFF. Sinon, l'état de la liaison de droite est OFF.
Contact de détection de transitions positives		Dans le cas de contacts de détection de transitions positives, la liaison de droite est ON pour un cycle de programme, si un passage de OFF à ON du paramètre réel booléen (xxx) associé a lieu et qu'en même temps, l'état de la liaison de gauche est ON. Sinon, l'état de la liaison de droite est 0. Voir aussi *Reconnaissance de front, page 381*.
Contact de détection de transitions négatives		Dans le cas de contacts de détection de transitions négatives, la liaison de droite est ON pour un cycle de programme, si un passage de ON à OFF du paramètre réel booléen (xxx) associé a lieu et qu'en même temps, l'état de la liaison de gauche est ON. Sinon, l'état de la liaison de droite est 0. Voir aussi *Reconnaissance de front, page 381*.

Bobines

Présentation

Une bobine est un élément LD permettant de transférer l'état de la liaison horizontale sur la gauche, inchangée, vers la liaison horizontale sur la droite. L'état est stocké dans le paramètre booléen réel respectif.

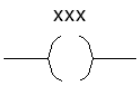
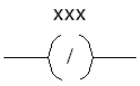
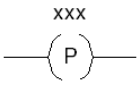
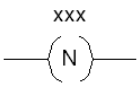
Normalement, les bobines suivent des contacts ou des FFB, mais elles peuvent aussi être suivies par des contacts.

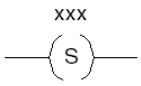
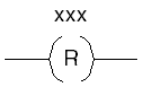
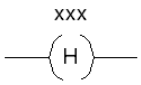
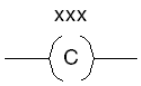
Les bobines occupent une cellule.

Sont autorisés comme paramètres réels :
- Variables booléennes
- Adresses booléennes (adresses topologiques ou symboliques)

Types de bobines

Les bobines suivantes sont disponibles :

Désignation	Représentation	Description
Bobine		Dans le cas de bobines, l'état de la liaison de gauche est transféré vers le paramètre booléen réel associé (indiqué par xxx) et la liaison de droite.
bobine inverse		Dans le cas de bobines inverses, l'état de la liaison de gauche est copié sur la liaison de droite. L'état inversé de la liaison de gauche est copié vers le paramètre booléen réel associé (indiqué par xxx). Si la liaison de gauche est OFF, alors la liaison de droite sera également OFF et le paramètre booléen réel associé sera ON.
Bobine de détection de transitions positives		Dans le cas de bobines de détection de transitions positives, l'état de la liaison de gauche est copié sur la liaison de droite. Le paramètre réel associé du type de données EBOOL (indiqué par xxx) est 1 pour un cycle de programme, si un passage de 0 à 1 de la liaison gauche est effectué. Voir aussi *Reconnaissance de front, page 381*.
Bobine de détection de transitions négatives		Dans le cas de bobines de détection de transitions négatives, l'état de la liaison de gauche copié sur la liaison de droite. Le paramètre réel booléen associé (indiqué par xxx) est 1 pour un cycle de programme, si un passage de 1 à 0 de la liaison gauche est effectué. Voir aussi *Reconnaissance de front, page 381*.

Désignation	Représentation	Description
Bobine d'enclenchement		Avec une bobine d'enclenchement, l'état de la liaison de gauche est copié sur la liaison de droite. Le paramètre booléen réel associé (indiqué par xxx) est défini sur ON si l'état de la liaison de gauche est ON, sinon il reste inchangé. Le paramètre booléen réel associé peut être réinitialisé via la bobine de réinitialisation. Voir aussi *Reconnaissance de front, page 381*.
Bobine de réinitialisation		Avec une bobine de réinitialisation, l'état de la liaison de gauche est copié sur la liaison de droite. Le paramètre booléen réel associé (indiqué par xxx) est défini sur OFF si l'état de la liaison de gauche est ON, sinon il reste inchangé. Le paramètre booléen réel associé peut être enclenché via la bobine d'enclenchement. Voir aussi *Reconnaissance de front, page 381*.
Bobine d'arrêt		Avec des bobines d'arrêt, si le statut de la liaison de gauche est 1, l'exécution du programme est arrêtée immédiatement. (Avec des bobines d'arrêt, l'état de la liaison de gauche n'est pas copié sur la liaison de droite.)
Bobine d'appel		Avec des bobines d'appel, l'état de la liaison de gauche est copié vers la liaison de droite. Si l'état de la liaison de gauche est ON alors le sous-programme associé (indiqué par xxx) est appelé. Le sous-programme à appeler doit se trouver dans la même tâche que la section LD appelante. Il est possible d'appeler des sous-programmes au sein de sous-programmes. Les sous-programmes sont un complément de la norme CEI 61131-3 et doivent être activés de manière explicite. Dans les sections d'actions SFC, les bobines d'appel (appels de sous-programmes) ne sont autorisés que si le mode Multitoken a été activé.

Fonctions élémentaires, blocs fonction élémentaires, blocs fonction dérivés et procédures (FFB)

Introduction

FFB est le terme générique pour :

- les fonctions élémentaires (EF) *(voir page 361)*
- les blocs fonction élémentaires (EFB) *(voir page 362)*
- les blocs fonction dérivés (DFB) *(voir page 363)*
- Procédure *(voir page 363)*

Les FFB occupent une largeur de 1 à 3 colonnes (en fonction de la longueur des noms des paramètres formels) et une longueur de 2 à 33 lignes (en fonction du nombre de lignes des paramètres formels).

Fonction élémentaire

Les fonctions n'ont pas d'état interne. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie est la même à chaque exécution de la fonction. Par exemple, l'addition de deux valeurs donne toujours le même résultat.

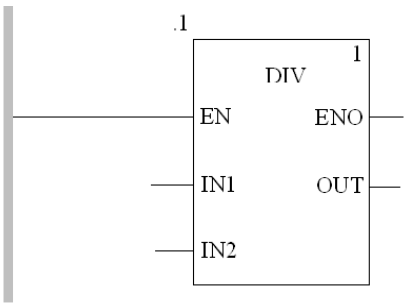
Une fonction élémentaire est représentée graphiquement comme un cadre avec des entrées et une sortie. Les entrées sont toujours représentées sur la gauche et la sortie toujours sur la droite du cadre.

Le nom de la fonction, c'est-à-dire le type de fonction, est affiché au centre du cadre.

Le numéro d'exécution *(voir page 390)* de la fonction apparaît à droite du type de fonction.

Le numéro de fonction est affiché au-dessus du cadre. Le numéro de fonction représente le numéro courant de la fonction dans la section actuelle. Les numéros de fonction ne peuvent pas être modifiés.

Fonction élémentaire



Pour certaines fonctions élémentaires, il est possible d'augmenter le nombre d'entrées.

Bloc fonction élémentaire

Les blocs fonction élémentaires ont des états internes. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie peut être différente pour toutes les exécutions de la fonction. Par exemple, pour un compteur, la valeur de sortie augmente.

Un bloc fonction élémentaire est représenté graphiquement sous forme de cadre avec des entrées et des sorties. Les entrées sont toujours représentées sur la gauche et les sorties toujours sur la droite du cadre. Le nom du bloc fonction, c'est-à-dire le type de bloc fonction, est affiché au centre du cadre. Le nom d'instance est affiché au-dessus du cadre.

Les blocs fonction peuvent avoir plusieurs sorties.

Le nom du bloc fonction, c'est-à-dire le type de bloc fonction, est affiché au centre du cadre.

Le numéro d'exécution *(voir page 390)* du bloc fonction apparaît à droite du type de bloc fonction.

Le nom d'instance est affiché au-dessus du cadre.

Le nom d'instance permet d'identifier précisément le bloc fonction dans un projet.

Le nom d'instance est généré automatiquement et présente la structure suivante : `FBI_n`

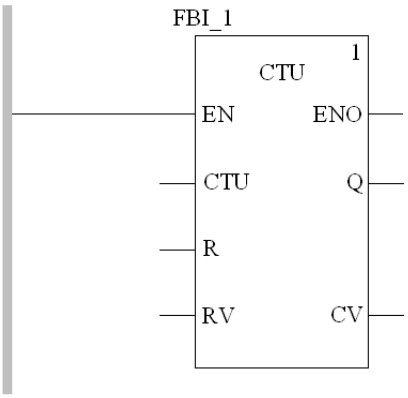
`FBI` = Instance de bloc fonction

`n` = numéro courant du bloc fonction au sein du projet

Vous pouvez modifier ces noms générés automatiquement pour rendre la vue d'ensemble plus claire. Le nom d'instance (32 caractères maximum) doit être unique dans tout le projet ; aucune distinction n'est faite ici entre majuscules et minuscules. Le nom d'instance doit respecter les conventions de noms générales.

NOTE : conformément à la norme CEI 61131-3, les noms d'instance doivent commencer par une lettre. Si vous voulez également utiliser des chiffres comme premier caractère, vous devez activer cette fonction.

Bloc fonction élémentaire

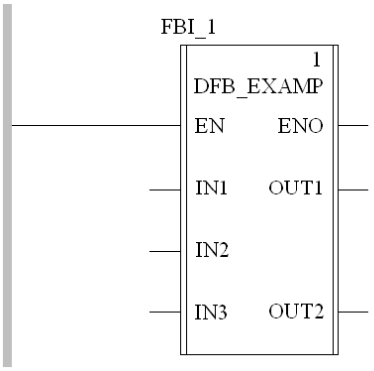


DFB

Les blocs fonction dérivés (DFB) ont les mêmes caractéristiques que les blocs fonction élémentaires. Ils sont cependant créés par l'utilisateur dans les langages FBD, LD, IL et/ou ST.

L'unique différence par rapport aux blocs fonction élémentaires est que le bloc fonction dérivé est représenté graphiquement sous forme de cadre avec deux lignes verticales.

Bloc fonction dérivé



Procédure

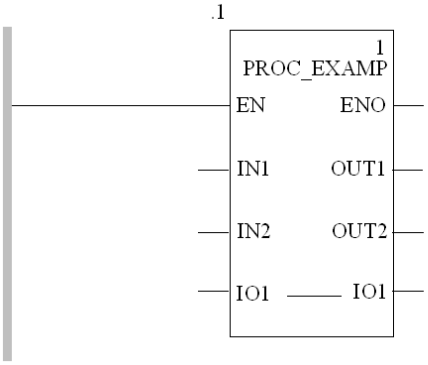
Techniquement, les procédures sont des fonctions.

L'unique différence par rapport aux fonctions élémentaires est que les procédures peuvent comprendre plus d'une sortie et qu'elles prennent en charge le type de données `VAR_IN_OUT`.

Il n'y a pas de différence physique entre les procédures et les fonctions élémentaires.

Les procédures sont une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.

Procédure

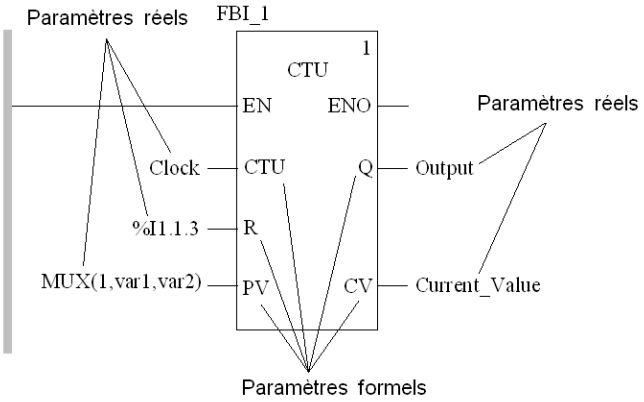


Paramètres

Pour importer des valeurs dans le FFB ou exporter des valeurs du FFB, des entrées et des sorties sont nécessaires. Elles sont appelées paramètres formels.

Les paramètres formels sont liés à des objets qui comprennent les états courants du traitement. Ces états sont appelés paramètres réels.

Paramètres formels et réels :



Durant l'exécution du programme, les valeurs sont transmises, par le biais des paramètres réels, du processus au FFB, et renvoyées à nouveau à la sortie après le traitement.

Seul un objet (paramètre réel) du type de données suivant peut être relié aux entrées FFB :

- contact
- variable
- adresse
- libellé
- expression ST
  Les expressions ST des entrées FFB représentent une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.
- Liaison

Les combinaisons d'objets (paramètres réels) suivantes peuvent être reliées aux sorties FFB :

- une ou plusieurs bobines
- un ou plusieurs contacts
- une variable
- une variable et une ou plusieurs liaisons (non valable pour les sorties VAR_IN_OUT *(voir page 371)*)
- une adresse,
- une adresse et une ou plusieurs liaisons (non valable pour les sorties VAR_IN_OUT *(voir page 371)*)
- une ou plusieurs liaisons (non valable pour les sorties VAR_IN_OUT *(voir page 371)*)

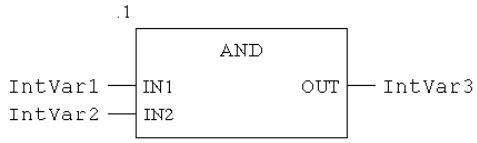
Le type des données de l'objet à relier doit correspondre au type des données de l'entrée/la sortie FFB. On choisira un type de données adapté pour le bloc fonction, si tous les paramètres réels sont constitués de valeurs littérales.

Exception : pour les entrées/sorties génériques FFB de type de données `ANY_BIT`, des objets de type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être reliés.

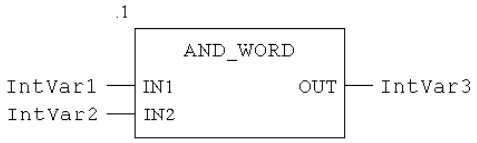
Il s'agit d'une extension de la norme CEI 61131-3 et doit donc être activée de manière explicite.

Exemple :

Autorisé :



Non autorisé :



(Dans ce cas `AND_INT` doit être utilisé.)

Il n'est en principe pas nécessaire d'affecter un paramètre réel à tous les paramètres formels. Cependant, cela n'est pas valable pour les broches inversées. Un paramètre réel doit toujours leur être affecté. Cela est également impératif pour certains types de paramètre formel. Pour connaître les types concernés, veuillez vous reporter au tableau suivant.

Tableau des types de paramètre formel :

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB : Entrée	-	+	+	+	/	+	/	+
DFB : Sortie	-	-	+	/	/	-	/	+
EFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB : Entrée	-	+	+	+	/	+	/	+
DFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB : Sortie	-	-	+	+	+	-	/	+
EF : Entrée	-	-	+	+	+	+	+	+
EF : VAR_IN_OUT	+	+	+	+	+	+	/	+

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EF : Sortie	-	-	-	-	-	-	/	-
Procédure : Entrée	-	-	+	+	+	+	+	+
Procédure : VAR_IN_OUT	+	+	+	+	+	+	/	+
Procédure : Sortie	-	-	-	-	-	-	/	+
+ paramètre réel impératif								
- Paramètre réel non impératif								
/ Non applicable								

Les FFB utilisant aux entrées des paramètres réels, auxquels aucune valeur n'a encore été affectée, fonctionnent avec les valeurs initiales de ces paramètres réels.

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Si aucune valeur n'est affectée à un paramètre formel et que le bloc fonction/DFB a été instancié à plusieurs reprises, les instances appelées par la suite travaillent avec l'ancienne valeur.

Variables publiques

Certains blocs fonction disposent non seulement d'entrées et de sorties, mais également de variables publiques (Public Variables).

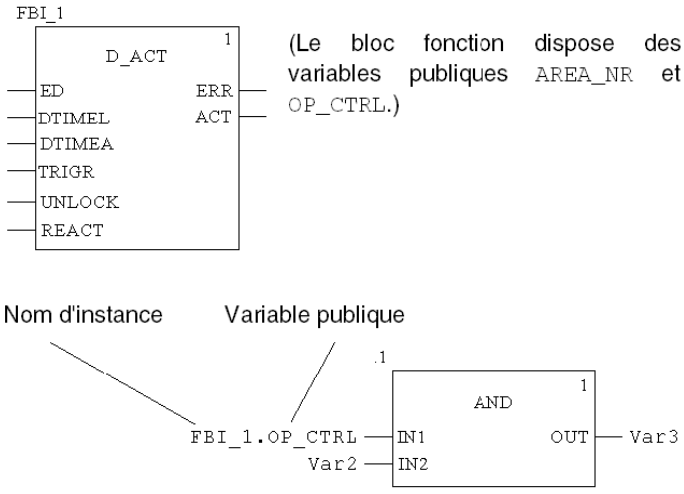
Ces variables permettent de transmettre des valeurs statiques (valeurs non influencées par le procédé) au bloc fonction. Elles sont donc utilisées lors du paramétrage du bloc fonction.

Les variables publiques sont une extension de la norme CEI 61131-3.

Les valeurs sont affectées aux variables publiques via leur valeur initiale.

Les valeurs des variables publiques sont ensuite lues à partir du nom d'instance du bloc fonction et du nom de la variable publique.

Exemple :



Variables privées

Certains blocs fonction disposent non seulement d'entrées, de sorties et de variables publiques, mais également de variables privées.

A l'instar des variables publiques, ces variables permettent de transmettre des valeurs statistiques (valeurs non influencées par le procédé) au bloc fonction.

Le programme utilisateur ne peut pas accéder à ces variables. Seule la table d'animation en a la capacité.

NOTE : les DFB imbriqués sont déclarés comme des variables privées du DFB parent. Ainsi, leurs variables ne sont pas accessibles via la programmation, mais via la table d'animation.

Les variables privées sont une extension de la norme CEI 61131-3.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

- Les FFB ne sont traités que lorsqu'ils sont connectés directement ou indirectement à la barre d'alimentation gauche.
- Si le FFB doit être exécuté de façon conditionnelle, l'entrée EN peut être préalablement reliée par des contacts ou d'autres FFB (voir également EN et ENO *(voir page 368)*).
- Les entrées et sorties booléennes peuvent être inversées.
- Des conditions particulières s'appliquent lors de l'utilisation de variables VAR_IN_OUT *(voir page 371)*.
- Les instances de DFB ou bloc fonction peuvent être appelées à plusieurs reprises (voir aussi Appel multiple d'une instance de bloc fonction *(voir page 368)*).

Appel multiple d'une instance de bloc fonction

Les instances de DFB ou bloc fonction peuvent être appelées à plusieurs reprises, à l'exception des instances d'EFB de communication et blocs fonction/DFB ayant une sortie ANY mais pas d'entrée ANY, qui ne peuvent être appelées qu'une seule fois.

L'appel multiple de la même instance de DFB/bloc fonction est par exemple utile dans les cas suivants :

- si le bloc fonction/DFB ne comporte aucune valeur interne ou si celle-ci n'est plus nécessaire pour un traitement ultérieur.
  Dans ce cas, l'appel multiple de la même instance de DFB/bloc fonction permet d'économiser de l'espace mémoire, car le code du bloc fonction/DFB n'est alors chargé qu'une seule fois.
  Le bloc fonction/DFB est pour ainsi dire traité comme une "fonction".
- si le bloc fonction/DFB comprend des valeurs internes et que celles-ci doivent être influencées à différents endroits du programme, la valeur d'un compteur, par exemple, doit être augmentée à différents endroits du programme.
  Dans ce cas, l'appel multiple de la même instance de bloc fonction/DFB permet d'économiser la mémoire des résultats intermédiaires pour un traitement ultérieur à un autre endroit du programme.

EN et ENO

Une entrée EN et une sortie ENO peuvent être configurées pour tous les FFB.

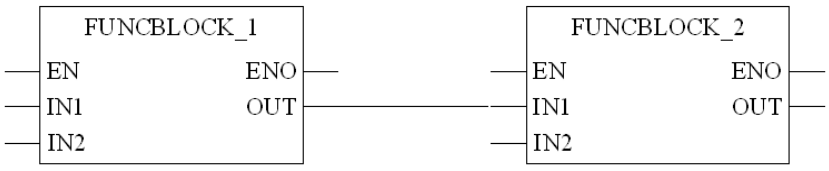
Si la valeur de EN est déjà à "0", lors de l'appel de FFB, les algorithmes définis par FFB ne sont pas exécutés et ENO est réglé sur 0.

Si la valeur de EN est déjà à "1", lors de l'appel de FFB, les algorithmes définis par FFB sont exécutés. Après l'exécution sans erreur de ces algorithmes, la valeur de ENO est mise à "1". Si une erreur se produit durant l'exécution de ces algorithmes, ENO est réglé sur 0.

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »). Reportez-vous à la section *Maintenir les liens de sortie sur les EF désactivés (voir Unity Pro, Modes de marche,)*.

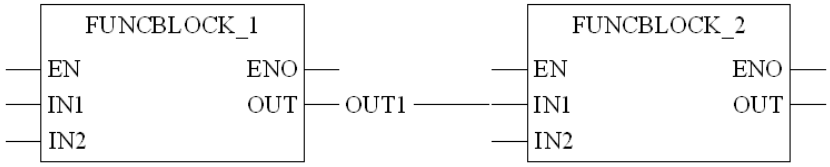
Si ENO est réglé sur 0 (car EN = 0 ou en raison d'une erreur d'exécution) :

- Blocs fonction
  - Traitement EN/ENO pour les blocs fonction ayant (seulement) une liaison comme paramètre de sortie :



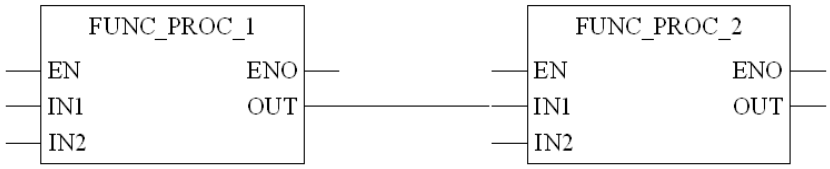
Lorsque EN est réglé sur 0 par FUNCBLOCK_1, la liaison à la sortie OUT de FUNCBLOCK_1 conserve l'ancien statut qu'elle avait lors du dernier cycle correct.

- Traitement EN/ENO pour les blocs fonction ayant une variable et une liaison comme paramètre de sortie :



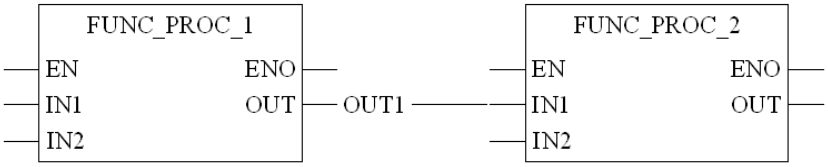
Lorsque EN est réglé sur 0 par FUNCBLOCK_1, la liaison à la sortie OUT de FUNCBLOCK_1 conserve l'ancien statut qu'elle avait lors du dernier cycle correct. La variable OUT1 située sur la même broche conserve son ancien statut ou peut être modifiée depuis l'extérieur sans avoir d'influence sur la liaison. La variable et la liaison sont enregistrées indépendamment l'une de l'autre.

- Fonctions/Procédures
  Selon la définition CEI 61131-3, les sorties de fonctions désactivées (entrée EN mise à "0") sont indéfinies. (Le même principe s'applique aux procédures.)
  Voici, néanmoins, une explication des états des sorties dans un tel cas :
  - Traitement EN/ENO pour les fonctions/procédures ayant (seulement) une liaison comme paramètre de sortie :



Si EN de FUNC_PROC_1 est réglé sur 0, la valeur de la liaison sur la sortie OUT de FUNC_PROC_1 dépend du paramètre de projet **Maintenir les liens de sortie sur les EF désactivés**, disponible depuis Unity Pro 4.1.
Si ce paramètre de projet est réglé sur 0, la valeur de la liaison est réglée sur 0.
Si ce paramètre de projet est réglé sur 1, la liaison conserve la valeur qu'elle avait lors du dernier cycle exécuté correctement.
Pour plus d'informations, consultez la section *Maintenir les liens de sortie sur les EF désactivés (voir Unity Pro, Modes de marche,)*.

- Traitement EN/ENO pour les fonctions/procédures ayant une variable et une liaison comme paramètre de sortie :



Si `EN` de `FUNC_PROC_1` est réglé sur 0, la valeur de la liaison sur la sortie `OUT` de `FUNC_PROC_1` dépend du paramètre de projet **Maintenir les liens de sortie sur les EF désactivés**, disponible depuis Unity Pro 4.1.
Si ce paramètre de projet est réglé sur 0, la valeur de la liaison est réglée sur 0.
Si ce paramètre de projet est réglé sur 1, la liaison conserve la valeur qu'elle avait lors du dernier cycle exécuté correctement.
Pour plus d'informations, consultez la section *Maintenir les liens de sortie sur les EF désactivés (voir Unity Pro, Modes de marche,)*.
La variable `OUT1` située sur la même broche conserve son ancien statut ou peut être modifiée depuis l'extérieur sans avoir d'influence sur la liaison. La variable et la liaison sont enregistrées indépendamment l'une de l'autre.

Le comportement aux sorties des FFB est indépendant du fait que les FFB soient appelés sans `EN`/`ENO` ou avec `EN` = 1.

NOTE : pour les blocs fonction désactivés (`EN` = 0) équipés d'une fonction d'horloge interne (par exemple, le bloc fonction `DELAY`), le temps semble continuer de s'écouler, car il est calculé à l'aide d'une horloge système et est, par conséquent, indépendant du cycle du programme et de la libération du bloc.

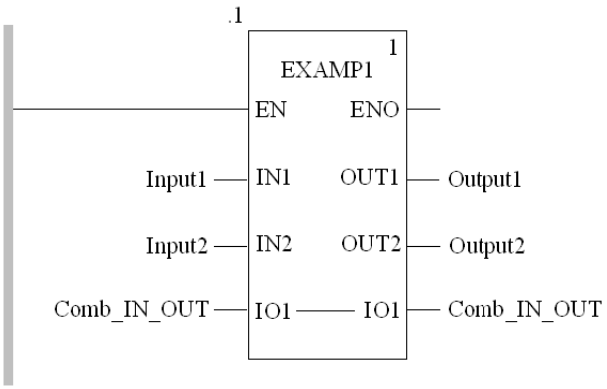
Variable VAR_IN_OUT

Les FFB sont souvent utilisés pour lire une variable à l'entrée (variables d'entrée), pour la traiter et pour transmettre de nouveau les valeurs modifiées de cette **même** variable (variables de sortie).

Ce cas exceptionnel d'une variable d'entrée/de sortie est également appelé variable `VAR_IN_OUT`.

Dans le FFB, une ligne indique que les variables d'entrée et de sortie sont liées l'une à l'autre.

Variable `VAR_IN_OUT`



Il convient de noter les particularités suivantes en cas d'utilisation de FFB avec des variables `VAR_IN_OUT` :

- une variable doit être affectée à toutes les entrées `VAR_IN_OUT`.
- les liaisons graphiques permettent uniquement de relier des sorties `VAR_IN_OUT` à des entrées `VAR_IN_OUT`.
- seule une liaison graphique unique peut être reliée à une entrée/sortie `VAR_IN_OUT`.
- une combinaison de variables/d'adresses et de liaisons graphiques n'est pas possible pour les sorties `VAR_IN_OUT`.
- il est interdit de relier des valeurs littérales ou des constantes à des entrées/sorties `VAR_IN_OUT`.
- il est interdit d'utiliser des négations au niveau des entrées/sorties `VAR_IN_OUT`.
- des variables/composantes de variables différentes peuvent être reliées à l'entrée `VAR_IN_OUT` et à la sortie `VAR_IN_OUT`. Dans un tel cas, la valeur de la variable/composante de variable à l'entrée est copiée dans la variable/composante de variable à la sortie.

Contrôles

Introduction

Les éléments de commande servent à l'exécution de sauts au sein d'une section LD et au retour prématuré dans le programme principal depuis un sous-programme (SRx) ou un bloc fonction dérivé (DFB).

Les éléments de commande occupent une cellule.

Contrôles

Les contrôles suivants sont disponibles.

Désignation	Représentation	Description
Jump		Si l'état de la liaison gauche est 1, un saut est exécuté jusqu'à l'étiquette (dans la section courante). Pour générer un saut inconditionnel, l'objet saut est placé directement sur la barre d'alimentation gauche. Pour générer un saut conditionnel, l'objet saut est placé à la fin d'une rangée de contacts.
Libellé	`LABEL:`	Les repères (destinations de saut) sont représentés comme du texte avec deux-points à la fin. Le texte est limité à 32 caractères et doit être unique dans l'ensemble de la section. Le texte doit respecter les conventions de nommage générales. Les étiquettes de saut ne peuvent être placées que dans la première cellule directement sur la barre d'alimentation gauche. **Note :** Les étiquettes de saut ne doivent "couper" aucun réseau, c'est-à-dire qu'une ligne imaginaire entre l'étiquette de saut et la marge droite de la section ne doit être coupée par aucun objet. Cela s'applique également aux liaisons booléennes et FFB.

Désignation	Représentation	Description

Return		Des objets `RETURN` ne peuvent pas être utilisés dans le programme principal.
		● Dans un DFB, un objet `RETURN` force le retour au programme qui a appelé le DFB.
		  ● Le reste de la section de DFB contenant l'objet `RETURN` n'est pas exécuté.
		  ● Les sections suivantes du DFB ne sont pas exécutées.
		  Le programme qui a appelé le DFB sera exécuté après retour du DFB.
		  Si le DFB est appelé par un autre DFB, le DFB appelant sera exécuté après le retour.
		● Dans un SR, un objet `RETURN` force le retour au programme qui a appelé le SR.
		  ● Le reste du SR contenant l'objet `RETURN` n'est pas exécuté.
		  Le programme qui a appelé le SR sera exécuté après retour du SR.

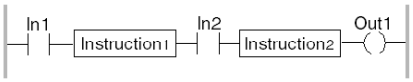
Blocs opération et blocs comparaison

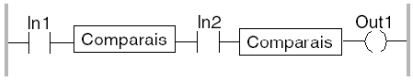
Introduction

En plus des objets définis dans la norme CEI 61131-3, il existe d'autres blocs servant à l'exécution d'instructions ST *(voir page 509)* et d'expressions ST *(voir page 509)* et à des opérations de comparaison simples. Ces blocs sont exclusivement disponibles dans le langage de programmation LD.

Objets

Les objets suivants sont disponibles :

Désignation	Représentation	Description
Bloc opération		Si l'état de la liaison gauche est 1, l'instruction ST comprise dans le bloc est exécutée. Toutes les instructions ST *(voir page 509)* sont permises **sauf** les instructions de commande : • (`RETURN`, • `JUMP`, `IF`, • `CASE`, • `FOR` • etc.) Pour les blocs opération, quel que soit le résultat de l'instruction ST, l'état de la liaison gauche est transmis à la liaison droite. Un bloc peut contenir jusqu'à 4 096 caractères. Si tous les caractères ne peuvent pas être affichés, les premiers caractères seront affichés suivis de points de suspension (...). Un bloc opération occupe 1 ligne et 4 colonnes. Exemple :  Dans l'exemple, `Instruction1` est exécutée si `In1`=1. `Instruction2` est exécutée si `In1`=1 et `In2`=1 (le résultat de `Instruction1` ne joue aucun rôle pour l'exécution de `Instruction2`). Out1 est 1, si `In1`=1 et `In2`=1 (les résultats de `Instruction1` et `Instruction2` n'influent pas sur l'état de `Out1`).

Désignation	Représentation	Description
Bloc de comparaison horizontal		Les blocs de comparaison horizontaux servent à exécuter une expression de comparaison (<, >, <=, >=, =, <>) dans le langage de programmation ST. (Remarque : La même fonctionnalité est également disponible via les expressions ST *(voir page 509)*.) Un bloc comparaison exécute un ET de sa broche d'entrée gauche et du résultat de sa condition de comparaison, puis affecte le résultat du ET de façon inconditionnelle à la broche de sortie droite. Par exemple, lorsque l'état de la liaison gauche est 1 et que le résultat de la comparaison est 1, l'état de la liaison droite est 1. Un bloc de comparaison horizontal peut contenir jusqu'à 4 096 caractères. Si tous les caractères ne peuvent pas être affichés, les premiers caractères seront affichés suivis de points de suspension (...). Un bloc de comparaison horizontal occupe une ligne et deux colonnes. Exemple :  Dans l'exemple, `Comparaison1` est exécutée si `In1`=1. `Comparaison2` est exécutée si `In1`=1 , `In2`=1, résultat de `Comparaison1`=1. Out1 est 1, si`In1`=1, `In2`=1, le résultat de `Comparaison1`=1 et le résultat de `Comparaison2`=1.

Liaisons

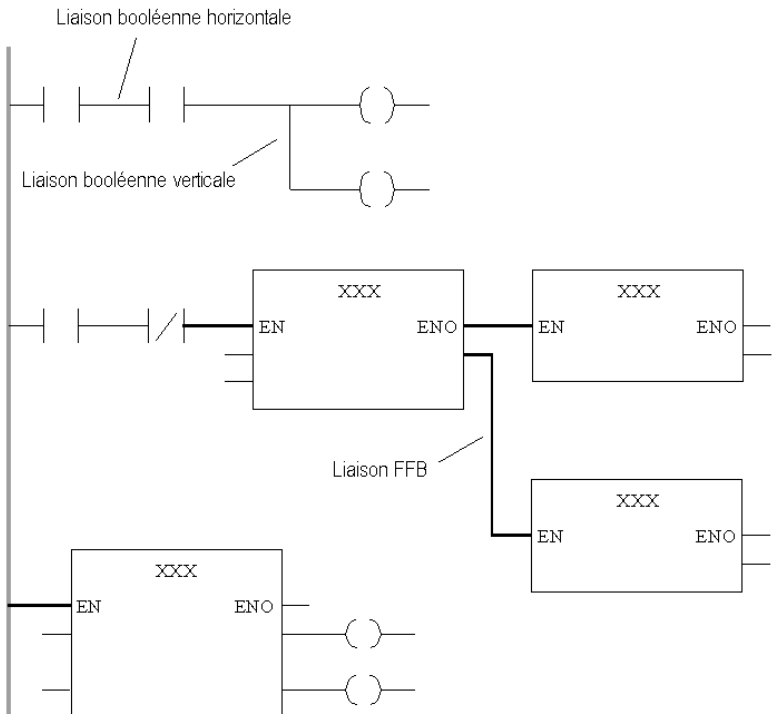
Description

Les liaisons sont des liens entre des objets LD (contacts, bobines, FFB, etc.).

Une différence est faite entre deux types de liaison.

- Liaison booléenne
  Les liaisons booléennes comprennent un ou plusieurs segments qui relient entre eux des objets booléens (contacts, bobines).
  Pour les liaisons booléennes, une différence est faite entre :
  - Les liaisons booléennes horizontales
    Les liaisons booléennes horizontales permettent une liaison en série de contacts et bobines.
  - Les liaisons booléennes verticales
    Les liaisons booléennes verticales permettent une liaison en parallèle de contacts et bobines.
- Liaisons FFB
  Les liaisons FFB comprennent une combinaison de segments horizontaux et verticaux qui relient les entrées/sorties FFB avec d'autres objets.

Liaisons :



Remarques générales sur la programmation

Veuillez observer les remarques générales qui suivent sur la programmation :

- Les types de données respectifs des entrées/sorties à relier doivent correspondre les uns aux autres.
- Les liaisons entre des paramètres de longueur variable (ex : `ANY_ARRAY_INT`) ne sont pas permises.
- Plusieurs liaisons peuvent être reliées à une entrée (côté droit d'un contact/d'une bobine, sortie FFB). Cependant une seule liaison est possible avec chaque entrée (côté gauche d'un contact/d'une bobine, entrée FFB).
- Les contacts, bobines et entrées de FFB non reliés obtiennent par défaut la valeur "0".
- Les boucles ne peuvent pas être configurées par le biais de liaisons, étant donné que, dans ce cas, l'ordre d'exécution dans la section ne peut pas être défini de façon unique. Les boucles doivent être résolues par le biais de paramètres réels (voir *Boucles non permises, page 392*).

Remarques sur la programmation de liaisons booléennes

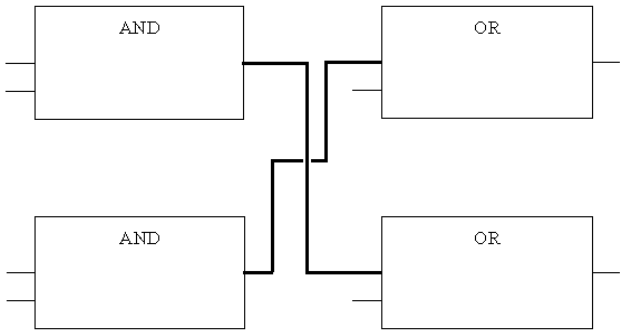
Remarques sur la programmation de liaisons booléennes :

- Le chevauchement des liaisons booléennes avec d'autres objets **n'est pas** admis.
- Le flux de signaux (passage de courant) d'une liaison booléenne va de gauche vers à droite. C'est pourquoi les liaisons dirigées vers l'arrière ne sont pas autorisées.
- Si deux liaisons booléennes se croisent, un lien entre les deux liaisons est automatiquement créé. Etant donné que le croisement de liaisons booléennes n'est pas possible, les liaisons ne sont pas identifiées de manière particulière.

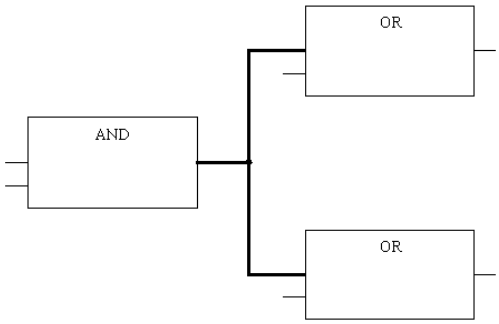
Remarques sur la programmation des liaisons FFB

Remarques sur la programmation des liaisons FFB :

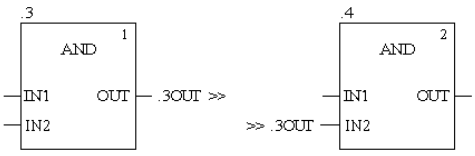
- Au moins un côté d'une liaison FFB doit être relié à une entrée ou une sortie FFB.
- Les liaisons FFB sont représentées par un trait double afin de les différencier des liaisons booléennes.
- Le flux de signaux (passage de courant) d'une liaison FFB va de la sortie FFB vers l'entrée FFB, indépendamment de la direction. C'est pourquoi les liaisons dirigées vers l'arrière sont autorisées.
- Seules des entrées FFB et des sorties FFB peuvent être reliées ensemble. La liaison de plusieurs sorties FFB entre elles n'est pas possible. Cela signifie qu'en LD aucune liaison OU n'est possible via des liaisons FFB.
- Le chevauchement des liaisons FFB avec d'autres objets est admis.
- Le croisement de liaisons FFB est admis. Les croisements sont représentés par une liaison interrompue.



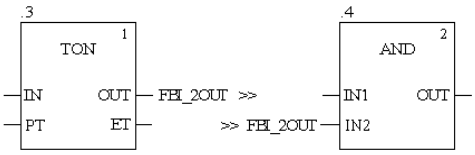
- Dans le cadre de liaisons FFB, les points de liaison entre plusieurs liaisons FFB sont représentés par un cercle rempli.



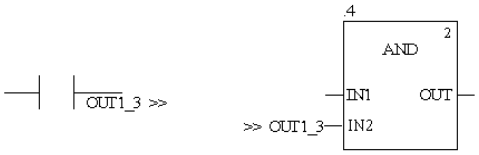
- Afin d'éviter le croisement de liaisons, les liaisons FFB peuvent également être représentées sous forme de connecteurs.
  A cette occasion, la source et la cible de la liaison FFB sont caractérisées par un nom unique au sein de la section.
  Suivant le type d'objet source, le nom du connecteur est formé comme suit :
  - pour les fonctions : " numéro de fonction/paramètre formel " de la source de la liaison.



  - pour les blocs fonction : " nom d'instance/paramètre formel " de la source de la liaison.



- Pour les contacts : "OUT1_numéro courant".



Liaisons verticales

La "liaison verticale" constitue un cas particulier de liaison. La liaison verticale tient lieu de OU logique. Avec cette forme de liaison OU, 32 entrées (contacts) et 64 sorties (bobines, liaisons) sont possibles.

Objet texte

Présentation

Dans le plan de contacts LD, les textes peuvent être placés sous forme d'objets texte. La taille de ces objets texte est fonction de la longueur du texte. Selon la longueur du texte, la taille de l'objet peut être agrandie, dans les sens vertical et horizontal, d'unités de grille supplémentaires. Les objets texte peuvent chevaucher d'autres objets.

Reconnaissance de front

Introduction

Pendant la reconnaissance d'un front, un bit est surveillé pendant un passage de 0 à 1 (front positif ou ascendant) ou de 1 à 0 (front négatif ou descendant).

Pour ce faire, la valeur du bit du cycle précédent est comparée à la valeur du bit du cycle en cours. Dans ce cas, non seulement la valeur en cours est requise mais également l'ancienne valeur.

C'est pourquoi, lors de la détection de fronts, 2 bits (valeurs courante et ancienne) sont requis au lieu d'un.

Comme le type de données `BOOL` n'offre qu'un seul bit (valeur courante), un autre type de données est utilisé pour la détection de fronts, `EBOOL` (`BOOL` étendu). Outre la détection des fronts, le type de données `EBOOL` offre une option pour le forçage. Vous devez donc également l'enregistrer que le forçage de bit soit actif ou non.

Le type de données `EBOOL` enregistre les données suivantes :
- la valeur courante du bit dans le *bit de valeur*,
- la valeur précédente du bit dans le *bit d'historique*,
  (au début de chaque cycle, le contenu du bit de valeur est copié dans le bit d'historique)
- les informations sur l'activation du forçage du bit dans le *bit de forçage*,
  (0 = forçage inactif, 1 = forçage actif)

Restrictions d'EBOOL

⚠ ATTENTION

COMPORTEMENT IMPREVU DE L'EQUIPEMENT

Pour une bonne détection des fronts, `%M` doit être mis à jour à chaque cycle de tâche. Lors d'une écriture unique, le front est infini.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

En utilisant une variable EBOOL pour les contacts afin de reconnaître les fronts positifs (P) et négatifs (N) (fronts ascendants et descendants, avec une EF), vous devez respecter les restrictions ci-après.

EBOOL avec %M non écrit dans le programme

Une variable EBOOL avec une adresse %M qui n'est pas écrite dans le programme, mais fournie directement (par une table d'animation, un écran ou une interface, par exemple), fonctionnera de manière imprévue. La valeur du front reste indéfiniment TRUE car %M n'est écrit qu'une seule fois.

NOTE : pour éviter cela, %M doit être écrit à la fin de la tâche pour mettre à jour les informations de valeur anciennes.

L'ancienne valeur n'est mise à jour que lorsque le bit %M est écrit. Par conséquent, si vous n'écrivez le bit qu'une fois, la détection de front est infinie.

Valeur ancienne	Valeur actuelle	Détection de front	Description
0	0	0	état 0 (avant d'écrire le bit)
0	1	1	Ecrivez 1 dans le bit (avec une table d'animation, par exemple).
0	1	1	Si vous ne l'écrivez pas à nouveau, le front demeure indéfiniment.
1	1	0	Ecrivez à nouveau 1 dans le bit, la valeur ancienne est actualisée et la détection de front est réglée sur 0.

EBOOL avec %M écrit dans le programme

Pour une variable EBOOL avec une adresse %M écrite dans le programme, vous devez respecter les limitations suivantes :

- N'utilisez pas le bit avec une bobine d'enclenchement ou de déclenchement. Dans ce cas, l'ancienne valeur n'est pas actualisée. Vous pouvez donc établir un front infini.
- N'écrivez pas le bit de manière conditionnelle. Une logique simple telle que `IF NOT %M1 THEN %M1 := TRUE; END_IF` cause un front éternel car elle n'est écrite qu'une fois.

EBOOL avec %I

Pour une variable EBOOL avec une adresse %I, vous devez respecter les restrictions suivantes :

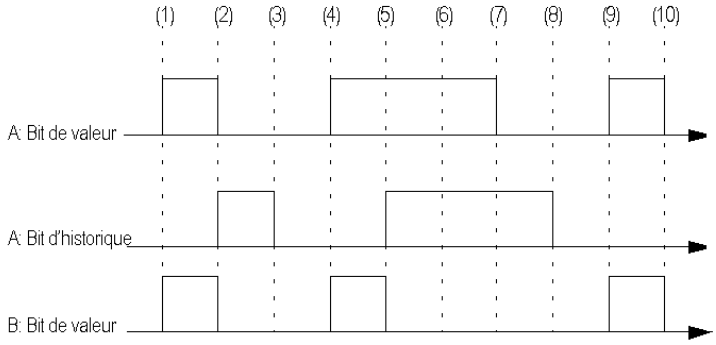
- En fonctionnement multitâche, la vérification du front %I doit être effectuée dans la tâche où il est actualisé. Il est déconseillé d'utiliser la détection du front d'un %I planifiée dans une tâche de priorité supérieure.
  Exemple : si vous avez une tâche fast qui actualise %I, n'utilisez pas la détection de front dans la tâche mast. Selon la planification, le front peut être détecté ou non.

Détection de fronts positifs

Pour détecter des fronts positifs, vous devez utiliser un contact pour la détection de fronts positifs. Avec ce contact, la connexion de droite pour un cycle de programme est 1 lorsque la transition du paramètre associé réel (A) passe de 0 à 1 et que, simultanément, l'état de la connexion de gauche est 1. Sinon, l'état de la liaison de droite est 0.

Dans l'exemple, un front positif de la variable A doit être reconnu et B doit donc être défini pendant un cycle.

Chaque fois que le bit de valeur de A est égal à 1 et que le bit d'historique est égal à 0, B est réglé sur 1 pendant un cycle (cycle 1, 4 et 9).

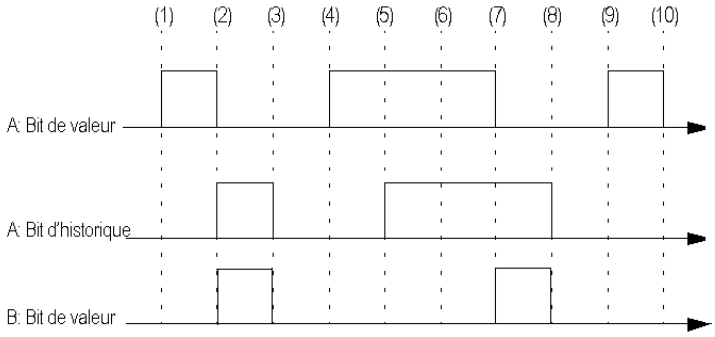


Détection de fronts négatifs

Pour détecter des fronts négatifs, vous devez utiliser un contact pour la détection de fronts négatifs. Avec ce contact, la connexion de droite pour un cycle de programme est 1 lorsque la transition du paramètre associé réel (A) passe de 1 à 0 et que, simultanément, l'état de la connexion de gauche est 1. Sinon, l'état de la liaison de droite est 0.

Dans l'exemple, un front négatif de la variable A doit être reconnu et B doit donc être défini pendant un cycle.

Chaque fois que le bit de valeur de A est égal à 0 et que le bit d'historique est égal à 1, B est réglé sur 1 pendant un cycle (cycle 2 et 8).

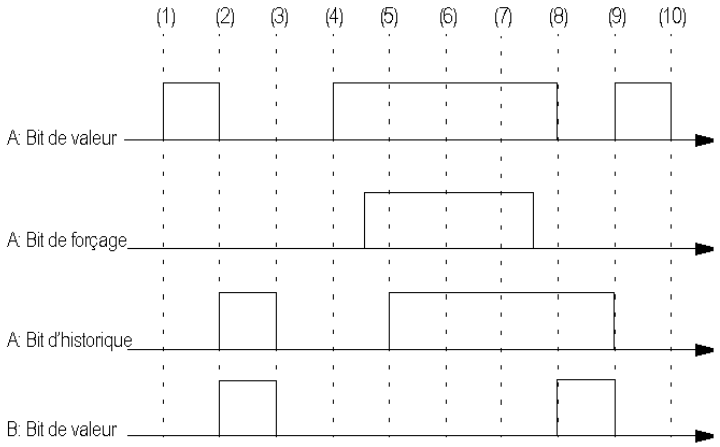


Forçage de bits

Lors du forçage de bits, la valeur des variables indiquée par la logique est écrasée par la valeur de forçage.

Dans l'exemple, un front négatif de la variable A doit être reconnu et B doit donc être défini pendant un cycle.

Chaque fois que le bit de valeur ou le bit de forçage de A est égal à 0 et que le bit d'historique est égal à 1, B est réglé sur 1 pendant un cycle (cycle 1 et 8).



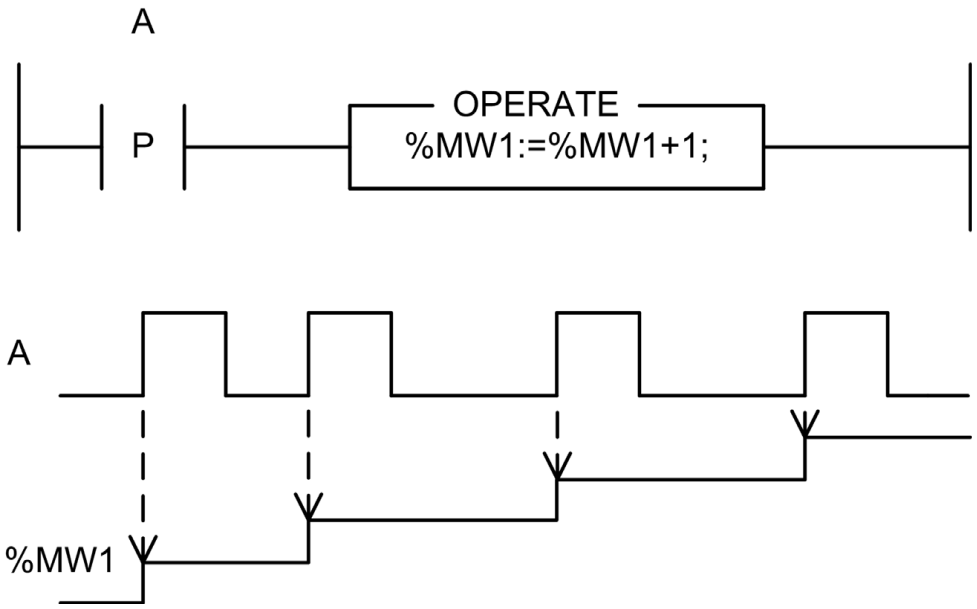
Utilisation des variables BOOL et EBOOL

Le comportement de la détection de fronts peut varier selon que vous utilisez le type de variable `BOOL` ou `EBOOL` :

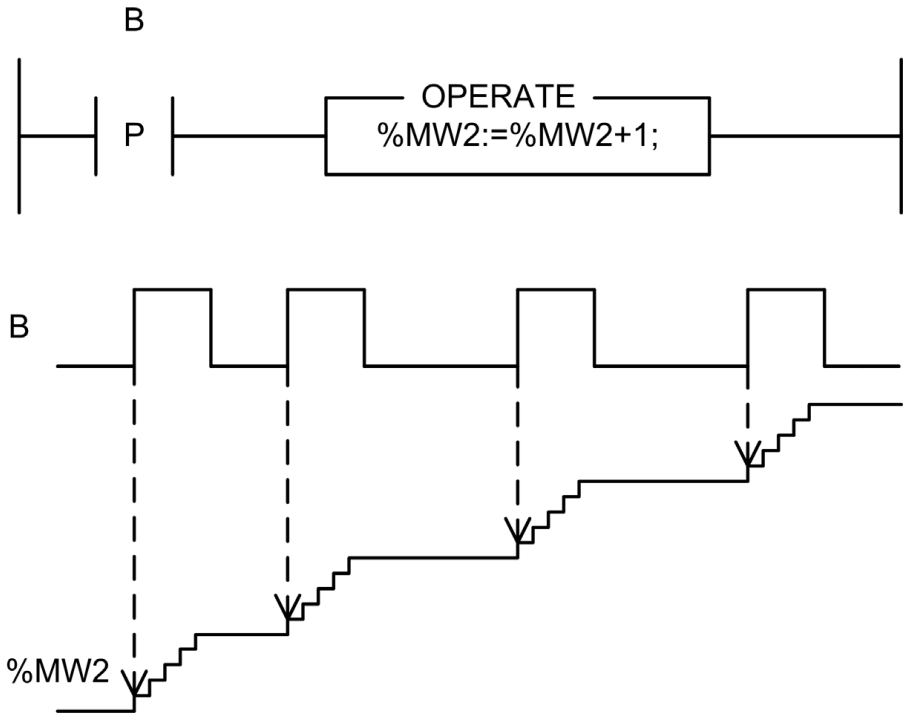
- Lorsque vous utilisez une variable `BOOL`, le système gère l'historique en permettant la détection de front pendant l'exécution du contact.
- Lorsque vous utilisez une variable `EBOOL`, le bit d'historique est actualisé pendant l'exécution de la bobine.

Les exemples suivants montrent les différents comportements selon le type de variable.

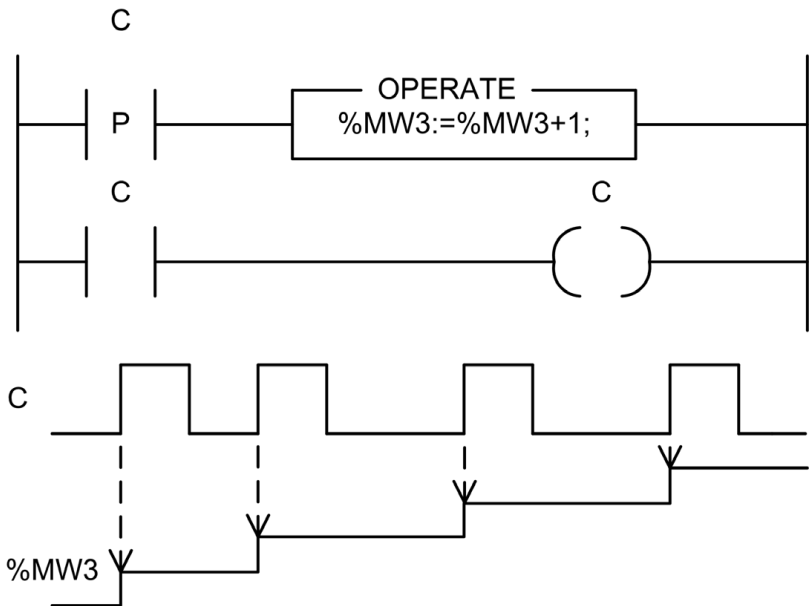
La variable `A` est définie comme `BOOL` ; chaque fois que `A` est réglé sur 1, `%MW1` est incrémenté de 1.



La variable B est définie comme EBOOL ; son comportement diffère de celui de la variable A. Lorsque B est réglé sur 1, %MW2 est incrémenté de 1 parce que le bit d'historique n'est pas actualisé.



La variable C est définie comme EBOOL ; son comportement est identique à celui de la variable A. Le bit d'historique est actualisé.

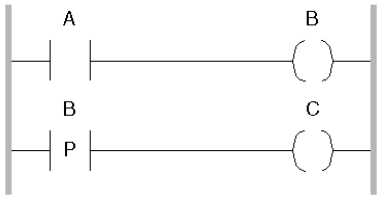


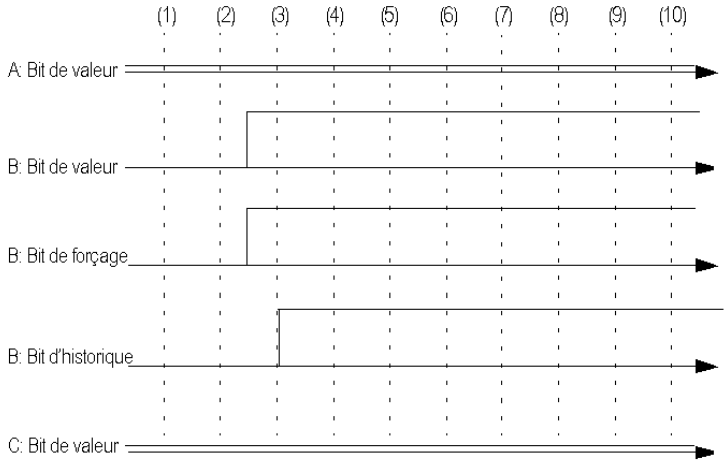
Perte de la détection de fronts probablement due au forçage de bobines

Le forçage de bobines peut causer la perte de la détection de fronts.

Dans l'exemple, lorsque A est égal à 1, B devrait être égal à 1, et avec un front montant de A, la bobine B sera définie pendant un cycle.

Dans cet exemple, la variable B est d'abord affectée à la bobine, puis à la liaison pour reconnaître les fronts montants.





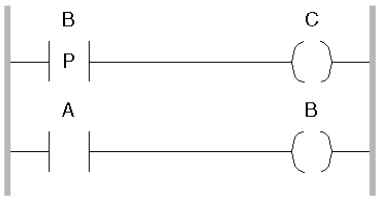
Au début du deuxième cycle, le bit de valeur de B est égal à 0. En cas de forçage de B dans ce cycle, le bit de forçage et le bit de valeur sont réglés sur 1. Pendant le traitement de la première ligne de logique dans le troisième cycle, le bit d'historique de la bobine (B) est également réglé sur 1.

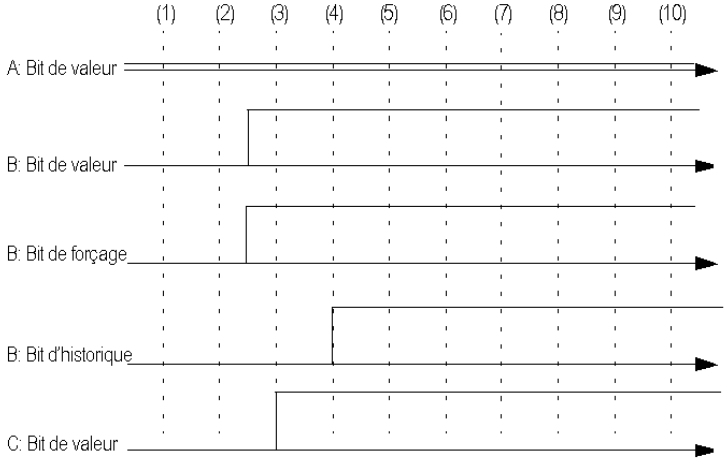
Problème :

Pendant la détection de front (comparaison du bit de valeur et du bit d'historique), dans la deuxième ligne de logique, aucun front n'est détecté car, en raison de la lise à jour, le bit de valeur et d'historique de la ligne 1 de B sont toujours identiques.

Solution :

Dans cet exemple, la variable B est d'abord affectée à la liaison pour reconnaître les fronts montants, puis à la bobine.





Au début du deuxième cycle, le bit de valeur de B est égal à 0. En cas de forçage de B dans ce cycle, le bit de forçage et le bit de valeur sont réglés sur 1. Pendant le traitement de la première ligne de logique dans le troisième cycle, le bit d'historique de la liaison (B) reste réglé sur 0.

La détection de front reconnaît la différence entre les bits de valeur et les bits d'historique et règle la bobine (C) sur 1 pour un cycle.

Perte de la détection de fronts probablement due à l'utilisation des bobines d'enclenchement ou de déclenchement

L'utilisation de bobines d'enclenchement ou de déclenchement peut causer la perte de la détection de fronts avec les variables EBOOL.

La variable au-dessus de la bobine d'enclenchement/de déclenchement (variable C dans l'exemple) est toujours affectée par la valeur de la liaison de gauche.

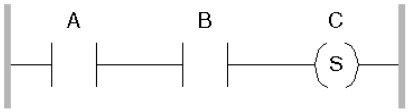
Si la liaison de gauche est 1, le bit de valeur bit (variable C dans l'exemple) est copié dans le bit d'historique et le bit de valeur est réglé sur 1.

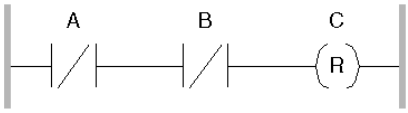
Si la liaison de gauche est 0, le bit de valeur bit (variable C dans l'exemple) est copié dans le bit d'historique, mais le bit de valeur n'est pas modifié.

Ainsi, quelle que soit la valeur prise par la liaison de gauche avant la bobine d'enclenchement ou de déclenchement, le bit d'historique est toujours mis à jour.

Dans l'exemple, un front positif de la variable C devrait être reconnu et définir D pendant un cycle.

Ligne de code	Comportement en LD	Equivalence en ST

1	Situation initiale : C = **0**, Bit d'historique = **0**
	 
	A = 1,
	B = 1,
	C = **1**, Bit d'historique = **0**
		`IF A AND B`
		`  THEN C := 1;`
		`ELSE C := C;`
		`END_IF;`

2	
	A = 1,
	B = 1,
	C = **1**, Bit d'historique = **1**
		`IF NOT(A) AND NOT(B)`
		`THEN  C := 0;`
		`ELSE C := C;`
		`END_IF;`

3	 C = **1**, Bit d'historique = **1** D = **0**, car le bit de valeur et le bit d'historique de C sont identiques. Le front montant de C, représenté dans ligne 1 du code, n'est par reconnu par le code de la ligne 2, car cela force l'actualisation du bit d'historique. (Si la condition est FALSE, la valeur présente de C est à nouveau attribuée à C, voir l'instruction ELSE de la ligne de code 2 dans l'exemple ST, par exemple.)	-

Ordre d'exécution et flux de signaux

Ordre d'exécution des réseaux

Les règles suivantes s'appliquent à l'ordre d'exécution des réseaux :

- l'exécution d'une section a lieu réseau pour réseau via les liaisons d'objets du haut vers le bas.
- les boucles ne peuvent pas être configurées par le biais de liaisons, étant donné que, dans ce cas, l'ordre d'exécution ne peut pas être défini de façon unique. Les boucles doivent être résolues par le biais de paramètres réels (voir *Configuration de boucles, page 392*).
- l'ordre d'exécution de réseaux reliés ensemble uniquement par la barre gauche d'alimentation est déterminé par l'ordre graphique (du haut vers le bas) dans lequel ces réseaux sont reliés à la barre gauche d'alimentation. Cela ne s'applique pas si l'ordre a été influencé par des éléments de commande.
- le calcul d'un réseau doit être complètement terminé avant que le calcul du réseau suivant puisse commencer.
- aucun élément d'un réseau n'est considéré comme calculé avant que l'état de toutes les entrées de cet élément n'ait été calculé.
- le calcul d'un réseau est considéré comme terminé lorsque toutes les sorties de ce réseau sont calculées. Cela s'applique également si le réseau comprend un ou plusieurs éléments de commande.

Flux de signaux dans un réseau

Les règles suivantes s'appliquent au flux de signaux au sein d'un réseau (rung) :

- le flux de signaux pour les liaisons booléennes est
  - de la gauche vers la droite pour les liaisons booléennes horizontales et
  - du haut vers le bas pour les liaisons booléennes verticales.
- le flux de signaux d'une liaison FFB va de la sortie FFB vers l'entrée FFB, indépendamment de la direction.
- un FFB n'est calculé que lorsque tous les éléments (sorties FFB, etc.) qui sont reliés à ses entrées sont calculés.
- l'ordre d'exécution des FFB reliés à différentes sorties du même FFB va du haut vers le bas.
- l'ordre d'exécution des objets n'est pas influencé par leur position au sein du réseau.
- l'ordre d'exécution de FFB apparaît sous forme de numéro d'exécution au dessus du FFB.

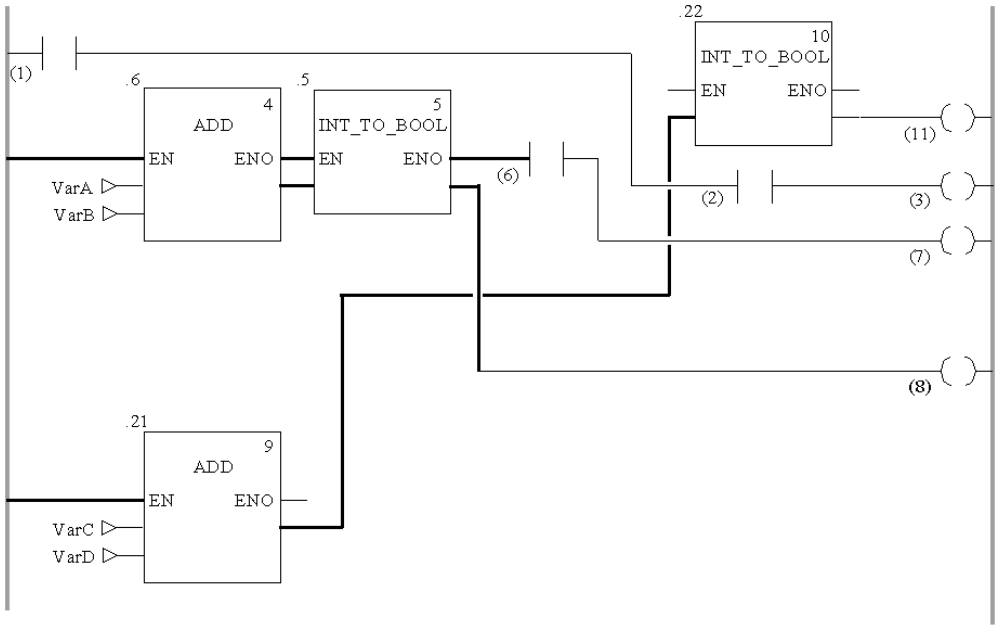
Priorités

Priorités lors de la détermination du flux de signaux au sein d'une section :

Priorité	Règle	Description
1	Liaison	Les liaisons ont la priorité la plus élevée lors de la détermination du flux de signaux au sein d'une section LD.
2	Réseau pour réseau	Le calcul d'un réseau doit être complètement terminé avant que le calcul du réseau suivant puisse commencer.
3	Ordre des sorties	Les sorties du même bloc fonction ou les sorties de liaisons verticales sont calculées du haut vers le bas.
4	Rung pour rung	Priorité la moins élevée. L'ordre d'exécution des réseaux reliés ensemble uniquement par la barre gauche d'alimentation est déterminé par l'ordre graphique (du haut vers le bas) dans lequel ces réseaux sont reliés à la barre gauche d'alimentation. (Cela ne s'applique que si aucune autre règle n'intervient.)

Exemple

Exemple de l'ordre d'exécution des objets dans une section LD :



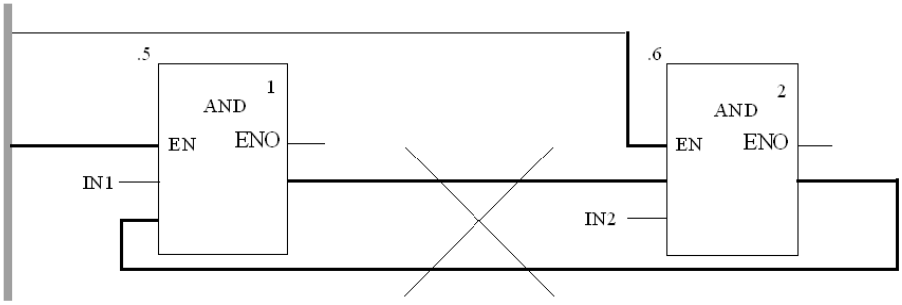
NOTE : Les numéros d'exécution de contacts et de bobines n'est pas affiché. Ils n'ont été indiqués dans le graphique que pour fournir une meilleure vue d'ensemble.

Configuration de boucles

Boucles non permises

La configuration de boucles exclusivement par le biais de liaisons n'est pas permise, étant donné que, dans ce cas, une détermination unique du flux de signaux n'est pas possible (la sortie d'un FFB est l'entrée du FFB suivant, et la sortie de celui-ci est à son tour l'entrée du premier).

Boucles non permises par le biais de liaisons :



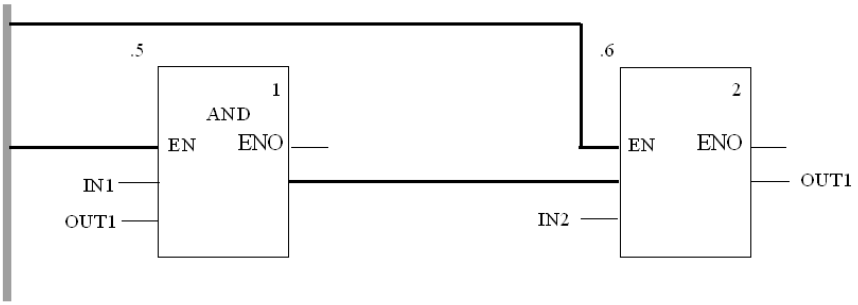
Résolution par le biais d'un paramètre réel

Une telle logique doit être résolue par le biais de variables de réaction, afin que le flux de signaux puisse être défini de façon unique.

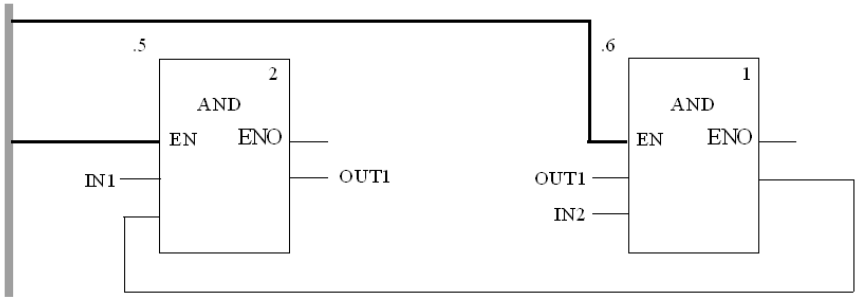
Les variables de réaction doivent être initialisées. La valeur initiale est utilisée lors de la première exécution de la logique. Une fois la première exécution effectuée, la valeur initiale est remplacée par la valeur actuelle.

Respectez pour les deux variantes l'ordre d'exécution (numéro entre parenthèses après le nom d'instance) des deux blocs.

Boucle résolue par le biais d'un paramètre réel : Variante 1



Boucle résolue par le biais d'un paramètre réel : Variante 2



Modification de l'ordre d'exécution

Présentation

L'ordre d'exécution des réseaux et l'ordre d'exécution des objets au sein d'un réseau sont définis par une série de règles *(voir page 390)*.

Dans certains cas, il est nécessaire de modifier l'ordre d'exécution proposé par le système.

Pour définir/modifier l'ordre d'exécution des réseaux, vous disposez des possibilités suivantes :

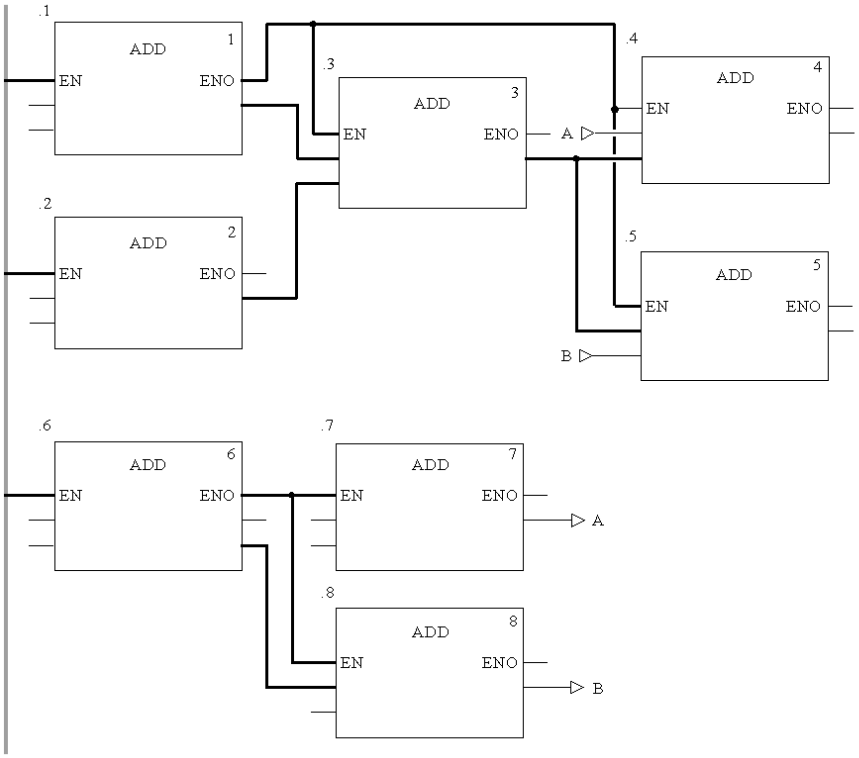
- utiliser des liaisons au lieu des paramètres réels,
- position des réseaux,

Pour définir/modifier l'ordre d'exécution au sein des réseaux, vous disposez des possibilités suivantes :

- position des objets.

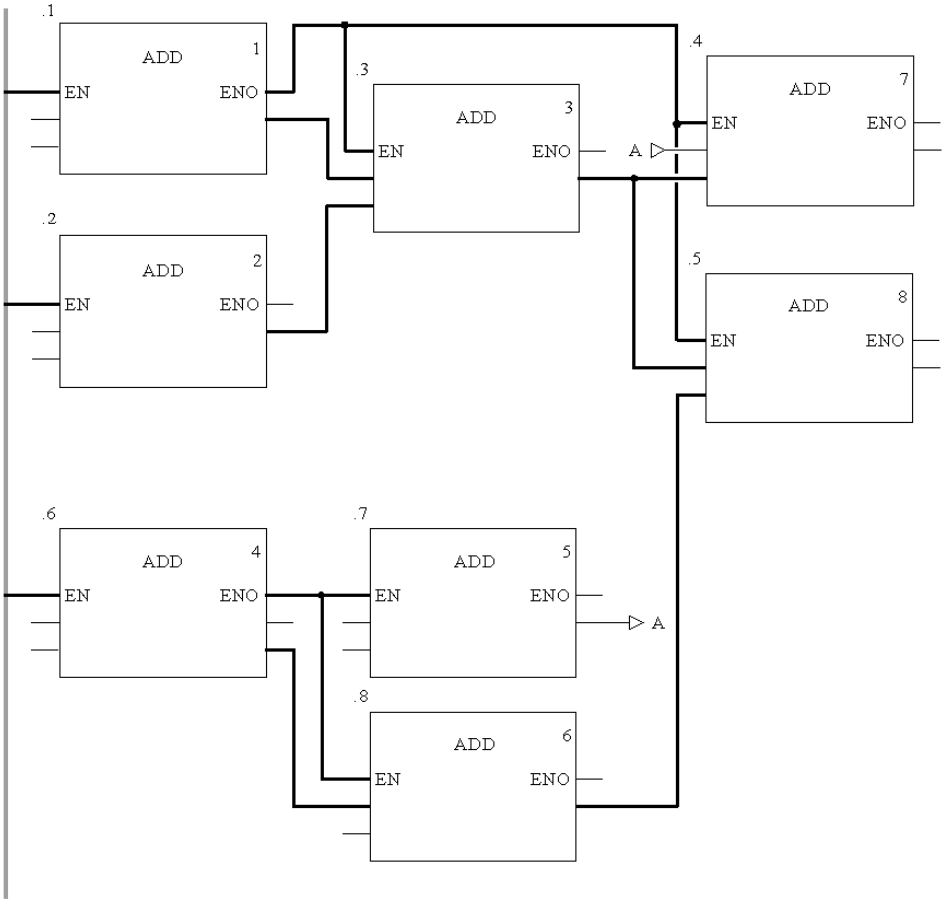
Situation de sortie

L'image suivante illustre deux réseaux dont l'ordre d'exécution est déterminé uniquement par leur position au sein de la section, indépendamment du fait que les blocs `0,4/0,5` et `0,7/0,8` nécessitent un ordre d'exécution différent.



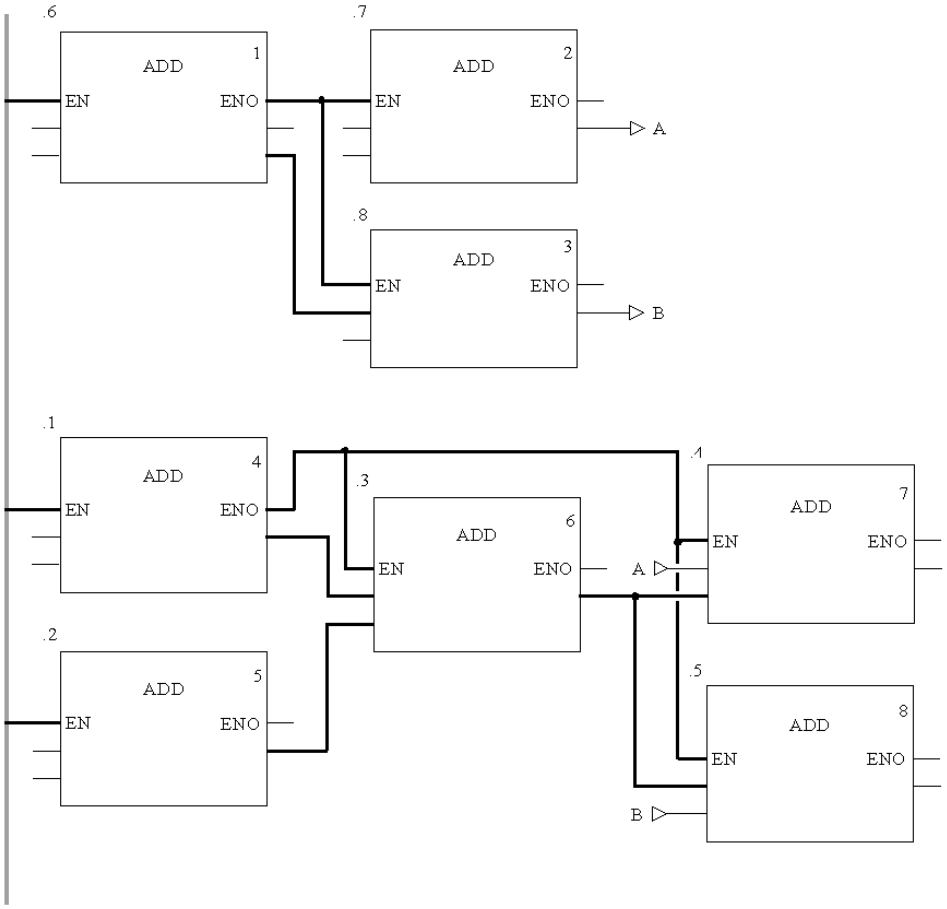
Liaison au lieu d'un paramètre réel

Lorsque l'on utilise une liaison au lieu d'une variable, les deux réseaux sont exécutés dans l'ordre correct (voir également *Situation de sortie, page 394*).



Position des réseaux

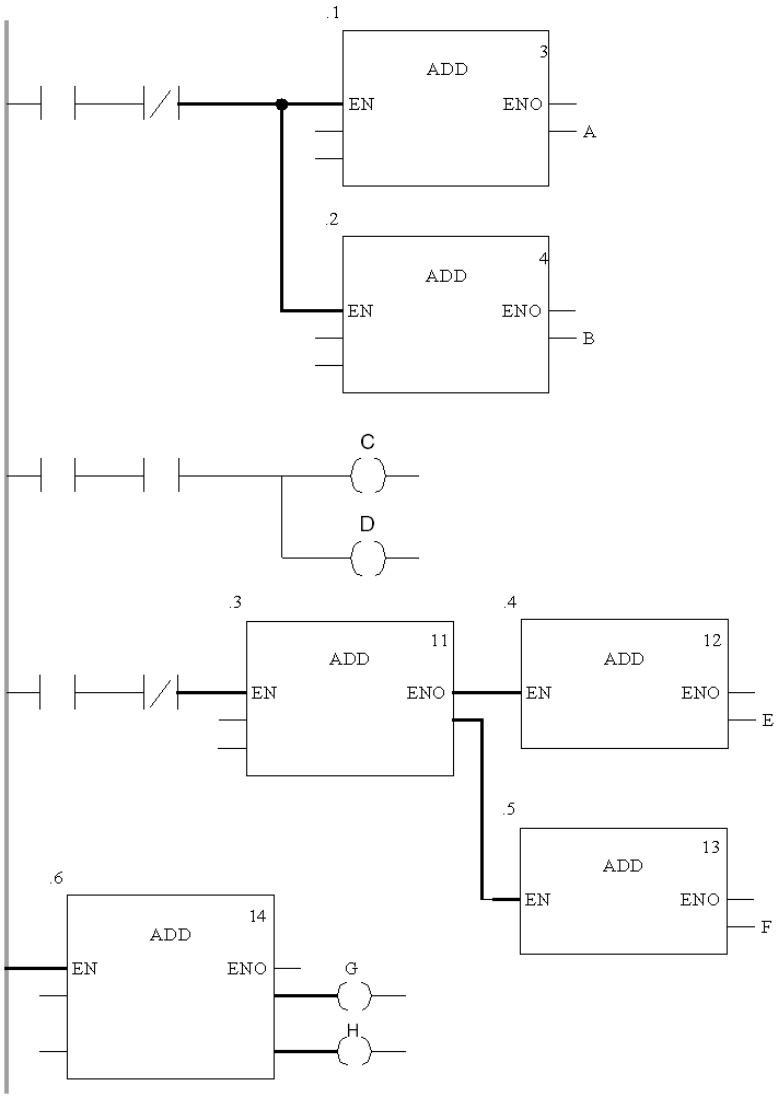
L'ordre d'exécution correct peut être obtenu en modifiant les positions des réseaux dans la section (voir également *Situation de sortie, page 394*).



Position des objets

La position des objets influe alors sur l'ordre d'exécution uniquement si plusieurs entrées (liaison gauche de contacts/bobines, entrées FFB) sont reliées à la même sortie de l'objet "appelant" (liaison droite de contacts/bobines, sorties FFB) (voir également *Situation de sortie, page 394*).

Situation de sortie :



Les positions du bloc `0,1` et `0,2` sont permutées dans le premier réseau. Dans ce cas (source commune des deux entrées de bloc), l'ordre d'exécution des deux blocs est également permuté (traitement du haut vers le bas). La même chose s'applique pour la permutation des bobines `C` et `D` dans le deuxième réseau.

Les positions du bloc `0,4` et `0,5` sont permutées dans le troisième réseau. Dans ce cas (source différente des entrées de bloc) l'ordre d'exécution des deux blocs n'est pas permuté (traitement dans l'ordre des sorties de bloc à appeler). La même chose s'applique pour la permutation des bobines `G` et `H` dans le dernier réseau.