

Séance 2 – Ladder Logic : bases, fonctions avancées et bonnes pratiques

1)	De la logique câblée à la normalisation : l'histoire du langage Ladder.....	3
2)	Le langage Ladder : définition et usages.....	4
3)	Pourquoi le langage Ladder est-il si populaire ?	5
3.1)	Langage graphique proche des schémas électriques :	5
3.2)	Visualisation et débogage aisés :	5
3.3)	Simplicité des concepts de base :	5
4)	Schéma Ladder : présentation et composants de base.....	6
4.1)	Rails (barres d'alimentation) :	6
4.2)	Rungs (échelons) :	6
4.3)	Entrées (contacts) :	6
4.4)	Sorties (bobines) :	7
4.5)	Expressions logiques :	7
4.6)	Adresses (notations) et tags :	7
4.7)	Commentaires :	8
5)	Lecture et exécution d'un programme Ladder.....	9
5.1)	Évaluation des logiques :	9
6)	Mise à jour des E/S :	10
7)	Fonctions logiques de base en Ladder (ET, OU, NON, etc.)	11
7.1)	Fonction ET (AND logique) :	11
7.2)	Fonction OU (OR logique) :	11
7.3)	Fonction NON (NOT logique) :	11
7.4)	Fonction NAND (NON-ET) :	12
7.5)	Fonction NOR (NON-OU) :	12
7.6)	Fonction XOR (OU exclusif) :	12
8)	Temporisateurs, compteurs et autres fonctions avancées du Ladder	13
8.1)	Les temporisateurs (timers) :	14
8.2)	Les compteurs (counters) :	18
8.3)	Les comparateurs et opérations mathématiques :	23
8.4)	Les bascules, mémoires et fonctions séquentielles :	24
9)	Le Ladder : un langage à la fois simple et puissant	25

1) De la logique câblée à la normalisation : l'histoire du langage Ladder

Avant 1968, l'automatisation des usines reposait sur des armoires de relais électromécaniques. Chaque commande, comme le démarrage d'un moteur, l'arrêt d'une pompe ou la surveillance d'un capteur, était réalisée par un câblage physique associant contacts et bobines. Ces schémas ressemblaient déjà à une échelle, avec deux rails verticaux représentant l'alimentation et des barreaux horizontaux qui traduisaient les liaisons logiques. C'est cette représentation visuelle simple qui a inspiré le Ladder tel que nous le connaissons.

En 1968, General Motors, confronté aux limites des armoires à relais trop complexes et coûteuses à modifier, lança un appel d'offres pour trouver une solution plus souple. Richard Morley, ingénieur chez Bedford Associates, développa alors le premier **PLC (Programmable Logic Controller)**, connu sous le nom de **Modicon 084**. Pour rassurer et convaincre les électriciens de l'époque, il choisit de reprendre la logique des schémas à relais sous forme programmable. C'est ainsi que naquit le **Ladder Diagram**, un langage qui servait de pont culturel entre le monde électrique et le monde logiciel.

Durant les années 1970 et 1980, les principaux constructeurs d'automates tels que Modicon, Allen-Bradley, Siemens ou Omron intégrèrent tous le Ladder à leurs produits. Chacun développa ses propres conventions, avec des symboles ou des syntaxes légèrement différentes, mais la philosophie restait la même. Le Ladder devint très vite le langage de référence dans les ateliers, car il était visuel, simple à utiliser et surtout familier pour les techniciens venant de l'électrotechnique.

En 1993, une étape décisive fut franchie avec la publication de la norme **IEC 61131-3** par l'International Electrotechnical Commission. Cette norme avait pour objectif d'uniformiser la programmation des automates et de mettre fin à la fragmentation entre constructeurs. Elle définit cinq langages standards :

1. Le Ladder Diagram (LD),
2. Le Function Block Diagram (FBD),
3. Le Structured Text (ST),
4. L'Instruction List (IL, depuis abandonné),
5. Le Sequential Function Chart (SFC, inspiré du Grafcet).

Dès lors, le Ladder fut officiellement reconnu comme un langage standard à l'échelle mondiale.

Depuis les années 2000 et jusqu'à aujourd'hui, le Ladder est resté dominant dans l'industrie malgré l'essor de langages plus puissants comme le Structured Text ou le Function Block Diagram. Il a continué d'évoluer en intégrant des fonctions avancées telles que les temporisations, les compteurs, les comparateurs arithmétiques ou encore les blocs de communication (Ethernet, Profibus, Profinet). Les grands environnements modernes comme TIA Portal, Rockwell Studio 5000 ou Codesys l'ont pleinement intégré.

2) Le langage Ladder : définition et usages

Le **langage Ladder** (ou *Ladder Diagram*, en français souvent appelé *schéma à contacts*) est un langage de programmation graphique destiné aux **automates programmables industriels (API)**. Inspiré directement des anciens schémas de câblage à relais, il reprend l'apparence d'une échelle avec deux rails verticaux et des barreaux horizontaux appelés *rungs*. Cette représentation visuelle facilite sa lecture et son appropriation, en particulier pour les techniciens et ingénieurs ayant une expérience en électrotechnique. Le mot *Ladder*, qui signifie *échelle* en anglais, illustre bien cette analogie.

Le Ladder est aujourd'hui largement utilisé dans l'industrie pour automatiser des tâches séquentielles et des opérations logiques. On le retrouve par exemple dans :

- Systèmes de convoyeurs en logistique et manutention (transporter et trier des produits),
- Machines d'emballage et de cerclage de palettes,
- Systèmes de lubrification (par ex. un broyeur à boulets),
- Dosages en génie chimique (mélange de ciment, de liquides, etc.),
- Lignes d'embouteillage et d'étiquetage dans l'agroalimentaire,
- Contrôle de niveau de cuves ou silos,
- Régulation de débit et de pression pour des fluides ou de l'air,

Plus largement, tout processus industriel combinatoire ou séquentiel peut être programmé en Ladder afin d'en automatiser le fonctionnement.

3) Pourquoi le langage Ladder est-il si populaire ?

Le Ladder est apprécié des automaticiens car il offre une **prise en main rapide et intuitive**. Plusieurs raisons expliquent sa popularité :

3.1) Langage graphique proche des schémas électriques :

Un programme Ladder se lit comme un schéma de circuit électrique à relais. Les symboles utilisés (contacts, bobines) **évoquent directement des composants électriques** (boutons, relais), ce qui facilite la transition pour les électriciens et techniciens de maintenance habitués aux plans de câblage. En somme, le Ladder permet de « programmer sans coder » en dessinant un schéma logique.

3.2) Visualisation et débogage aisés :

La représentation sous forme d'échelle permet de **voir le flux logique circuler** de la gauche vers la droite. Lorsque l'automate tourne, on peut généralement observer en surbrillance les contacts activés et les chemins alimentés, ce qui rend le dépannage très visuel. Il est plus facile d'identifier pourquoi une sortie n'est pas activée en suivant le circuit logique sur le diagramme, qu'en analysant du code texte équivalent.

3.3) Simplicité des concepts de base :

Le Ladder repose sur des **états binaires (Vrai/Faux)** et des opérations logiques fondamentales (ET, OU, NON) faciles à comprendre. Il a été conçu pour être suffisamment **simple et intuitif** pour être appris sans formation informatique poussée. Beaucoup le considèrent comme le **langage PLC le plus facile à apprendre** pour un débutant.

Le Ladder offre une **courbe d'apprentissage rapide** pour résoudre la majorité des besoins de logique combinatoire en automatisme.

4) Schéma Ladder : présentation et composants de base

Un **diagramme Ladder** se compose graphiquement de **lignes horizontales** appelées *rungs* (ou *échelons*) reliant une barre verticale gauche à une barre verticale droite, qui sont les **rails** d'alimentation.

Visuellement, cela ressemble à une échelle d'où le nom *Ladder*. Chaque rung contient une combinaison de **symboles logiques** représentant les conditions d'entrée et les actions de sortie.

L'ensemble du diagramme se lit un peu comme un tableau : de **gauche à droite** pour chaque rung, et de **haut en bas** pour enchaîner les rungs.

Pour bien comprendre la structure d'un programme Ladder, détaillons ses **principaux éléments constitutifs** :

4.1) Rails (barres d'alimentation) :

Ce sont deux lignes verticales, à gauche et à droite du schéma, qui figurent les pôles d'alimentation (le potentiel positif à gauche, le retour à droite, par analogie avec un circuit électrique). Dans un API, ils représentent le début et la fin de l'exécution de chaque ligne de code Ladder – on évalue la logique de gauche (rail gauche) jusqu'au rail droit comme si le courant circulait de gauche à droite.

4.2) Rungs (échelons) :

Chaque rung est une **ligne horizontale** reliant le rail gauche au rail droit. Il correspond à un segment de programme qui exécute une logique particulière. En schéma électrique câblé, un "barreau" équivaut à un fil reliant la source aux composants en série jusqu'à la masse. En Ladder, le rung représente le **chemin logique** que doivent parcourir les signaux pour aboutir à un résultat (sortie). Les rungs sont exécutés séquentiellement de haut en bas par l'automate, et sont souvent numérotés.

4.3) Entrées (contacts) :

Les entrées du Ladder correspondent aux **capteurs ou commandes externes** câblés sur l'API (boutons poussoirs, fin de course, capteurs divers...).

Dans le programme Ladder :

- Une entrée est représentée par un **symbole de contact** normalement ouvert (NO) → symbolisé par | |.
- Une sortie est représentée par un **symbole de contact** normalement fermer (NC) → symbolisé par | / |.

4.4) Sorties (bobines) :

Les sorties représentent les **actionneurs** pilotés par l'automate (moteurs, vérins, voyants, électrovannes...).

Dans le programme Ladder :

- Une sortie est représentée par une bobine généralement dessinée comme une paire de parenthèses () placée à droite du rung.
- Quand la logique du rung est vraie (chemin établi), la bobine s'énergise → l'actionneur relié est activé (ou la mémoire passe à 1).
- Quand la logique du rung est fausse (chemin interrompu), la bobine est relâchée → l'actionneur est désactivé (ou la mémoire repasse à 0).

4.5) Expressions logiques :

Ce terme désigne l'ensemble des combinaisons de contacts et de bobines qui réalisent une fonction de contrôle donnée. Un rung peut comporter plusieurs contacts connectés en série et/ou en parallèle formant une **expression logique** dont la bobine de sortie est la résultante.

- Les contacts peuvent être placés **en série** → cela correspond à la fonction **ET (AND)**.
 - Par exemple, "Entrée A ET Entrée B"
- Les contacts peuvent être placés **en parallèle** → cela correspond à la fonction **OU (OR)**.
 - Par exemple, "Entrée C OU Entrée D"

4.6) Adresses (notations) et tags :

Chaque contact ou bobine dans le Ladder est associé à une adresse correspondant à une localisation mémoire de la variable.

Exemple :

- %I → une entrée physique,
- %Q → une sortie,
- %M → un registre interne (selon les fabricants).

Les tags sont des noms symboliques ou des commentaires attribués à ces adresses afin de mieux les identifier.

- Exemple : I0.0 = "Capteur température".

Une bonne documentation des adresses avec des noms clairs facilite grandement la compréhension, la lecture et la maintenance d'un programme Ladder.

4.7) Commentaires :

Finalement, les commentaires de programme sont des **notes textuelles** qu'on peut ajouter dans le Ladder pour expliquer le fonctionnement de la logique. Ils n'affectent pas l'exécution, mais sont essentiels pour la maintenance et le dépannage en décrivant ce que fait chaque section du code. Un Ladder bien commenté est beaucoup plus facile à lire et à transmettre à d'autres techniciens.

En combinant ces éléments de base, on construit un programme Ladder complet. Voyons maintenant comment se déroule son exécution dans l'automate.

5) Lecture et exécution d'un programme Ladder

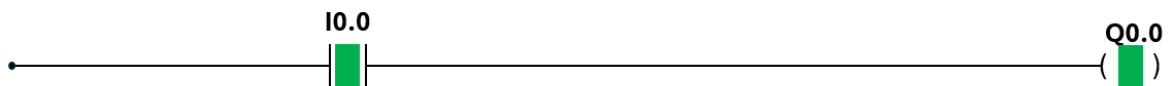
Un programme Ladder s'exécute de manière cyclique dans l'automate, en évaluant les rungs **de haut en bas** et **chaque rung de gauche à droite**. Autrement dit, l'automate balaie en permanence le schéma : il commence au rung 1, évalue la condition logique de ce rung de la barre gauche jusqu'à la bobine à droite, met à jour la sortie correspondante, puis passe au rung 2, et ainsi de suite jusqu'au dernier rung, après quoi il retourne au début (c'est ce qu'on appelle le *scan* PLC). Ce cycle de scan est très rapide (de l'ordre de la milliseconde à quelques millisecondes selon le nombre de rungs), si bien que le programme semble tourner en continu.

5.1) Évaluation des logiques :

Lorsqu'un **rung** est exécuté, l'automate vérifie s'il existe un **chemin logique continu** reliant le rail gauche à la bobine de droite. Chaque contact agit alors comme un interrupteur :

- **Fermé (actif)** → il laisse passer le courant logique,
- **Ouvert (inactif)** → il bloque le passage.

Si, au final, le courant atteint la **bobine**, celle-ci s'active (état 1). Dans le cas contraire, elle reste désactivée (état 0). On peut assimiler cela à une condition de type *si... alors...* : si toutes les conditions posées sur le rung sont vraies, alors la sortie est alimentée.



Lorsque le rung comporte des **branches parallèles** (contacts en dérivation), l'automate les évalue de gauche à droite et de haut en bas. On peut se représenter chaque branche comme un chemin alternatif partant du rail gauche et rejoignant la bobine :

- Si au moins une branche fournit un passage valide, la sortie sera activée,
- Si toutes les branches sont ouvertes, la bobine reste inactive.

Le **solveur logique** interne applique systématiquement les règles de l'**algèbre de Boole**, que ce soit pour les enchaînements en série (fonction ET) ou pour les dérivations en parallèle (fonction OU). Dans le **scan PLC**, les branches parallèles d'un même rung sont généralement analysées de la branche supérieure vers la branche inférieure, toujours dans le sens gauche → droite à l'intérieur de chaque branche.

6) Mise à jour des E/S :

Le fonctionnement d'un automate suit un **cycle en trois étapes** :

1. En **début de cycle**, il lit l'état de toutes les **entrées physiques** (capteurs, boutons, fins de course) et les stocke en mémoire.
2. Ensuite, il exécute le **programme Ladder** à partir de ces valeurs, pour déterminer l'état logique des **sorties** en mémoire.
3. Enfin, en **fin de cycle**, il applique ces résultats aux **sorties physiques** (actionneurs, moteurs, voyants).

Ce cycle se répète en continu, généralement en quelques millisecondes, donnant l'illusion d'une réaction instantanée.

Le Ladder agit ainsi comme un **chef d'orchestre logique**, reliant les entrées et les sorties du système selon les règles définies dans le schéma.

7) Fonctions logiques de base en Ladder (ET, OU, NON, etc.)

Ladder est essentiellement une traduction graphique de la logique booléenne. Il permet de reproduire toutes les fonctions logiques fondamentales en combinant astucieusement des contacts en série ou en parallèle.

Voici les principales fonctions logiques réalisables et leur équivalent en Ladder :

7.1) Fonction ET (AND logique) :

Cette fonction est vraie uniquement si **toutes les conditions** d'entrée sont vraies. En Ladder, on la réalise en mettant les **contacts en série** les uns à la suite des autres. Par exemple, si l'on veut que la sortie Y s'active lorsque *Entrée A* et *Entrée B* sont toutes deux actives, on place le contact A suivi du contact B, puis la bobine Y.

- Si $A = 1$ et $B = 1 \rightarrow$ le chemin est complet, la bobine Y est alimentée.
- Si l'un des deux est ouvert (0), le courant est interrompu et Y reste inactive.

7.2) Fonction OU (OR logique) :

Cette fonction est vraie si au moins une des conditions est vraie. En Ladder, elle s'implémente avec des contacts en parallèle (branches dérivées rejoignant la même bobine).

Concrètement, à partir du rail gauche, on crée deux branches : l'une avec le contact A menant à Y, l'autre avec le contact B menant aussi à Y.

- Si $A = 1$, la branche supérieure suffit à alimenter Y.
- Si $B = 1$, la branche inférieure suffit également.
- Si A et $B = 0$, alors Y reste inactive.

7.3) Fonction NON (NOT logique) :

C'est l'opération d'inversion : elle est vraie lorsque l'entrée est fausse, et inversement.

En Ladder, elle se traduit simplement par un contact normalement fermé (NC).

Si ce contact est associé à l'entrée A, alors :

- Tant que $A = 0$, le contact NC est passant, et la sortie Y est activée.
- Dès que $A = 1$, le contact s'ouvre, coupant le courant et désactivant Y.

7.4) Fonction NAND (NON-ET) :

La sortie est vraie **sauf si toutes les entrées sont vraies**. Autrement dit, elle n'est fausse que dans le cas où toutes les conditions sont remplies simultanément.

En Ladder, on peut :

- Soit réaliser un montage **ET** puis inverser le résultat avec un contact NC en sortie,
- Soit remplacer chaque contact NO par un **NC** pour inverser chaque entrée.
Exemple : $Y = \text{NAND}(A, B) \rightarrow Y$ est faux uniquement si A et B sont vrais en même temps, et vrai dans tous les autres cas.

7.5) Fonction NOR (NON-OU) :

La sortie est vraie **seulement si aucune entrée n'est vraie**. Dès qu'une condition est active, la sortie devient fausse.

En Ladder, on peut :

- Placer A et B en parallèle (OU), puis inverser le résultat avec un contact NC en sortie,
- Ou utiliser directement des **contacts NC** en parallèle.
Exemple : $Y = \text{NOR}(A, B) \rightarrow Y$ sera vrai uniquement si A = 0 et B = 0 (aucune condition active).

7.6) Fonction XOR (OU exclusif) :

La sortie est vraie si **une seule entrée** est vraie, mais pas les deux à la fois.

En Ladder, on combine les opérateurs de base (ET, OU, NON) :

- Une branche pour $(A \text{ ET } \text{NON } B)$,
- Une branche pour $(\text{NON } A \text{ ET } B)$.
Ces deux branches sont mises en parallèle vers la même bobine.
Exemple : $Y = \text{XOR}(A, B) \rightarrow Y$ s'active si A = 1 et B = 0, ou si A = 0 et B = 1, mais reste inactif si A et B sont identiques.

En Ladder, il est possible de **mélanger librement les séries et les parallèles** dans un même rung. Cela permet de construire des logiques booléennes simples ou très complexes, en reproduisant exactement la logique souhaitée.

8) Temporisateurs, compteurs et autres fonctions avancées du Ladder

Si le Ladder excelle pour représenter des logiques combinatoires avec des contacts et bobines, il propose également tout un ensemble de **fonctions avancées** pour aller plus loin que la simple logique booléenne.

En effet, les environnements de programmation Ladder intègrent des blocs fonctionnels spéciaux (souvent sous forme de boîtes ou de symboles particuliers) qui permettent de réaliser des opérations que l'on ne pouvait pas implémenter avec des relais électromécaniques classiques

Parmi ces fonctions évoluées, on trouve notamment :

- Les temporisateurs (timers)
- Les compteurs (counters)
- Les comparateurs et opérations mathématiques
- Les bascules, mémoires et fonctions séquentielles

8.1) Les temporisateurs (timers) :

Ils permettent d'introduire la notion de **temps** dans la logique Ladder. Un temporisateur est un bloc qui, par exemple, active sa sortie avec un certain **retard** (délai à l'enclenchement) ou la désactive après une durée programmée, une fois sa condition d'entrée remplie. On les utilise pour créer des **temporisations** (par ex. allumer un moteur 5 secondes après l'appui sur un bouton, ou couper une pompe après un temps donné).

Différents types de timers existent, configurables en durée.

Fonction TON (Timer On Delay)

La fonction **TON** (Timer ON Delay) est un temporisateur qui déclenche sa sortie **avec un retard à l'enclenchement**.

En d'autres termes : la sortie ne s'active pas immédiatement lorsque l'entrée passe à 1, mais seulement **après un délai programmé** (*Preset Time, PT*).

Principe de fonctionnement

Quand l'entrée IN passe de 0 à 1 :

- Le temporisateur commence à compter le temps défini par PT.
- Durant ce délai, la sortie Q reste à 0 (inactive).

Si IN reste à 1 pendant toute la durée de PT :

- À la fin du délai, la sortie Q passe à 1 (active).

Si IN retombe à 0 avant que PT soit écoulé :

- Le compteur est réinitialisé, Q reste à 0, et le timer recommence au prochain front montant.

Variables utilisées

Input :

IN (BOOL) : condition d'entrée qui déclenche le comptage.

PT (TIME) : durée programmée (Preset Time).

Output :

Q (BOOL) : sortie qui devient vraie uniquement si IN est restée vraie pendant toute la durée PT.

ET (TIME) : temps écoulé pendant lequel IN est vraie (valeur intermédiaire du comptage).



Exemple d'utilisation

- Allumer un moteur 5 secondes après avoir appuyé sur un bouton.
- Retarder l'activation d'un système de refroidissement pour éviter un démarrage brutal.
- Échelonner le démarrage de plusieurs machines afin de limiter les pics de consommation électrique.

Fonction TOF (Timer Off Delay)

La fonction TOF est un temporisateur à retard à la coupure. Contrairement au TON, la sortie s'active immédiatement quand l'entrée passe à 1, puis elle ne se désactive qu'après un délai programmé lorsque l'entrée retombe à 0.

Principe de fonctionnement

Quand l'entrée IN est vraie (1) :

- La sortie Q devient immédiatement vraie (active),
- Le temporisateur n'entre pas encore en action.

Quand l'entrée IN repasse à faux (0) :

- La sortie Q reste encore vraie pendant la durée programmée PT,
- Le timer commence à décompter le temps.

À la fin du temps PT :

- Si IN est toujours faux → la sortie Q devient fausse,
- Si IN redevient vrai pendant le décompte → le timer est réinitialisé et Q reste active.

Variables utilisées

Input :

IN (BOOL) : condition d'entrée qui déclenche le timer.

PT (TIME) : durée programmée avant coupure (Preset Time).

Output :

Q (BOOL) : sortie qui suit l'entrée, mais avec un **retard à la désactivation**.

ET (TIME) : temps écoulé depuis le passage de IN à 0 (valeur de comptage).



Exemple d'utilisation

- Maintenir l'alimentation d'un système de sécurité quelques secondes après une coupure.
- Temporiser l'arrêt de ventilateurs ou de pompes pour assurer un **post-refroidissement**.
- Laisser tourner un extracteur ou une ventilation pendant un temps défini après l'arrêt d'une machine.

Fonction TP (Timer Pulse)

La fonction TP est un temporisateur à impulsion. Lorsque l'entrée devient vraie, la sortie s'active immédiatement, puis reste vraie pendant une durée déterminée (PT), même si l'entrée retombe à 0.

Principe de fonctionnement du TP

Quand l'entrée IN passe de 0 à 1 :

- La sortie Q devient immédiatement vraie,
- Le temporisateur commence à compter la durée programmée PT.

Pendant le temps PT :

- La sortie Q reste active,
- Peu importe l'état de IN (même si elle retombe à 0, le timer poursuit son décompte).

À la fin de PT :

- La sortie Q retombe automatiquement à 0,
- Le temporisateur est réinitialisé et prêt pour une nouvelle impulsion au prochain front montant de IN.

Variables utilisées

Input :

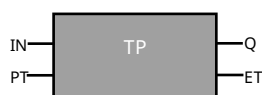
IN (BOOL) : condition d'entrée qui déclenche l'impulsion.

PT (TIME) : durée programmée de l'impulsion (Preset Time)

Output :

Q (BOOL) : sortie qui devient vraie immédiatement après l'activation de IN et reste vraie pendant toute la durée PT.

ET (TIME) : temps écoulé depuis le déclenchement de l'impulsion (valeur de comptage).



Exemple d'utilisation

- Allumer une lampe témoin pendant 10 secondes après un appui bref sur un bouton.
- Générer une impulsion de démarrage d'un moteur, quelle que soit la durée d'appui.
- Créer un signal temporaire de validation pour un cycle machine.

Ces fonctions n'existent pas en câblage pur : il aurait fallu ajouter des relais temporisés externes, alors qu'en Ladder c'est une simple instruction.

8.2) Les compteurs (counters) :

Ils fournissent la capacité de **compter des événements**. Un compteur incrémente une valeur chaque fois que son entrée est déclenchée (par exemple à chaque impulsion d'un capteur) et peut activer une sortie lorsqu'un seuil est atteint.

En Ladder, on insère un bloc compteur qui compte les fronts d'entrée et mémorise le total.

Fonction CTU (Counter Up)

La fonction CTU est un compteur incrémental. À chaque front montant détecté sur son entrée, il augmente sa valeur interne de 1. Il est utilisé pour compter des événements successifs (par exemple des pièces passant devant un capteur) et déclencher une sortie une fois un seuil atteint.

Principe de fonctionnement du CTU

Quand un front montant est détecté sur l'entrée **CU** :

- La valeur du compteur **CV** augmente de 1.
- Si **CV** $\geq$ **PV** (valeur de consigne), la sortie **Q** devient vraie.

Lorsque l'entrée **RESET** est activée :

- La valeur du compteur **CV** est réinitialisée à 0.

Variables utilisées

Input :

CU (BOOL) : À chaque front montant de CU, la valeur du compteur le compteur s'incrémente d'un.

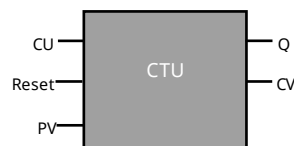
RESET (BOOL) : Remet CV à zéro.

PV (WORD) : valeur programmée qui déclenche Q

Output :

Q (BOOL) : devient vrai lorsque $CV \geq PV$.

CV (WORD) : valeur actuelle du compteur.



Exemple d'utilisation

- Compter le nombre de pièces passant sur un convoyeur.
- Déclencher une alarme lorsqu'un seuil de production est atteint.
- Suivi du nombre de cycles d'une machine.

Fonction CTD (Counter Down)

La fonction CTD est un compteur décrémental. À chaque front montant détecté sur son entrée, il diminue sa valeur interne de 1. Il est utilisé pour décompter un nombre défini d'événements jusqu'à atteindre zéro.

Principe de fonctionnement du CTD

Quand un front montant est détecté sur l'entrée **CD** :

- La valeur du compteur **CV** diminue de 1.
- Si **CV = 0**, la sortie **Q** devient vraie.

Lorsque l'entrée **LOAD** est activée :

- La valeur du compteur **CV** est chargée avec PV (valeur initiale).

Variables utilisées

Input :

CD (BOOL) : À chaque front montant de CD, la valeur du compteur se décrémente de 1 unité

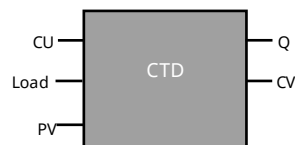
LOAD (BOOL) : À chaque front montant de LOAD, la valeur du compteur sera égalisée à PV

PV (WORD) : Valeur qui doit être chargée dans le compteur

Output :

Q (BOOL) : devient vrai lorsque CV = 0.

CV (WORD) : valeur actuelle du compteur.



Exemple d'utilisation

- Décompter le nombre de cycles restants avant une maintenance.
- Réaliser une production par lots (par ex. 50 pièces puis arrêt).
- Commande d'un distributeur de lots.

Fonction CTUD (Counter Up/Down)

La fonction CTUD est un compteur bidirectionnel. Il peut être incrémenté ou décrémenté en fonction des entrées activées. Il est utilisé lorsque le comptage doit prendre en compte à la fois des ajouts et des retraits.

Principe de fonctionnement du CTUD

- À chaque front montant de CU → la valeur CV augmente de 1.
- À chaque front montant de CD → la valeur CV diminue de 1.
- Si $CV \geq PV$ → la sortie QU devient vraie.
- Si $CV = 0$ → la sortie QD devient vraie.
- RESET remet CV à 0, LOAD charge la valeur PV.

Variables utilisées

Input :

CU (BOOL) : À chaque front montant de CU, la valeur du compteur augmentera de 1 unité

CD (BOOL) : À chaque front montant de CD, la valeur du compteur diminuera de 1 unité

RESET (BOOL) : À chaque front montant de RESET, la valeur du compteur sera égalisée à PV

LOAD (BOOL) : À chaque front montant de LOAD, la valeur du compteur deviendra égale à PV

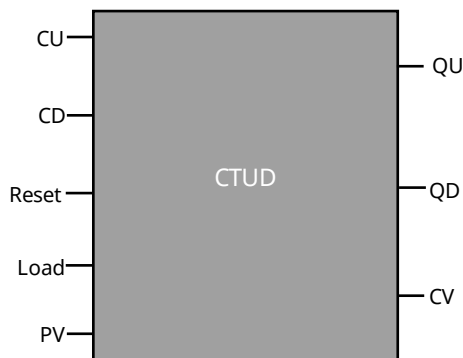
PV (WORD) : Valeur utilisée pour régler la sortie QU et chargée avec l'entrée LOAD

Output :

QU (BOOL) : vrai si $CV \geq PV$.

QD (BOOL) : vrai si $CV = 0$.

CV (WORD) : valeur actuelle du compteur.



Exemple d'utilisation

- Suivi du nombre de personnes présentes dans une salle (entrée = CU, sortie = CD).
- Gestion de stock avec ajouts et retraits d'articles.
- Comptage bidirectionnel sur une ligne de convoyage avec entrée et sortie de pièces.

Ceci est très utile pour des tâches comme compter le nombre de pièces passées sur un convoyeur ou le nombre de cycles d'une machine.

Là encore, en logique câblée, il aurait fallu un compteur électromécanique séparé, tandis qu'avec l'API c'est réalisé par logiciel.

8.3) Les comparateurs et opérations mathématiques :

Au-delà du pur booléen, le Ladder sait manipuler des **valeurs numériques** grâce à des blocs comparateurs (>, <, =, etc.) et des blocs arithmétiques (addition, soustraction, etc.). On peut par exemple comparer une mesure analogique de température à une consigne et activer une alarme si la valeur dépasse le seuil.

Ces éléments étendent le Ladder vers le contrôle continu et le calcul, ce qui sort du cadre strict du relais tout ou rien.

8.4) Les bascules, mémoires et fonctions séquentielles :

Le Ladder propose aussi des instructions pour **mémoriser des états** (par exemple les bobines *SET/RESET* qui verrouillent une sortie à 1 ou 0 jusqu'à nouvel ordre) ou pour détecter des fronts (détecteurs d'**impulsion** au front montant ou descendant). Il est ainsi possible de créer des **séquences** (par exemple démarrer un cycle au front d'un capteur et l'arrêter sur un autre) ou de garder un moteur enclenché même après relâchement du bouton start (fonction *latch*).

Ces mécanismes sont très utilisés pour structurer des automatismes plus complexes (ex : gestion d'un cycle de machine étape par étape en Ladder combiné avec du Grafcet).

9) Le Ladder : un langage à la fois simple et puissant

En résumé, le langage Ladder ne se limite pas aux contacts et bobines de base. Il offre un éventail complet de fonctions qui en font un outil **puissant et polyvalent** pour l'automaticien : on bénéficie de la **simplicité de la logique relais** tout en ayant la **souplesse du logiciel** (ajout de temporisations, de comptages, de mémoires, etc., sans aucun câblage supplémentaire).

En conclusion, le Ladder demeure un langage de prédilection pour les automaticiens du fait de son **accessibilité** et de son **efficacité** sur le terrain. Il permet de **traduire des logiques de commande complexes en un schéma clair** qui parle autant aux programmeurs qu'aux électriciens. Si son principe est ancien, il s'est modernisé en s'intégrant aux outils actuels (logiciels de conception, simulation, diagnostic en ligne) et reste un savoir indispensable pour tout technicien ou ingénieur en automatisme industriel souhaitant concevoir ou dépanner des systèmes pilotés par API.