1 .Grafcet et langage LADDER

Même si la forme est très différente, ces deux langages ont de nombreux points communs.

- tous deux décrivent un automatisme séquentiel sous forme graphique
- le fonctionnement est découpé en structures élémentaires que le Grafcet appelle **étapes**
- la progression d'une étape à l'autre se fait à la suite de la survenue d'un événement

Dans un cas comme dans l'autre, la structure élémentaire, l'**étape** en Grafcet, est constituée d'une fonction mémoire.

Cette fonction mémoire est mise à 1 si deux conditions sont vraies simultanément :

- l'étape précédente est active (à 1)
- l'évènement associé est vrai. (la **réceptivité** du Grafcet)

La mémoire est mise à 0 lorsque l'étape / la mémoire suivante est à 1 ou pendant la phase initiale.

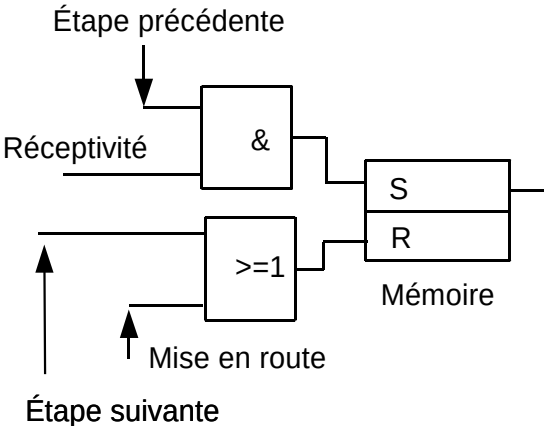


figure 1

L'étape initiale possède une spécificité, elle doit être mise à 1 lors de la phase initiale du fonctionnement. Sa structure reprend la précédente avec une modification.

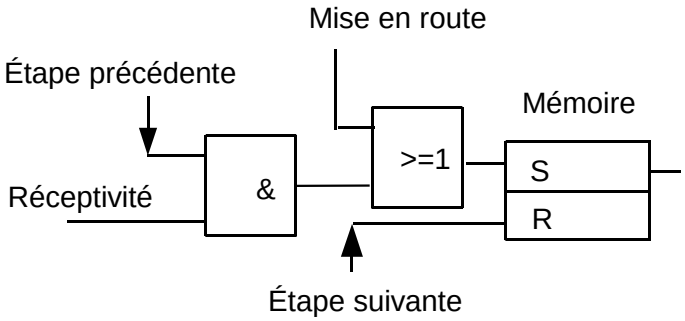


figure 2

Le Grafcet cache la structure montrée ci-dessus derrière le carré de l'étape et les différents traits.

Le LADDER demande que l'utilisateur explicite complètement la structure, il met en œuvre un graphisme de norme américaine.

2 .Traduction d'un fonctionnement explicité sous forme d'un Grafcet en son équivalent LADDER

Rappel de la fonction mémoire de l'électricien

Cette fonction mémoire utilise des dispositifs électromécaniques, des boutons poussoir, des relais avec leurs contacts.

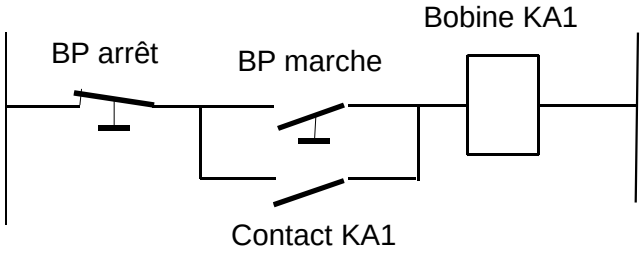


figure 3 : Mémoire avec priorité à l'arrêt

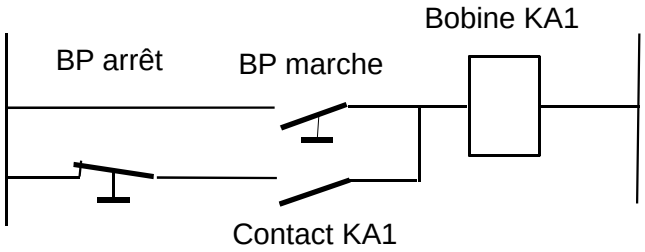


figure 4 : Mémoire avec priorité à la marche

Généralisation

La fonction mémoire, sous l'une ou l'autre forme, comprend :

- un relais et son contact
- un ordre de mise en marche, représenté ici par le BP marche
- un ordre de mise à l'arrêt, représenté ici par le BP arrêt

Traduction Grafcet vers LADDER

Le langage LADDER est graphique, il utilise la représentation des circuits à contacts, comme ci-dessus, même pour écrire un programme destiné à un automate programmable.

Compte tenu des règles d'évolution du Grafcet, on emploiera la structure de mémoire avec priorité à la marche.

Le principe de traduction est le suivant :

à partir du schéma de la figure 4 remplacer le BP marche par les contacts représentant l'ordre de mise en marche comme expliqué en figure 1

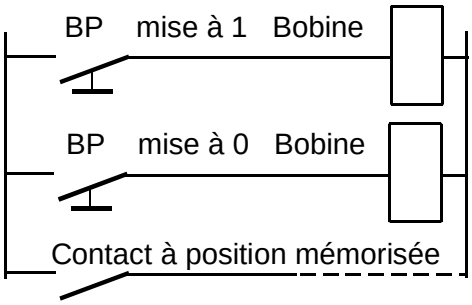
même chose pour remplacer le BP arrêt.

Voir ci-dessous, une autre possibilité de traduction

3 .Les relais à commande bistable

Un relais ordinaire possède une bobine. Lorsqu'elle est alimentée, les contacts sont attirés (s'ouvrent ou se ferment selon leur configuration). Lorsqu'elle n'est plus alimentée, les contacts retombent. Ce qui entraine une consommation d'énergie pendant toute la durée d'activation des contacts. On peut contourner ce problème en dotant le relais d'une mémoire mécanique qui accroche les contacts en envoyant une impulsion sur une bobine et qui les décroche par une autre impulsion sur une deuxième bobine.

En résumé
Un relais bistable possède deux bobines,
–une première pour l'activation des contacts
–une seconde pour la désactivation
son fonctionnement est fondé sur l'utilisation d'une mémoire mécanique.
C'est l'équivalent de le mémoire RS de l'électronicien.



Ci-dessous, une autre méthode encore plus proche du Grafcet

4 .Organisation d'un programme d'automate programmable

L'exécution du programme d'un API est confiée à un microcontrôleur, il ne peut réaliser qu'une tâche à la fois. Or, un circuit doit réagir rapidement aux sollicitations des entrées. Le programme d'un API doit être organisé afin de respecter cette contrainte.

On a choisi de concevoir le programme afin qu'il s'exécute de manière cyclique avec une durée la plus courte possible ou tout au moins d'une durée acceptable.

Pour un programme reproduisant un Grafcet on trouve le cycle, simplifié suivant :

- lecture de toutes les entrées
- mise à jour de la table représentative de l'état des entrées
- consultation de la table représentant l'état des étapes
- prise de décision concernant l'activation et la désactivation des étapes
- mise à jour de la table représentant l'état des étapes
- application de l'état des étapes aux sorties
- mise à jour de la table représentant l'état des sorties
- retour au début.

La durée de cycle s'étend de quelques millisecondes à quelques dizaines de millisecondes

On remarque que ce modèle fait une distinction entre l'état des étapes et l'état des sorties. Une sortie pouvant être active sur plusieurs étapes, une étape pouvant activer plusieurs sorties.

La prise de décision se fait en comparant la table des entrées à la table des étapes, selon les règles d'évolution du Grafcet.

La réaction de l'API n'est pas instantanée mais de durée suffisamment courte pour être acceptable.

5 .Commentaires sur le manuel CUBLOC

5.1 Les registres

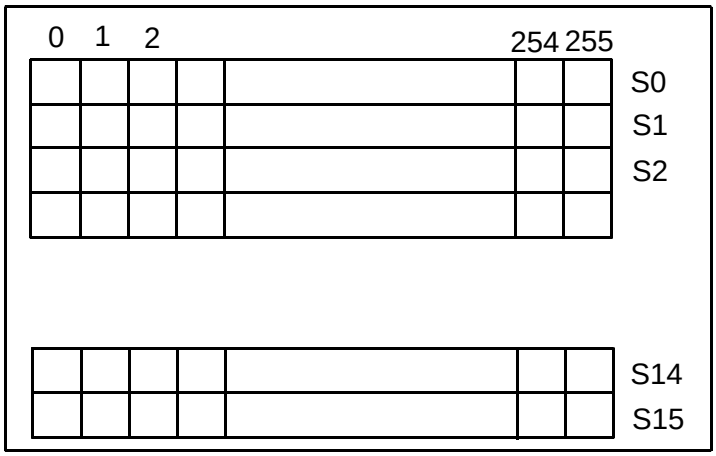
Dans son acception générale, le terme registre désigne un ensemble de cellules mémoire ayant un rôle commun. Ici, on semble confondre, dans la dénomination, le registre et la cellule.

Le registre P, d'une capacité de 128 bits, contient à la fois la table des entrées et la table des sorties évoquées plus haut. Chaque bit désigne soit une entrée soit une sortie.

Le registre M, d'une capacité de 512 bits, est utilisé pour les données internes au programme.
Le registre T contient 100 mots de 16 bits qui permettent de constituer des chronomètres.
Le registre C contient 50 mots de 16 bits destinés à réaliser des compteurs.
Pour stocker des données numériques, on utilisera un des 100 mots du registre D

Le registre S est destiné à la programmation du type Grafcet. Le terme registre S désigne un ensemble constitué de 16 entités de 256 places chacune.

Registre S



Le registre S contient la table des états évoquée plus haut. Chaque cellule[1] contient une image de l'état d'une étape. Pour s'adresser à une cellule on emploie la syntaxe suivante **Sn:m**
où **n** appartient à [0 ; 15] et **m** à [0 ; 255]

6 .Comment accéder aux éléments d'un registre

6.1 Accès direct on utilise le formalisme suivant

Registre P : mémorise les états des entrées et des sorties de l'automatisme
m = Px ou Py = n où x et y sont des entiers appartenant à [0 ; 127] ; m et n sont des variables

Registre M : mémorise des valeurs logiques internes
m = Mx ou My = n où x et y sont des entiers appartenant à [0 ; 511] ; m et n sont des variables

Registre T : mots de 16 bits pour les temporisations
Tx = m où x est un entier appartenant à [0 ; 99] La résolution dépend des instructions utilisées

Registre C : mots de 16 bits pour les compteurs
Cx = m où x est un entier appartenant à [0 ; 49]

Registre D : mots de 16 bits pour les données numériques
Dx = m où x est un entier appartenant à [0 ; 99]

1 un carré de la figure ci-dessus,

6.2 Le programme Basic accède à la mémoire du programme Ladder

Les CUBLOC sont à même d'exécuter deux programmes simultanément

- un programme écrit en Basic
- un programme écrit en Ladder

Chaque programme possède une zone de mémoire qui lui est propre.

Le programme Basic peut accéder à la zone mémoire du Ladder par l'intermédiaire d'entités dites mémoires système.

Si P0 est une entité utilisée en Ladder, le programme Basic pourra y accéder en utilisant l'écriture _P(0). L'accès aux autres registres se fait grâce au même formalisme.

Pour M5 on aura _M(5)

7 .Programmation d'un automatisme séquentiel

Le Grafcet et le Ladder modélisent le comportement d'un automatisme séquentiel.

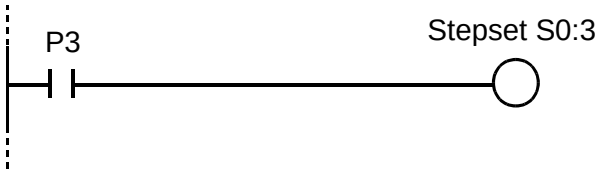
La programmation d'un automatisme séquentiel peut se faire, en Ladder, par les instructions Stepset et Stepout.

7.1 Stepset

Stepset reprend le fonctionnement indiqué à la figure 1 c'est à dire

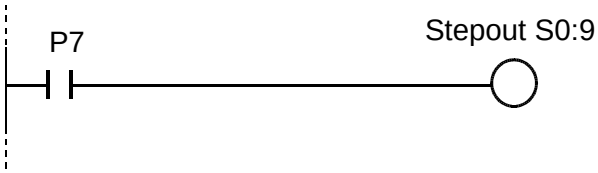
- l'activation sous les deux conditions
- la désactivation par l'activation de l'étape suivante.

Exemple :



L'activation de S0:3 se fait lorsque P3 = 1 à condition que S0:2 soit active préalablement
L'activation de S0:3 désactivera S0:2
L'activation de S0:4 désactivera S0:3

7.2 Stepout



L'activation de S0:9 n'est soumise qu'à P7. S0:9 désactivera toute autre mémoire S0:x
La désactivation de S0:9 se fera par l'activation de S0:10
La programmation en Ladder exige que les numéros d'ordre des étapes soit continus.

7.3 Étape initiale

Le fonctionnement par Stepset nécessite qu'au moins une mémoire soit active au lancement du programme. C'est le rôle des mémoires de type Sn:0

7.4 Le retour au début

Il se fait par l'activation, fugitive, d'une dernière étape, qui relance la première.

7.5 En résumé

Dans le registre Sn,
Sn:0 est active au lancement du programme
La cellule mémoire Sn:m+1 ne pourra être active que si Sn:m est active.
Cette dernière sera désactivée par l'activation de Sn:m+1

On ne peut trouver qu'une seule cellule active à un instant donné. Si on a besoin de deux étapes actives simultanément, il faut utiliser deux registres différents, Sn et Sm

Stepout force l'activation d'une cellule et désactive toutes les autres

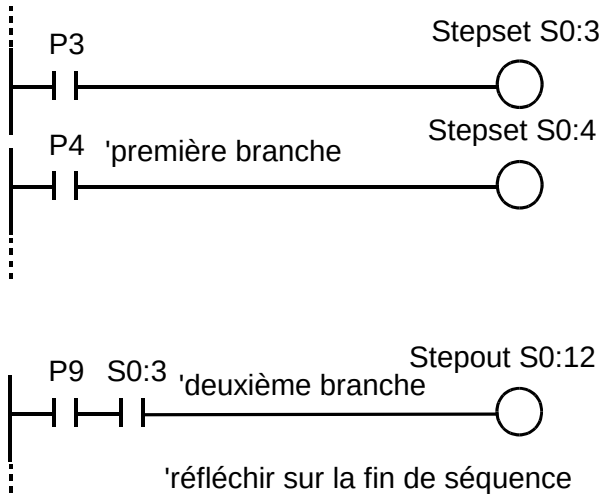
8 .Implémentation des structures du Grafcet

8.1 La boucle unique

Elle se fait au sein d'un même registre Sn.
Sn:0 représente l'étape initiale
La survenue du dernier événement active la mémoire Sn:fin qui ne restera active que le temps d'activer Sn:0

8.2 Le choix de séquences

Reste au sein d'un même registre Sn.
Exemple :



La deuxième branche commence par Stepout, elle se poursuit avec des Stepset

8.3 Les séquences simultanées

Il faut autant de registres que de séquences.
Chaque registre possède son étape initiale, Sm:0.
Les séquences sont lancées par un même événement
Prévoir une étape de synchronisation à la fin de chaque séquence
L'événement de fin peut être « toujours vrai »

9 .Terminologie

PLC = Automate programmable industriel (API)
Scan time : durée de cycle. Voir le paragraphe 4
Step : la traduction littérale est pas, marche d'un escalier. Ici désigne l'équivalent de l'étape d'un Grafcet