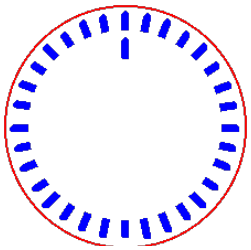
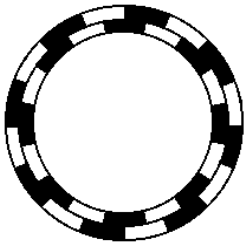
Master PAIP Mécatronique 1
Automatisme et Supervision, second cours 28/1/21

1 – Justification code GRAY : voir ma fiche capteurs rotatifs. D'abord le capteur incrémental : sur un disque, on prévoit des graduations, soit contact, soit peinture réfléchissante ou non (émetteur et récepteur optiques du même côté), soit disque opaque et orifices, soit disque transparent et peinture opaque (émetteur et récepteur optiques de chaque côté)… Si une seule piste (et un capteur) on ne connaît pas le sens (mais si c'est nous qui commandons le moteur, on peut s'en douter). On prévoit dans ce cas un moyen de trouver un zéro (home), soit sur le disque, soit un capteur de fin de course sur le dispositif.





Pour connaître le sens de rotation, on peut prévoir deux pistes (et 2 capteurs). Si l'on choisit un code Gray (00, 01, 11, 10), à chaque modification il suffit de lire les deux capteurs et d'avoir mémorisé la valeur précédente. Un des avantages du code Gray, est qu'il suffit de décaler un des capteurs d'un pas (et les marques sur les pistes) pour simplifier grandement la peinture (ou les orifices).

Ce dispositif était par exemple utilisé dans les anciennes souris : une boule (d'environ 2cm) appuie sur 2 axes perpendiculaires, au bout desquels se trouvent deux disques d'environ 1 cm. Les capteurs optiques, au lieu d'être juste décalés d'un pas, le sont d'un nombre impair, environ des deux côtés pour plus de facilité.

2 – comment code-t-on ? : C'est du Tout ou Rien. On désire qu'en passant d'un nombre à son suivant (+1) ou précédent (-1), on n'aie qu'un seul bit qui change. On désire de plus que les zéro rajoutés à gauche d'un nombre ne soient pas significatifs. On commence par 0, puis 1. N'ayant plus de possibilités sur un chiffre, on on se met sur 2 bits. On rajoute un 0 devant les cas précédents donc 00 puis 01. Le suivant commence par 1, comme un seul bit a le droit de changer, le second reste 1 donc 11. puis 10. Sur 3 bits, on gardera les mêmes premiers codes (précédés d'un zéro). La combinaison suivante débutera donc obligatoirement par 1, donc les deux autres bits ne peuvent pas changer. On continuera à prendre les mêmes codes, en ordre inverse, débutant par 1 : 110, 111, 101 et 100. En passant à 4 bits, on précède ces 8 cas d'un 0, les 8 suivants étant les mêmes, dans l'ordre inverse, précédés d'un 1. Donc, sur 4 bits :

0000 0001 0011 0010 | 0110 0111 0101 0100 | 1100 1101 1111 1110 1010 1011 1001 1000

Ce codage est utilisé dans les cas où des valeurs ne peuvent varier que par incrémentation ou décrémentation, voir l'explication dans la fiche. De plus, si l'on voit que plus d'un bit a changé entre deux valeurs, c'est qu'il y a eu un problème (en général le nombre a changé trop vite, le système n'a pas eu le temps de lire toutes les valeurs, par exemple en cas de choc). Il faut par contre passer en binaire naturel pour tout autre calcul que l'incrémentation.

3 – Convertisseur Gray <-> binaire

C'est un problème combinatoire : pour chaque code Gray on a un code binaire, et un seul. Nous nous limitons à 3 bits. Pour convertir du binaire en Gray, on fait une table de vérité (une table de vérité prend en compte tous les cas, mais dans n'importe quel ordre). On essaye de trouver, sortie par sortie, l'équation en fonction des entrées. On peut les déterminer directement avec un peu d'expérience, comme ci-contre. Pour l'autre sens, nous allons utiliser des tableaux de Karnaugh (c'est aussi une table de vérité, mais pour une seule sortie, et l'ordre est imposé : code Gray) :

	$b_2 b_1 b_0$	$g_2 g_1 g_0$
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 0	0 1 1
3	0 1 1	0 1 0
4	1 0 0	1 1 0
5	1 0 1	1 1 1
6	1 1 0	1 0 1
7	1 1 1	1 0 0

$$g_2 = b_2$$

$$g_1 = \overline{b_2}.b_1 + b_2.\overline{b_1} = b_1 \oplus b_2$$

$$g_0 = \overline{b_1}\, b_0 + \overline{b_0}\, b_1 = b_0 \oplus b_1$$

$b2 = g2$

$b1 = g1.\overline{g2} + \overline{g1}.g2 = g1 \oplus g2 = g1 \oplus b2$

$\overline{b1} = \overline{g1.\overline{g2} + \overline{g1}.g2} = (g1.\overline{g2}).(\overline{g1}.g2)$

$= (\overline{g1}+g2).(g1+\overline{g2})$

$= \overline{g1}.g1 + \overline{g1}.\overline{g2} + g2.g1 + \overline{g2}.g2$

$= \overline{g1}.\overline{g2} + g1.g2$

$bo = g2.\overline{g1}.\overline{go} + \overline{g2}.\overline{g1}.go + g2.g1.go + \overline{g2}.g1.\overline{go}$

$= go(\overline{g2}.\overline{g1} + g2.g1) + \overline{go}(\overline{g2}.g1 + g2.\overline{g1})$

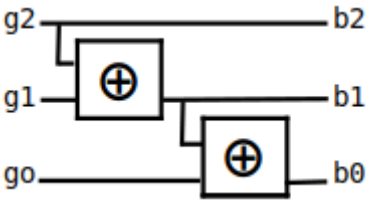
$= go.\overline{b1} + \overline{go}.b1 = go \oplus b1$

g1g0

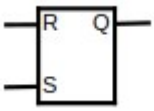
g2 \ b1	00	01	11	10
0	0	0	1	1
1	1	1	0	0

g1g0

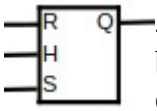
g2 \ b0	00	01	11	10
0	0	1	0	1
1	1	0	1	0



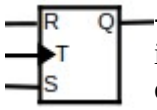
4 – Les bascules : Une bascule est l'élément de base d'une mémoire. Quand on l'allume (Set), elle reste allumée jusqu'à ce qu'on l'éteigne (Reset). Il y a 3 sortes de bascules :



- RS : S l'allume, R l'éteint. Si ni R ni S, la bascule reste dans l'état précédent. Si vous demandez en même temps S et R, ça dépend, on peut prévoir des bascules à priorité enclenchement (notées RS chez Siemens) ou priorité déclenchement (notées SR). Ce sont celles que nous utiliserons principalement.



- RSH (H=hold, bloquer) : quand H=0 elle obéit aux ordres S et R, quand H=1 elle est bloquée, c'est à dire qu'elle garde son état même si on donne un ordre R ou S. On peut avoir diverses variantes, dont la RST : T (trigger) est l'inverse de H (obéit si T=1, bloque sinon)



- Maître-esclave (MS) : 3 entrées RST. Obéit (R ou S) au prochain front montant (c.a.d quand il passe de 0 à 1) de T. Si entre temps il y a eu plusieurs ordres, c'est le dernier qui est pris en compte (même s'il n'est plus activé au moment du front montant de T). Les diverses variantes se distinguent sur le comportement en cas de S et R simultanés, par ex. la JK clignote.

Ce qui distingue ce 3 classes de bascules est quand elles obéissent à R et S : soit tout le temps, soit dans des plages de temps donné, soit à un instant précis. Pour plus de détails : mon cours, ou Google

5 – exercice : Une porte de garage peut soit monter (M) soit descendre (D). On dispose, de 5 entrées : la position de la porte en haut (h) ou en bas (b), et 3 boutons : ouvrir (o), fermer (f), stop (s).

Résoudre un problème d'automatisation, c'est d'abord déterminer les Entrées/Sortie, en déduire le type de problème : ici ToR séquentiel, car 1) on ne veut pas appuyer sur le bouton tout le temps 2) ce sont des conditions différentes qui allument/éteignent les sorties (o allume M, h l'éteint).

Donc nous allons utiliser deux bascules, et devons déterminer 4 équations SM, RM, SD, RD (quand allumer et quand éteindre chaque sortie). Il faut prévoir ce qui se passe si je demande o et f en même temps, ou o (donc M) puis f avant d'être arrivé en h (ne pas oublier d'éteindre M)… Il y a donc beaucoup de cas à prévoir.

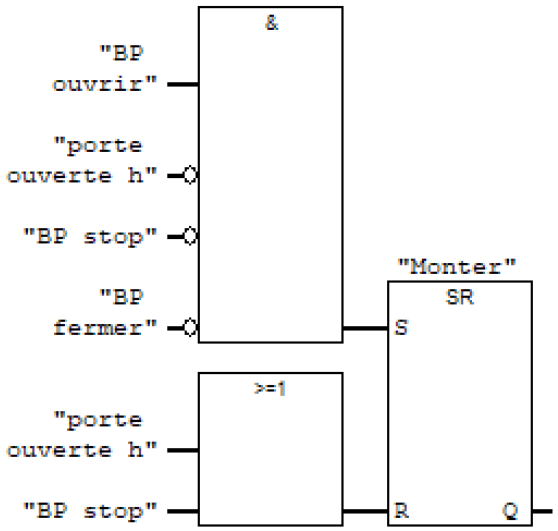
Je propose SM=o. $\overline{h}$. $\overline{f}$. $\overline{s}$ RM=s+h SD=f. $\overline{b}$. $\overline{s}$. $\overline{o}$ RD=s+b+M

6 – mise en œuvre sur automates Siemens (en Step7)

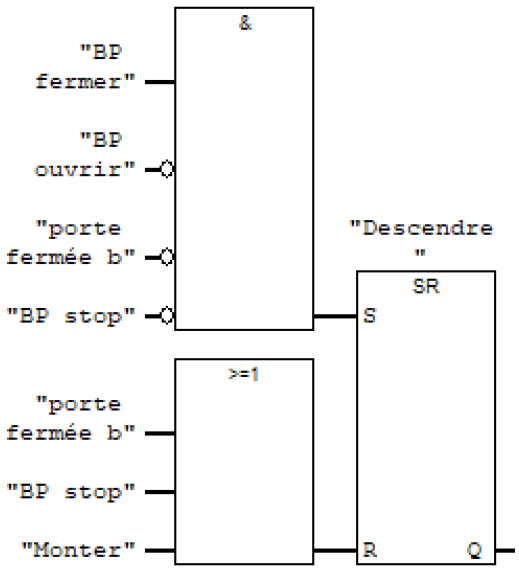
Voir ma doc Step7 sur moodle. Et page suivante.

7 – exercice : On dispose d'un portique, il peut Mo monter, De descendre, Av avancer, Ar reculer. 4 capteurs b, h, g, d disent s'il est en bas, en haut, à gauche ou à droite. Initialement, on est en b et g. à l'appui de dcy (départ cycle) on fait le cycle : Mo, Av, De, Mo, Ar, De. A résoudre pour la prochaine fois.

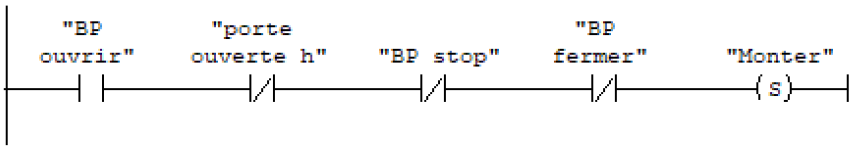
Réseau 1 : montée



Réseau 2 : descente

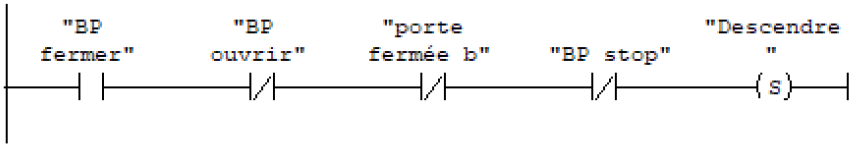


Réseau 1 : Set montée



Réseau 2 : Reset montée

Réseau 3 : Set descente



Réseau 4 : Reset descente

Réseau 1 : Set montée

```
U     "BP ouvrir"
UN    "porte ouverte h"
UN    "BP stop"
UN    "BP fermer"
S     "Monter"
```

Réseau 2 : Reset montée

```
O     "porte ouverte h"
O     "BP stop"
R     "Monter"
```

Réseau 3 : Set descente

```
U     "BP fermer"
UN    "BP ouvrir"
UN    "porte fermée b"
UN    "BP stop"
S     "Descendre"
```

Réseau 4 : Reset descente

```
O     "porte fermée b"
O     "BP stop"
O     "Monter"
R     "Descendre"
```

Ci-dessous une version avec rayon optique et temporisation de fermeture :