FICHE CONNAISSANCE

C 3-4 Traitement combinatoire de l'information

OBJECTIF *Effectuer des traitements logiques combinatoires sur des données binaires.*

Voir fiche C 3.3

1 Pourquoi traiter l'information ?

L'information codée disponible dans un système n'est pas directement adaptée à la prise de décision par une unité de commande automatique. Par exemple, sur le scooter hybride, on dispose des informations suivantes :

a : la clef de contact est en position active ;

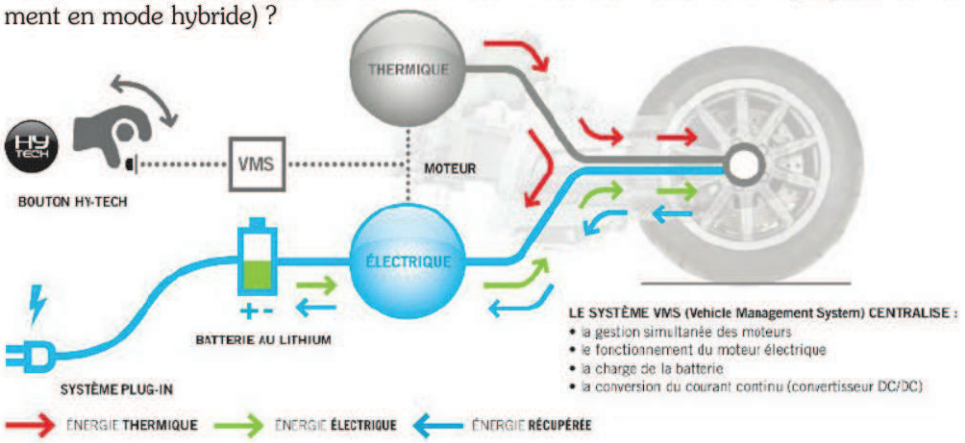
b : la batterie est correctement chargée ;

c : le moteur thermique est en fonctionnement ;

d : le véhicule roule en palier à plus de 30 km/h.

La commande automatique doit-elle mettre en marche le moteur électrique (fonctionnement en mode hybride) ?

Voir fiche S2



Document 1 La chaîne d'énergie du scooter hybride Piaggio est contrôlée par un traitement logique d'informations

La décision de commande automatique résultant de ces informations est obtenue par un traitement du type : « SI **a** ET **b** ET **c** ET **d** sont vraies (valeur **1**), ALORS **activer** (donner la valeur **1**) le mode hybride ». On pourrait aussi écrire : « SI **a** OU **b** OU **c** OU **d** est fausse (valeur **0**), ALORS désactiver (valeur **0**) le mode hybride ». Un traitement de ce type est appelé **traitement combinatoire**, car il consiste à définir quelles sont les ***combinaisons*** des variables binaires **a**, **b**, **c**, **d** qui activent la fonction résultante.

2 Traitements combinatoires de base

Tous les traitements combinatoires sont réalisés à partir de trois fonctions de base appelées « **ET** », « **OU** » et « **NON** ». Les **tables de vérité** de ces fonctions décrivent les combinaisons dont le résultat est **1**.

Variable a	Variable b	Résultat a ET b
0	0	0
0	1	0
1	0	0
1	1	1

Document 2 Table de vérité de la fonction ET

La fonction « **ET** » donne un résultat « **1** » si les deux variables d'entrée sont à « **1** ».

On peut généraliser à plus de deux entrées : toutes les entrées doivent être à **1** pour que la sortie soit à **1**.

Variable a	Variable b	Résultat a OU b
0	0	0
0	1	1
1	0	1
1	1	1

Document 3 Table de vérité de la fonction OU

La fonction « **OU** » donne un résultat « **1** » si l'une au moins des deux variables d'entrée est à « **1** ».

On peut généraliser à plus de deux entrées : une entrée au moins doit être à **1** pour que la sortie soit à **1**.

Variable a	Résultat NON a
0	1
1	0

Document 4 Table de vérité de la fonction NON

La fonction « **NON** » ne donne un résultat « **1** » que si l'entrée est à « **0** ».

Il n'y a qu'une entrée. Cette fonction donne le « **complément logique** » de la variable.

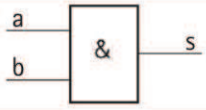
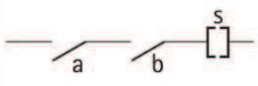
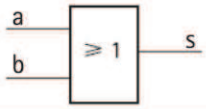
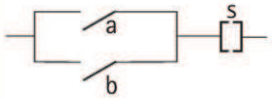
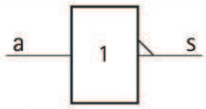
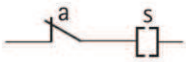
EXEMPLE

Fonctions combinatoires du tableau de bord du scooter hybride Piaggio

Le tableau de bord du scooter hybride donne en permanence à l'utilisateur l'état de fonctionnement des éléments de la chaîne d'énergie, tels que la batterie et les moteurs. Les informations affichées sont déduites des informations obtenues par des capteurs.

Variables d'entrée fournies par un capteur	Variable de sortie : état des voyants	Fonction
a = « le moteur électrique est en marche »	s = « le voyant **moteur électrique à l'arrêt** est allumé »	s = NON a
a = « la batterie est déchargée » b = « le chargeur de batterie est connecté »	s = « le voyant **batterie en charge** est allumé »	s = a ET b
a = « la batterie est chargée » b = « le chargeur de batterie est déconnecté »	s = « le voyant **batterie en charge** est éteint »	s = a OU b

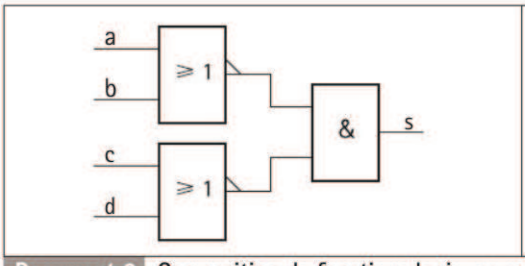
3 Représentations symboliques des fonctions logiques

Fonction	Forme algébrique	Forme électronique	Forme électrique
s = a ET b	$s = a \cdot b$		
s = a OU b	$s = a + b$		
s = NON a	$s = \overline{a}$		

Document 5 Différentes représentations utilisées pour les fonctions logiques combinatoires

4 Composition des fonctions de base

Afin de décrire des combinaisons complexes mettant en jeu de nombreuses variables, on peut composer les fonctions précédentes, comme indiqué dans l'exemple suivant :



$$s = \overline{(a + b)} \cdot \overline{(c + d)}$$

Document 6 Composition de fonctions logiques combinatoires NON, OU et ET

La représentation détaillée du résultat d'une fonction composée est toujours réalisable à l'aide d'une table de vérité, comme dans l'exemple suivant :

	(a , b) = (0 , 0)	(a , b) = (1 , 0)	(a , b) = (1 , 1)	(a , b) = (0 , 1)
(c , d) = (0 , 0)	0	0	0	0
(c , d) = (1 , 0)	0	1	1	1
(c , d) = (1 , 1)	0	1	1	1
(c , d) = (0 , 1)	0	1	1	1

Document 7 Table de vérité de la fonction $s = (a + b) \cdot (c + d)$

On retiendra que le nombre de combinaisons possibles dans une fonction combinatoire à **n** variables d'entrée est : $N = 2^n$. Ce nombre peut rapidement devenir excessivement grand pour une étude par table de vérité.

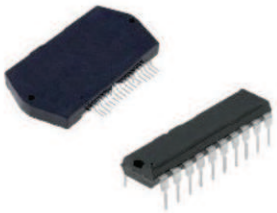
5 Réalisation des fonctions combinatoires

Pour réaliser les fonctions logiques, on peut employer soit des circuits électroniques logiques élémentaires reliés électriquement ensemble, soit des composants programmables.

5.1 Circuits électroniques logiques élémentaires

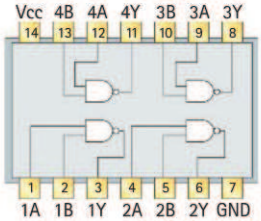
On trouve, chez les fabricants, des circuits intégrés réalisant directement les fonctions logiques élémentaires ainsi que des fonctions composées simples :

NON ET (*NAND*) $s = \overline{a \cdot b}$



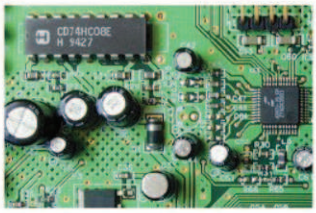
Circuits intégrés contenant des « portes » logiques

NON OU (*NOR*) $s = \overline{a + b}$



Câblage interne d'un circuit intégré à 4 portes logiques NON ET (représentation américaine)

OU EXCLUSIF (*XOR*) $s = b \cdot \overline{a} + a \cdot \overline{b}$



Implantation d'un circuit de portes logiques sur un circuit imprimé

Document 8 Mise en œuvre des fonctions logiques à l'aide de circuits intégrés

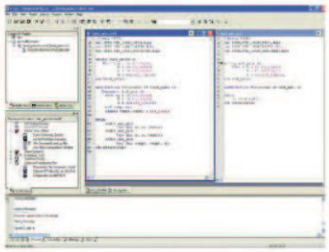
5.2 Composants programmables

Les microprocesseurs sont capables de réaliser toutes ces opérations logiques de manière efficace, et on peut les programmer pour réaliser toutes sortes de traitements logiques combinatoires. On trouve aussi de nombreux composants programmables spécialisés.

Circuit intégré programmable (NXP)

```
35  ENTITY nand_gate IS
36    port
37    (
38     a,b : in  std_logic;
39     c   : out std_logic
40    );
41  END nand_gate;
42
43
44  ARCHITECTURE dataflow OF nand_gate IS
45
46  BEGIN -- dataflow
47    c <= not ( a and b ) ;
48  END dataflow;
```

Programmation d'une porte NON ET (NAND) dans un langage spécialisé



Implantation d'un ensemble de fonctions logiques en langage de programmation

Document 9 Mise en œuvre des fonctions logiques à l'aide de composants programmables

5.3 Autres réalisations

Il est possible de réaliser les fonctions logiques avec des composants pneumatiques ainsi qu'à l'aide de contacts et de relais électriques. Les commandes obtenues sont souvent volumineuses et on les implante dans des systèmes spécialisés. Ces technologies sont de moins en moins usitées.

6 Conception des fonctions

L'analyse et la synthèse des fonctions de logique combinatoire sont grandement facilitées par l'existence d'une théorie élégante et efficace : l'algèbre de Boole. En voici les principaux résultats :

$a \cdot a = a$	$1 \cdot a = a$	$0 \cdot a = 0$
$a + a = a$	$1 + a = 1$	$0 + a = a$
$\overline{\overline{a}} = a$	$\overline{a \cdot b} = \overline{a} + \overline{b}$	$\overline{a + b} = \overline{a} \cdot \overline{b}$
$a + (b \cdot c) = (a + b) \cdot (a + c)$		$a \cdot (b + c) = a \cdot b + a \cdot c$

Document 10 Algèbre de Boole

À moi de le faire !

Écrire sous forme de logigramme les fonctions suivantes (plusieurs solutions sont possibles) :

$s = a \cdot \overline{b} + \overline{a} \cdot b$; $s = \overline{a \cdot \overline{b \cdot c}}$; $s = \overline{a + (\overline{b \cdot c})}$