Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 69 (1978)

Heft: 22: Sondernummer Elektrotechnik 1978 = Edition spéciale Electrotechnique 1978

Artikel: Arbres de décision pour systèmes logiques câblés ou programmés

Autor: Mange, D.

DOI: https://doi.org/10.5169/seals-914960

Arbres de décision pour systèmes logiques câblés ou programmés

Par D. Mange

164.053:621.3.049.73:681.3.06

Tout système logique combinatoire peut être décrit par un mode de représentation semblable à un organigramme: l'arbre de décision binaire. Les propriétés de cet arbre et son éventuelle transformation en un algorithme de décision binaire sont étudiées. On montre enfin qu'un réseau de démultiplexeurs (système combinatoire) ou qu'un processeur à un bit, la machine de décision binaire (système séquentiel), peuvent réaliser n'importe quel arbre ou algorithme.

Jedes System logischer Verknüpfungen kann mittels einer organigrammähnlichen Darstellung beschrieben werden, dem binären Entscheidungsbaum. Dessen Eigenschaften und mögliche Umwandlung in einen Algorithmus für binäre Entscheidungen werden untersucht. Abschliessend wird gezeigt, dass ein Netz von Demultiplexern (kombinatorisches System) oder ein 1-Bit-Prozessor, die binäre Entscheidungsmaschine (sequentielles System), jeden Entscheidungsbaum oder Algorithmus verwirklichen kann.

1. Introduction

A l'aide d'un exemple simple, un comparateur de deux nombres binaires, on montre que tout système logique peut être représenté par un arbre de décision binaire; celui-ci peut, dans certains cas, être transformé en un mode de représentation plus général, l'algorithme de décision binaire. Tout algorithme ou tout arbre de décision binaire est réalisable par un réseau de démultiplexeurs (système logique combinatoire) ou par un programme pour un processeur à un bit (système logique séquentiel).

Les arbres et algorithmes de décision binaire décrivent aussi bien le matériel (systèmes logiques câblés) que le logiciel (systèmes logiques programmés), et suggèrent une unification possible du langage des informaticiens.

2. Arbres de décision binaire

2.1 Exemple: comparateur de deux nombres

Deux nombres décimaux A et B sont représentés dans le système de numération binaire à l'aide de deux bits chacun: A_1, A_0 pour le nombre A et B_1, B_0 pour le nombre B (table I). Un système logique combinatoire doit déterminer si A est supérieur à B ($A > B$), égal ($A = B$) ou inférieur à B ($A < B$) (fig. 1). La table de vérité (table II) est une représentation possible du comportement de ce système: les quatre variables A_1, B_1, A_0, B_0 déterminent $2^4 = 16$ états d'entrée (numérotés de 0 à 15), tandis que la comparaison des nombres A et B produit trois états de sortie Z distincts ($A > B$, $A = B$, $A < B$); un codage possible de ces trois états est donné dans la table III.

Représentation binaire des nombres A et B Table I

A_1 B_1	A_0 B_0	A B
0	0	0
0	1	1
1	0	2
1	1	3

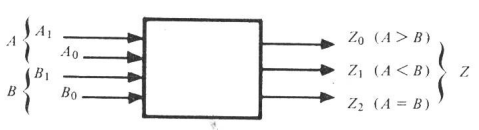

Fig. 1 Système logique combinatoire comparant les nombres A et B

Les formes canoniques décimales et algébriques des trois fonctions Z_2, Z_1 et Z_0 découlent de la table de vérité (table II):

$$Z_2(A_1, B_1, A_0, B_0) = \sum 0, 3, 12, 15 = \overline{A}_1 \overline{B}_1 \overline{A}_0 \overline{B}_0 + \overline{A}_1 \overline{B}_1 A_0 B_0 + A_1 B_1 \overline{A}_0 \overline{B}_0 + A_1 B_1 A_0 B_0 \quad (1)$$

$$Z_1(A_1, B_1, A_0, B_0) = \sum 1, 4...7, 13 = \overline{A}_1 \overline{B}_1 \overline{A}_0 B_0 + \overline{A}_1 B_1 \overline{A}_0 \overline{B}_0 + \overline{A}_1 B_1 \overline{A}_0 B_0 + \overline{A}_1 B_1 A_0 \overline{B}_0 + \overline{A}_1 B_1 A_0 B_0 + A_1 B_1 \overline{A}_0 B_0 \quad (2)$$

$$Z_0(A_1, B_1, A_0, B_0) = \sum 2, 8...11, 14 = \overline{A}_1 \overline{B}_1 A_0 \overline{B}_0 + A_1 \overline{B}_1 \overline{A}_0 \overline{B}_0 + A_1 \overline{B}_1 \overline{A}_0 B_0 + A_1 \overline{B}_1 A_0 \overline{B}_0 + A_1 \overline{B}_1 A_0 B_0 + A_1 B_1 A_0 \overline{B}_0 \quad (3)$$

La réalisation des expressions (1), (2) et (3), éventuellement simplifiées, à l'aide d'opérateurs logiques tels que ET, OU, NON, NAND, NOR, constitue la méthode classique de synthèse des systèmes combinatoires [1][1]).

2.2 Arbre de décision binaire

Un *arbre de décision binaire* (fig. 2a) est un mode de représentation des systèmes logiques combinatoires qui est constitué par l'assemblage de deux types d'éléments ou *instructions:*

– une instruction de *test* (en anglais «*if...then...else*»), représentée par un losange; chaque instruction de test est définie par un numéro décimal ou *adresse* i ($1 \leq i \leq 15$), une variable a ($a \in \{A_1, B_1, A_0, B_0\}$), une entrée (provenant d'une instruction précédente ou réalisant le début de l'arbre) et deux sorties ($a = 1$, $a = 0$) conduisant à une instruction suivante selon l'état de la variable a.

– une instruction de *sortie*, représentée par un rectangle; chaque instruction de sortie est définie par son adresse i ($16 \leq i \leq 31$), un état de sortie ($Z \in \{0, 1, 2\}$) et une entrée (provenant d'une instruction précédente).

Les règles d'assemblage sont définies ainsi:

– il existe une, et une seule, instruction initiale ($i = 1$ dans la figure 2a);

– une sortie d'une instruction de test n'est reliée qu'à une seule entrée d'une instruction suivante;

– une entrée d'une instruction (de test ou de sortie) n'est reliée qu'à une seule sortie d'une instruction précédente.

Chaque chemin reliant l'instruction initiale à une instruction de sortie est une *branche* de l'arbre. Il existe ainsi seize branches dans l'arbre de la figure 2a: chacune d'elles représente un état d'entrée du système combinatoire (table II) ainsi que l'état de sortie associé. L'arbre de décision binaire de la figure 2a est un mode de représentation équivalent à celui de la table de vérité (table II).

[1]) Voir la bibliographie à la fin de l'article.

Table de vérité du comparateur des nombres A (A_1, A_0) et B (B_1, B_0) Table II

N°	A_1	B_1	A_0	B_0	Z	Z_2	Z_1	Z_0
0	0	0	0	0	2	1	0	0
1	0	0	0	1	1	0	1	0
2	0	0	1	0	0	0	0	1
3	0	0	1	1	2	1	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	0	1	0
6	0	1	1	0	1	0	1	0
7	0	1	1	1	1	0	1	0
8	1	0	0	0	0	0	0	1
9	1	0	0	1	0	0	0	1
10	1	0	1	0	0	0	0	1
11	1	0	1	1	0	0	0	1
12	1	1	0	0	2	1	0	0
13	1	1	0	1	1	0	1	0
14	1	1	1	0	0	0	0	1
15	1	1	1	1	2	1	0	0

Codage des trois états de sortie Table III

	Z	Z_2	Z_1	Z_0
$A > B$	0	0	0	1
$A < B$	1	0	1	0
$A = B$	2	1	0	0

2.3 Arbre complet et arbre partiel

On remarque dans la figure 2a, comme dans la table II, que l'état d'entrée A_1, $B_1 = 01$ produit l'état de sortie $Z = 1$ indépendamment des valeurs de A_0 et B_0; de même, l'état d'entrée A_1, $B_1 = 10$ produit l'état de sortie $Z = 0$. Il en découle un nouvel arbre de décision binaire, équivalent au précédent et représenté dans la figure 2b.

De façon plus générale, on appellera *arbre complet* (ou *canonique*) tout arbre réalisant les 2^n états de n variables avec 2^n branches (cas de la figure 2a avec $n = 4$ et $2^n = 16$); on admet de plus que, dans une même branche, une même variable n'apparaît qu'une fois: chaque branche contient donc n instructions de test. Dans le cas où le nombre de branches b est inférieur à 2^n (cas de la fig. 2b avec $b = 10 < 16$), on dira que l'arbre est *partiel* (ou *simplifié*).

Dans un arbre qui possède m instructions de test, il existe $2m$ sorties de ces instructions et m entrées dont l'une est l'entrée de l'instruction initiale et les $(m - 1)$ autres sont des entrées reliées à des sorties précédentes. Il existe alors b branches avec

$$b = 2m - (m - 1) = m + 1 \tag{4}$$

donc

$$m = b - 1 \tag{5}$$

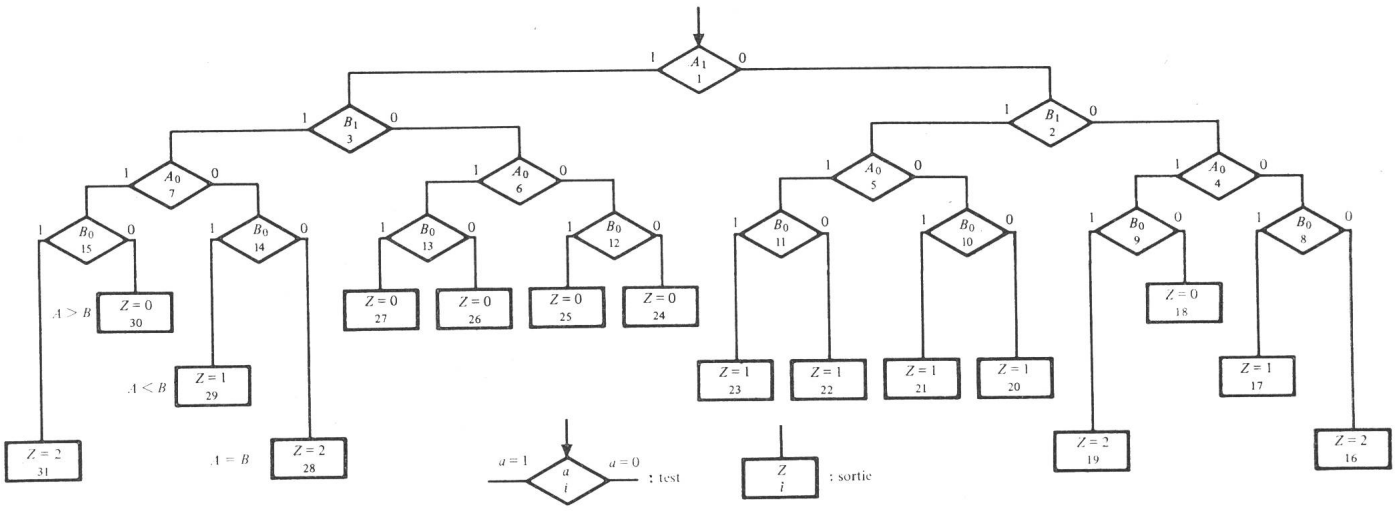

Fig. 2a Arbre complet du comparateur

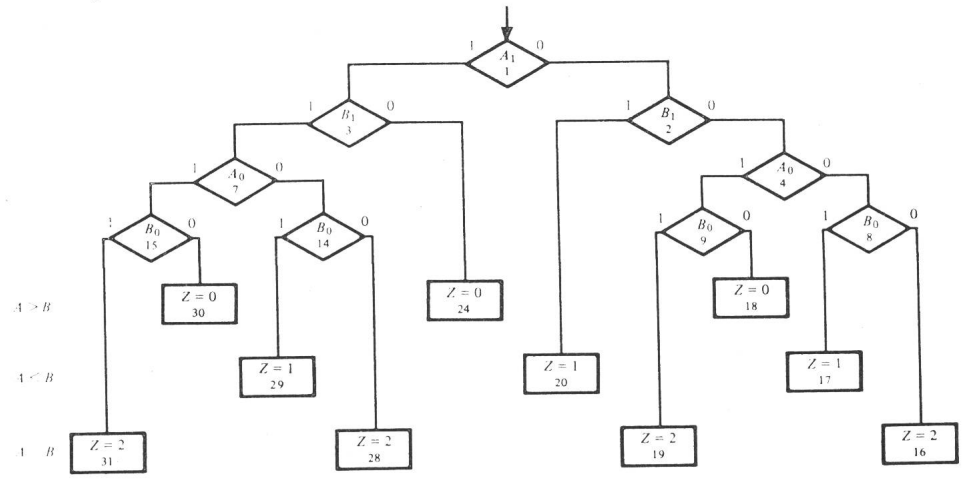

Fig. 2b Arbre partiel du comparateur

L'arbre (complet) de la fig. 2a comporte en effet seize branches et quinze instructions de test, celui de la fig. 2b dix branches et neuf instructions de test.

2.4 *Représentations algébrique et tabulaire*

Chaque branche d'un arbre complet représente un état d'entrée de toutes les variables du système combinatoire (A_1, B_1, A_0, B_0 dans la fig. 2a) et réalise ainsi un minterme de ces variables. Chaque branche d'un arbre partiel réalise un minterme ou un produit partiel (monôme) des variables d'entrée; on déduit de la fig. 2b les formes algébriques suivantes (polynômes) des trois fonctions Z_2, Z_1 et Z_0:

$$Z_2 = \overline{A_1}\,\overline{B_1}\,\overline{A_0}\,\overline{B_0} + \overline{A_1}\,\overline{B_1}\,A_0\,B_0 + A_1\,B_1\,\overline{A_0}\,\overline{B_0} + A_1\,B_1\,A_0\,B_0 \quad (6)$$

$$Z_1 = \overline{A_1}\,\overline{B_1}\,\overline{A_0}\,B_0 + \overline{A_1}\,B_1 + A_1\,B_1\,\overline{A_0}\,B_0 \quad (7)$$

$$Z_0 = \overline{A_1}\,\overline{B_1}\,A_0\,\overline{B_0} + A_1\,\overline{B_1} + A_1\,B_1\,A_0\,\overline{B_0} \quad (8)$$

En utilisant la représentation bidimensionnelle [2, pp. 19 à 37] définie par

$$a \cdot b = |\,ab\,| \qquad a + b = \left|\begin{matrix} a \\ b \end{matrix}\right| \quad (9)$$

les relations (6), (7), (8) peuvent s'écrire sous la forme

$$\left|\begin{matrix} \overline{A_1} \left|\begin{matrix} \overline{B_1} \left|\begin{matrix} \overline{A_0} \left|\begin{matrix} \overline{B_0} \in Z_2 \\ B_0 \in Z_1 \end{matrix}\right. \\ A_0 \left|\begin{matrix} \overline{B_0} \in Z_0 \\ B_0 \in Z_2 \end{matrix}\right. \end{matrix}\right. \\ B_1 \in Z_1 \end{matrix}\right. \\ A_1 \left|\begin{matrix} \overline{B_1} \in Z_0 \\ B_1 \left|\begin{matrix} \overline{A_0} \left|\begin{matrix} \overline{B_0} \in Z_2 \\ B_0 \in Z_1 \end{matrix}\right. \\ A_0 \left|\begin{matrix} \overline{B_0} \in Z_0 \\ B_0 \in Z_2 \end{matrix}\right. \end{matrix}\right. \end{matrix}\right. \quad (10)$$

qui reflète exactement la topologie de l'arbre partiel; en particulier, chaque trait vertical de (10) représente une instruction de test de la figure 2b.

Chaque minterme ou produit partiel réalisé par la branche d'un arbre est appelé *produit de branche*. Les dix produits de branche de l'arbre partiel (fig. 2b) peuvent être représentés dans la table de *Karnaugh* de la figure 3.

2.5 *Propriétés*

L'examen de l'expression (10) et de la table de Karnaugh (fig. 3) met en évidence les trois propriétés d'un arbre de décision binaire:

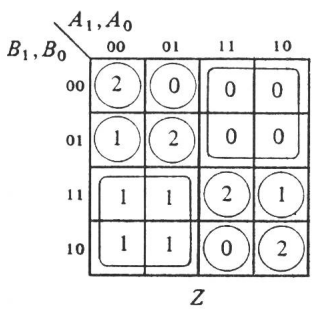

Fig. 3 Table de Karnaugh de l'arbre partiel du comparateur
($Z = 0 : A > B$; $Z = 1 : A < B$; $Z = 2 : A = B$)

Programme de l'arbre partiel de la figure 2b Table IV

i	a	i^+ ($a = 1$)	i^+ ($a = 0$)
1	A_1	3	2
2	B_1	$Z = 1$	4
3	B_1	[7] 4	$Z = 0$
4 [7]	A_0	9	8
[7	A_0	(15) 9	(14) 8]
8 (14)	B_0	$Z = 1$	$Z = 2$
9 (15)	B_0	$Z = 2$	$Z = 0$
(14	B_0	$Z = 1$	$Z = 2$)
(15	B_0	$Z = 2$	$Z = 0$)

a) L'ensemble des produits de branche recouvre les 2^n mintermes des n variables (dans la figure 3, les dix produits de branche recouvrent les seize mintermes des quatre variables). Cette propriété est appelée la *couverture*.

b) L'ensemble des produits de branche définit une partition sur l'ensemble des mintermes; chaque minterme est recouvert par un seul produit, et chaque produit ne contient qu'un seul état de sortie (fig. 3): c'est la propriété de *séparation*.

c) Chaque produit de branche contient la variable (vraie ou inversée) de l'instruction de test initiale ($\overline{A_1}$ ou A_1 dans la figure 2b et dans l'expression (10)); de façon plus générale, chaque portion de branche, comprise entre l'entrée d'une instruction de test définie par la variable a et une instruction de sortie, réalise un produit contenant la variable a ou son complément $\overline{a}$ (dans la figure 2b, par exemple, les quatre portions de branche, comprises entre l'instruction de test 4 et les instructions de sortie, réalisent quatre produits de branche qui contiennent chacun la variable A_0 ou son complément $\overline{A_0}$: $\overline{A_0}\,\overline{B_0}$, $\overline{A_0}\,B_0$, $A_0\,\overline{B_0}$, $A_0\,B_0$). Cette propriété est la *compatibilité*; un ensemble de produits logiques vérifie cette propriété s'il peut être représenté par une forme bidimensionnelle telle que (10).

Un ensemble de produits de branche qui vérifie les trois propriétés précédentes (couverture, séparation, compatibilité) est un *ensemble standard*. Un tel ensemble est toujours réalisable par un arbre et, inversement, un arbre produit toujours un ensemble standard [3].

2.6 *Arbre minimal*

Une même table de vérité peut généralement être représentée par plusieurs arbres de décision binaire distincts, mais équivalents. Ceux d'entre eux qui comportent le plus petit nombre d'instructions de test, c'est-à-dire le plus petit nombre de branches (en vertu de la relation (4)), sont les *arbres minimaux* de cette table. La détermination d'un arbre minimal nécessite la recherche d'un ensemble standard comportant un nombre minimal de produits de branche (§ 2.5).

2.7 *Conception des arbres*

Pour certains systèmes logiques, en particulier ceux décrits par des fonctions symétriques [4, pp. 508–515, 585–586], l'intuition permet d'établir directement un arbre, complet ou partiel [5]. La recherche systématique d'un arbre est toujours possible si l'on détermine, dans une table de *Karnaugh* par exemple, un ensemble standard de produits de branche. En minimisant cet ensemble standard [6; 7], on peut alors déterminer un arbre minimal.

3. Algorithmes de décision binaire

3.1 Programme

Une autre représentation tabulaire de l'arbre de décision binaire (fig. 2b) peut être obtenue en procédant comme suit (table IV):

– on trace autant de lignes que d'instructions de test (neuf lignes, avec $1 \leqq i \leqq 15$);

– chaque instruction de test (chaque ligne) est définie par son adresse i, sa variable a, les adresses de l'instruction suivante i^+ pour $a = 1$ et pour $a = 0$;

– lorsque l'instruction suivante est une instruction de sortie, on remplace l'adresse de celle-ci par l'état de sortie ($Z = 2,1$ ou 0).

La table obtenue est le *programme* de l'arbre.

3.2 Réduction d'un programme

Deux instructions de test d'adresses i et j sont *identiques* (ou *équivalentes*) si elles ont la même variable ($a_i = a_j = a$), les mêmes adresses de l'instruction future ($i_i^+ = i_j^+$) et/ou les mêmes états de sortie ($Z_i = Z_j$), pour $a = 1$ comme pour $a = 0$. Dans un tel cas, une seule instruction résultante d'adresse i peut remplacer la paire d'instructions initiales d'adresses i et j.

Les instructions 9 et 15 de la table 4 sont identiques: elles ont la même variable ($a = B_0$) et les mêmes états de sortie ($Z = 2$ pour $a = 1$, $Z = 0$ pour $a = 0$); les instructions 8 et 14 sont également identiques. Chaque groupe d'instructions identiques ($9 \equiv 15$, $8 \equiv 14$) est représenté par une seule instruction résultante (9, 8) et l'on remplace dans la table IV toute apparition des adresses supprimées (15, 14) par les adresses résultantes (9, 8): ces transformations sont indiquées entre parenthèses.

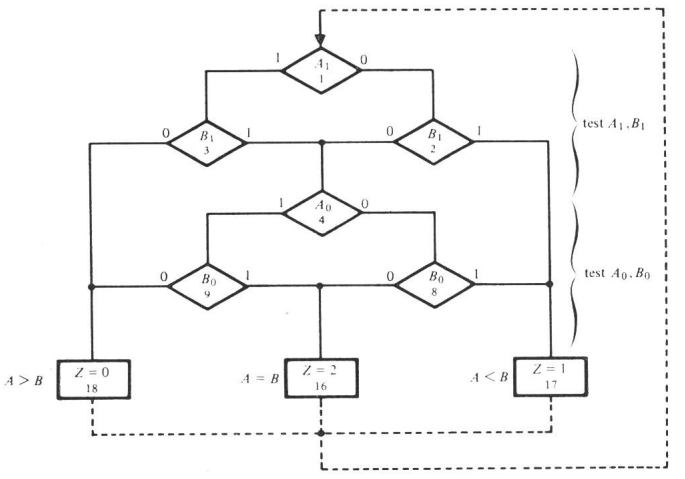

Fig. 4 Algorithme de décision binaire du comparateur des nombres A (A_1, A_0) et B (B_1, B_0)

Il apparaît alors que les instructions 4 et 7 sont identiques à leur tour et peuvent être remplacées par une instruction unique (4): cette transformation est indiquée entre crochets. Aucune autre réduction n'est ensuite possible.

Le programme original à neuf instructions de test est donc transformé en un programme équivalent, mais plus simple, à six instructions; une représentation possible, apparentée à celle d'un arbre, est donnée par la figure 4.

3.3 Algorithme de décision binaire

Un *algorithme de décision binaire* (fig. 4, sans les connexions en trait discontinu) est un mode de représentation des systèmes combinatoires qui a toutes les propriétés d'un arbre de décision binaire (§ 2.2), à l'exception de la règle d'assemblage suivante:

une entrée d'une instruction (de test ou de sortie) peut être reliée aux sorties de plusieurs instructions précédentes (l'entrée de l'instruction 4, p. ex., est reliée aux instructions 2 et 3).

L'arbre de décision binaire constitue un cas particulier de l'algorithme de décision binaire. Tout algorithme peut être transformé en un arbre équivalent [3], mais la réciproque n'est pas toujours vraie.

Le calcul des algorithmes minimaux (c'est-à-dire ceux qui réalisent une table de vérité donnée avec un nombre minimal d'instructions de test) constitue un problème combinatoire très complexe et n'a pas encore fait l'objet de recherches systématiques [8]; une méthode exhaustive peut être envisagée à partir des algorithmes proposés pour la simplification des arbres [6; 7], si l'on réduit les programmes de tous les arbres possibles d'une table de vérité donnée.

3.4 Généralisation

On remarque, dans la figure 4, que les instructions 1, 2, 3 (test A_1, B_1) et 4, 8, 9 (test A_0, B_0) constituent des assemblages dont la topologie est identique: l'algorithme met en évidence le caractère répétitif de la comparaison des bits de même poids des deux nombres A et B; on pourrait aisément généraliser l'algorithme pour comparer deux nombres ayant un nombre de bits plus élevé.

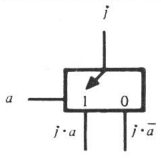

Fig. 5a Schéma d'un démultiplexeur d'une variable a

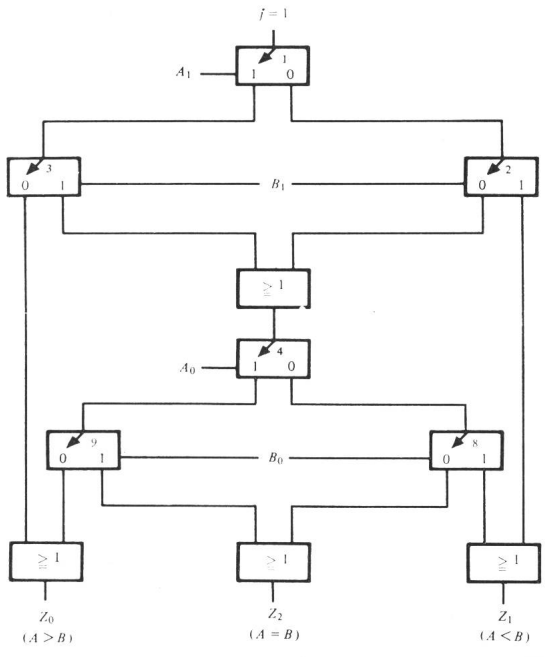

Fig. 5b Réseau de démultiplexeurs et de portes OU réalisant l'algorithme de la figure 4

4. Réalisation combinatoire: réseau de démultiplexeurs

4.1 Démultiplexeur

Un *démultiplexeur* d'une variable est un système logique (fig. 5a) réalisant les équations

$$x = j\,\overline{a} \qquad y = j\,a \tag{11}$$

où a est la variable de test, j la variable d'entrée et x, y les variables de sortie. Le démultiplexeur réalise ainsi une instruction de test selon la définition du § 2.2.

4.2 Réseau de démultiplexeurs

Un *réseau de démultiplexeurs* est un assemblage de démultiplexeurs et de portes OU (fig. 5b). Le réseau réalisant les fonctions Z_2 (6), Z_1 (7), Z_0 (8) du comparateur de nombres est directement construit à partir de l'algorithme de la figure 4, en procédant comme suit:

- chaque instruction de test est réalisée par un démultiplexeur;
- chaque instruction de sortie est réalisée par une porte OU;
- l'entrée d'une instruction de test, qui est commune à plusieurs sorties d'instructions précédentes, est réalisée par une porte OU (entrée de l'instruction 4);
- la variable d'entrée de l'instruction de test initiale ($i = 1$) prend la valeur $j = 1$.

En appliquant les relations (11) à chacun des démultiplexeurs de la figure 5b, on peut calculer les produits de branche du réseau et retrouver les équations (6), (7) et (8).

4.3 Réalisation spatiale d'un algorithme

Le réseau de démultiplexeurs est une réalisation *spatiale* (ou combinatoire) de l'algorithme de décision binaire; si les opérateurs logiques (démultiplexeurs, portes OU) sont idéaux, c'est-à-dire sans délai, un état d'entrée A_1, B_1, A_0, B_0 produit sans délai l'état de sortie Z_2, Z_1, Z_0 correspondant.

Programme de l'algorithme bouclé de la figure 4 Table V

i	T	a	i^+ $(a = 1)$	i^+ $(a = 0)$
1	1	A_1	3	2
2	1	B_1	17	4
3	1	B_1	4	18
4	1	A_0	9	8
8	1	B_0	17	16
9	1	B_0	16	18
i	T	a	i^+	Z
18	0	∅	1	0
17	0	∅	1	1
16	0	∅	1	2

Codage des variables d'entrée Table VI

a	a_1	a_0
A_1	0	0
B_1	0	1
A_0	1	0
B_0	1	1

5. Réalisation séquentielle: machine de décision binaire

5.1 Réalisation temporelle d'un algorithme

Le programme de la table IV suggère une réalisation *temporelle* (ou séquentielle) des fonctions Z_2, Z_1, Z_0; à chaque période d'un signal de référence (ou signal d'horloge), un système séquentiel exécute une instruction de l'algorithme. Sitôt qu'un état de sortie Z est déterminé, il faut reprendre le calcul à l'instruction initiale pour tenir compte d'une éventuelle variation de l'état d'entrée A_1, B_1, A_0, B_0: l'algorithme doit être

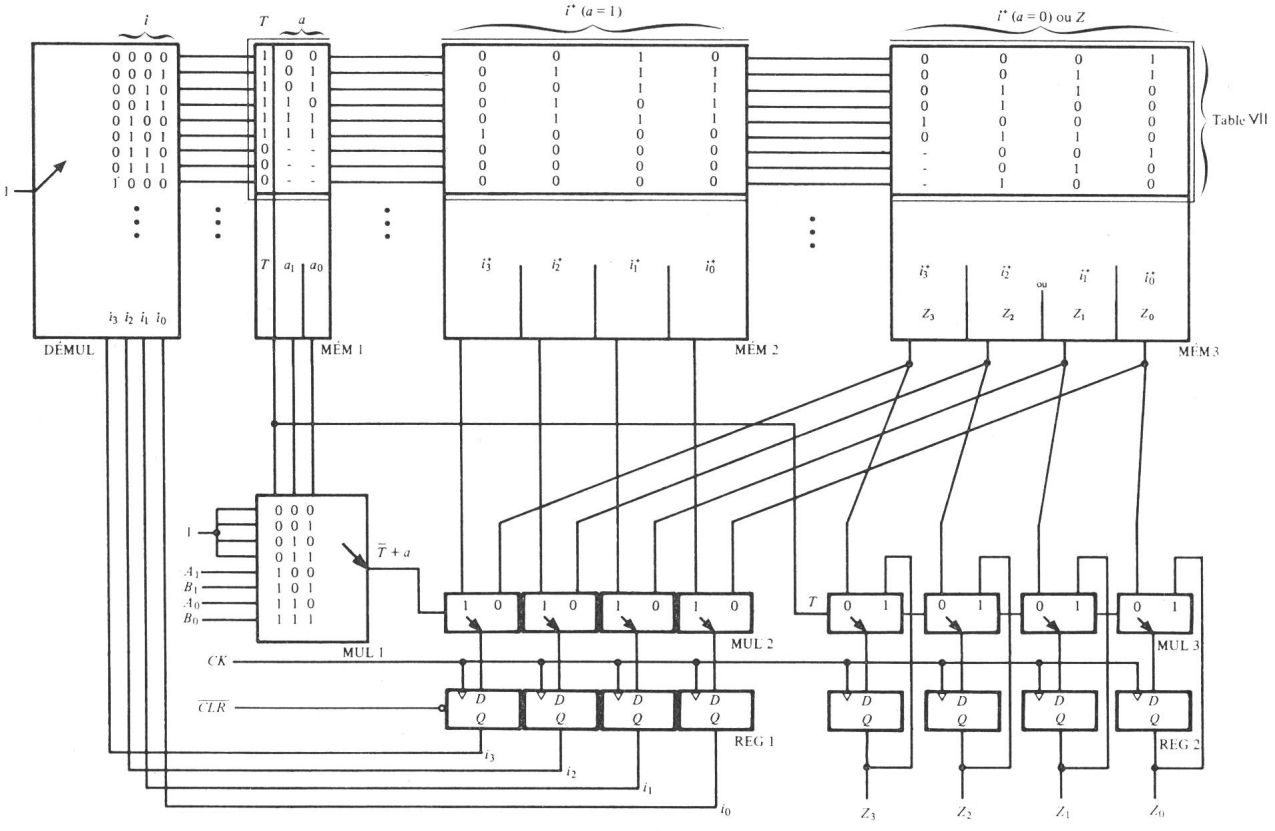

Fig. 6 Machine de décision binaire exécutant le programme de la table VII

Programme de la table V codé

Table VII

i	i_3	i_2	i_1	i_0	T	a_1	a_0	i_3^+	i_2^+	i_1^+	i_0^+	i_3^+ ou Z_3	i_2^+ Z_2	i_1^+ Z_1	i_0^+ Z_0
1	0	0	0	0	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
2	0	0	0	1	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**
3	0	0	1	0	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
4	0	0	1	1	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**
8	0	1	0	0	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**
9	0	1	0	1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
18	0	1	1	0	**0**	–	–	**0**	**0**	**0**	**0**	–	**0**	**0**	**1**
17	0	1	1	1	**0**	–	–	**0**	**0**	**0**	**0**	–	**0**	**1**	**0**
16	1	0	0	0	**0**	–	–	**0**	**0**	**0**	**0**	–	**1**	**0**	**0**

complété par une *boucle,* représentée en trait discontinu dans la figure 4. Pour effectuer l'algorithme complet, c'est-à-dire calculer un état de sortie à partir de l'instruction de test initiale, il faut, dans l'exemple traité, cinq périodes du signal d'horloge au plus (produit de branche $\overline{A}_1\,\overline{B}_1\,\overline{A}_0\,\overline{B}_0$ et sortie $Z = 2$) et trois périodes au moins (produit $\overline{A}_1\,B_1$ et sortie $Z = 1$).

5.2 Programmes

L'algorithme bouclé de la figure 4 peut être représenté par le programme de la table V dans lequel on distingue par une variable logique T les instructions de test ($T = 1$) des instructions de sortie ($T = 0$); pour chacune de celles-ci, on donne:

– l'adresse de l'unique instruction suivante (i^+) qui ne dépend pas de la variable a ($a = \varnothing$);
– l'état de sortie (Z).

En codant les variables d'entrée (A_1, B_1, A_0, B_0) à l'aide de deux variables a_1, a_0 (table VI) et les neuf instructions du programme de la table V à l'aide de quatre variables i_3, i_2, i_1, i_0, on obtient enfin le *programme codé* de la table VII.

5.3 Machine de décision binaire

On appelle *machine de décision binaire* tout système séquentiel capable de réaliser un algorithme de décision binaire bouclé (fig. 4 par exemple), c'est-à-dire d'exécuter un programme codé tel celui de la table VII. Il existe une grande variété d'architectures [9, pp. 75–96] et [10], et l'on propose ici une réalisation possible dans laquelle on s'est efforcé de rechercher la plus grande simplicité de structure (logigramme) et d'utilisation (programmation).

Le logigramme de la figure 6 est un assemblage d'éléments combinatoires (multiplexeurs: MUL; démultiplexeur: DÉMUL) et d'éléments séquentiels (bascules bistables du type D, regroupées en registres: REG; mémoires: MÉM).

Si l'instruction présente est une instruction de test ($T = 1$), les variables a_1, a_0 du multiplexeur MUL 1 choisissent la variable d'entrée ($a \in \{A_1, B_1, A_0, B_0\}$); selon la valeur de cette variable a, les multiplexeurs MUL 2 choisissent l'adresse de l'instruction future i_3^+, i_2^+, i_1^+, i_0^+ dans la mémoire MÉM 2 ($a = 1$) ou dans la mémoire MÉM 3 ($a = 0$), tandis que les multiplexeurs MUL 3, commandés par la variable T ($T = 1$), admettent l'état de sortie présent Z_3, Z_2, Z_1, Z_0. A la montée du signal d'horloge CK, l'adresse i_3^+, i_2^+, i_1^+, i_0^+ est transférée aux sorties des bascules du *registre d'instructions* REG 1, puis décodée par le démultiplexeur DÉMUL, tandis que l'état de sortie est inchangé.

Si l'instruction présente est une instruction de sortie ($T = 0$), les multiplexeurs MUL 2 choisissent l'unique adresse de l'instruction future i_3^+, i_2^+, i_1^+, i_0^+ dans la mémoire MÉM 2, tandis que les multiplexeurs MUL 3 choisissent l'état de sortie Z_3, Z_2, Z_1, Z_0 dans la mémoire MÉM 3, pour le présenter à l'entrée du *registre de sortie* REG 2. A la montée du signal d'horloge CK, l'adresse i_3^+, i_2^+, i_1^+, i_0^+ est transférée à la sortie du registre d'instructions REG 1, tandis que l'état de sortie apparaît à la sortie du registre REG 2.

Le signal $\overline{\text{CLR}}$ permet de remettre à zéro le registre REG 1 et impose l'instruction de test initiale (i_3, i_2, i_1, $i_0 = 0000$).

6. Conclusions

Une ou plusieurs fonctions logiques, complètement ou incomplètement définies, peuvent toujours être représentées par un arbre de décision binaire et, éventuellement, par un algorithme de décision binaire. Cet arbre ou cet algorithme sont directement réalisables par un système combinatoire (un réseau de démultiplexeurs) ou par un système séquentiel (une machine de décision binaire); le premier est un système logique câblé (l'information réside dans l'assemblage des démultiplexeurs), le second est un système logique programmé (l'information réside dans une mémoire).

La conception des machines de décision binaire (cas particulier des *processeurs à un bit*) et leur utilisation constituent l'axe principal des recherches entreprises actuellement à la Chaire de systèmes logiques de l'EPFL. Sur le plan pédagogique, il semble que les arbres et algorithmes de décision binaire soient spécialement bien adaptés pour faire le pont entre les systèmes logiques câblés et programmés, entre la théorie du matériel et celle du logiciel.

Bibliographie

[1] *D. Mange:* Analyse et synthèse des systèmes logiques. Traité d'Electricité. Vol. V. Saint-Saphorin, Editions Georgi, 1978.
[2] *R.L. Vallée:* Analyse binaire. Tome 1: Théorie et applications aux circuits combinatoires. Paris, Masson, 1970.
[3] *M. Davio* et *A. Thayse:* Sequential evaluation of Boolean functions. MBLE Report R 341. Bruxelles, Manufacture Belge de Lampes et de Matériel Electronique, 1977.
[4] *F.J. Hill* and *G.R. Peterson:* Introduction to switching theory and logical design. Second Edition. New York, John Wiley, 1974.
[5] *S.B. Akers:* Binary decision diagrams. IEEE Trans. C 27(1978)6, S. 509...516.
[6] *D. Mange* et *E. Sanchez:* Synthèse des fonctions logiques avec des multiplexeurs. Digital Processes 4(1978)1, p. 29...44.
[7] *E. Cerny, D. Mange* and *E. Sanchez:* Synthesis of minimal demultiplexer, multiplexer, and binary-decision trees. Montreal/Canada, Concordia University, 1977 (IEEE-Repository-No. R 77-372).
[8] *M. Silva Suarez:* Contributions à la synthèse programmée des automatismes logiques. Thèse de l'Institut National Polytechnique de Grenoble, 1978.
[9] *C.R. Clare:* Designing logic systems using state machines. New York, McGraw-Hill, 1973.
[10] *R.T. Boute:* The binary decision machine as programmable controller. Euromicro Newsletter 2(1976)1, p. 16...22.

Adresse de l'auteur

Daniel Mange, Professeur EPFL, Chaire de systèmes logiques, 16, chemin de Bellerive, 1007 Lausanne.