

The Design of Current Mode CMOS Multiple-Valued Circuits[†]

Young-hoon Chang
Dept. of Electr. Eng. & Comp. Sci.
Northwestern University
Evanston, IL 60208

Jon T. Butler
Dept. of Electr. & Comp. Eng.
Naval Postgraduate School, Code EC/Bu
Monterey, CA 93943-5004

Abstract

We propose an algorithm for the design of multiple-valued current-mode CMOS logic (CMCL) circuits that is based on the cost-table technique. The algorithm is a heuristic search technique (AO* algorithm) [10,11] applied to an AND-OR tree. It is significantly faster than Exhaustive Search while providing realizations that are almost as good. A new cost-table is also proposed that results in better realizations than obtained with a previous cost-table [14].

1 Introduction

The development of multiple-valued current-mode CMOS circuits (CMCL) [3,12,13,14] has resulted in a need for design techniques for such circuits. The development of charge-coupled device (CCD) circuits has inspired a close examination of the cost-table technique [2,8,9,14,16]. A natural extension of this work is a cost-table technique for CMCL [14]. In this paper, we show an improved cost-table for CMCL. The improvement occurs in two ways. First, the table is extended to include more operations. Second, an Exhaustive Search technique is replaced by an efficient search technique which is similar to AO* algorithm [10,11].

This paper is divided as follows; Section 2 is a description of notation, Section 3 describes the new cost-table and decomposition method, Section 4 is a description of improvements obtained, and Section 5 is the conclusions.

2 Background, Notation and Basic Circuit Elements

Let $R = \{0, 1, \dots, r-1\}$ be a set of r logic values, where $r \geq 2$. Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of n variables, where x_i takes on values from R . A function $f(X)$ is a mapping $f: R^n \rightarrow R$. If X is a single variable x , $f(x)$ is represented as an r -tuple, $\langle f(0), f(1), \dots, f(r-1) \rangle$. For example, if $r = 4$, then $f(x) = \langle 3, 2, 1, 0 \rangle$ represents a complement function in which 0 maps to 3, 1 to 2, 2 to 1, and 3 to 0. Let \bigcup_n^r be the set of all r -valued functions of n r -valued inputs. Let $c(f)$, the cost of function f , be a mapping $c: \bigcup_n^r \rightarrow R^{0+}$, where R^{0+} is the

set of real numbers. The cost function $c(f)$ introduced by Kerkhoff and Robroek [8] for the design of 4-valued CCD logic circuits correlates closely with the chip area occupied by the most compact implementation of f .

In the realization of a given function by cost-table, cost-table functions are combined using a connecting operation. The connecting operation $+$ between functions used in this paper is similar to ordinary addition with logic values viewed as integers. That is, if $f(X)$ is represented as the sum $f(X) = f_1(X) + f_2(X) + \dots + f_m(X)$, then, for any assignment ν of values to X , $f(\nu) = f_1(\nu) + f_2(\nu) + \dots + f_m(\nu)$, where each $f_i(\nu)$ is taken as an integer and where $+$ is ordinary addition, except when the sum exceeds $r-1$, the highest output logic value, in which case $+$ is undefined. For example, if $f_1(x) = \langle 0, 1, 2, 3 \rangle$ and $f_2(x) = \langle 3, 2, 1, 0 \rangle$, then $f_1(x) + f_2(x) = \langle 3, 3, 3, 3 \rangle$ and $f_1(x) + f_1(x)$ is undefined. The first example shows that the sum of the identity function $\langle 0, 1, 2, 3 \rangle$ and the complement function $\langle 3, 2, 1, 0 \rangle$ is the constant function $\langle 3, 3, 3, 3 \rangle$.

Let s be the cost of realizing the sum operation ($+$) between two functions. Thus, the cost of the realization $f = f_1 + f_2 + \dots + f_m$ is $c(f_1) + c(f_2) + \dots + c(f_m) + (m-1)s$, where the last term is the cost of $(m-1)$ two-input adders.

Function f is a basis function if and only if $f(X)$ is 1 for exactly one assignment of values to X and is 0 otherwise. Let BT be the union of all basis functions and the constant 0 function. BT is called the basis cost-table. F is a cost-table if and only if $BT \subseteq F \subseteq \bigcup_n^r$. The condition $BT \subseteq F$ guarantees that all functions can be realized as the sum ($+$) of cost-table functions. For example, in \bigcup_1^4 , $BT = \{\langle 0, 0, 0, 0 \rangle, \langle 0, 0, 0, 1 \rangle, \langle 0, 0, 1, 0 \rangle, \langle 0, 1, 0, 0 \rangle, \langle 1, 0, 0, 0 \rangle\}$. If $\langle 0, 0, 0, 1 \rangle$ is missing, it is impossible to realize certain functions, including $\langle 0, 0, 0, 1 \rangle$ itself. Conversely, any function $\langle a_0, a_1, a_2, a_3 \rangle$ can be realized as the sum of functions exclusively from BT , specifically a_0 functions of the form $\langle 1, 0, 0, 0 \rangle$, a_1 $\langle 0, 1, 0, 0 \rangle$, a_2 $\langle 0, 0, 1, 0 \rangle$, and a_3 $\langle 0, 0, 0, 1 \rangle$.

$c_F(f)$, the cost of realizing $f \in \bigcup_n^r$ with respect to cost-table F , is the minimal cost realization, specifically

$$c_F(f) = \min_{f_1, \dots, f_m \in F} \{c(f_1) + \dots + c(f_m) + (m-1)s\},$$

where c is the cost function. $f = f_1 + f_2 + \dots + f_m$ is said to be a minimal cost realization of f .

[†]Research supported by NATO Grant 423/84, by NSF Grant MIP-8706553, and by an NPS Direct Funded Grant in cooperation with the Naval Research Laboratory.

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE MAY 1991		2. REPORT TYPE		3. DATES COVERED	
4. TITLE AND SUBTITLE The Design of Current Mode CMOS Multiple-Valued Circuits				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School, Department of Electrical and Computer Engineering, Monterey, CA, 93943				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT We propose an algorithm for the design of multivalued current-mode CMOS logic (CMCL) circuits that is based on the cost-effective technique. The algorithm is a heuristic search technique (AO* algorithm) [1] - Exhaustive Search while providing realizations that are almost as good. A new cost-table is also proposed which results in better realizations than obtained with a previous cost-table [Id].					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 9	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

Two cost functions are of special interest.

Transistor Count

This cost function was introduced in [14] as a measure of the area in an implementation of a circuit. Area correlates closely with the number of transistors needed in the minimal circuit implementation. It is straightforward to calculate given the circuit, and we use it here. Because the adder uses no transistors, its cost, s , is 0.

Because it is useful in predicting cost in the AO^* algorithm, we also use the following.

Sum Cost

Given $f(x) = \langle x_0, x_1, \dots, x_{r-1} \rangle$, the sum cost is

$$SC(f(x)) = x_0 + x_1 + \dots + x_{r-1}.$$

For example, $SC(\langle 1, 1, 1, 1 \rangle) = 4$ and $SC(\langle 3, 0, 1, 2 \rangle) = 6$.

The elements in the cost-table of [14] were chosen by analyzing many CMCL circuits [5,6,12,13,14,17]. We consider here two kinds of logic element sets. One set includes sum, constant, and mirror, which were used in [14]. The other includes threshold detector logic. In addition to the set of basic elements in [14], threshold detector logic in [5,6,17] and their modifications are also used.

- **Sum.** The simplest operation in the current domain is arithmetic addition. The sum is the only connective operator in the set of basic circuit elements. The circuit realization of the sum, as shown in Fig. 1, is simply an interconnection between all the inputs and the output.
- **Constant.** The second basic element is the constant generator, as shown in Fig. 1. It can provide any positive or negative integer value between $\frac{(r-1)}{r}$ (a negative logic value) and $(r-1)$, where r denotes the radix. A negative logic value, \bar{i} , represents a logic value corresponding to the same quantity of current that is represented by i , except that the flow is in the opposite direction. The circuit of a constant generator consists of one or more MOS N-type transistor(s) or P-type transistor(s). The gate(s) of the transistor(s) are connected to a reference voltage, which can be generated locally.
- **N-type and P-type mirror.** The mirrors are one-input, multiple output logic operators of which there are two kinds, an N-type and a P-type. The mathematical functions $bndp$ (bound-positive) and $bndn$ (bound-negative) are used in the algebraic representation of these mirrors. They are defined as:

$$bndp(x) = \max(0, x)$$

$$bndn(x) = \min(0, x)$$

For example, if the input string with component x is $\{ \underline{3} \underline{2} \underline{1} \underline{0} \underline{1} \underline{2} \underline{3} \}$, the string obtained by applying

$bndp(x)$ and $bndn(x)$ is $\{ 0 \ 0 \ 0 \ 1 \ 2 \ 3 \}$ and $\{ \underline{3} \ \underline{2} \ \underline{1} \ 0 \ 0 \ 0 \}$, respectively. The circuits of these mirrors are shown in Fig. 1. Multiple copies of the output signals are obtained by using separate output transistors. Multiplication of the output signals by an integer is obtained by connecting the drains of several identical output transistors together.

There are two kinds of threshold detector circuits. When input x is m or greater, one produces x at the output, while the other produces a constant c . When x is less than m , both kinds produce 0.

- **Threshold detector.** A voltage-switched current source threshold detector can be implemented with a constant generator and a pass-transistor, as shown in Fig. 1. A threshold detector with constant generator was designed with those basic elements in [5,17]. Fig. 2a shows a circuit that is identical except for the addition of a current mirror to invert the input current direction. Considering Fig. 2a, the operation of the threshold detector is as follows: if the control input current \bar{x} in the upper side of the box in the symbol diagram of Fig. 2 is smaller than or equal to the threshold m , the output current is 0. On the other hand, if the control input current \bar{x} is greater than m , the gate output is connected by pass-transistor T1 to a current source T2. In Fig. 2b, however, T2 produces the value " x ". The function realized is defined as

$$TD(m, x) = \begin{cases} \text{constant } c \text{ or input } x & \text{if } m < x \\ 0 & \text{otherwise.} \end{cases}$$

The cost $Q(f)$ of $TD(m, c)$ (Fig. 2.a) is $2 + m + c$. The 2 occurs because there are 3 transistors of cost 1, one of which is shared with other cost-table functions (the P-type input transistor). The m occurs because the transistor in Fig. 2.a labeled m occupies m times the area of each of the three transistors discussed above. The c occurs because of the transistor in Fig. 2.a labeled by c . The cost of $TD(m, x)$ (Fig. 2.b) is $5 + m$. There are six transistors of size 1 (including T1 and T2) of which one, the input transistor, is not counted because it is shared with other cost-table functions. The m occurs because of the transistor labeled m in Fig. 2.b.

- **Modified threshold detector.** From the above threshold detector, a modified threshold detector, as shown in Fig. 1, was used in [5]. This is defined as

$$MTD(m1, m2, x) = \begin{cases} \text{constant } c \text{ or input } x & \text{if } m1 < x < m2 \\ 0 & \text{otherwise.} \end{cases}$$

The cost of $MTD(m1, m2 : c)$ is $4 + m1 + m2 + c$. The 4 occurs because of the 2 transistors of size 1 shown in Fig. 1 and 2 additional transistors (not shown) needed for the current mirror. Again, one of

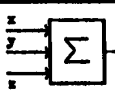
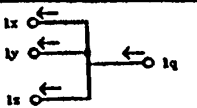

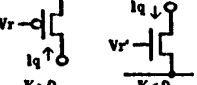
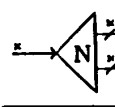
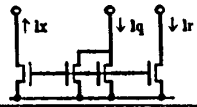
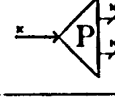
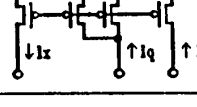

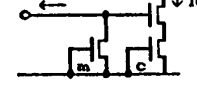

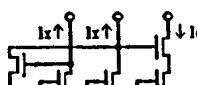

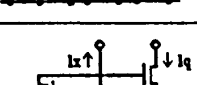
Name	Logic Operation	Symbol	Circuit Realization
Arithmetic Summator	$q=x+y+\dots+z$		
Constant	$q=K$		
N-type Mirror	$x' = bndn(x)$ $q = -Mx'$ $r = -Nx'$		
P-type Mirror	$x' = bndp(x)$ $q = -Mx'$ $r = -Nx'$		
Threshold Detector	$TD(m, c)$		
Modified Threshold Detector	$MTD(m1, m2 : c)$		
Inverse Threshold Detector	$ITD(m, c)$		

Figure 1: The basic set of CMCL circuit

these is not counted because it is shared with other functions. Similarly, the cost of $MTD(m1, m2 : x)$ can be calculated as $7 + m1 + m2$. The operation of the modified threshold detector logic is same as the literal generator and is useful in implementing the Universal Unary Programmable Circuit(UUPC) or T-gate, and Programmable Logic Arrays(PLAs).

- **Inverse threshold detector.** The inverse threshold detector or down-literal generator operates as the inverse operation of threshold detector. It can be implemented as a modification of the threshold detector and is defined as

$$ITD(m, x) = \begin{cases} \text{constant } c \text{ or input } x & \text{if } m > x \\ 0 & \text{otherwise.} \end{cases}$$

Its symbol and circuit diagram are shown in Fig. 1. The cost $Q(f)$ of $ITD(m, c)$ is $3 + m + c$ with

constant value "c" or $5 + m$ for $ITD(m, x)$ with the pass-transistor.

3 The Proposed Cost-table Technique

In this section, a cost-table design algorithm is shown which yields improved realizations over a previous algorithm. The improvements are due to insight gained by an operation called Vertical Partitioning (VP).

3.1 Cost-table circuits

The circuit structure of CMCL cost-table functions consists of three parts ([14]); a distribution circuit for the input, circuits that realize the cost-table functions, and an output summator circuit.

Cost-table functions in [14] were realized using the circuits shown in Fig. 3.a and 3.b. We augment the cost-table in [14] with functions that can be realized

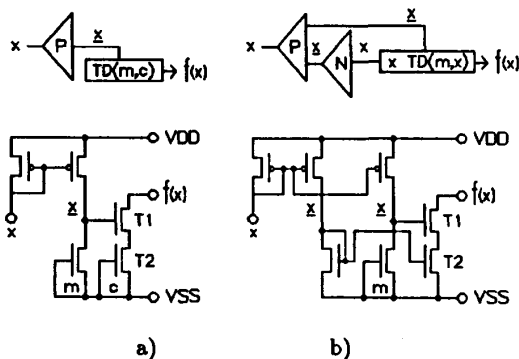


Figure 2: The operation of threshold detector logic

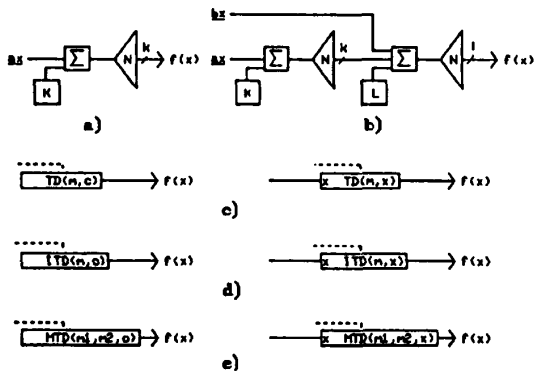


Figure 3: Cost-table circuit structures

using a multiplication. For example, instead of just $\langle 0,0,0,1 \rangle$, we choose to include $\langle 0,0,0,2 \rangle$ and $\langle 0,0,0,3 \rangle$ also. Table 1 shows the augmented cost-table. This increases the cost-table size from 37 to 53. Although the cost-table size is increased by more than 40%, the algorithm is simplified using this cost-table.

A further improvement can be obtained by using the threshold operations explained in the previous section. Specifically, the threshold detector $TD(m, c)$ and $TD(m, x)$, $MTD(m_1, m_2, c)$ and $MTD(m_1, m_2, x)$, and $ITD(m, c)$ and $ITD(m, x)$ are used. This results in an improvement in certain costs in Table 1. The resulting cost-table is shown in Table 2. As with Table 2, this table lists the circuit structure type, the values of the parameters, the cost, and the values of the cost functions.

In CMCL, the area of the circuit correlates closely with the number of transistors. As in the studies of CCD, it is of interest to compare the costs of functions obtained

Table 1: Cost-table with the circuits of Fig. 3a(circuit a), 3b(circuit b).

no.	Q	f(c)	circ.	parameters
00	1	1111	a)	
01	2	0113	a)	$K=0$ $km1$ $am1$
02	4	0013	a)	$K=1$ $km1$ $am1$
03	5	0001	a)	$K=2$ $km1$ $am1$
04	6	1000	b)	$K=0$ $Lm-1$ $km1$ $lm1$ $am1$ $bm0$
05	6	0002	a)	$K=2$ $km2$ $am1$
06	7	0111	b)	$K=1$ $Lm0$ $km1$ $lm1$ $am1$ $bm1$
07	7	1100	b)	$K=1$ $Lm-1$ $km1$ $lm1$ $am1$ $bm0$
08	7	3100	a)	$K=0$ $Lm-2$ $km1$ $lm1$ $am1$ $bm0$
09	7	0003	b)	$K=2$ $km3$ $am1$
10	7	2000	b)	$K=0$ $Lm-1$ $km1$ $lm2$ $am1$ $bm0$
11	8	1110	b)	$K=2$ $Lm-1$ $km1$ $lm1$ $am1$ $bm0$
12	8	2210	b)	$K=1$ $Lm-2$ $km1$ $lm1$ $am1$ $bm0$
13	8	0122	b)	$K=2$ $Lm0$ $km1$ $lm1$ $am1$ $bm1$
14	8	0100	b)	$K=1$ $Lm0$ $km2$ $lm1$ $am1$ $bm1$
15	8	2200	b)	$K=1$ $Lm-1$ $km1$ $lm2$ $am1$ $bm0$
16	8	0222	b)	$K=1$ $Lm0$ $km1$ $lm2$ $am1$ $bm1$
17	8	3000	b)	$K=0$ $Lm-1$ $km1$ $lm3$ $am1$ $bm0$
18	8	3100	b)	$K=0$ $Lm-3$ $km2$ $lm1$ $am1$ $bm0$
19	9	3310	b)	$K=1$ $Lm-3$ $km2$ $lm1$ $am1$ $bm0$
20	9	0011	b)	$K=2$ $Lm1$ $km1$ $lm1$ $am1$ $bm1$
21	9	0121	b)	$K=2$ $Lm0$ $km2$ $lm1$ $am1$ $bm1$
22	9	1210	b)	$K=1$ $Lm-1$ $km2$ $lm1$ $am1$ $bm1$
23	9	3300	b)	$K=1$ $Lm-1$ $km1$ $lm3$ $am1$ $bm0$
24	9	2220	b)	$K=2$ $Lm-1$ $km2$ $lm2$ $am1$ $bm0$
25	9	0200	b)	$K=1$ $Lm0$ $km1$ $lm2$ $am1$ $bm1$
26	9	0333	b)	$K=1$ $Lm0$ $km1$ $lm3$ $am1$ $bm1$
27	10	0300	b)	$K=1$ $Lm0$ $km2$ $lm3$ $am1$ $bm1$
28	10	3330	b)	$K=2$ $Lm-1$ $km1$ $lm3$ $am1$ $bm0$
29	10	0110	b)	$K=3$ $Lm0$ $km1$ $lm1$ $am2$ $bm1$
30	10	0010	b)	$K=2$ $Lm1$ $km2$ $lm1$ $am1$ $bm1$
31	10	0120	b)	$K=2$ $Lm0$ $km3$ $lm1$ $am1$ $bm1$
32	10	0210	b)	$K=1$ $Lm0$ $km3$ $lm1$ $am1$ $bm2$
33	10	1200	b)	$K=1$ $Lm-1$ $km3$ $lm1$ $am1$ $bm1$
34	10	0022	b)	$K=2$ $Lm1$ $km1$ $lm2$ $am1$ $bm1$
35	11	0133	b)	$K=2$ $Lm1$ $km2$ $lm1$ $am1$ $bm2$
36	11	2310	b)	$K=1$ $Lm-2$ $km3$ $lm1$ $am1$ $bm1$
37	11	0023	b)	$K=2$ $Lm1$ $km1$ $lm3$ $am1$ $bm1$
38	11	0220	b)	$K=3$ $Lm0$ $km1$ $lm2$ $am2$ $bm1$
39	11	0020	b)	$K=2$ $Lm1$ $km2$ $lm2$ $am1$ $bm1$
40	12	0132	b)	$K=2$ $Lm1$ $km3$ $lm1$ $am1$ $bm2$
41	12	0321	b)	$K=1$ $Lm0$ $km4$ $lm1$ $am1$ $bm3$
42	12	1230	b)	$K=2$ $Lm-1$ $km4$ $lm1$ $am1$ $bm1$
43	12	1310	b)	$K=1$ $Lm-1$ $km4$ $lm1$ $am1$ $bm2$
44	12	0330	b)	$K=3$ $Lm0$ $km1$ $lm3$ $am2$ $bm1$
45	13	0021	b)	$K=2$ $Lm2$ $km3$ $lm1$ $am1$ $bm2$
46	13	0131	b)	$K=2$ $Lm1$ $km4$ $lm1$ $am1$ $bm2$
47	13	0310	b)	$K=1$ $Lm0$ $km5$ $lm1$ $am1$ $bm3$
48	13	1300	b)	$K=1$ $Lm-1$ $km5$ $lm1$ $am1$ $bm2$
49	13	0030	b)	$K=2$ $Lm1$ $km3$ $lm3$ $am1$ $bm1$
50	14	0130	b)	$K=2$ $Lm1$ $km5$ $lm1$ $am1$ $bm2$
51	14	1220	b)	$K=5$ $Lm-1$ $km1$ $lm1$ $am3$ $bm1$
52	17	0031	b)	$K=2$ $Lm3$ $km5$ $lm1$ $am1$ $bm3$

from a rigidly specified table of costs to costs derived directly from the functions themselves, that is, costs of four previously defined cost functions TC [16], TTS [16], SUM [16], and BC [1]. We do the same in this study.

For the decomposition program, it is interesting to know how the transistor counts of the cost-table functions and the mathematical cost functions are related to each other. Table 2 also lists the relations between the TTS , SUM , BC , and TC cost functions and transistor count Q for the functions in the cost-table in [14]. For the given functions, the mathematical cost functions are fixed. Therefore, the values of the mathematical cost functions are not duplicated in Table 1.

As seen in [20], for all cost-table functions, $TTS < 7$, $SUM < 10$, $BC < 4$ and $TC < 2$. Even though we add the threshold detector logic circuits as basic circuit elements, the correlation between the cost Q and the mathematical cost functions is weak as shown in Table 2. Because of this negative correlation, these cost functions are not useful in a direct way to realize a function by the cost-table technique. However, one, the SUM , is

Table 2: Proposed Cost-table for 4-valued CMCL functions with the circuits of Fig. 3a(circuit a), 3b(circuit b), 3c(circuit c), 3d(circuit d) and 3e(circuit e).

no.	Q	f(s)	circ.	parameters	TC	TTS	SUM	BC
00	1	1111			0	0	4	0
01	3	0123	a)	K=0 k=1 a=1	0	3	6	0
02	3.5	0111	c)	TD(0.5,1)	0	1	3	0
03	4	0012	a)	K=1 k=1 a=1	0	2	3	0
04	4.5	0222	e)	TD(0.5,2)	0	2	6	0
05	4.5	0011	c)	TD(1.5,1)	0	1	2	0
06	5	0001	a)	K=2 k=1 a=1	0	1	1	0
07	5.5	0333	c)	TD(0.5,3)	0	3	9	0
08	5.5	0022	c)	TD(1.5,2)	0	2	4	0
09	5.5	1000	d)	ITD(0.5,1)	1	2	1	1
10	6	0002	a)	K=2 k=2 a=1	0	2	2	0
11	6.5	2000	d)	ITD(0.5,2)	1	4	2	1
12	6.5	1100	d)	ITD(1.5,1)	1	1	2	1
13	6.5	0033	c)	TD(1.5,3)	0	3	6	0
14	6.5	0023	c)	K=0 k=1 a=1 and TD(1.5,x)	0	3	5	0
15	7	0003	a)	K=2 k=3 a=1	0	3	3	0
16	7	0100	e)	MTD(0.5,1.5:1)	1	2	1	1
17	7	2100	b)	K=0 L=2 k=1 l=1 a=1 b=0	1	3	3	2
18	7.5	3000	d)	ITD(0.5,3)	1	6	3	1
19	7.5	2200	d)	ITD(1.5,2)	1	2	4	1
20	7.5	1110	d)	ITD(2.5,1)	1	1	3	1
21	8	2210	b)	K=1 L=2 k=1 l=1 a=1 b=0	1	2	5	1
22	8	0122	b)	K=2 L=0 k=1 l=1 a=1 b=1	0	2	5	0
23	8	3210	b)	K=0 L=3 k=1 l=1 a=1 b=0	1	4	6	3
24	8	0110	e)	MTD(0.5,2.5:1)	1	2	2	1
25	8	0200	e)	MTD(0.5,1.5:2)	1	4	2	1
26	8.5	3300	d)	ITD(1.5,3)	1	3	6	1
27	8.5	2220	d)	ITD(2.5,2)	1	2	6	1
28	9	3100	b)	K=0 L=3 k=2 l=1 a=1 b=0	1	5	4	2
29	9	3310	b)	K=1 L=3 k=2 l=1 a=1 b=0	1	3	7	2
30	9	0121	b)	K=2 L=0 k=2 l=1 a=1 b=1	1	3	4	1
31	9	1210	b)	K=1 L=1 k=2 l=1 a=1 b=1	1	3	4	1
32	9	0220	e)	MTD(0.5,2.5:2)	1	4	4	1
33	9	0010	e)	MTD(1.5,2.5:1)	1	2	1	1
34	9	0300	e)	MTD(0.5,1.5:3)	1	6	3	1
35	9.5	3330	d)	ITD(2.5,3)	1	3	9	1
36	10	0120	d)	K=2 L=0 k=3 l=1 a=1 b=1 or MTD(0.5,2.5:2)	1	4	3	1
37	10	0210	b)	K=1 L=0 k=3 l=1 a=1 b=2	1	4	3	2
38	10	1200	b)	K=1 L=1 k=3 l=1 a=1 b=1	1	3	3	1
39	10	0020	e)	MTD(1.5,2.5:2)	1	4	2	1
40	10	0330	e)	MTD(0.5,2.5:3)	1	6	6	1
41	10.5	1230	b)	K=0 L=1 k=1 l=1 a=1 b=0 and ITD(2.5:2)	1	5	6	1
42	11	2310	b)	K=1 L=2 k=3 l=1 a=1 b=1	1	4	6	2
43	11	0030	e)	MTD(1.5,2.5:3)	1	6	3	1
44	12	0132	b)	K=2 L=1 k=3 l=1 a=1 b=2	1	4	6	1
45	12	0321	b)	K=1 L=0 k=4 l=1 a=1 b=3	1	5	6	2
46	12	1310	b)	K=1 L=1 k=4 l=1 a=1 b=2	1	5	6	2
47	13	0021	c)	K=2 L=2 k=3 l=1 a=1 b=2	1	3	3	1
48	13	0131	b)	K=2 L=1 k=4 l=1 a=1 b=2	1	5	5	1
49	13	0310	b)	K=1 L=0 k=5 l=1 a=1 b=3	1	6	4	2
50	13	1300	b)	K=1 L=1 k=5 l=1 a=1 b=2	1	5	4	1
51	14	0130	b)	K=2 L=1 k=5 l=1 a=1 b=2	1	6	4	1
52	14	1220	b)	K=5 L=1 k=1 l=1 a=3 b=1	1	3	5	1

useful indirectly, as we show later.

3.2 The proposed cost-table decomposition algorithm.

The addition of the threshold detector logic to the basic logic in [14] allows functions to be generated with less cost than without this logic. Fig. 4 shows how the various functions in the two cost-tables are distributed. Data points in boxes indicate number of functions from the old cost-table (Table 1), while data points in x's indicate number of functions from the new cost-table (Table 2). The lower cost of the new cost-table is shown by the grouping of x's to the left of the boxes. However, the increase in the functions of the cost-table renders the direct application of the existing decomposition algorithms difficult, because the decomposition algorithm searches the cost-table at each step. An analysis of the functions in the cost-table reveals the following:

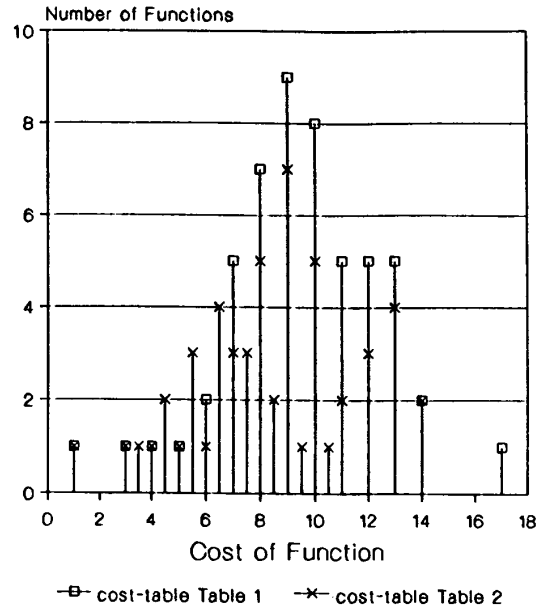


Figure 4: Comparison of the cost functions in the cost-tables

- Constant functions are the least expensive functions.
- Monotonously increasing functions (stair-case functions) such as $\langle 0, 1, 2, 3 \rangle$, $\langle 0, 0, 1, 2 \rangle$, $\langle 0, 0, 0, 1 \rangle$, $\langle 0, 1, 2, 0 \rangle$, $\langle 0, 1, 0, 0 \rangle$ and $\langle 1, 2, 3, 0 \rangle$ can be obtained at moderate cost.
- All step-up functions (e.g. $\langle 0, 0, 2, 2 \rangle$) and all step-down functions (e.g. $\langle 2, 2, 0, 0 \rangle$) are in the table.
- Most functions like $\langle x_0, x_1, 0, 0 \rangle$ and $\langle 0, 0, x_2, x_3 \rangle$ are in the cost-table, where x_0, x_1, x_2, x_3 are any values in radix 4 (i.e. 0, 1, 2 and 3). The exceptions are $\langle 0, 0, 3, 2 \rangle$, $\langle 2, 3, 0, 0 \rangle$, and $\langle 3, 2, 0, 0 \rangle$.

The following definitions are useful in formulating the decomposition method. We allow the radix r to take on any value, and so we consider general r -valued unary functions $f(x) = \langle x_0, x_1, \dots, x_{r-1} \rangle$.

Definition 1: Let i_{max} be the smallest index i such that $x_i = 0$ and $x_{i+1} \neq 0$. i_{max} is said to be the *rightmost internal 0* of $f(x)$. If no x_i is 0 or the only 0's extend consecutively from $i = r - 1$ to lower values, the rightmost internal 0 does not exist.

Example: The functions $\langle 1, 3, 0, 1 \rangle$, $\langle 2, 0, 0, 3 \rangle$, and $\langle 0, 0, 0, 1 \rangle$ all have a rightmost internal 0 at

$i = 2$. The leftmost internal 0 is defined in a similar manner. However, we do not make use of it here.

Definition 2: If j is the index of the rightmost internal 0, then $LF(x) = \langle x_0, x_1, \dots, x_{j-1}, 0, \dots, 0 \rangle$ is the *left function* and $RF(x) = \langle 0, \dots, 0, x_j, x_{j+1}, \dots, x_{r-1} \rangle$ is the *right function*.

Example: For the functions $\langle 1, 3, 0, 1 \rangle$, $\langle 2, 0, 0, 3 \rangle$ and $\langle 0, 0, 0, 1 \rangle$, $LF(x) = \langle 1, 3, 0, 0 \rangle$, $\langle 2, 0, 0, 0 \rangle$, and $\langle 0, 0, 0, 0 \rangle$, respectively, while $RF(x)$ is $\langle 0, 0, 0, 1 \rangle$, $\langle 0, 0, 0, 3 \rangle$, and $\langle 0, 0, 0, 1 \rangle$, respectively.

These concepts are useful in decomposing a given function into two subfunctions. That is, the algorithm shown later accepts a function $f(x)$ and attempts to decompose it into two subfunctions $LF(x)$ and $RF(x)$.

Definition 3: $f(x) = \langle c, c, \dots, c \rangle$ is a constant function for $0 < c \leq r - 1$.

Example: $\langle 1, 1, 1, 1 \rangle$, $\langle 2, 2, 2, 2 \rangle$, and $\langle 3, 3, 3, 3 \rangle$ are the three constant functions for 4-valued unary functions.

Definition 4: $f(x) = \langle x_0, x_1, \dots, x_{r-1} \rangle$ is an *up-staircase (down-staircase) function* if $f(x)$ is not a constant function and $x_i = \text{bndp}(i - c)$, for $0 \leq c \leq r - 2$ ($x_i = \text{bndp}(c - i)$, for $1 \leq c \leq r - 1$). $f(x)$ is a *staircase function* if it is either an up-staircase function or a down-staircase function.

Example: $\langle 0, 1, 2, 3 \rangle$, $\langle 0, 0, 1, 2 \rangle$, and $\langle 0, 0, 0, 1 \rangle$ are up-staircase functions, while $\langle 3, 2, 1, 0 \rangle$, $\langle 2, 1, 0, 0 \rangle$, and $\langle 1, 0, 0, 0 \rangle$ are down-staircase functions.

Definition 5: $f(x) = \langle x_0, x_1, \dots, x_{r-1} \rangle$ is a *partial up-staircase (partial down-staircase function)*, if $f(x)$ is not a constant function, and $x_i = \text{bndp}(i - c)$ for $0 \leq c \leq r - 2$ ($x_i = \text{bndp}(c - i)$, for $1 \leq c \leq r - 1$), where $0 \leq i \leq m$ ($m \leq i \leq r - 1$) and $x_i = 0$ for $m < i$ ($i < m$), where $m \geq c + 1$ ($m \leq c - 1$). $f(x)$ is a *partial staircase function* if it is either a partial up-staircase function or a partial down-staircase function.

Example: $\langle 0, 1, 2, 3 \rangle$, $\langle 0, 0, 1, 2 \rangle$, $\langle 0, 0, 0, 1 \rangle$, $\langle 0, 1, 0, 0 \rangle$, $\langle 0, 1, 2, 0 \rangle$, and $\langle 0, 0, 1, 0 \rangle$ are partial up-staircase functions, while $\langle 3, 2, 1, 0 \rangle$, $\langle 2, 1, 0, 0 \rangle$, $\langle 1, 0, 0, 0 \rangle$, $\langle 0, 0, 1, 0 \rangle$, $\langle 0, 2, 1, 0 \rangle$, and $\langle 0, 1, 0, 0 \rangle$ are partial down-staircase functions.

Definition 6: $f(x) = \langle x_0, x_1, \dots, x_{r-1} \rangle$ is a block function iff $x_i = c$ for $i_{\min} \leq i \leq i_{\max}$ and $x_i = 0$ for $i < i_{\min}$ and $i_{\max} < i$, where $1 \leq c \leq r - 1$ and $0 \leq i_{\min} < i_{\max} \leq r - 1$, such that i_{\min} and i_{\max} are not 0 and $r - 1$ simultaneously.

Example: When $r = 4$, there are 15 block functions $\langle 0, 1, 1, 1 \rangle$, $\langle 0, 2, 2, 2 \rangle$, $\langle 0, 3, 3, 3 \rangle$, $\langle 0, 0, 1, 1 \rangle$, $\langle 0, 0, 2, 2 \rangle$, $\langle 0, 0, 3, 3 \rangle$, $\langle 0, 1, 1, 0 \rangle$, $\langle 0, 2, 2, 0 \rangle$, $\langle 0, 3, 3, 0 \rangle$, $\langle 1, 1, 0, 0 \rangle$, $\langle 2, 2, 0, 0 \rangle$, $\langle 3, 3, 0, 0 \rangle$, $\langle 1, 1, 1, 0 \rangle$, $\langle 2, 2, 2, 0 \rangle$, and $\langle 3, 3, 3, 0 \rangle$.

In the algorithm described next, a given function $f(x)$ is tested to determine if a nonzero constant function can be subtracted from it. Next, an internal 0 is sought and the corresponding right and left functions $RF(x)$

and $LF(x)$ are extracted. These are then decomposed further.

This process is performed using an AND_OR tree. From certain nodes, there are AND arcs which lead to two or more nodes, all of which must be solved to solve the given node. Alternatively, if a node has OR arcs leading to two or more nodes, only one of those nodes must be solved in order for the initial node to be solved. This algorithmic process is similar to the well-known A^* algorithm [4]. But specifically, we use the tree structure of the AO^* algorithm [10,11] to find solutions in the AND_OR tree, and our terminology is similar to that of [15]. The algorithm described here finds a path from the starting node to a set of nodes representing the cost-table functions.

The proposed Vertical Partitioning algorithm (VP) proceeds by searching an AND_OR tree [15]. In this tree, nodes represent functions. As the search proceeds, cost estimates are generated at each node. These estimates are updated as more information is gathered in the search and, in turn, are used to direct the search toward productive paths.

Vertical Partitioning Algorithm

1. Let the initial AND_OR tree consist only of the node representing $f = f(x)$, which is called INIT. Set this node's cost, $Q(f)$, to ∞ , and denote this level as 0.
2. Generate levels 1, 2, ... and thus the AND_OR tree (AO-tree), until all leaves become a cost-table function or FUTILITY. The AO-tree has AND node(s) on levels of odd value and has OR node(s) on levels of even value. The algorithm proceeds until INIT is labeled SOLVED. Repeat the following operation at each level until all leaves generated are marked SOLVED or FUTILITY:
 - a. At each AND node function f , i.e., level 0, 2, 4, ... , let z be the rightmost internal 0. If z exists, partition f into two subfunctions f_1 and f_2 , where f_1 is the left function (LF) and f_2 is the right function (RF) of f . If z does not exist, let $f_1 = f$, and mark f_2 as FUTILITY, which is equivalent to specifying the AND node as not partitioned. Form the AO-tree nodes f_1 and f_2 . If f_1 or f_2 is in the cost-table, mark the corresponding node SOLVED, and assign it the Q value of the function's cost from the cost-table. If f_1 or f_2 is not in the cost-table, do the following:
 - b. At each OR node function, f , level 1, 3, 5, ... , generate a component function f' of f of the following three types 1. a partial staircase function (assigned to the left-hand OR_successor), 2. a block function (assigned to the middle OR_successor), and 3. a non-zero constant function (assigned to the right-hand OR_successor). Each component function f' has the property that 1. f' is in the cost-table, 2. $f'(x) \leq f(x)$ for $0 \leq x \leq r - 1$

in radix r , and 3. there is no other component function of the same type with a larger *SUM* cost function. Furthermore, among the choices of component functions with the same largest *SUM* cost, select the one with the lowest Q (cost-table cost). If no function of any of the three types of functions can be subtracted, the corresponding OR_successor node is marked FUTILITY. Next,

- i. choose the arc associated with a component function, f' , that has the lowest Q value.
 - ii. generate the OR_successor by subtracting the corresponding component function from the OR node function.
 - iii. if the OR_successor is found in the cost-table, label it SOLVED and assign its Q value from the cost-table.
 - iv. depth first recursion: If an OR_successor cannot be found in the cost-table, repeat the above AND/OR decomposition until all successive leaves are marked as SOLVED.
 - v. backtrack: When a node status changes to SOLVED and its cost changes, its parent node cost is re-evaluated to determine if a lower cost can be assigned to the parent node.
 - vi. repeat Steps (ii) to (iv) for other OR_successors if the Q values for their component functions are less than the cost of the OR node found from the completed evaluation of the first OR_successor.
3. Besides the decompositions considered in Steps 1 and 2 above, which are based on vertical partitions about the leftmost internal 0, consider other vertical partitions. Compute the cost of all vertical partitions of the given function, as it is decomposed into subfunctions $\langle x_0, x_1, \dots, x_i, 0, \dots, 0 \rangle$ and $\langle 0, \dots, 0, x_{i+1}, \dots, x_{r-1} \rangle$, where both subfunctions are in the cost-table (if either one or both are not in the cost-table, discard the composition). There are $r - 1$ such decompositions. From among these decompositions and the decompositions chosen in Steps 2 and 3, choose the one with the lowest cost.

3.3 Examples

Fig. 5 shows two examples of the Vertical Partitioning algorithm.

Example: Fig. 5a shows how $\langle 3, 0, 2, 3 \rangle$ is decomposed with the cost-table shown in Table 1. At level 0, this function is partitioned into $\langle 3, 0, 0, 0 \rangle$ and $\langle 0, 0, 2, 3 \rangle$. Function $\langle 3, 0, 0, 0 \rangle$ is in the cost-table and is therefore marked SOLVED, with a cost $Q = 8$. Since the other function $\langle 0, 0, 2, 3 \rangle$ is not in Table 1, it is decomposed as $\langle 0, 0, 1, 1 \rangle$ or $\langle 0, 0, 0, 1 \rangle$ by subtracting the staircase function

$f' = \langle 0, 0, 1, 2 \rangle$, with a cost $Q(f') = 4$ or the block function $f' = \langle 0, 0, 2, 2 \rangle$, with a cost $Q(f') = 10$. Because it is not possible to subtract a non-zero constant, the corresponding node is marked FUTILITY. Both $\langle 0, 0, 1, 1 \rangle$ and $\langle 0, 0, 0, 1 \rangle$ are cost-table functions and the minimal realization can be obtained as $\langle 3, 0, 0, 0 \rangle + \langle 0, 0, 1, 2 \rangle + \langle 0, 0, 1, 1 \rangle$ with a total cost of 21.

Example: Fig. 5b shows the realization of $\langle 3, 2, 1, 1 \rangle$ with the cost-table shown in Table 2. Since $\langle 3, 2, 1, 1 \rangle$ contains no leftmost internal 0, there is no partition at level 0. Thus, the left function is marked FUTILITY and consequently the right function is a copy of parent node function. Next generate three potential component functions $\langle 1, 1, 1, 1 \rangle$ (constant function), $\langle 2, 2, 0, 0 \rangle$ (block function) and $\langle 3, 2, 1, 0 \rangle$ (staircase function) with costs 1, 7.5 and 8, respectively. First, $\langle 3, 2, 1, 1 \rangle$ is decomposed as $\langle 1, 1, 1, 1 \rangle + \langle 2, 1, 0, 0 \rangle$ at the right-hand OR_successor, since its cost is the lowest among the component functions at its tree level. Node $\langle 2, 1, 0, 0 \rangle$ is marked SOLVED because it is a cost-table function at a cost $Q = 7$. At this point, one solution is found with the total cost 8 by summing $Q(f') = 1$ with $Q = 7$. Next, test the next lowest component, i.e., the middle OR_successor. Here, OR node function $\langle 3, 2, 1, 1 \rangle$ can be decomposed as $\langle 2, 2, 0, 0 \rangle + \langle 1, 0, 1, 1 \rangle$, the latter of which can further be partitioned into $\langle 1, 0, 0, 0 \rangle$ and $\langle 0, 0, 1, 1 \rangle$ (since $\langle 1, 0, 1, 1 \rangle$ is not a cost-table function). Both $\langle 1, 0, 0, 0 \rangle$ and $\langle 0, 0, 1, 1 \rangle$ are cost-table functions and are marked SOLVED with cost $Q = 5.5$ and $Q = 4.5$, respectively. Now that the middle OR_successor is solved. The total cost for this path is calculated by summing the cost Q for $\langle 1, 0, 0, 0 \rangle$, $\langle 0, 0, 1, 1 \rangle$, and $\langle 1, 0, 1, 1 \rangle$ for a total cost larger than the solution already solved for the right-hand OR_successor. Therefore, the solution for the middle OR_successor is discarded. The remaining left-hand OR_successor can also be discarded, since its total cost will surely be greater than cost of the component function, $\langle 3, 2, 1, 0 \rangle$, which is 8. Thus, the solution is $\langle 1, 1, 1, 1 \rangle + \langle 2, 1, 0, 0 \rangle$ with a cost $Q = 8$.

Example: In [14], $\langle 1, 3, 0, 3 \rangle$ is realized at a cost $Q = 24$ using $\langle 1, 0, 0, 0 \rangle$ and $\langle 0, 0, 1, 0 \rangle$ with a multiplication factor of 2 and 3 respectively followed by complementation with respect to a constant 3. With the cost-table shown in Table 1 presented here, it is realized at a cost of $Q = 20$ as shown in Fig. 6.

Example: Using the cost-table of Table 2, the Vertical Partitioning algorithm decomposes $\langle 0, 2, 3, 2 \rangle$ into $\langle 0, 2, 2, 2 \rangle$ and $\langle 0, 0, 1, 0 \rangle$ at a cost of $Q = 13.5$. However, the Exhaustive Search algorithm decomposes $\langle 0, 2, 3, 2 \rangle$ into $\langle 0, 1, 1, 1 \rangle$ and $\langle 0, 1, 2, 1 \rangle$ at a cost of 12.5, which is better than that produced by the Vertical Partitioning algorithm.

4 Overall Comparison Results

An example in the previous section shows that the results of the Vertical Partitioning are sometimes worse than Exhaustive Search. In this section, we further

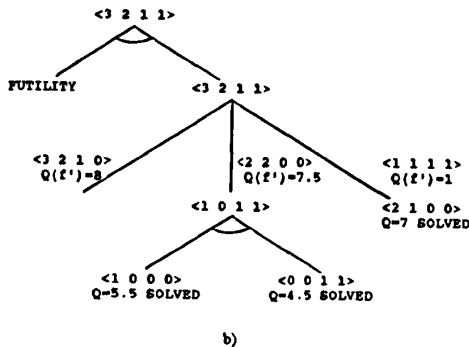
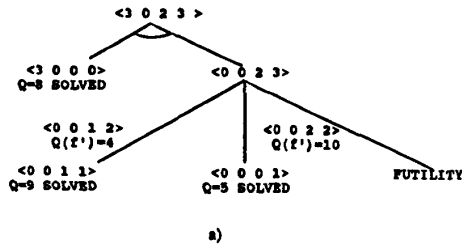


Figure 5: Examples of the application of the vertical partitioning algorithm

quantify the performance of the two algorithms. Fig. 7 shows the result of applying the Vertical Partitioning algorithm to 255 4-valued unary functions using two cost-tables, that of Table 1 and Table 2. Plotted vertically is the number of functions with a minimum cost of realization shown along the horizontal axis. Data points in boxes represent the cost-table of Table 1, while data points marked with x's correspond to the cost-table shown in Table 2. For example, the box at (4,1) means there is one function realized at a cost of 4 using the Vertical Partitioning algorithm on the cost-table of Table 1. The advantage of the cost-table of Table 2 is clearly seen as a bulk of the new cost-table histogram is to the left of the bulk of the old cost-table histogram. The average cost over all 255 functions is $Q=12.1804$ with the cost-table of Table 2 compared to $Q=13.6039$ with the cost-table of Table 1. A similar experiment was performed using the Exhaustive Search algorithm on the cost-table of Table 2. The cost-tables of Table 2 and Table 1 yielded an average cost of $Q=12.1235$ and $Q=13.5686$, respectively. Table 3 shows the computation time of Vertical Partitioning algorithm as it compares to the Exhaustive Search. That is, the Vertical Partitioning algorithm required 19.3 sec. of CPU time to compute the minimal realization of all 255 4-valued unary functions. This is five times faster than EX(3) which is Exhaustive Search for the minimal solution over all combinations of three cost-table functions. It is 265 times faster than EX(4) which is Exhaustive Search over all combi-

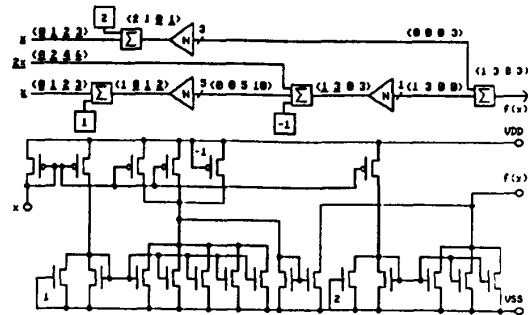


Figure 6: $f(x) = \langle 1, 3, 0, 3 \rangle$

Table 3: Relative Computation Times of Three Algorithms

Algorithm	CPU Time (sec.)	Relative Duration
VP	19.3	1
EX(3)	96.6	≈ 5
EX(4)	$\approx 53 \times \text{EX}(3)$	≈ 265

nations of four cost-table functions. We have verified that the cost-table implementation EX(3) is the same as Exhaustive Search. These observations hold because of the increased size of the cost-table to 53. As a result, we didn't need to run EX(4). Therefore, the CPU time for EX(4) is an estimate in Table 3.

5 Conclusions

We have proposed the Vertical Partitioning algorithm for the design of current-mode CMOS multiple-valued logic. It partitions a given function according to the location of logic 0's. The algorithm requires search and proceeds in a manner similar to the heuristic algorithm(AO*). In addition, the cost-table of Table 2 is proposed in which threshold detector logic circuits augment operations used in a previous cost-table. The cost of realizing functions using the cost-table of Table 2 is about 10% less than with Table 1. That is, using the Vertical Partitioning algorithm, the average cost over 255 4-valued functions was $Q=12.1804$ for the cost-table of Table 2 versus $Q=13.6039$ for the cost-table of Table 1, a reduction of 11.7%. The new algorithm does not achieve the performance of Exhaustive Search which finds a guaranteed minimal solution. However, it is not far off, only 0.0047% (calculated from the total cost difference between Vertical Partitioning algorithm and Exhaustive Search divided by the total cost from Exhaustive Search for all 4-valued 255 functions) and it is faster.

An advantage of the Vertical Partitioning method is that it is easily extended to higher radices in a natural way. We expect this, because functions like the constant and staircase will be relatively inexpensive in higher radices as well.

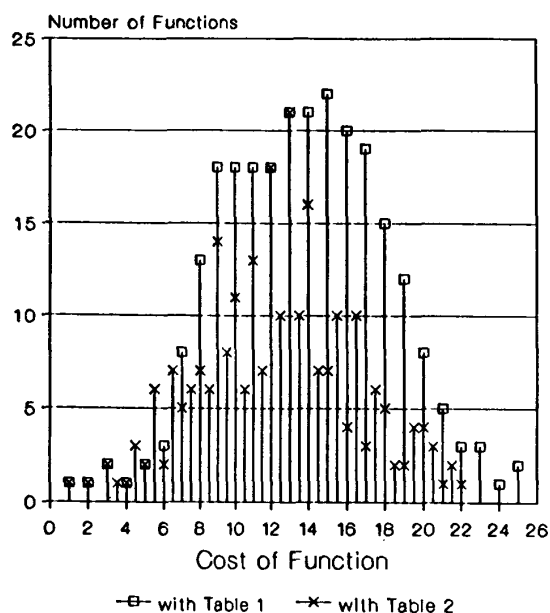


Figure 7: Comparison of 255 4-valued unary functions in two cost-tables

References

- [1] M.H. Abd-El-Barr, Z.G. Vranesic, and S.C. Zaky, "Synthesis of MVL functions for CCD implementations," *Proc. of the 16th Inter. Symp. on Multiple-Valued Logic*, pp. 116-127, May 1986.
- [2] M.H. Abd-El-Barr, T.D. Hoang, and Z.G. Vranesic, "The incremental-cost approach for synthesis of CCD 4-valued unary functions," *Proc. of the 18th Inter. Symp. on Multiple-valued Logic*, pp. 82-89, May 1988.
- [3] D.A. Freitas and K.W. Current, "A quaternary logic encoder-decoder circuit design using CMOS," *Proc. of the 13th Inter. Symp. on Multiple-Valued Logic*, pp. 190-195, May 1983.
- [4] P.E. Hart, N.J. Nilsson and B. Raphael, "Correction to 'A formal basis of the heuristic determination of minimum cost paths'," *SIGART Newsletter*, Vol. 37, 1972.
- [5] M. Kameyama, T. Sekibe and T. Higuchi, "Design of highly parallel residue arithmetic circuits based on multiple-valued bidirectional current-mode MOS technology," *Proc. of the 18th Inter. Symp. on Multiple-Valued Logic*, pp. 6-13, May, 1988.
- [6] T.S. Kawahito, M. Kameyama, T. Higuchi, and H. Yamada, "A high-speed compact multiplier based on multiple-valued bi-directional current-mode circuits," *Proc. of the 17th Inter. Symp. on Multiple-Valued Logic*, pp. 172-180, May 1987.
- [7] H.G. Kerkhoff and M.L. Tervoert, "Multiple-valued logic charge coupled devices," *IEEE Trans. on Comp.*, vol. C-30, no. 9, pp. 644-652, September 1981.
- [8] H.G. Kerkhoff and H.A.J. Robroek, "The logic design of multiple-valued logic functions using charge-coupled devices," *Proc. of the 12th Inter. Symp. on Multiple-Valued Logic*, pp. 35-44, May 1982.
- [9] J.K. Lee and J.T. Butler, "Tabular methods for the design of CCD multiple-valued logic," *Proc. of the 13th Inter. Symp. on Multiple-Valued Logic*, pp. 162-170, May 1983.
- [10] A. Martelli and U. Montanari, "Additive And/Or graphs," *Proc. of IJCAI 3*, 1973.
- [11] A. Martelli and U. Montanari, "Optimization decision trees through heuristically guided search," *Communication of the ACM*, Vol. 21, No. 12, 1978.
- [12] S.P. Onneweer and H.G. Kerkhoff, "Current-mode high-radix circuits," *Proc. of the 16th Inter. Symp. on Multiple-Valued Logic*, pp. 60-69, May 1986.
- [13] S.P. Onneweer and H.G. Kerkhoff, "High-radix current-mode CMOS circuits based on the truncated-difference operator," *Proc. of the 17th Inter. Symp. on Multiple-Valued Logic*, pp. 188-195, May 1987.
- [14] S.P. Onneweer, H.G. Kerkhoff, and J.T. Butler, "Structural computer-aided design of current-mode CMOS logic circuits," *Proceedings of the 18th International Symposium on Multiple-Valued Logic*, pp. 21-30, May 1988.
- [15] E. Rich, *Artificial Intelligence*, McGraw-Hill 1983.
- [16] K.A. Schueller, P.P. Tirumalai, and J.T. Butler, "An analysis of the costtable approach to the design of multiple-valued circuits," *Proc. of the 16th Inter. Symp. on Multiple-Valued Logic*, May 1986, pp. 42-50.
- [17] T. Yamakawa, "CMOS multivalued circuits in hybrid mode," *Proc. of the 15th Inter. Symp. on Multiple-Valued Logic*, pp. 144-151, May 1985.