Implementation of Multiple-Valued CAM Functions by LUT Cascades

Tsutomu Sasao

Department of Computer Science
and Electronics
Kyushu Institute of Technology
Iizuka, 820-8502 JAPAN

Jon T. Butler

Department of Electrical
and Computer Engineering
Naval Postgraduate School
Monterey, CA, 93943-5121 U.S.A.

Abstract

*In this paper, we introduce three types of multiple-valued content-addressable memories (CAMs): ordinary CAMs (CAMs), distance d CAMs, and *CAMs. Ordinary CAMs require an exact match, while *CAMs allow wild-card matches. In a distance d CAM, a match occurs even if at most d digits differ. Then, we define multiple-valued CAM functions represented by these CAMs. Next, we show an approach to realize each CAM function by an LUT cascade, which is a series connection of RAMs. Experimental results for both two-valued and multi-valued cases are shown.*

1 Introduction

Ordinary memory produces the data at a given address. Conversely, a content-addressable memory (CAM) [5] produces the address of the given data. If the given data does not exist anywhere in the CAM, a special address (e.g. $00\ldots0$) is produced. CAMs are often used in pattern matching applications, since they are much faster than software implementations. Other applications include routers for the internet [2, 4, 6, 15], processor caches, translation lookaside buffers (TLB), data compression applications, database accelerators, and neural networks. Several implementations of multiple-valued CAMs have been proposed [3, 14].

p-valued CAMs store p-nary vectors. We consider three types of p-valued CAMs: ordinary CAM (CAM), distance d CAMs, and *CAM. CAMs produce an address only if there is an exact match to the input data. A distance d CAM is similar to a CAM except that there can be d or fewer mismatches. Distance d CAMs are used in pattern matching applications. A *CAM is similar to a CAM except there can be a mismatch only in positions designated by a *. Therefore, a binary *CAM stores a three valued vector, where each element is chosen from $\{0, 1, *\}$ [1] . Such CAMs are often used in routers for the internet.

We present a method to implement a CAM function by an LUT cascade, which is a series connection of RAMs. We show that this results in a simple implementation. The LUT cascade uses ordinary RAMs instead of special semiconductor CAMs, and show significant promise in reducing power dissipation, which is the problem in existing CAMs [9, 15].

Example 1.1 *Consider an e-mail system in a company with 50,000 employees. Each employee has an employee number consisting of 7 digits. Also, each employee has a login name of an e-mail account consisting of 8 letters from the 26-letter English alphabet.*

CAM: *Consider a table that yields the employee number from a given login name. Since each letter can be represented by 5 bits, the table has* $5 \times 8 = 40$ *binary inputs and* $\lceil \log_2 10^7 \rceil = 24$ *binary outputs. A direct implementation by a single conventional memory requires* $2^{40} \times 24 = 2.6 \times 10^{13}$ *bits, which is 3 terabytes. Although there exist* $27^8 \cong 2.82 \times 10^{11}$ *different login names (each consisting of 26 letters and a space), only 50,000 are used. We can use a CAM to greatly reduce the memory needed by the direct implementation.*

Distance 1 CAM: *The E-mail administrator sometimes wants to know the correct login name from an incorrect login name. Suppose that the correct login name is MORAGA. However, due to the smudged FAX or an unclear handwritten document, the login name may appear as MOROGA or MARAGA or MORAGO. It would be convenient if the system showed the correct login name when there is no employee named MOROGA or MARAGA or MORAGO.*

***CAM:** *The E-mail administrator wants to know the correct login name from an uncertain login name. Suppose that the administrator knows only a part of the login name: ***KOWSKI, for example, where * corresponds to any letter. It would be convenient if the system can show the possible correct login names, such as PERKOWSKI and FALKOWSKI.* *(End of Example)*

[1] *CAMs are sometimes called *ternary* CAMs or TCAM[6].

Table 2.1. CAM Table

Address	Vectors
1	0010
2	0111
3	1101
4	0101
5	0011
6	1011
7	0001

Table 2.2. CAM function truth table

x_1	x_2	x_3	x_4	f_2	f_1	f_0
0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	0	1	0	0	0	1
0	0	1	1	1	0	1
0	1	0	0	0	0	0
0	1	0	1	1	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	0
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	1	1	0
1	1	0	0	0	0	0
1	1	0	1	0	1	1
1	1	1	0	0	0	0
1	1	1	1	0	0	0

2 Content Addressable Memory

We adopt the following definitions, which are intended to model certain physical devices [2] .

2.1 CAM

Definition 2.1 *An n-input, m-output, p-valued, k-entry* **CAM table** *stores k n-element p-nary vectors at addresses* 1 *through k. An address is an m-element p-nary vector, where* $m = \lceil \log_p(k+1) \rceil$. *Let* $P = \{0, 1, \ldots, p-1\}$. *The corresponding* **CAM function** *is a logic function* $\vec{f} : P^n \rightarrow P^m$, *where* $\vec{f}(\vec{x})$ *is the address of the entry that matches* $\vec{x}$ *exactly. If no such entry exists,* $\vec{f}(\vec{x}) = 0^m$.

Example 2.1 *Consider the CAM table shown in Table 2.1. Here,* $n = 4$, $m = 3$, $p = 2$, *and k=7. The truth table of the CAM function is shown in Table 2.2. The function's output is an address if there is an exact match (e.g.,* $\vec{f}(1,1,0,1) = 011$*). If the input vector is not stored in the CAM table, the output is 000 (e.g.,* $\vec{f}(1,1,1,0) = 000$*). (End of Example)*

2.2 Distance d CAM

A distance d CAM is used for pattern recognition and electronic dictionary [7, 8] applications.

Definition 2.2 *Given a CAM table, the corresponding* **distance d CAM function** *is a logic function* $\vec{f} : P^n \rightarrow P^m$, *where* $\vec{f}(\vec{x})$ *is the address of a CAM table entry that differs in no more than d digits from* $\vec{x}$, *i.e., the distance is d or less. If no such entry exists,* $\vec{f}(\vec{x}) = 0^m$. *If more than one entry qualifies, the address is determined by identifying all the entries that are the smallest distance from* $\vec{x}$ *and then by choosing the smallest address from these entries.*

[2] Some CAMs [1] have both *Address* and *Match Flag* outputs. In this case, the vectors are stored from address 0 (instead of 1), and $MatchFlag = 1$ if at least one element in the CAM table matches. Our CAM is different from such CAMs.

Table 2.3. Distance 1 CAM function truth table

x_1	x_2	x_3	x_4	f_2	f_1	f_0
0	0	0	0	0	0	1
0	0	0	1	1	1	1
0	0	1	0	0	0	1
0	0	1	1	1	0	1
0	1	0	0	1	0	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	0	0	0	0	0	0
1	0	0	1	0	1	1
1	0	1	0	0	0	1
1	0	1	1	1	1	0
1	1	0	0	0	1	1
1	1	0	1	0	1	1
1	1	1	0	0	0	0
1	1	1	1	0	1	0

Example 2.2 *Table 2.3 shows the truth table of the distance 1 CAM function corresponding to the CAM table of Table 2.1. For* $\vec{x} = (1,0,0,1)$, *there is no entry that matches exactly. However, the entries* $(1,1,0,1)$, $(1,0,1,1)$, *and* $(0,0,0,1)$ *at addresses 3, 6, and 7 differ in only one bit. Since the smallest address is 3,* $\vec{f}(1,0,0,1) = (0,1,1)$. *(End of Example)*

2.3 *CAM

Definition 2.3 *An n-input, m-output, p-valued, k-entry* ***CAM table** *stores k n-element* $(p+1)$*-nary vectors at addresses* 1 *through k. Each* $(p+1)$*-nary entry consists of* $0, 1, \ldots, p-1$ *and * (don't care). An address is an m-element p-nary vector, where* $m = \lceil \log_p(k+1) \rceil$. *The corresponding* ***CAM function** *is a logic function* $\vec{f} : P^n \rightarrow P^m$, *where* $\vec{f}(\vec{x})$ *is the smallest address of an entry that is identical to* $\vec{x}$ *except for don't care values. If no such entry exists,* $\vec{f}(\vec{x}) = 0^m$.

Example 2.3 *Table 2.4 shows a *CAM table that stores seven ternary vectors, where* $n = 4, m = 3, p = 2$, *and* $k = 7$. *The corresponding *CAM function truth table is shown in Table 2.5. The vector* $\vec{x} = (1,0,1,1)$ *matches the entries at addresses 5 and 6 except for don't care values. Since 5 is smaller,* $\vec{f}(1,0,1,1) = (1,0,1)$. *(End of Example)*

*CAMs for $p = 2$ are extensively used in routing tables for the internet. A routing table specifies an interface identifier corresponding to the longest prefix that matches an incoming packet, in a process called **Longest Prefix Match**

Table 2.4. *CAM Table

Address	Vectors
1	*010
2	0011
3	1101
4	1100
5	*011
6	1*11
7	*001

Table 2.5. *CAM function truth table

x_1	x_2	x_3	x_4	f_2	f_1	f_0
0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	1	0
1	0	0	1	1	1	1
1	0	1	0	0	0	1
1	0	1	1	1	0	1
1	1	0	0	1	0	0
1	1	0	1	0	1	1
1	1	1	0	0	0	0
1	1	1	1	1	1	0

(LPM). In the *CAM table for LPM, the $(p+1)$-nary vectors have restricted patterns: the prefix consists of only 0's, 1's,..., and $p-1$'s, and the postfix consists of only *'s (*don't cares*).

Definition 2.4 *The* **LPM table** *stores distinct $(p+1)$-nary vectors of the form $VEC_1 \cdot VEC_2$, where VEC_1 consists of 0's, 1's, ..., and $p-1$'s, and VEC_2 consists of *'s. To assure that the longest prefix address is produced, *CAM entries are stored in descending prefix length. The* **LPM function** *is the logic function represented by the *CAM function for the LPM table.*

Example 2.4 *Consider the LPM table shown in Table 2.6. In the third vector VEC_1=01 and VEC_2=**, while in the last vector VEC_1=0 and VEC_2=***. Clearly, this is a CAM table for LPM. The truth table for the LPM function is shown in Table 2.7. (End of Example)*

Table 2.6. LPM Table

Address	Vectors
1	1000
2	010*
3	01**
4	1***
5	0***

Table 2.7. LPM function truth table

x_1	x_2	x_3	x_4	f_2	f_1	f_0
0	0	0	0	1	0	1
0	0	0	1	1	0	1
0	0	1	0	1	0	1
0	0	1	1	1	0	1
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	0	0	1
1	0	0	1	1	0	0
1	0	1	0	1	0	0
1	0	1	1	1	0	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	1	0	0

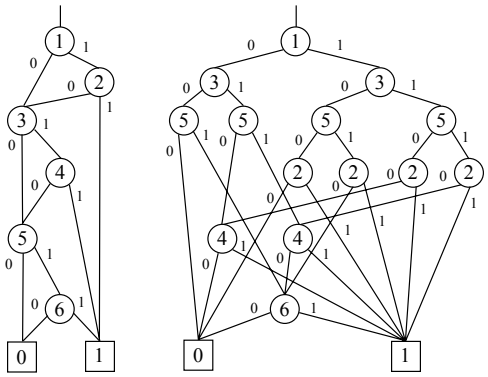

Figure 3.1. BDD for two different orderings.

3 Properties of CAM Functions

3.1 C-measure of a Logic Function

Definition 3.1 *Consider a function $\vec{f}(\vec{X}) : P^n \rightarrow P^q$, where $P = \{0,1,\ldots,p-1\}$, and $\vec{X} = (x_1, x_2, \ldots, x_n)$. Let $(\vec{X}_L, \vec{X}_H)$ be a partition of the variable set $\vec{X}$. The* **decomposition chart** *for f is a two-dimensional matrix, where the column labels have all possible assignments of values to variables in $\vec{X}_L$, the row labels have all possible assignments of values to variables in $\vec{X}_H$, and the corresponding matrix value is equal to $\vec{F}(\vec{X}_L, \vec{X}_H)$. Among the decomposition charts for $\vec{F}$, the one where $\vec{X}_L = (x_1, x_2 \ldots, x_{n_L})$ and $\vec{X}_H = (x_{n_{L+1}}, x_{n_L+2}, \ldots, x_n)$ is a* **standard decomposition chart**, *where $1 \leq n_L \leq n$. The number of different column patterns in the decomposition chart is the* **column multiplicity of the decomposition chart**.

Definition 3.2 The C-measure of a logic function *f is the maximum value of all possible column multiplicities of the standard decomposition charts for f, where we assume that the ordering of variables is fixed at $(x_1, x_2, \ldots, x_n)$.*

The column multiplicity of a decomposition chart is equal to the width of the MTMDD (multi-terminal multi-valued decision diagram). So, the C-measure of a logic function is equal to the maximum width of the MTMDD for the given ordering of the input variables.

Example 3.1 *Consider two functions: $f_1 = x_1x_2 \vee x_3x_4 \vee x_5x_6$ and $f_2 = x_1x_5 \vee x_2x_6 \vee x_3x_4$. Fig. 3.1 shows the BDDs for f_1 and f_2. In this case, C-measures of f_1 and f_2 are 3 and 8, respectively. (End of Example)*

For a given logic function and variable ordering, the C-measure is easy to obtain and uniquely defined. Functions

having small C-measures have efficient LUT cascade realizations.

Lemma 3.1 *For a given function f, let k be the number of the input combinations that produce non-zero output values. Then, the C-measure of f is at most $k+1$.*

3.2 CAM

Theorem 3.1 *Consider a CAM table that stores k vectors. The C-measure of the CAM function is at most $k+1$.*

(Proof) The number of non-zero outputs of the CAM function is at most k. From Lemma 3.1, the C-measure is at most $k+1$. $(Q.E.D.)$

Theorem 3.1 shows that the worst case value of the C-measure is $k+1$. A worst case occurs when the non-zero CAM table entries are widely scattered. However, in practical applications, such as TLBs, the virtual page address will be clustered due to frequent access to the same blocks of memory (e.g., the program or stack). Thus, we expect the actual value of the C-measure to be smaller in practical applications.

3.3 Distance 1 CAM

Theorem 3.2 *Consider a CAM table that stores k vectors of n digits. Then, the C-measure of the distance 1 CAM function is at most $kn(p-1)+k+1$.*

(Proof) In the CAM table, for each vector, generate $n(p-1)$ adjacent vectors, and generate the truth table of the CAM function that shows exact matches and distance 1 matches. Note that the number of non-zero output values of the CAM function is at most $k(n(p-1)+1)$. From Lemma 1, the C-measure is at most $k(n(p-1)+1)+1$. $(Q.E.D.)$

3.4 *CAM

Theorem 3.3 *Consider a *CAM table that stores k vectors, where each vector has at most t don't cares. Then, the C-measure of the corresponding *CAM function is at most $p^t k+1$.*

(Proof) In the *CAM table, replace each * by 0,1, ..., and $p-1$, and generate the table without *, i.e., the CAM table. For each pattern, we can generate at most p^t vectors. So, the number of non-zero outputs of the *CAM function is at most $p^t k$. From Lemma 1, the C-measure is at most $p^t k+1$. $(Q.E.D.)$

Theorem 3.4 *The C-measure of an LPM function with k vectors is at most $k+1$.*

(Proof) We prove the theorem for the case where all k entries in the LPM table map to distinct output values. This is the worst case, since forcing certain entries to have the same output value can only reduce the column multiplicity.

We prove the theorem by counting the number of distinct columns in the standard decomposition chart as LPM vectors are added to the LPM table. Reorder the LPM vectors so that those vectors with the most * entries are first and those with the fewest are last. Consider a standard decomposition chart where assignments of values to $x_1, x_2, \ldots, x_i$ label the columns, and assignments of values to $x_i, x_{i+1}, \ldots, x_n$ label the rows. An "empty" standard decomposition chart has a unique column pattern (all 0's). Let the first vector be $\vec{\alpha} = (a_1, a_2, ..., a_m, *, *, ..., *)$, where $a_j \in P$. If $m > i$, then the first vector changes only a proper subset of elements in one column. If $m = i$ ($m < i$), then the new vector changes all elements in one (or more) complete column(s) to the vector's output value in the LPM table. In either case, at most one distinct column pattern is added to the standard decomposition chart.

Because the second vector has no more * entries than the first vector, adding it will change columns only among a subset of the two distinct columns so far in the standard decomposition chart. Let the new vector be $\vec{\beta} = (b_1, b_2, ..., b_{m'}, *, *, ..., *)$, where $b_j \in P$. If $b_i = a_i$, for all $1 \le i \le m'$, then a subset of the columns created by adding $\bar{\alpha}$ to the empty standard decomposition chart are changed. Otherwise, a subset of the columns containing all 0's are changed. In either case, at most one additional column pattern is added. This process continues until all vectors are exhausted. In all, at most $k+1$ column patterns are created. The theorem follows. $(Q.E.D.)$

4 LUT Cascade

It is possible to realize a CAM function by a standard RAM. For example, the 7-entry CAM shown in Table 2.1 can be realized by a 16-word RAM, where each word has 3 bits, as shown in Table 2.2. The size of the RAM is exponential in the number of bits n used to store the CAM data, even though the CAM contains relatively few data words. The LUT cascade takes advantage of this, offering a way to reduce memory requirements substantially.

Theorem 4.1 *For a given function f, let $\vec{X}_L$ be the column variables, and let $\vec{X}_H$ be the row variables, and let μ be the column multiplicity of the decomposition chart. Then, function f is realizable with the network shown in Fig. 4.1. In this case, the number of p-nary signal lines that connect two blocks H and G is $\lceil \log_p \mu \rceil$.*

When the number of p-nary signal lines that connect two blocks is smaller than the number of variables in $\vec{X}_L$, we can

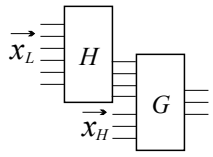

Figure 4.1. Realization of logic functions by decomposition.

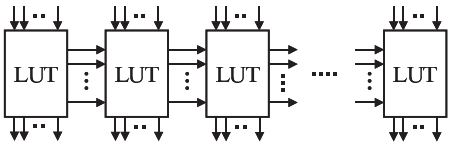

Figure 4.2. LUT cascade with intermediate outputs.

often reduce the size of memory to implement the function. This technique is **functional decomposition**.

By applying functional decomposition repeatedly to the given function, we have the **LUT cascade** [11] shown in Fig. 4.2. The cascade consists **cells**, and the wires connecting adjacent cells are **rails**. Functions with small C-measure require fewer rails, and thus have more compact LUT cascade realizations. To derive C-measures, we need not use decomposition charts. We can efficiently obtain the C-measure from a binary decision diagram (BDD_for_CF) that represents the characteristic function for the multiple-output function [12].

Theorem 4.2 *[11] A logic function with C-measure μ can be implemented by a LUT cascade consisting of cells with at most $\lceil \log_p \mu \rceil + 1$ inputs and $\lceil \log_p \mu \rceil$ outputs .*

Theorem 4.3 *[13] Consider an LUT cascade for a function f. Let n be the number of primary inputs, s be the number of cells, r be the maximum number of rails (i.e., the number of lines between cells), K be the maximum number of inputs of a cell, and μ be the C-measure of f. If $K \geq \lceil \log_p \mu \rceil + 1$, then there is an LUT cascade for f that satisfies the relation:*

$$s \leq \left\lceil \frac{n-r}{K-r} \right\rceil .$$

5 Experiment: Two-Valued Case

We generated random CAM tables, and converted the corresponding CAM functions into MTBDDs.

Table 5.1. C-measure and Size of memory for CAM functions.

n	k	# nodes MTBDD	C-meas	Mem M bit
32	1,000	23,156	1,001	0.44
32	2,000	44,316	2,001	0.94
32	3,000	64,628	3,001	1.88
32	4,000	84,512	4,001	1.88
32	5,000	104,125	5,001	3.75
32	6,000	123,408	6,001	3.86
32	7,000	142,402	7,001	3.86
32	8,000	161,237	8,001	3.86
32	9,000	179,879	9,001	7.41

5.1 CAM

For CAM tables, we choose the probability of a 0 and a 1 in the entries to be the same. Table 5.1 shows the result. n denotes the number of variables in the CAM function, and k denotes the number of vectors in the CAM table. *C-meas* shows the C-measure (i.e., the maximum width of the MTBDD). This confirms that the C-measure upper bound (Theorem 3.1) of $k+1$ is firm.

In these example runs, the numbers of CAM table entries are small compared to the total number of possible entries. For example, when $n = 32$, the CAM table with $k = 1000$ represents $\frac{1000}{2^{32}} = 2.3 \times 10^{-7}$ of the total number of entries in the truth table. We expect, therefore, the entries to be widely scattered.

5.2 Distance 1 CAM

Tables 5.2 and 5.3 show the results for different numbers of vectors k, and different numbers of inputs n, respectively.

The expression for the upper bound on the C-measure for distance 1 CAM functions from Theorem 3.2, $nk(p-1)+k+1$, suggests an approximately equal dependence on n and k, when $p = 2$, and especially when n and k are large. However, in comparing the experimental values for the C-measure shown in Tables 5.2 and 5.3, one sees a much stronger dependence on k than on n. That is, Table 5.2, where n is fixed, shows that the C-measure is approximately a linear function of k. However, Table 5.3, where k is fixed, shows that the C-measure not strongly dependent n.

5.3 *CAM

5.3.1 General *CAM Table

Here, we assume that $p = 2$. For *CAM tables, we choose the probability of a 0, 1, and $*$ *(don't care)* in the entries of the *CAM table to be the same. Table 5.4 shows the

Table 5.2. C-measure and Size of memory for Distance 1 CAM functions

n	k	# nodes MTBDD	C-meas	Mem M bit
32	100	5,751	373	0.11
32	200	11,424	801	0.25
32	300	16,976	1,249	0.49
32	400	22,433	1,765	0.54
32	500	27,997	2,260	0.76
32	1,000	54,931	4,867	1.64
32	1,500	82,020	7,610	2.31
32	2,000	108,700	10,363	3.38
32	2,500	135,340	12,975	5.34
32	3,000	161,352	16,048	5.81

Table 5.3. C-measure and Size of memory for Distance 1 CAM functions

n	k	# nodes MTBDD	C-meas	Mem M bit
12	500	3,903	1,457	0.04
14	500	7,893	2,119	0.13
16	500	11,056	2,240	0.22
24	500	19,858	2,260	0.54
32	500	27,997	2,260	0.76
40	500	35,670	2,265	0.96
48	500	43,807	2,271	1.18

result. In this case, the C-measure is much greater than in Table 5.1.

In Table 5.4, we take δ as 11, about one-third of the 32 variables, where δ is the number of *don't cares* in the entries. This implies that the number of non-zero outputs for *CAM functions is at most $2^{11} = 2048$ times greater than that of CAM functions.

Table 5.4. C-measure and Size of memory for General *CAM functions.

n	k	# nodes MTBDD	C-meas	Mem M bit
32	1,000	82,514	11,886	3.89
32	2,000	237,291	34,561	10.97
32	3,000	460,106	70,835	20.25
32	4,000	716,909	114,814	34.00
32	5,000	978,984	151,577	61.00
32	6,000	1,320,175	213,503	66.62
32	7,000	1,685,556	273,292	106.50
32	8,000	2,039,238	335,097	134.00
32	9,000	2,469,748	418,833	139.75

Table 5.5. C-measure and Size of memory for LPM functions.

n	k	# nodes MTBDD	C-meas	Mem M bit
32	1,000	15,095	1,001	0.42
32	2,000	28,197	2,001	0.67
32	3,000	40,552	3,001	1.31
32	4,000	52,450	4,001	1.31
32	5,000	64,010	5,001	2.53
32	6,000	75,251	6,001	2.64
32	7,000	86,292	7,001	2.64
32	8,000	97,037	8,001	2.64
32	9,000	107,597	9,001	4.78

Table 6.1. C-measures for Multiple-valued Functions.

Name	CAM	dist 1 CAM	* CAM	LPM CAM
u552	548	1135	553	548
u1245	1238	2528	1246	1238
u2061	2048	4286	2056	2048
u3366	3367	6891	3313	3367

5.3.2 LPM Table

Here, we assume that $p = 2$. In the LPM tables, we generated random prefixes consisting of 0's and 1's. The lengths of the prefixes were chosen from 22 to 26, and we chose the probability of a 0 and 1 to be the same. Table 5.5 shows that the C-measure is $k+1$. Table 5.5 shows that Theorem 3.4 holds. Compared with the result in [10], the C-measure in our experimental results is about 10 times larger. This is because practical routing tables were used in [10], while we used randomly generated LPM tables.

6 Experiment: Multi-Valued Case

For the multiple-valued case, instead of using randomly generated function, we used lists of English words as benchmark functions. We only considered English words with at most 8 letters. For the words with fewer than 8 letters, blanks are appended to the end of the words to make them 8 letters. So, the function has $5 \times 8 = 40$ input variables. $u552$ consist of 552 words having 8 letters. $u1245$ consist of 1245 words having 7 or 8 letters. $u2061$ consist of 2061 words having 6 or 7 or 8 letters. $u3366$ consist of 3366 words having 3 to 8 letters. Table 6.1 shows the experimental results for CAMs, distance 1 CAMs, * CAMs, and LPM CAMs.

CAM: For $u3366$, the C-measure is $k+1 = 3367$. However, for other functions, the C-measures are slightly

smaller than k.

Distance 1 CAM: The C-measures are approximately $2k$. The bound given by Theorem 3.2 is $kn(p-1)+k+1 \simeq k(40\times 25)$, which is much larger than the actual values.

***CAM:** To generate *random* *CAMs, for each word, we replaced just one letter with a "*". The C-measures are always larger than those of corresponding CAMs. The bound given from Theorem 3.3 is $p^t k+1 = 26\times k+1$, which is much larger than the actual values.

LPM CAM: The C-measures are the same as that of the CAM cases. The bound given from Theorem 3.4 is $k+1$. Note that the exact value matches that of CAMs.

6.1 LUT Cascade Realization

Consider the CAM realization of $u3366$. The C-measure is 3367. Note that $\lceil \log_2 3367 \rceil = 12$. Thus, by Theorem 4.1, the CAM can be implemented by binary cells with 13 inputs and 12 outputs. By Theorem 4.2, the number of cells is

$$s = \lceil \frac{n-r}{K-r} \rceil = \lceil \frac{40-12}{13-12} \rceil = 28,$$

since $r = \lceil \log_2 3367 \rceil = 12$, and $K = 13$. Thus, the total amount of memory is at most $s \cdot 2^K \cdot r = 28 \times 2^{13} \times 12 = 336 \times 2^{13} \simeq 2.7 \times 10^6$ bits. Note that the straightforward implementation by a single memory requires $2^{40} \times 12 \simeq 1.3 \times 10^{13}$ bits.

7 Concluding Remarks

In this paper, we showed a method to implement multiple-valued CAM functions by using LUT cascades. We also defined the C-measure that shows the complexity of LUT cascade. The C-measure is easy to obtain from the MTMDD of the given function. As shown in this paper, LUT cascades are promising for the realization of CAM, LPM, and distance 1 CAM functions. However, an LUT cascade may be too large for the replacement of *CAMs of general functions especially when the *CAM table has many *don't cares*.

Acknowledgment

This work was supported by a grant from the Japanese Ministry of MEXT via Kitakyushu Innovative Cluster Project, the Aid for Scientific Research of the Japan Society for the Promotion of Science (JSPS), and NSA Contract RM A54.

References

[1] ALTERA, " Implementing high-speed search applications with Altera CAM," Application Note 119, Altera Corporation.

[2] P. Gupta, S. Lin , and N. McKeown, "Routing lookups in hardware at memory access speeds," *Proc. INFOCOM*, IEEE Press, Piscataway, N.J., 1998, pp. 1240-1247.

[3] T. Hanyu, N. Kanagawa, and M. Kameyama, "Design of a one-transistor-cell multiple-valued CAM," *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 11, Nov 1996, pp. 1699-1674.

[4] M. Kobayashi, T. Murase, and A. Kuriyama, "A longest prefix match search engine for multigigabit IP processing," *Proc. Int'l Conf. on Communications (ICC 2000)*, IEEE Press, Piscataway, N.J., 2000, pp. 1360-1364.

[5] T. Kohonen, *Content-Addressable Memories*, Springer Series in Information Sciences, Vol. 1, Springer Berlin Heidelberg 1987.

[6] H. Liu, "Routing table compaction in ternary CAM," *IEEE Micro*, Vol. 22, No.1, Jan.-Feb. 2002, pp. 58-64.

[7] H. J. Mattausch, T. Gyohten, Y. Soda and T. Koide, "Compact associative-memory architecture with fully-parallel search capability for the minimum Hamming distance", *IEEE Journal of Solid-State Circuits*, Vol.37, No. 2, Feb. 2002, pp. 218-227.

[8] M. Motomura et al., "A 1.2-million-transistor, 33-MHz, 20-b dictionary search processor (DISP) ULSI with a 160-Kbyte CAM," *IEEE J. Solid-State Circuits*, Vol. 25, No. 5, Oct. 1990, p. 1158- 1165.

[9] K. Pagiamtzis and A. Sheikholeslami, "A Low-power content-addressable memory (CAM) using pipelined hierarchical search scheme," *IEEE Journal of Solid-State Circuits*, Vol. 39. No. 9, Sept. 2004, pp.1512-1519.

[10] A. Prakash, R. Kotla, T. Mandal, and A. Aziz, "A high-performance architecture and BDD-based synthesis methodology for packet classification," *IEEE Trans. on CAD*, Vol. 22, No. 6, pp. 698-709, June 2003.

[11] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *International Workshop on Logic and Synthesis (IWLS01),* Lake Tahoe, CA, June 12-15, 2001, pp. 225-230.

[12] T. Sasao and M. Matsuura, "BDD representation for incompletely specified multiple-output logic functions and its applications to functional decomposition," *Design Automation Conference*, June 2005, pp.455-462.

[13] T. Sasao, "Analysis and synthesis of weighted-sum functions," *IEEE Transaction on CAD*, (accepted for publication.)

[14] A. Sheikholeslami, P. G. Gulak, and T. Hanyu, "A multiple-valued ferroelectric content-addressable memory," *26th International Symposium on Multiple-Valued Logic (ISMVL '96)*, May 1996, pp.74-79.

[15] F. Zane, G. Narlikar, and A. Basu, "CoolCAM: Power-efficient TCAMs for forwarding engines", *Proceeding of IEEE INFOCOM '03*, April, 2003.