AN UPDATE METHOD FOR A CAM EMULATOR USING AN LUT CASCADE BASED ON AN EVMDD (K)

[1]*Hiroki Nakahara* [2]*Tsutomu Sasao* [3]*Munehiro Matsuura*

[1]Kagoshima University, Japan [2]Meiji University, Japan [3]Kyushu Institute of Technology, Japan

ABSTRACT

Core routers perform longest prefix matching (LPM) using content addressable memories (CAMs). With the rapid growth of the Internet, LPM has become the bottleneck in network traffic management. In a previous publication, we have proposed an area-efficient and high-performance CAM emulator using an LUT cascade based on an edge-valued multi-valued decision diagram (EVMDD (k)). In the internet, registered vectors must be updated frequently. In this paper, we propose an algorithm to update an LUT cascade. We implemented the proposed algorithm on the ARM processor. Its update time is shorter than the peak update time of the BGP protocol. Experimental results show that, as for the lookup speed per area, our architecture outperforms existing CAM realizations on FPGAs.

1. INTRODUCTION

1.1. Demands of LPM Architecture

Routers forward packets in IP address lookups using **longest prefix matching (LPM)**. With the rapid growth of the Internet, LPM has become the bottleneck in the network traffic management. Previously, ternary content addressable memories (TCAMs) were widely used in routers to realize LPM. With the rapid increase of traffic, core routers dissipate the major part of the total network power [17]. Thus, we cannot use TCAMs any more, since they dissipate too much power. Le and Prassana [7] proposed a memory-based IP lookup architecture on a field programmable gate arrays (FPGAs), which dissipate lower power than TCAMs.

In this paper, we consider a CAM emulator using an LUT cascade on the FPGA, which has the following features:

High throughput per area: Recently, core routers work at the 100 Gbps link speed for the minimum packet size (40 bytes). A parallel processing is an effective method to increase the system throughput. In this case, the throughput per area is an important measure [4]. A modern FPGA consists of lookup-tables (Slices), on-chip memories (BRAMs), arithmetic circuits (DSP48Es), and so on. Thus, a balanced usage of hardware resources in FPGAs is the key to achieve a high throughput per area.

High-speed updatable: The IP addresses on routers are frequently updated (added and deleted). For a border gateway protocol (BGP), its peak number of updates per second is about 10,000 [1]. The simplest method to update the LPM architecture on an FPGA is direct rewriting of its interconnections using the new configuration data. However, since the time to generate the new configuration is very long, it is infeasible. Thus, the high-speed update on the LPM architecture is essential.

1.2. Proposed Method

In the previous publications, we proposed CAM emulators based on the edge-valued multi-valued decision diagrams (EVMDD (k)s) [10] for the IP address matching [12] and the packet classification [11]. They are more efficient than other FPGA implementations. However, they did not consider the update method.

Previous work showed that the LUT cascade based on the EVMDD (k) is smaller than one based on the MTMDD (k). The addition and deletion can be done in time that is proportional to the number of cells in the LUT cascade based on the multi-terminal MDD (MTMDD (k)) [12]. In this paper, we apply its method into the LUT cascade based on the EVMDD (k). Thus, the proposed LUT cascade based on the EVMDD (k) satisfies above conditions.

1.3. Organization of the Paper

The rest of the paper is organized as follows: Chapter 2 defines an LPM function; Chapter 3 introduces the LUT cascade based on an EVMDD (k); Chapter 4 shows the update method for the LUT cascade based on an EVMDD (k); Chapter 5 shows experimental results; and Chapter 6 concludes the paper.

2. DEFINITION OF A LONGEST PREFIX MATCHING (LPM) FUNCTION

Definition 2.1 *The* **LPM table** *stores ternary vectors of the form $VEC_1 \cdot VEC_2$, where VEC_1 consists of $0's$ and $1's$, and VEC_2 consists of $*'s$ (don't cares). The* **length** *of prefix is the number of bits in VEC_1. To assure that the longest prefix address is produced, entries are stored in the descending prefix length. Let $B \in \{0, 1\}$. The* **LPM function** *[14] is the logic function $\vec{f} : B^n \rightarrow B^m$, where $\vec{f}(x)$ is the min-*

0195-623X/14 $31.00 © 2014 IEEE
DOI 10.1109/ISMVL.2014.9

Table 1. Example of LPM function.

x_0	x_1	x_2	x_3	Rule
0	0	0	0	1
0	0	0	1	2
1	0	0	0	3
1	0	0	1	4
0	0	1	*	5
1	0	0	*	6
0	1	*	*	7
otherwise				0

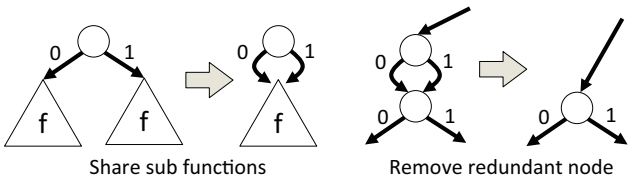

Fig. 1. Conversion of a binary tree node into an MTBDD node.

imum address of VEC_1 corresponding to $\vec{x}$. If there is no such vector, $\vec{f}(\vec{x}) = 0^m$.

We can assign an arbitrary monotone increasing index to the LPM table. In this paper, we use an M_1**-monotone increasing function** [8] to reduce the amount of memory.

Definition 2.2*[8] Let Z be the set of integers, and I be a set of integers including 0. An integer function $f\ (X) : I \rightarrow Z$ such that $0 \leq f\ (X+1) - f\ (X) \leq 1$ and $f\ (0) = 0$ is* **an M_1-monotone increasing function** *on I. That is, for an M_1-monotone increasing function $f\ (X)$, $f\ (0) = 0$, and the increment of X by one increases the value of $f(X)$ by at most one.*

Example 2.1 *Table 1 shows an LPM function that is also an M_1-monotone increasing function.* ■

3. CAM EMULATOR USING AN LUT CASCADE BASED ON AN EVMDD (K)

3.1. LUT Cascade Based on an MTMDD (k)

Definition 3.3 *A* **binary decision diagram (BDD)** *[2] is obtained by applying* **Shannon expansions** *repeatedly to a logic function f. Each non-terminal node labeled with a variable x_i has two outgoing edges which indicate nodes representing cofactors of f with respect to x_i.*

Definition 3.4 A multi-terminal BDD (MTBDD) *[3] is an extension of a BDD and represents an integer-valued function. In the MTBDD, the terminal nodes are labeled by integers.*

Definition 3.5 *Let $X = (X_1, X_2, \ldots, X_u)$ be a partition of the input variables, and $|X_i|$ be the number of input variables in X_i. X_i is called* **a super variable**. *When the*

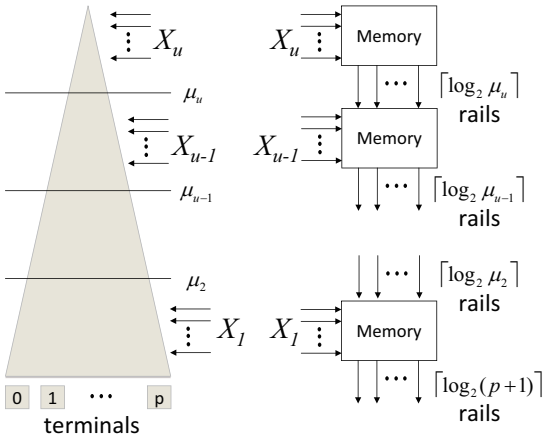

Fig. 2. An LUT cascade based on an MTMDD (k).

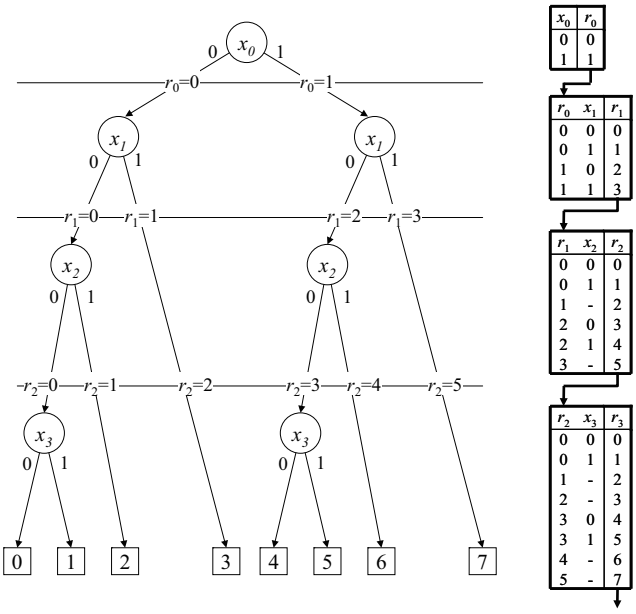

Fig. 3. Example of an LUT cascade based on an MTMDD (k).

Shannon expansions are performed with respect to super variables X_i, where $|X_i| = k$, all the non-terminal nodes have 2^k edges. In this case, we have a **multi-valued multi-terminal decision diagram (MTMDD(k))** *[5]. Note that, an MTMDD(1) means an MTBDD.*

Definition 3.6 The width of the MDD (k) at the height k *is the number of edges crossing the section of the MDD (k) between super variables X_{i+1} and X_i, and denoted by μ_i where the edges incident to the same node are counted as one.*

Let p be the number of rules, and $|X| = n$. An M_1-monotone increasing function can be realized by **an LUT cascade** [15] shown in Fig. 2. Connections between LUT_i

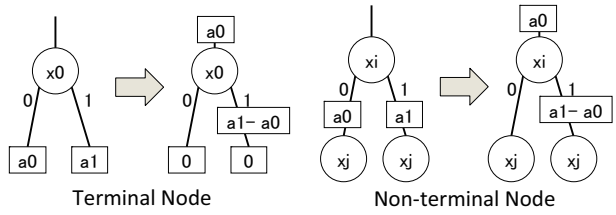

Fig. 4. Conversion of an MTBDD node into an EVBDD node.

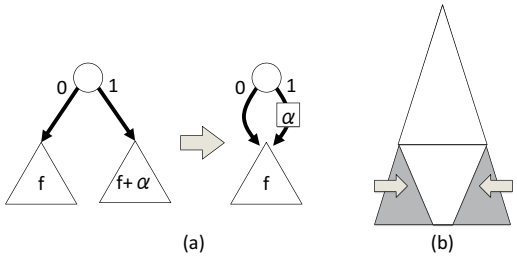

Fig. 5. Principle of reduction of width in an EVBDD.

and LUT_{i-1} requires $r_i = \lceil log_2\, \mu_i \rceil$ rails. Since a modern FPGA has BRAMs and distributed RAMs (realized by Slices), LUT cascades are easy to implement. The amount of memory for LUT_i based on an MTMDD (k) is $r_i \cdot 2^{(k+r_{i+1})}$ Thus, the total amount of memory for an LUT cascade is. $M = \sum_{i=0}^{u} r_i \cdot 2^{(k+r_{i+1})}$.

Example 3.2 *Fig. 3 shows an example of an LUT cascade based on an MTMDD (k).* ■

As for an M_1-monotone increasing function, the upper bound on the number of rails in the LUT cascade has been analyzed [13] [1].

Theorem 3.1 *Let p be the number of unique indices for the M_1-monotone increasing function. The upper bound on the number of rails in the LUT cascade is $r = \lceil log_2(p+1) \rceil$.*

3.2. CAM Emulator Using an LUT Cascade Based on an EVMDD (k)

To reduce the amount of memory for an LUT cascade, we introduce an LUT cascade based on **an edge-valued multi-valued decision diagram (EVMDD (k))**, which is an extension of an EVBDD [6]. An EVBDD consists of one terminal node representing zero and non-terminal nodes with a weighted 1-edge, where the weight has an integer value α. An EVBDD is obtained by recursively applying the conversion shown in Fig. 4 to each non-terminal node in an MTBDD. Note that, in the EVBDD, 0-edges have zero weights.

In an M_α-monotone increasing function, subfunction f' is obtained by adding α to subfunction f. Thus, an EVBDD may have smaller widths by sharing f and f' with an α edge (Fig. 5 (a)). The MTBDD only shares prefixes, while the EVBDD shares both prefixes and postfixes (Fig. 5 (b)).

[1] In [13], the M_1-monotone increasing function is called segment index encoder function.

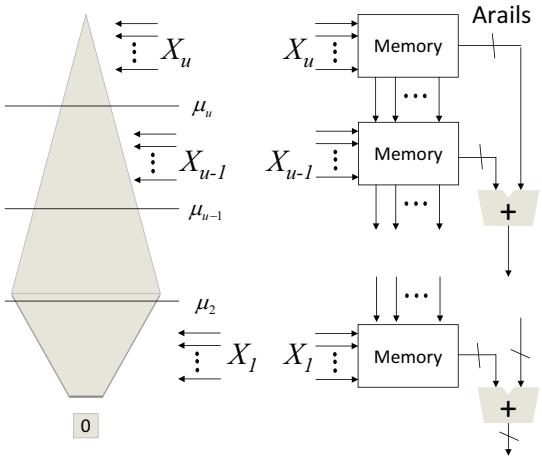

Fig. 6. An LUT cascade based on an EVMDD (k).

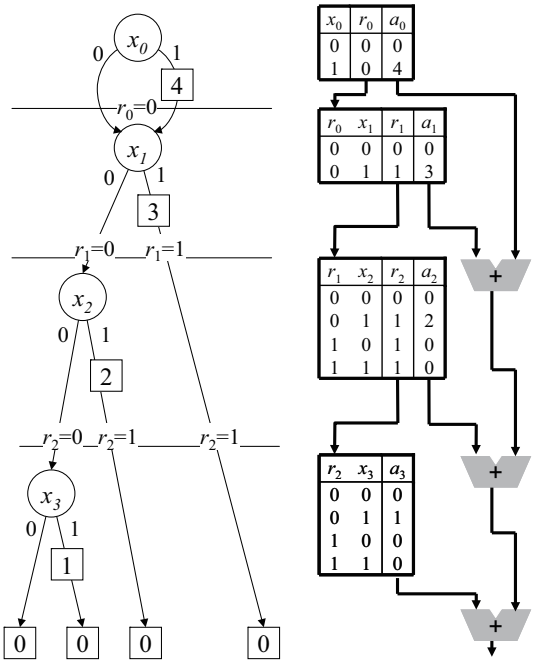

Fig. 7. Example of an LUT cascade based on an EVMDD (k).

Definition 3.7 An edge-valued MDD (k) (EVMDD (k)) *[9] is an extension of the MDD (k), and represents a multi-valued input M_1-monotone increasing function. It consists of one terminal node representing zero and non-terminal nodes with edges having integer weights, and 0-edges always have zero weights.*

Let p be the number of rules, and $|X| = n$. An M_1-monotone increasing function is efficiently realized by an LUT cascade with adders [10] shown in Fig. 6. In this case,

the rails represent sub-functions in the EVMDD (k). Each LUT_i has an additional rail representing the weight of the edge. We call such an output **Arail** which consists of a_i rails. Since the width of the EVMDD (k) for M_1-monotone increasing function is often smaller than that of the MTMDD (s), we can reduce the amount of memory for the LUT cascade by using an EVMDD (k). Since adders are realized by DSP blocks (DSP48Es), FPGA resources are efficiently used.

Example 3.3 *Fig. 7 shows an example of an LUT cascade based on an EVMDD (k).* ■

The amount of memory for LUT_i is $(r_i + a_i) \cdot 2^{k+r_{i+1}}$. Let $|X| = n$ be the number of inputs, and $k = |X_i|$. The LUT cascade requires $u = \lceil \frac{n}{k} \rceil$ LUTs. Let M be the amount of memory for the LUT cascade based on an EVMDD (k). Then, we have

$$M = \sum_{i=0}^{u} (r_i + a_i) \cdot 2^{k+r_{i+1}}. \quad (1)$$

Also, it requires u adders. Generally, an increase of k increases the amount of memory, while decreases the number of adders.

4. UPDATE METHOD FOR THE LUT CASCADE BASED ON THE EVMDD (K)

4.1. Definition of the Update

Definition 4.8 The update for the EVMDD (k) *is a change of a constant value in the function.*

The update of the LUT cascade is decomposed into an **addition** and a **deletion**. The addition is archived by rewriting the index corresponding to non-zero, while the deletion is archived by rewriting the index corresponding to zero. Thus, the update requires both an addition and a deletion.

4.2. Update of the LUT Cascade Based on the EVMDD (k)

To update the LUT cascade based on the EVMDD (k), first, we update the EVMDD (k) corresponding to the update vector. We show an algorithm to update the EVMDD (k) as follows:

Algorithm 4.1

1. *Traverse the EVMDD (k) from the root node to the terminal node corresponding to the update vector by converting the EVMDD node into the MTMDD node.*
2. *When it reaches the terminal node, then rewrite the terminal value.*
3. *Return to the root node by converting the MTMDD node to the EVMDD node shown in Fig. 4.*
4. *Terminate.*

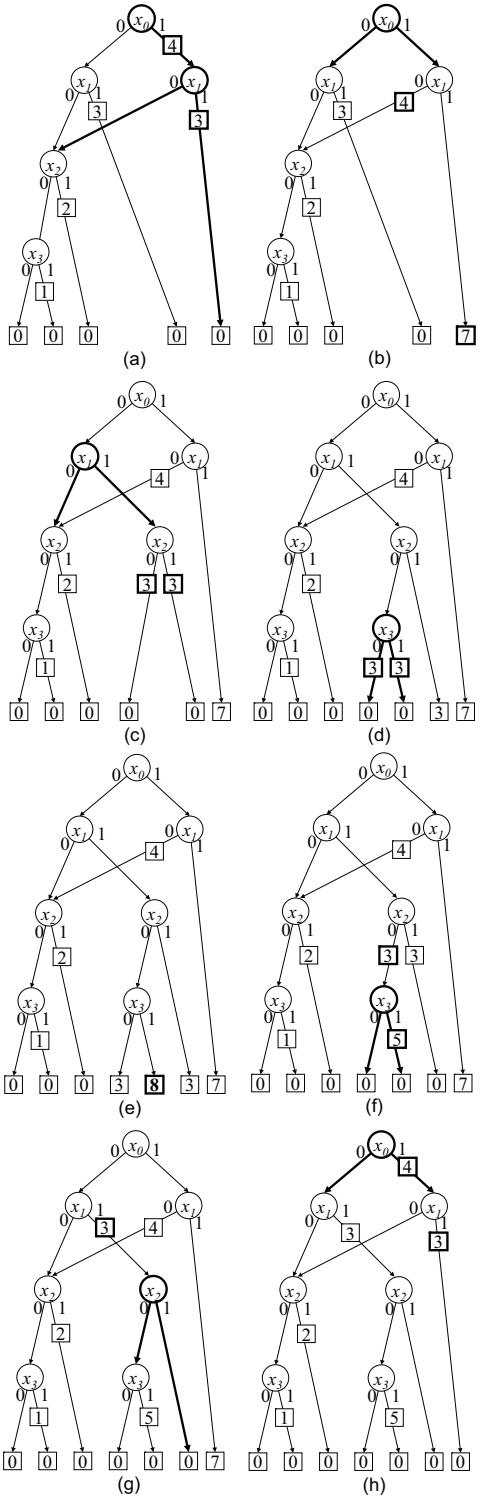

Fig. 8. Example of the update for the EVMDD (1).

Example 4.4 *Fig. 8 updates the index "0" of the vector "0101" into the index "8". First, it traverses the EVMDD (*1*) corresponding to the vector "0101" (Fig. 8 (a)〜(d)) by converting the EVMDD node into the MTMDD node. Then, it*

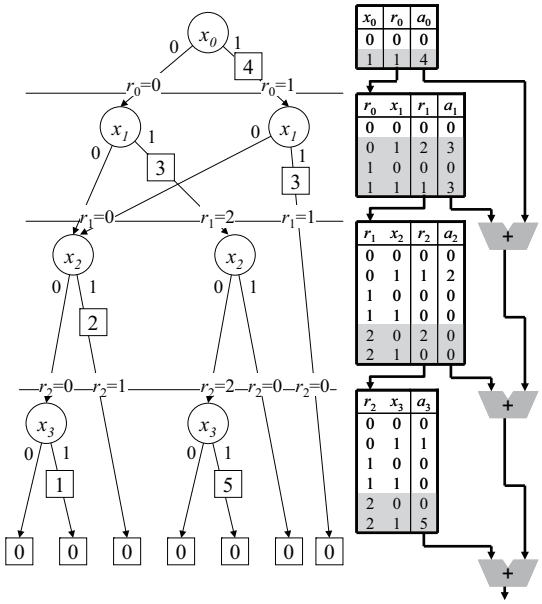

Fig. 9. Example of the update for the LUT cascade based on the EVMDD (1).

rewrites the terminal value into "8" (Fig. 8 (e)). Finally, it returns to the root node by converting the MTMDD node into the EVMDD node (Fig. 8 (f)〜(h)). ■

Then, we modify the memory of the LUT cascade according to the modified part of the EVMDD (k). Modification of the LUT cascade can be done as follows:

Algorithm 4.2

1. *Apply the Algorithm 4.1.*
2. *Traverse the modified EVMDD (k) corresponding to the update vector. Then, modify the memory of the LUT cascade corresponding to the modified node on the EVMDD (k).*
3. *Terminate.*

Example 4.5 *Fig. 9 shows the modification of the memory on the LUT cascade (indicated in gray) according to modified part of the EVMDD shown in Fig. 8.* ■

4.3. Analysis of the Memory Size of the LUT Cascade

We analyze the upper bound on the memory size with respect to the number of update vector p'.

Theorem 4.2 *Let p be the width of the EVMDD (k) representing M_1 monotone increasing function. When p' vectors are updated, the width of the EVMDD (k) is at most $p + p' + 1$.*

(Proof) As for the EVMDD (k), by shifting down all the edge values to the terminal node, we have the MTMDD (k). From Theorem 3.1, the width of the MTMDD (k) increases at most p'. Thus, the width of the EVMDD (k) is at most $p + p' + 1$ after the update of p' vectors. (Q.E.D.)

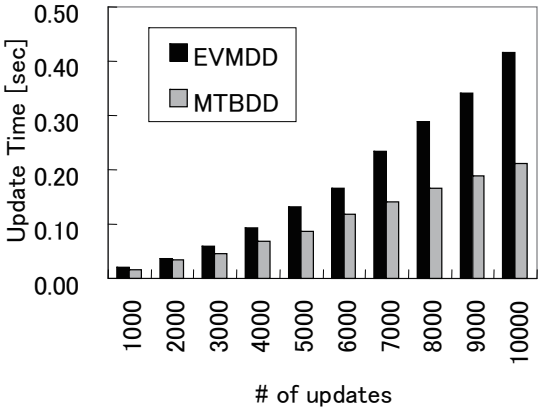

Fig. 10. Comparison with Update Time.

By Theorem 3.1, we have an upper bound of the number of rails on the LUT cascade from the upper bound of the width of the EVMDD (k)

Theorem 4.3 *Let p be the width of the EVMDD (k) representing M_1 monotone increasing function. After p' vectors are updated, the number of rails on the LUT cascade based on the EVMDD (k) is at most $r = \lceil log_2(p + p' + 1) \rceil$.*

(Proof) From Theorem 4.2, the width of the EVMDD (k) is at most $p + p' + 1$. Obviously, the number of rails is at most $r = \lceil log_2(p + p' + 1) \rceil$. (Q.E.D.)

Theorem 4.3 introduces the upper bound of the number of rails. In the LPM function, the length of vector n is fixed. For example, that for the IPv4 address is 32, while that for the IPv6 address is 128. Therefore, Expr. (1) shows the upper bound of the memory size of the LUT cascade based on the EVMDD (k).

Corollary 4.1 *Assume that p vectors are registered on the LUT cascade based on the EVMDD (k). When p' vectors are updated, then, its memory size is at most $\frac{n}{k} 2^{n/k+1} \lceil log_2(p + p' + 1) \rceil$, where n is the length of the vector.*

(Proof) The upper bound of both the adder rail and the rail is $p + p'1 + 1$ respectively. Thus, the number of outputs for each LUT is at most $2\lceil log_2(p + p' + 1) \rceil$. The number of words for each memory is $2^{n/k}$, and the number of memories on the LUT cascade is $\frac{n}{k}$. Thus, we have $\frac{n}{k} 2^{n/k+1} \lceil log_2(p + p' + 1) \rceil$. (Q.E.D.)

5. EXPERIMENTAL RESULTS

5.1. Comparison with Update Time

We implemented Algorithm 4.2 using the ARM Cortex-A9 MPCore (666 MHz, L1 cache 32KB I/D, L2 cache 512KB) on the Avanet Corp. Zedboard which has a 512 MB DDR3

Table 2. Comparison with other realizations.

	$p = 255$				$p = 511$				$p = 1023$			
Realization	4-LUT	BRAM +LUT	Cascade (MT)	Cascade (EV)	4-LUT	BRAM +LUT	Cascade (MT)	Cascade (EV)	4-LUT	BRAM +LUT	Cascade (MT)	Cascade (EV)
# of 4LUTs	3156	1271	—	—	6383	2806	—	—	13294	6133	—	—
# of Block RAMs	—	32	18	16	—	64	21	18	—	128	32	22
Equivalent # of 4LUTs	3156	7415	3456	3072	6383	15094	4032	3456	13294	30709	6144	4224
Max. Freq. (MHz)	55.1	46.5	188.7	181.1	52.6	42.1	172.9	168.8	50.1	38.4	165.7	152.3
Efficiency (KHz/LUT)	17.4	6.2	54.6	**58.9**	8.2	2.7	42.8	**48.8**	3.7	1.2	26.9	**36.0**

SDRAM. The operating system (OS) was Ubuntu 12.04 LTS. We wrote Algorithm 4.2 by C-language. Then, we generated the execution code by gcc compiler with an optimize option -O3. The size of the execution code was 96.3 KB. Thus, the proposed program and the work area (stack and heap) fit in the available memory. As for the update time of LUT cascades with respect to the number of updates, Fig. 10 compares the EVMDD (k) based one with the MT-MDD (k) based one [12]. Although the update time for the EVMDD (k) based one is longer than that for the MT-MDD (k) based one, it is about a half of the required time for the BGP protocol which requires 10,000 updates per second. Thus, its update time is acceptable.

5.2. Comparison of Area-Performance Efficiency

We assumed that the length of the vector is 32. We implemented the Xilinx Inc. CAM IPs [18] on the Xilinx Inc. FPGA (Spartan III: XC3S256FG). The synthesis tool was Xilinx Inc. ISE Web Pack 7.2i. As for the number of vectors p, Table 2 compares the EVMDD (k) based one with the 4-input LUT based CAM IP (4-LUT), and the block RAM and 4-input LUT based CAM IP (BRAM+LUT). Since the different realization uses different resources, to do fair comparison, we assume that one 4-input LUT corresponds to 96 bits of a BRAM [16]. We used **the equivalent number of 4-input LUTs** as follows:

$$\begin{aligned}\text{Equivalent \# of 4LUTs} &= \text{\# of 4-input LUTs} \\ &\quad + \text{\# of BRAMs} \times 192.\end{aligned}$$

Since the LPM architecture on the router requires high throughput per area, we used **efficiency [kHz/LUT]**, which shows the clock frequency per a 4-input LUT. Table 2 shows that the LUT cascade based on the EVMDD (k) has the highest efficiency.

6. CONCLUSION

This paper showed an update method for a CAM emulator using an LUT cascade based on an EVMDD (k). Since the EVMDD (k) represents the M_1-monotone increasing function, it is suitable for the LPM function, which is used for the router. The experimental result showed that the proposed update method is acceptable for the BGP protocol which requires 100,000 updates per second. Compared with other CAM realizations, the LUT cascade based on the EVMDD (k) has a higher throughput per area.

7. ACKNOWLEDGEMENTS

This research is supported in part by the Grants in Aid for Scientistic Research of JSPS, and the Adaptable and Seamless Technology Transfer Program through target-driven R&D of JST.

8. REFERENCES

[1] The BGP Instability Report: http://bgpupdates.potaroo.net/instability/bgpupd.html

[2] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Compt.*, Vol. C-35, No. 8, 1986, pp. 677-691.

[3] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," *DAC1993*, 1993, pp. 54-60.

[4] W. Jiang and V. K. Prasanna, "Scalable packet classification on FPGA," *IEEE Trans. on VLSI*, Vol. 20, No. 9, 2012, pp. 1668-1680.

[5] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and applications," *Multiple-Valued Logic: An International Journal*, Vol. 4, No. 1-2, 1998, pp. 9-62.

[6] Y-T. Lai and S. Sastry, "Edge-valued binary decision diagrams for multi-level hierarchical verification," *DAC1992*, 1992, pp. 608-613.

[7] H. Le and V. K. Prasanna, "Scalable high throughput and power efficient IP-lookup on FPGA," *FCCM2009*, April, 2009.

[8] S. Nagayama and T. Sasao, "Complexities of graph-based representations for elementary functions" *IEEE Trans. on Comput.*, Vol. 58. No. 1, Jan. 2009, pp.106-119.

[9] S. Nagayama and T. Sasao, "Representations of elementary functions using edge-valued MDDs," *ISMVL2007*, 2007.

[10] S. Nagayama, T. Sasao, and J. T. Butler, "Design method for numerical function generators using recursive segmentation and EVBDDs," *IEICE Trans. on Fund.*, Vol. E90-A, No. 12, 2007, pp. 2752-2761.

[11] H. Nakahara, T. Sasao, and M. Matsuura, "A packet classifier using LUT cascades based on EVMDDs (k)," *FPL 2013*, 2013, pp. 1-6.

[12] H. Nakahara, T. Sasao and M. Matsuura, "A CAM emulator using look-up table cascades," *RAW2007*, CD-ROM RAW-9-paper-2.

[13] T. Sasao, *Memory-Based Logic Synthesis, Springer.*, 2011.

[14] T. Sasao and J. T. Butler, "Implementation of multiple-valued CAM functions by LUT cascades," *ISMVL2006*, 2006.

[15] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *IWLS2001*, 2001, pp. 225-230.

[16] T. Sproull, G. Brebner, and C. Neely, "Mutable codesign for embedded protocol processing," *FPL2005*, Aug. 24-26, 2005, pp. 51-56.

[17] R. Tucker, "Optical packet-switched WDM networks: a cost and energy perspective," *OFC/NFOEC2008*, 2008.

[18] Xilinx Inc., "Content-Addressable Memory," *Datasheet 253*, pp. 1-13.