A PACKET CLASSIFIER USING LUT CASCADES BASED ON EVMDDS (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

This paper presents a packet classifier using multiple LUT cascades based on edge-valued multi-valued decision diagrams (EVMDDs (k)). First, a set of rules for a packet classifier is partitioned into groups. Second, they are decomposed into field functions and Cartesian product functions. Third, they are represented by EVMDDs (k), and finally, they are converted to LUT cascades using adders. We implemented the proposed circuit on a Virtex 7 VC707 evaluation board. The system throughput is 345.60 Gbps for minimum packet size (40 Bytes). As for the normalized throughput (efficiency), the proposed one is 7.14 times better than existing FPGA implementations.

1. INTRODUCTION

1.1. Demands of Packet Classifier

A packet classification [20] is a key technology in routers and firewalls. A packet header includes a protocol number, a source address, a destination address, and a port number. The packet classifier performs a predefined action for a corresponding rule. Applications of the packet classifier include a firewall (FW), an access control list (ACL), and an IP chain for an IP masquerading technique.

With the rapid increase of traffic, core routers dissipate the major part the total network power [22]. Thus, we cannot use ternary content addressable memories (TCAMs), since they dissipate too much power. In addition, a reconfigurable architecture is necessary to update policy rules of the packet classifier. With the rapid growth of the Internet, packet classifiers have become the bottleneck in the network traffic management. Recently, a core router works at a 100 Gbps link speed for a minimum packet size (40 bytes). Thus, a parallel processing is an effective method to increase the system throughput. In this case, the throughput per area-efficiency is an important measure [12]. A modern FPGA consists of lookup-tables (Slices), on-chip memories (BRAMs), arithmetic circuits (DSP48Es), and so on. A balanced usage of hardware resources in FPGAs is the key to realize a high throughput per area. This paper considers a memory-based architecture on the FPGA, which dissipate lower power than TCAMs.

1.2. Contributions of the Paper

This paper proposes an architecture using multiple look-up-table (LUT) cascades based on edge-valued multi-valued decision diagrams (EVMDDs (k)). Our contributions are as follows:

1 We proposed a compact and high-speed packet classifier by LUT cascades based on EVMDDs (k). Conventional methods use only Slices and BRAMs, while the proposed architecture uses DSP48E blocks in addition to Slices and BRAMs. Thus, the proposed architecture uses available FPGA resources effectively.

2 We implemented packet classifier using multiple LUT cascades on an FPGA. Its system throughput is more than 300 Gbps. As for the efficiency measure (throughput per normalized memory size), the proposed architecture is higher than existing methods.

The rest of the paper is organized as follows: Chapter 2 introduces a packet classifier; Chapter 3 shows the LUT cascade based on an MTMDD (k); Chapter 4 shows the LUT cascade based on an EVMDD (k); Chapter 5 shows experimental results; and Chapter 6 concludes the paper.

2. PACKET CLASSIFIER

2.1. 5-tuple Packet Classification

A packet classification table consists of a set of **rules**. Each rule has five input **fields**: Source address (SA), destination address (DA), source port (SP), destination port (DP), and protocol number (PRT). Also, it generates a **rule number** (Rule). A field has **entries**. In this paper, since we consider a realization of the packet classifier for the Internet protocol version 4 (IPv4), we assume that SA and DA have 32 bits, DP and SP have 16 bits, and PRT has 8 bits. An entry for SA or DA is specified by an IP address; that for SP or DP is specified by **an interval [x,y]**, where x and y denote a port number; and that for PRT is specified by a protocol number. SA and DA are detected by **a longest prefix match**; SP and DP are detected by **a range match**; and PRT is detected by **an exact match**. **A packet classifier** detects matched rules using the packet classification table. In this paper, we assume that the rule with the largest number has the highest **priority**. Note that, any packet matches **a default rule** whose rule number is zero. Obviously, the default rule has the lowest priority. When two or more rules are matched, a rule having the highest priority is selected.

Example 2.1 *Table 1 shows an example of the packet classification table, where an asterisk '*' in an entry matches both 0 and 1, while a dash '-' in a field matches any pattern. In Table 1, each field has four bits, rather than the actual number of bits to simplify the example.*

Consider the packet classification table shown in Table 1. The packet header with $SA = 0000$, $DA = 1010$, $SP = 8$, $DP = 8$, *and* $PRT = TCP$ *matches rule 3,*

978-1-4799-0004-6/13/$31.00

Table 1. An example of a packet classification table.

in					out
SA	DA	SP	DP	PRT	Rule
1000	110*	[1,8]	[8,9]	ICMP	4
00**	1***	[2,9]	[6,8]	TCP	3
010*	0010	[8,15]	[7,14]	UDP	2
0***	10**	[8,9]	[4,11]	TCP	1
****	****	[0,15]	[0,15]	-	0 (default)

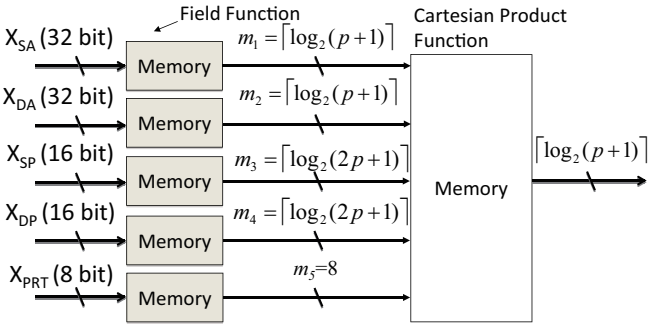

Fig. 1. Decomposition of packet classification table by Cartesian product method.

rule 1, and the default rule. Since the rule 3 has the highest priority, the rule 3 is selected. ■

2.2. Decomposition of Packet Classification Table by Cartesian Product Method

Let p be the number of rules. Since $|X_{SA}| = |X_{DA}| = 32$, $|X_{SP}| = |X_{DP}| = 16$, and $|X_{PRT}| = 8$, the direct memory realization requires $2^{104}\lceil log_2(p+1)\rceil$ bits, which is too large to implement. We use **Cartesian product method** [19] which decomposes the packet classification table into field functions and a Cartesian product function [1].

An entry of a rule can be represented by **an interval**

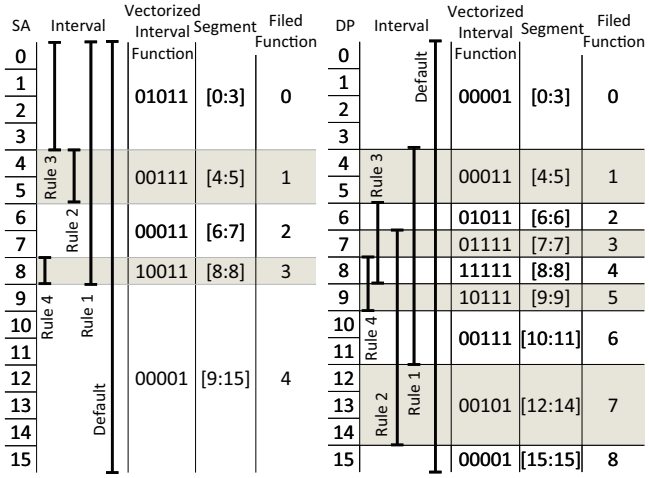

Fig. 2. Examples of segment.

[1] In [19], Cartesian product was called "cross product".

Field Functions

SA	IDX_{SA}
[0:3]	0
[4:5]	1
[6:7]	2
[8:8]	3
[9:15]	4

DA	IDX_{DA}
[0:1]	0
[2:2]	1
[3:7]	2
[8:11]	3
[12:13]	4
[14:15]	5

SP	IDX_{SP}
[0:0]	0
[1:1]	1
[2:7]	2
[8:8]	3
[9:9]	4
[10:15]	5

DP	IDX_{DP}
[0:3]	0
[4:5]	1
[6:6]	2
[7:7]	3
[8:8]	4
[9:9]	5
[10:11]	6
[12:14]	7
[15:15]	8

PRT	IDX_{PRT}
[0:0]	0
[1:1]	1
[2:2]	2
[1:1]	1

Cartesian Product Function

IDX_{SA}	IDX_{DA}	IDX_{SP}	IDX_{DP}	IDX_{PRT}	Rule
3	4	0	4	0	4
3	4	0	5	0	4
0	3	2	2	1	3
⋮	⋮	⋮	⋮	⋮	⋮

Fig. 3. An example of Cartesian product method.

function [16]:

$$IN(X : A, B) = \begin{cases} 1 & (A \le X \le B) \\ 0 & (otherwise) \end{cases} \quad (1)$$

where X, A, and B are integers. Let $x_i \in \{0, 1\}$, $y_i = *$, $\vec{v} = (x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m)$, and $A = \sum_{i=1}^{n} x_i 2^{i-1}$. Any entry for SA is represented by $IN(X_{SA} : A2^m, (A+1)2^m - 1)$. Similarly, any entry for DA can be represented by an interval function. Any entry for PRT is represented by $IN(X_{PRT} : b, b)$, where b is a protocol number.

As shown in Example 2.1, multiple rules may match in a packet classification table. In such a case, we use **a vectorized interval function**. Let r be the number of rules. A vectorized interval function is $\vec{H}(X) = \bigvee_{i=1}^{r} \vec{e}_i IN(X : A_i, B_i)$, where $\vec{e}_i$ is a unit vector with r elements, and only i-th bit is one and other bits are zeros.

For each value of $\vec{H}(X)$, we assign **a segment**, which is an interval or a set of intervals. Then, we define **a field function** $F(X)$, which generates an unique integer index I_i corresponding to the i-th segment $[C_i, D_i]$ satisfying $C_i \le X \le D_i$. Note that, to distinguish from an interval, we denote a segment consisting of an interval $[C, D]$ as $[C : D]$. Next, we define **Cartesian product function** $G : Y \to Z$, where $Y = I_1 \times I_2 \times \cdots \times I_k$ is a set of Cartesian products of indices generated by field functions. As shown in Fig. 1, the packet classification table is decomposed into field functions and a Cartesian product function.

We can assign an arbitrary index to a segment. In this paper, we assign indices to make an M_1**-monotone increasing function** [8] to reduce the amount of memory. Let I be a set of integers including 0. An integer function $f\ (X) : I \to Z$ such that $0 \le f\ (X+1) - f\ (X) \le 1$ and $f\ (0) = 0$ is **an** M_1**-monotone increasing function** on I.

Example 2.2 *Fig. 2 shows examples of segments for SA and DP shown in Table 1. Note that, rules are represented by intervals.* ■

3. LUT CASCADE BASED ON MTMDD (K)

3.1. Cascade Realization of an M_1-monotone Increasing Function

A **binary decision diagram (BDD)** [1] is obtained by applying **Shannon expansions** repeatedly to a logic function

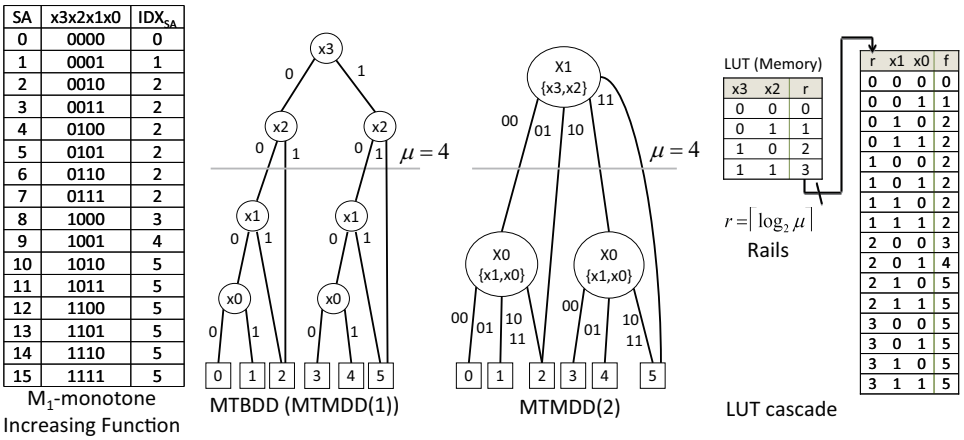

Fig. 4. An LUT cascade based on an MTMDD (k).

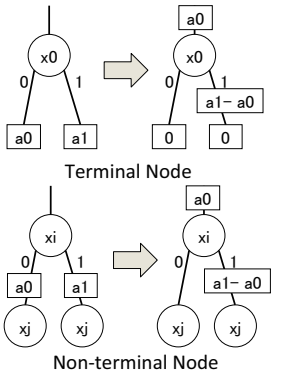

Fig. 5. Conversion of an MTBDD node into an EVBDD node.

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. **A multi-terminal BDD (MTBDD)** [2] is an extension of a BDD and represents an integer-valued function. In the MTBDD, the terminal nodes are labeled by integers.

Let $X = (X_1, X_2, \ldots, X_u)$ be a partition of the input variables, and $|X_i|$ be the number of inputs in X_i. X_i is called **a super variable**. When the 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))** [4]. Note that, an MTMDD(1) means an MTBDD. **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, 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** [17] shown in Fig. 6. Connections between LUT_i 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 $M = \sum_{i=0}^{u} r_i \cdot 2^{(k+r_{i+1})}$. The number of unique indices for the M_1-monotone increasing function is equal to the number of segments. A reduction of r_i reduces the amount of memory for an LUT cascade. Thus, to reduce the amount of memory for the LUT cascade, we partition rules into subrules which increases the least number of segments.

Example 3.3 *Fig. 4 converts the field function for SP shown in Fig. 3 into the LUT cascade. First, the given function is converted to the MTBDD. Then, it is converted to the MTMDD (k). Next, by realizing each index on the MTMDD (k) of the LUT, we have the LUT cascade. In this example, the amount of memory for the LUT cascade is* $2^2 \times 2 + 2^4 \times 3 = 54$ *bits.* ■

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

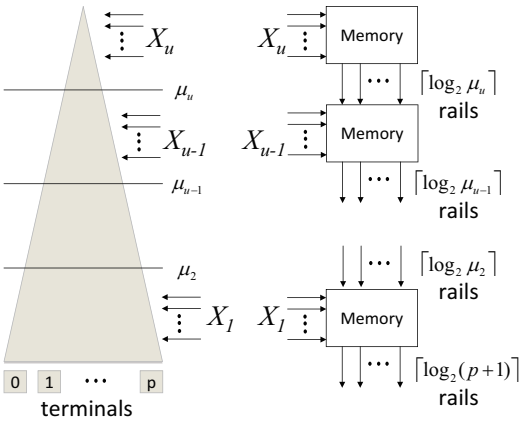

Fig. 6. Conversion of an LUT cascade from an MTMDD (k).

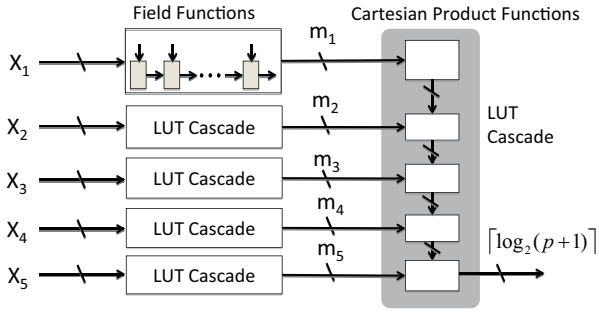

Fig. 7. Packet Classifier by LUT cascades based on an MTMDD (k).

Theorem 3.1 [2] *[15] 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. Partition of Rules by Greedy Algorithm

Since a field function produces at most $2p+1$ segments, it is compactly realized by an LUT cascade [10]. However, the

[2] In [15], the M_1-monotone increasing function is called segment index encoder function.

Cartesian product function produces $O(p^5)$ segments [19]. Thus, a direct realization by an LUT cascade is hard. To reduce the number of segments, we partition rules into **subrules**. Then, we realize subrules by circuits shown in Fig. 7. Since two or more rules may match at the same time, we attached the maximum selector to the output.

Let $[x, y]$ be an entry for a field. Then, $y-x$ is the **size of the interval**. We propose the greedy algorithm to partition rules as follows:

Algorithm 3.1 *(Partition of rules) Let $R = \{r_1, r_2, \ldots, r_p\}$ be the set of rules, p be the number of rules, $\mathcal{G} = \{G_1, G_2, \ldots, G_q\}$ be the partition of rules, and q be the number of groups of rules.*

1. *Compute the sum of sizes of intervals d for each rule. Then, sort the rules in decreasing order as $R' = (r'_1, r'_2, \ldots, r'_p)$.*
2. *$q \leftarrow 1$, $i \leftarrow 1$.*
3. *$G_q \leftarrow r'_i$.*
4. *Do Steps 4.1 to 4.4 until $i > p$.*
   - 4.1. *For $1 \leq j \leq q$, decompose $G_j \cup r'_i$ by Cartesian product method, then generate LUT cascades. And, obtain the amount of memory M_{grp} for the LUT cascades.*
   - 4.2. *Decompose r'_i by Cartesian product method, then generate LUT cascades. And, obtain the amount of memory M_{single} for the LUT cascades.*
   - 4.3. *If $M_{grp} < M_{single}$, then $G_j \leftarrow G_j \cup r'_i$. Otherwise, $G_{q+1} \leftarrow r'_i$, and $q \leftarrow q + 1$.*
   - 4.4. *$i \leftarrow i + 1$.*
5. *Terminate.*

Algorithm 3.1 partitions the packet classification table efficiently using its property. Real-life packet classification tables in an inherent data structure are analyzed in [7]. Since many packet classification tables are maintained by humans, global controls (wide range port) are used in the global networks, while detail controls (narrow range port) are used in the local networks. Thus, in practice, the number of rails seldom becomes the worst. A simple partition algorithm can suppress the increase of segments. As a result, we can reduce the memory size.

4. 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))** [6], which is an extension of an EVBDD. 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. 5 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 α edge (Fig. 8 (a)). The MTBDD only shares prefixes, while the EVBDD shares both prefixes and postfixes (Fig. 8 (b)). By rewriting

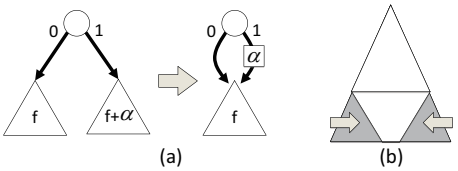

Fig. 8. Principle of reduction of width in an EVBDD.

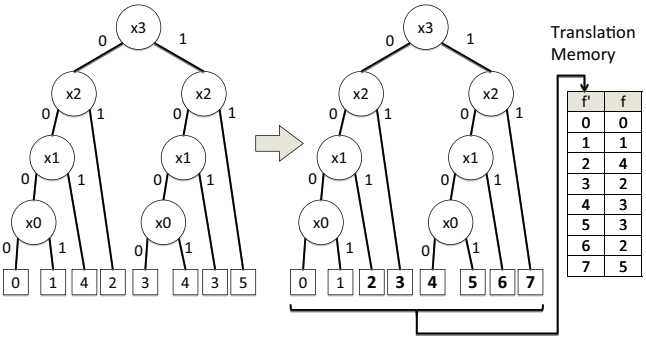

Fig. 9. An example of rewrite to M_1-monotone increasing function.

the terminals of the MTBDD for the Cartesian product function, we have the M_1-monotone increasing function. Fig. 9 shows an example to obtain an M_1-monotone increasing function. To recover the original function, we use **a translation memory**. The size of the translation memory is equal to the number of terminal nodes in the MTBDD. Experimental results shows that its amount memory tends to be small.

An edge-valued MDD (k) (EVMDD (k)) 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 [9] shown in Fig. 10. In this case, the rails represent sub-functions in the EVMDD (k). The outputs from each LUT_i other than rails represent the sum of weights of edges. We call such outputs **Arails** which consist of ar_i rails. Since the width of the EVMDD (k) for M_1-monotone increasing function is smaller than that of the MTMDD (s), we can reduce the amount of memory for the LUT cascade by using an EVMDD (k). Since we realize the adders by DSP blocks (DSP48Es), FPGA resources are efficiently used.

The amount of memory for LUT_i is $(r_i + ar_i) \cdot 2^{k+r_{i+1}}$. Let $|X| = n$ be the number of inputs, and $k = |X_i|$. The LUT cascade has $u = \lceil \frac{n}{k} \rceil$ LUTs. Thus, the LUT cascade based on an EVMDD (k) requires $\sum_{i=0}^{u}(r_i + ar_i) \cdot 2^{k+r_{i+1}}$ bit of memory in total. Also, it requires u adders. Generally, an increase of k increases the amount of memory, while decreases the number of adders. Thus, in this paper, we find a value of k that uses FPGA resources efficiently.

Example 4.4 *Fig. 11 shows the EVBDD obtained from the MTBDD shown in Fig. 4. At the first level, the width of the MTBDD is four, while that of the EVBDD is two. First,*

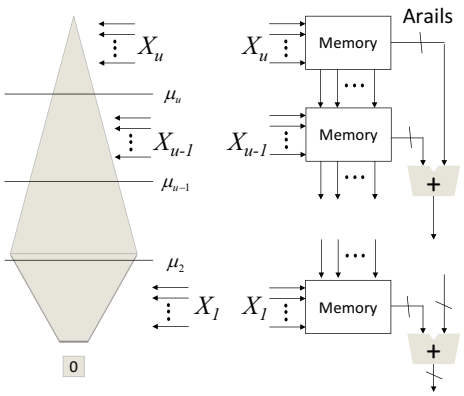

Fig. 10. Conversion of EVMDD (k) into an LUT cascade.

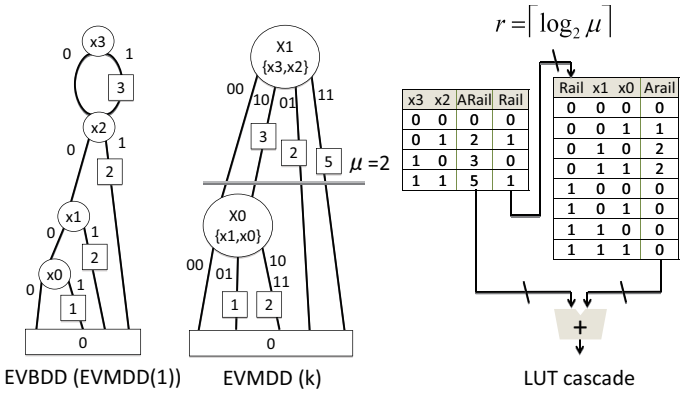

Fig. 11. Conversion of an EVBDD into an LUT cascade.

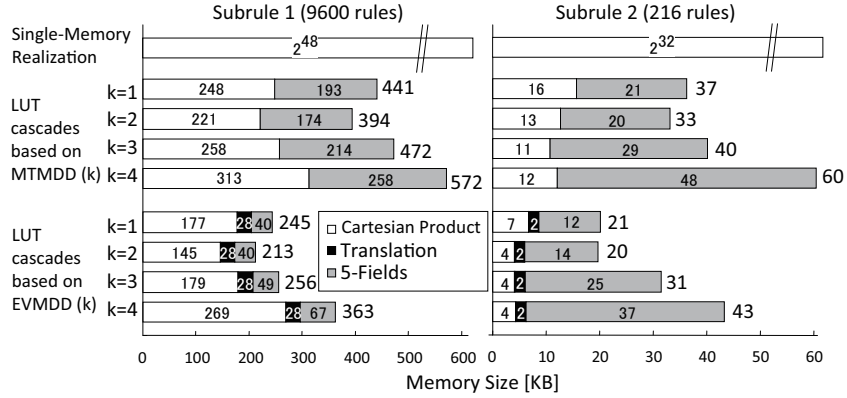

Fig. 12. Comparison of memory sizes [KB].

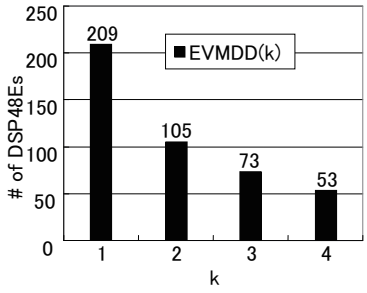

Fig. 13. Number of adders for EVMDD (k).

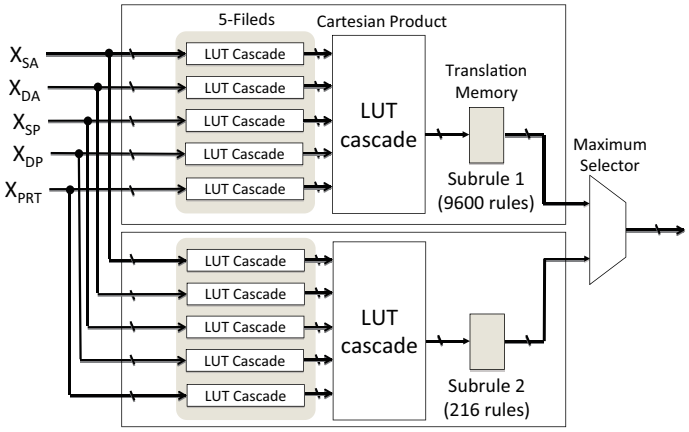

Fig. 14. Overall architecture.

convert the EVBDD to the EVMDD (k). Then, convert it to the LUT cascade. Its memory size is $2^2 \times (3+1) + 2^3 \times 2 = 32$ *bits. A single memory implementation of this function requires* $2^4 \times 3 = 48$ *bits. Thus, the LUT cascade can reduce the total memory size.* ■

5. EXPERIMENTAL RESULTS

5.1. Implementation Setup

We implemented the proposed circuit on the Virtex 7 VC707 evaluation board (FPGA: Xilinx, XC7VX485T-2FFG, 75,900 Slices, 1,030 36KbBRAMs, and 2,800 DSP48E Blocks). We used the Xilinx PlanAhead version 14.4 for the synthesis. As for the LUT cascade implementation, LUT_i whose size is equal to or more than 36Kb LUT_i is implemented by 36Kb BRAMs, while LUT_i whose size is less than 36Kb LUT_i is implemented by distributed RAMs using Slices. To increase the system throughput, we set the dual port mode to the memory. By Algorithm 3.1, we partitioned 9,816 ACL rules generated by ClassBench [21] into two: Subrule 1 (9,600 rules) and Subrule 2 (216 rules). Then, each subrule is decomposed into five field functions and a Cartesian product function. Finally, each function is realized by an LUT cascade based on an EVMDD (k). To reduce the widths for an EVMDD (k), we used the shifting method [14].

5.2. Comparison of EVMDD (k) with MTMDD (k) to Implement LUT Cascade

We realized the packet classifier by three different methods:

1 A single memory.
2 LUT cascades based on MTMDDs (k).
3 LUT cascades based on an EVMDDs (k).

To find the smallest LUT cascade, we changed the size of super variables k from one to four. Fig. 12 compares the memory sizes. It shows that, for all k, EVMDDs (k) produced smaller LUT cascades than MTMDDs (k). Also, the memory size takes its minimum when $k = 2$. As for Cartesian product functions, EVMDDs (k) required smaller memory than MTMDDs (k) even if the translation memories are

Table 2. Comparison with other methods.

Architecture	#Rules	Memory [KB]	Memory [B] /#Rule	Throughput [Gbps]	Efficiency [Gbps·Rules/KB]
BV-TCAM (FPGA 2005) [18]	222	16	73.80	10.00	138.7
Memory-based DCFL (FCCM 2008) [3]	128	221	1768.00	24.00	13.9
2sBFCE (FCCM 2008) [11]	4,000	178	45.56	2.06	46.3
Simplified Hyper Cuts (ANCS 2008) [5]	10,000	286	29.28	10.84	379.0
Cartesian-Product with Quadtrees (ASAP 2009) [13]	9,603	432	46.05	91.73	2039.1
Optimized Hyper Cuts (IEEE Trans. VLSI2012) [12]	9,603	612	65.25	80.23	1258.9
Multiple LUT cascades (Proposed)	9,816	233	24.30	345.60	14559.6

used. Fig. 13 shows the number of adders for EVMDD (k). Although EVMDD (k) requires DSP48Es, it requires less than 3.8% of available resources. Thus, the usage of DSP48Es is negligible. As shown in this part, the LUT cascade based on EVMDDs (k) efficiently utilize the resource of an FPGA.

5.3. Comparison with Other Methods

According to the result of Section 5.2, we implemented the packet classifier by the LUT cascades based on an EVMDD (k) shown in Fig. 14, which consumes 2,024 Slices (6.7%), 37 BRAMs (3.6%), and 105 DSP48E blocks (3.8%). Since the maximum clock frequency was 543.774 MHz, we set the system clock frequency to 540 MHz. Thus, the system throughput is 0.54 (MHz) $\times 2$ (ports)$\times 320$ (Bits) $= 345.60$ Gbps for minimum packet size (40 Bytes).

Table 2 compares the proposed method with other methods. Since different methods use different numbers of rules of ClassBench, we use **the efficiency measure** [12] by $\frac{Throughput\ [Gbps]}{normalizedarea\ (Memory\ [B]/\#Rules)}$ to compare them. The proposed architecture implemented 9,816 rules by 233 [KB] memory, and its system throughput was 345.60 [Gbps]. Thus, the efficiency measure is 14559.6 [Gbps·rules/KB]. From Table 2, the efficiency measure of the proposed architecture is 7.14 times higher than that of Cartesian-Product with Quadtrees method [13] that was the best among the existing methods. In this way, we implemented a high-speed and area efficient system.

6. CONCLUSION

In this paper, we showed a method to implement a packet classifier. First, the packet classification table was decomposed into two subrules. Second, they were decomposed into five field functions and a Cartesian product function. And finally, each function was realized by an LUT cascade based on an EVMDD (2). We implemented the proposed architecture on a Virtex 7 VC707 evaluation board. Experimental result showed that, the efficiency measure (throughput per normalized area) is 7.14 times higher than that of an existing method.

The rules for the packet classifier should be updated (added and deleted) frequently. The addition and deletion of a registered vector can be done in time that is proportional to the number of cells in the LUT cascade [10]. One of a future project is applying this update method in the proposed architecture.

7. ACKNOWLEDGMENTS

This research is supported in part by the Grants in Aid for Scientistic Research of JSPS. Reviewer's comments were useful to improve the paper.

8. REFERENCES

[1] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, 1986, pp. 677-691.

[2] 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.

[3] G. S. Jedhe, A. Ramamoorthy, and K. Varghese, "A scalable high throughput firewall in FPGA," *FCCM2008*, 2008, pp. 43-52.

[4] 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.

[5] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Low power architecture for high speed packet classification," *ANCS2008*, 2008, pp. 131-140.

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

[7] L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: From theory to practice," *INFOCOM2009*, 2009, pp. 648-656.

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

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

[10] H. Nakahara, T. Sasao and M. Matsuura, "A CAM emulator using look-up table cascades," *RAW2007*, 2007, CD-ROM-RAW, paper-2.

[11] A. Nikitakis and I. Papaefstathiou, "A memory-efficient FPGA-based classification engine," *FCCM2008*, 2008, pp. 53-62.

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

[13] W. Jiang and V. K. Prasanna, "A FPGA-based parallel architecture for scalable high-speed packet classification," *ASAP2009*, 2009, pp. 24-31.

[14] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *ICCAD1993*, pp. 42-47, 1993.

[15] T. Sasao, *Memory-based logic synthesis, Springer*, 2011.

[16] T. Sasao, "On the complexity of classification functions," *ISMVL2008*, 2008.

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

[18] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," *FPGA2005*, 2005, pp. 238-245.

[19] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," *SIGCOM1998*, 1998, pp. 191-202.

[20] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Curveys,* Vol. 37, No. 3, 2005, pp. 238-275.

[21] D. E. Taylor and J. S. Turner, "ClassBench: a packet classification benchmark," *INFOCOM2005*, 2005, Vol. 3, pp. 2068-2079.

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