LUT Cascades Based on Edge-Valued Multi-Valued Decision Diagrams: Application to Packet Classification

Hiroki Nakahara, *Member, IEEE*, Tsutomu Sasao, *Life Fellow, IEEE*, Hisashi Iwamoto, and Munehiro Matsuura

***Abstract*—This paper presents a packet classifier using multiple LUT cascades for edge-valued multi-valued decision diagrams (EVMDDs (k)). Since the proposed one uses both DSP blocks and on-chip memories, it can efficiently use the available FPGA resources. Thus, it can realize a parallel packet classifier on a single-chip FPGA for the next generation 400 Gb/s Internet link rate (IEEE 802.3). Since it is a memory-based one, the power consumption is lower than the TCAM-based one. Also, we proposed an on-line update method that can be done without intermitting the packet classification. Compared with the conventional off-line update which requires resynthesis of the re-generated HDL codes, it drastically reduces the update time. Although the proposed on-line update requires additional hardware, the overhead is only 8.5% of the original LUT cascades, which is acceptable. We implemented a two-parallel packet classifier on a Virtex 7 VC707 evaluation board. The system throughput is 640 Gb/s for minimum packet size (40 Bytes). For the performance per memory, the proposed architecture is 2.21 times higher than existing methods. For the power consumption per performance, the proposed architecture is 11.95 times lower than existing methods.**

***Index Terms*—Edge-valued decision diagram, field-programmable gate array (FPGA), multi-valued decision diagram, packet classification, LUT cascade.**

I. Introduction

A. Packet Classification

With the rapid growth of the Internet, various network applications [e.g., firewall, quality of service (QoS), virtual private networks, and network address translation (NAT)] have been developed. They distinguish incoming packets using the multiple fields of packet headers. Such a function is called **multi-field packet classification**. Traditionally, a packet classification problem considers the fixed 5-tuple fields consisting of 32-bit source/destination IP addresses, 16-bit source/destination port numbers, and an 8-bit transport layer protocol. Recently, multi-match packet classification for the network intrusion detection system (NIDS) [68] and 12-tuple packet classification for the OpenFlow [43] have been considered. Since packet classification spends a considerable fraction of the total computation time for these applications, a dedicated hardware is necessary. Although packet classification hardware has been widely studied for many years, we must develop novel and efficient packet classification [69] due to continuous growth of network bandwidth; ever-increasing complexity of network applications; and technology innovations of network systems.

Let N be the number of the rules and k be the number of the fields. The software-based realization requires either $O(N^k)$ space and $O(\log N)$ time, or $O(N)$ space and $O(\log^{k-1} N)$ time [44]. Thus, the software-based realization is too slow or consumes too much memory. The core router employs dedicated hardware for packet classification. Hardware realizations of packet classification are roughly divided into two: A ternary content addressable memory (TCAM) based one, and a field programmable gate array (FPGA) based one. Since the core routers consume the major part of the total network power dissipation [61], we cannot use the TCAM-based architecture that dissipates too much power [48]. Also, unlike static random access memories (SRAMs), TCAMs are not scalable with respect to the clock frequency [18]. Thus, the memory-based IP lookup architectures on the FPGA have been proposed. They dissipate lower power than the TCAM-based ones [20]. Since the state-of-the-art SRAM-based FPGA devices can utilize the leading-edge large scale integration (LSI) process, it achieves a high clock rate, a low-power dissipation and a large amount of on-chip memory. Since the direct implementation of the TCAM on the FPGA consumes too much hardware [40], hash-based and decision diagram based realizations are major for the FPGA-based packet classifier. Most of the existing hash-based realizations employ the bloom filter due to $O(1)$ time complexity with small memory size. However, it suffers from the potential collision that requires additional module to provide deterministic performance which reduces the system throughput [16]. Therefore, in the paper, we employ a decision diagram based architecture.

Rules for the packet classifier are frequently updated. For example, the SNORT, which is a kind of the NIDS, recommends

Manuscript received January 30, 2015; revised May 27, 2015; accepted July 08, 2015. Date of publication February 23, 2016; date of current version March 09, 2016. This research is supported in part by the Grants in Aid for Scientistic Research of JSPS. This work was presented in part at the 23rd International Conference on Field Programmable Logic and Applications (FPL2013), Porto, Portugal, Sep., 2013. This paper was recommended by Guest Editor N. Homma.

H. Nakahara is with the Department of Electrical and Computer Engineering, Ehime University, Ehime 7900911, Japan (e-mail: nakahara@cs.ehime-u.ac.jp).

T. Sasao is with the Department of Computer Science, Meiji University, Kanagawa 2148571, Japan (e-mail: sasao@cs.meiji.ac.jp).

H. Iwamoto is with the REVSONIC Corp., Osaka 5320011, Japan (e-mail: hisashi-iwamoto@revsonic.com).

M. Matsuura is with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Fukuoka 8208502, Japan (e-mail: matsuura@aries01.cse.kyutech.ac.jp).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2016.2528638

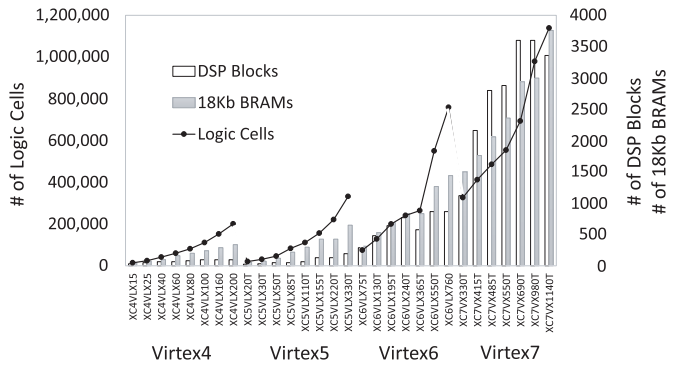

Fig. 1. Distribution of resources for Xilinx Inc. Virtex FPGA devices [66].

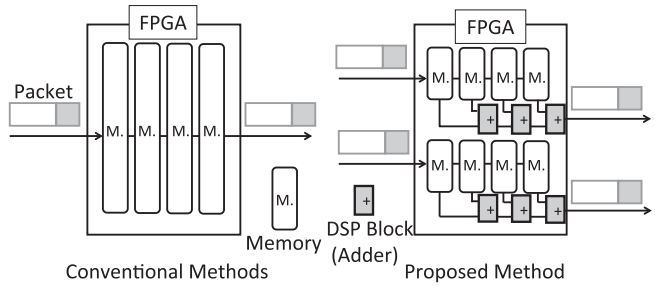

Fig. 2. Comparison of conventional and proposed methods.

to update the rules in every 15 min [42]. Two types of updates exist: the on-line update and the off-line update. In the off-line update, first, the hardware description language (HDL) codes are regenerated. Then they are converted into the bit-stream, and finally, it is downloaded it to the FPGA. The off-line update has two drawbacks. Since the synthesis takes a long time, it cannot keep up with the update interval. Also, the off-line update requires to suspend the packet classifier. In the paper, we proposed an on-line update method. Although it requires the additional hardware, update can be done without suspension of packet classification. The overhead of hardware is small enough.

B. Proposed Method

Since the next generation network transmission requires 400 Gb/s link throughput by IEEE 802.3 working group [14], a high-speed packet classifier is essential. Since the modern high-end FPGA operates at most 600 MHz clock frequency[1] [66], a parallel processing of packet classification is an effective method to keep up with the next generation link throughput. In this case, the throughput per area is an important measure. Fig. 1 shows the resource distribution of the Xilinx Inc. Virtex FPGA devices. It shows that a modern FPGA consists of logic cells, on-chip memories (BRAMs), and arithmetic circuits (DSP blocks), and their distributions are balanced. Thus, a balanced usage of hardware resources in FPGAs is the key to increase parallelization. Since the conventional method represents packet classification by the multi-terminal multi-valued decision diagram (k) (MTMDD (k)), it uses logic cells and BRAMs only. This paper presents a packet classifier using multiple LUT cascades for edge-valued multi-valued decision diagrams (EVMDDs (k)). The classifier is designed as follows: First, a set of rules for a packet classifier is partitioned into groups by using modified HiCuts [11]. Second, each group is decomposed into field functions and a Cartesian product function. Third, they are represented by EVMDDs (k), and finally, they are converted to LUT cascades using adders (DSP blocks). Thus, it can realize highly parallelization (Fig. 2).

In this paper, we proposed an on-line update method for the LUT cascade based on the EVMDD (k). To update a rule without suspension, we appended a small CAM. The CAM is used to store an update rule, while the host PC is rewriting the LUT cascades. After finishing the update, the rule stored in the CAM is cleared. Although our method requires additional hardware, it can update without suspending the operations. The off-line update requires re-synthesizes of the packet classifier. The experimental result shows that, the area overhead and the update time are acceptable for practical packet classifiers.

In our previous contributions [38], we note the following.

1) We proposed a compact and high-speed 5-tuple packet classifier by multiple LUT cascades for EVMDDs (k). Conventional methods used only logic cells and BRAMs, while the proposed architecture uses DSP48E blocks in addition. To best of our knowledge, only our architecture utilizes all the FPGA resources efficiently.
2) We found the smallest LUT cascade by changing the size of super variables k from one to four. The experimental results showed that the memory size takes its minimum when $k = 2$.

The following contributions in this paper are new.

1) We derived an the upper bound on the memory size of the proposed architecture for a given packet classification table.
2) We implemented the two-parallel packet classifier using multiple LUT cascades on an FPGA. Its system throughput achieved 640 Gb/s, which exceeds the next generation link speed. For the performance per memory, the proposed architecture is 2.21 times higher than existing FPGA realizations. Since this technique can be used for the 12-tuple classification and the multi-match classification problems, it will accelerate the existing decision diagram based hardware.
3) We proposed an on-line update method for the LUT cascades. It used an additional small CAM and the host PC. Compared with the off-line update that requires re-generation of the HDL codes, it can quickly update the cascades without suspending the operation. For different numbers of update rules, we compared the proposed on-line update method with the off-line one with respect to the area overhead and the update time. Experimental results showed that the amount of additional hardware is only 8.5% of the LUT cascade. Note that, the proposed on-line update requires no suspension of the operations.
4) We achieved the lowest power consumption for the LUT cascade by changing k. The experimental results shows that the power consumption takes its minimum when $k = 2$. Its power consumption is lower than the TCAM-

[1]The BRAM operates up to 601 MHz and the DSP48E block operates up to 741 MHz for the Virtex 7 FPGA with -3 speed grade. However, these values are hard to achieve in real designs.

based ones. For the power consumption per performance, the proposed architecture is 11.95 times lower than existing FPGA realizations.

C. Organization of the Paper

The rest of the paper is organized as follows.Section II surveys related works. Section III defines a 5-tuple packet classification problem. Section IV shows the LUT cascade for an MTMDD (k). Section V reviews the LUT cascade for an EVMDD (k). Section VI analyzes the upper bound of the memory size. Section VII proposes an update method for LUT cascades based on the EVMDD (k). Section VIII shows experimental results. Section IX concludes the paper. This paper is an extension version of the previous publication [38].

II. Related Work

Comprehensive surveys have been done in [13] and [59]. Due to the page limitation, we only introduce leading researches.

As for the packet classification rule set, theoretical and practical complexities for the real-life rules have been analyzed in [69], the access control list (ACL) has been analyzed in [4], and the fire wall has been analyzed in [9]. The synthesizer for the packet classification table (ClassBench [60]) is widely used in the experiment.

Among all existing realizations, we categorize them by 1) data structure and algorithm; and 2) implementation technique. As for data structure and algorithm, they are roughly divided into hashing schemes, rule partition schemes, rule compression schemes, trie schemes, and decision diagram schemes. Most of existing works combine these schemes. The works [5], [41], [45], [22], [68] proposed hash-based architecture to match a packet to its possible matching rules. The hash tables for multi-core processor have been proposed [70]. The works [13], [57] represented the packet classification table by tries. The outstanding partition algorithms are HSM [71], HiCuts [10], HyperCuts [55], and EffiCuts [62]. Other existing works extending above partition algorithms are [28], [46], [65]. In this paper, we partition the given packet classification table by a variation of HiCuts [11]. Compression techniques [31], [64] are based on the specialization of existing logic minimizer, such as Quine-McClusky method [47] or Espresso [30]. As for decision diagrams, a binary decision diagram (BDD) [1], [33] was extended to a multi-valued decision diagram (MDD) [21]. An edge-valued BDD (EVBDD) [25], [26] was used to compactly represent the monotone increasing logic function. The combination of the MDD and the EVBDD was proposed as an edge-valued multi-valued decision diagram (EVMDD) [34]. Since the memory size of the decision diagram depends on its variable ordering, the optimization algorithm for the variable order has been proposed [49]. Representations for the packet classification table by the decision diagram have been proposed [12], [55], [62]. Sasao *et al.* implemented the decision diagram by the cascade of the memory (**an LUT cascade**) [51], [54]. We implemented the dedicated processor based on the EVMDD for a small network [39]. Also, we analyzed the complexity of the decision diagram for the given packet classification table [53], [37].

As for the implementation technique, they are roughly divided into three: a software-based one, an application specific integrated circuit (ASIC) one, a TCAM-based one, and an FPGA-based one. Many sophisticated software-based implementations have been proposed, and its comprehensive survey was shown in [63]. ASIC implementations have been proposed for low-power packet classification [11], [23]. Existing works for optimization of the TCAM-based approach are divided into three categories; power reduction [72]; circuit modification [27]; and TCAM memory compression [32]. Recently, to achieve low-power and high-throughput, many researchers have developed various FPGA-based architectures: a combination of TCAM and the bit vector (BV) [56]; the modified HiCut and HyperCut with a dynamically clock changing [24]; an extension to distributed crossproducting of field labels (DCFLs) [15]; a dual-stage bloom filter classification engine (2sBFCE) [41]; parallel realization of coarse-grained independent rules and the cross-producting method [18]; the field-split parallel bit vector (FSBV) [17]; a combination of the cutting-based and the merging-based architecture [19]; a bit vector based lookup scheme and a parallel hardware architecture (StrideBV) [7]; a range-point conversion rule partitioning (ParaSplit) [6]; a combination of multi-bit tries including the expand trie and the tree bitmap trie to minimize the power consumption [20]; an improvement of the memory realization for the StrideBV [50]; and a combination of the StrideBV and modularized BV [8]. In this paper, we also employ the FPGA-based architecture.

III. 5-Tuple Packet Classification

A. Problem Statement

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, the rule having the highest priority is selected.

Example 3.1: Table I 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 I, each field has four bits, rather than the actual number of bits to simplify the example.

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

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

B. Decomposition of Packet Classification Table by Cartesian Product Method

Let p be the number of rules. When a prefix match and a range match are decompressed into all the exact match patterns, since $|X_{\mathrm{SA}}| = |X_{\mathrm{DA}}| = 32$, $|X_{\mathrm{SP}}| = |X_{\mathrm{DP}}| = 16$, and $|X_{\mathrm{PRT}}| = 8$, the direct memory realization storing these decompressed patterns requires $2^{104}\lceil \log_2(p+1) \rceil$ bits. It is too large to implement. We use **the Cartesian product method** [58] which decomposes the packet classification table into field functions and a Cartesian product function[2].

An entry of a rule can be represented by **an interval function** [53]:

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

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 $\mathrm{IN}(X_{\mathrm{SA}} : A2^m, (A+1)2^m - 1)$. Similarly, any entry for DA can be represented by an interval function. Any entry for PRT is represented by $\mathrm{IN}(X_{\mathrm{PRT}} : b, b)$, where b is a protocol number.

As shown in Example 3.1, 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 $\vec{H}(X) = \bigvee_{i=1}^{r} \vec{e}_i \mathrm{IN}(X : A_i, B_i)$, where $\vec{e}_i$ is a unit vector with r elements, and only ith 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 ith segment $[C_i, D_i]$ satisfying $C_i \le X \le D_i$. Note that, to distinguish a segment from an interval, we denote a segment consisting of an interval $[C, D]$ as $[C : D]$. Next, we define **the 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. 3, the packet classification table is decomposed into field functions and a Cartesian product function.

We must assign a different index to a different segment. In this paper, we assign indices to make an M_1**-monotone increasing function** [36] to reduce memory size. 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 3.2: Fig. 4 shows examples of segments for SA and DP shown in Table I. Note that, rules are represented by intervals. ■

Example 3.3: Fig. 5 shows decomposition by the Cartesian product method of the packet classification table shown in Table I. As for the PRT field, we assigned "0" to ICMP, "1" to TCP, and "2" to UDP. In this field, for each segment $[C : D]$, since $C = D$, $[0 : 0]$ denotes ICMP, $[1 : 1]$ denotes TCP, and $[2 : 2]$ denotes UDP. Note that, we omit a part of the Cartesian product function due to the space limitations. ■

[2]In [58], Cartesian product was called "cross product".

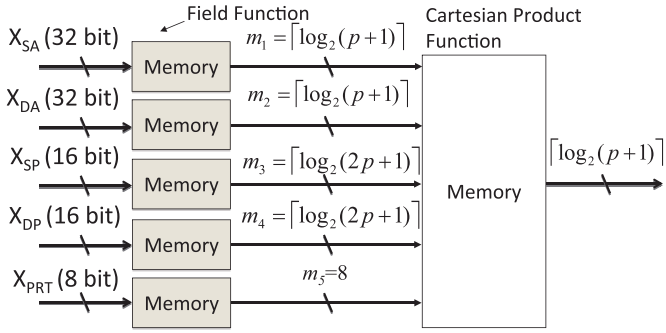

Fig. 3. Decomposition of packet classification table by Cartesian product method.

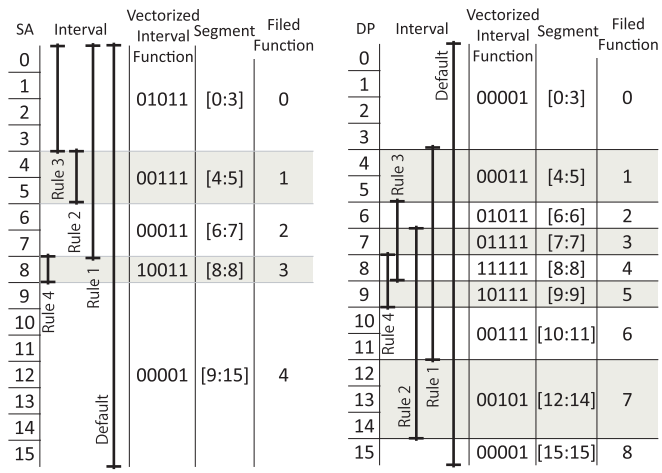

Fig. 4. Relations among rules, vectorized interval function, segments and field function.

TABLE I
EXAMPLE OF A PACKET CLASSIFICATION TABLE

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

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. 5. Example of Cartesian product method.

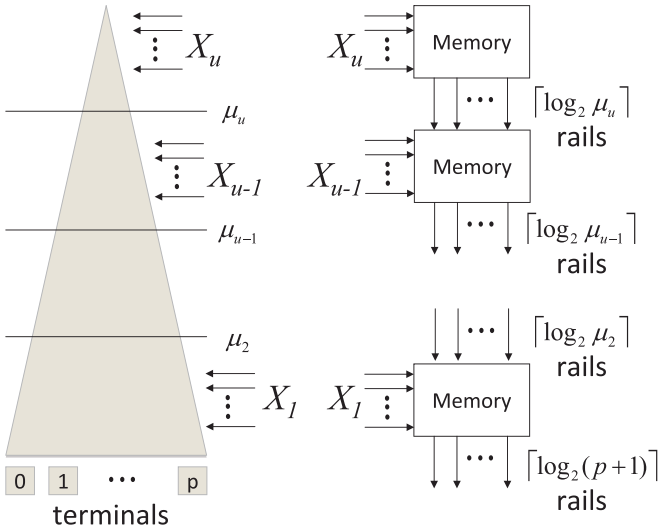

Fig. 6. Conversion of an LUT cascade from an MTMDD (k).

IV. LUT Cascade for MTMDD (k)

A. Cascade Realization of an M_1-Monotone Increasing Function

A **binary decision diagram (BDD)** [1], [33] is obtained by applying **Shannon expansions** repeatedly to a logic function f. Each nonterminal 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)** [3] 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 binary variables 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 nonterminal nodes have 2^k edges. In this case, we have a **multi-valued multi-terminal decision diagram** ($\mathbf{MTMDD}(k)$) [21]. Note that, an MTMDD(1) corresponds to an MTBDD. **The width of the MDD** (k) **at the height** i 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.

An M_1-monotone increasing function can be realized by **an LUT cascade** [51] shown in Fig. 6. Connections between LUT_i and LUT_{i-1} requires $r_i = \lceil \log_2 \mu_i \rceil$ rails. Let $|X| = n$ be the number of inputs, and $k = |X_i|$. The LUT cascade has $u = \lceil (n)/(k) \rceil$ LUTs. 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 for an MTMDD (k) is $r_i \cdot 2^{(k+r_{i+1})}$. Thus, the total amount of memory for an LUT cascade is $M = \sum_{i=1}^{u} r_i \cdot 2^{(k+r_{i+1})}$. The number of unique indices for the M_1-monotone increasing function is equal to the number of segments. A reduction of r_i also reduces the amount of memory for an LUT cascade. To reduce the amount of memory for the LUT cascade, we partition rules into sub rules that take a minimum number of segments.

Example 4.4: Fig. 7 illustrates the process of conversion from the field function for SP shown in Fig. 5 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 $2^2 \times 2 + 2^4 \times 3 = 56$ bits. ■

B. Partition of Rules by Greedy Algorithm

Since a field function for p rules produces at most $2p + 1$ segments (will be shown later), it is compactly realized by an LUT cascade. However, the Cartesian product function produces $O(p^5)$ segments [58]. 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. 8. 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 use the greedy algorithm to partition rules as follows.

Algorithm 4.1: (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 $1 \le j \le q$, decompose $G_j \cup r'_i$ by the 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 the Cartesian product method, then generate LUT cascades. And, obtain the amount of memory M_{single} for the LUT cascades.
   - 4.3. If $M_{\mathrm{grp}} < M_{\mathrm{single}}$, 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 4.1 partitions the packet classification table efficiently using its property. Real-life packet classification tables in an inherent data structure are analyzed in [69]. 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.

V. LUT Cascade for an EVMDD (K)

To farther reduce the total memory size for an LUT cascade, we introduce an LUT cascade for **an edge-valued multi-valued decision diagram** ($\mathbf{EVMDD}$ (k)) [25], which is an extension of an EVBDD. An EVBDD consists of one terminal node representing zero and non-terminal nodes with a weighted

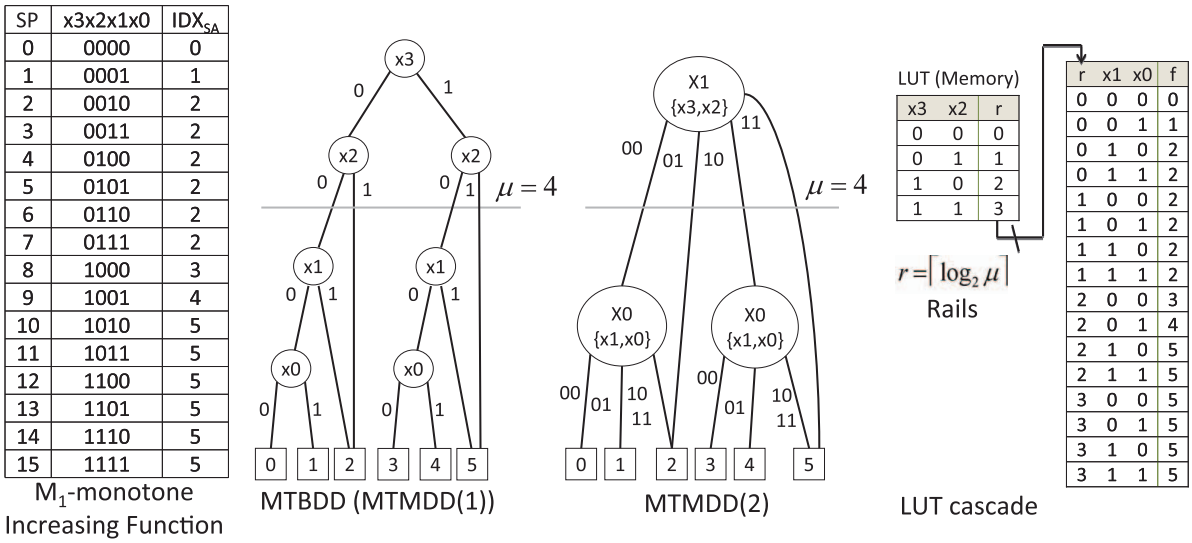

Fig. 7. LUT cascade for an MTMDD (k).

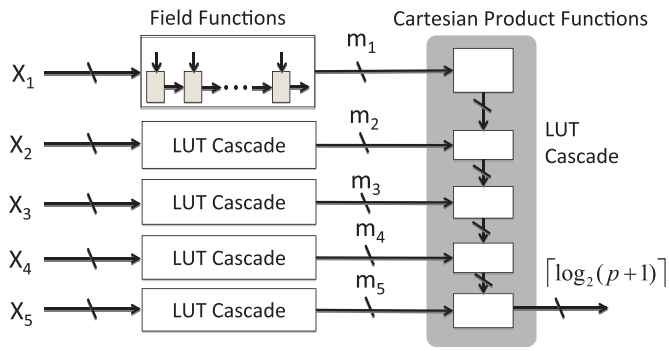

Fig. 8. Packet Classifier by LUT cascades for an MTMDD (k).

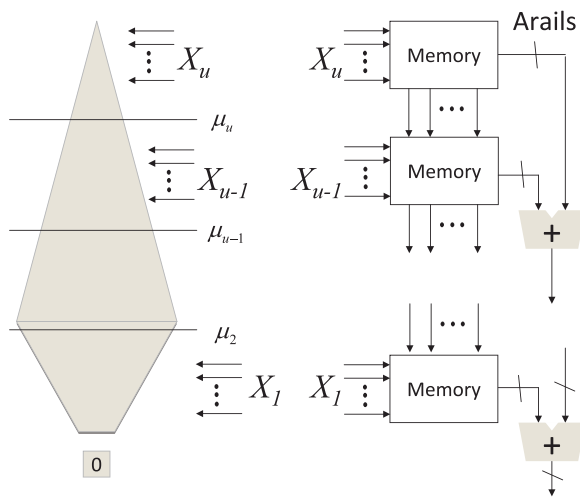

Fig. 9. Conversion of EVMDD (k) into an LUT cascade.

1-edge, where the weight has an integer value α. Note that, in the EVBDD, 0-edges have zero weights.

In an M_1-monotone increasing function, sub-function f' is obtained by adding α to sub-function f. Thus, an EVBDD may have smaller widths by sharing f and f' with α edge [Fig. 11(a)]. The MTBDD can share only prefixes, while the EVBDD can share both prefixes and postfixes [Fig. 11(b)]. By rewriting the terminals of the MTBDD for the Cartesian product function, we have the M_1-monotone increasing function. Fig. 12 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 proportional to the number of terminal nodes in the MTBDD. Experimental results show 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 nonterminal nodes with edges having integer weights, and 0-edges always have zero weights.

An M_1-monotone increasing function is efficiently realized by an LUT cascade with adders [35] as shown in Fig. 9. In this case, the rails represent sub-functions in the EVMDD (k). Each LUT_i produces other rails representing 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 (k), we can reduce the total memory size for the LUT cascade by using an EVMDD (k). Since the adders are realized 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 $u = \lceil (n)/(k) \rceil$ LUTs. Thus, the LUT cascade for an EVMDD (k) requires $\sum_{i=1}^{u} (r_i + ar_i) \cdot 2^{k+r_{i+1}}$ bits 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 k that minimizes the usage of FPGA resources.

Example 5.5: Fig. 10 shows the EVBDD obtained from the MTBDD shown in Fig. 7. At the first level, the width of the MTBDD is four, while that of the EVBDD is two. First, 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.

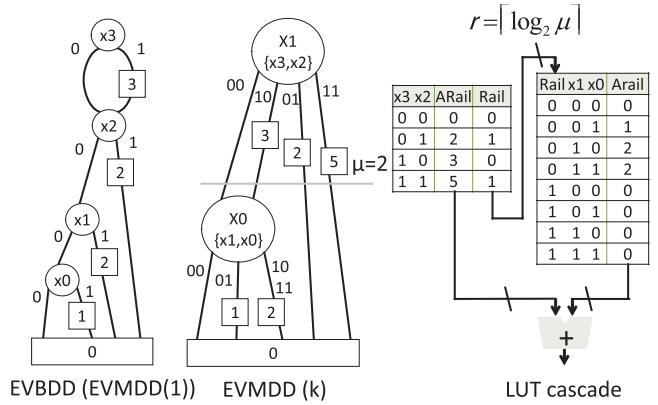

Fig. 10. Conversion of an EVBDD into an LUT cascade.

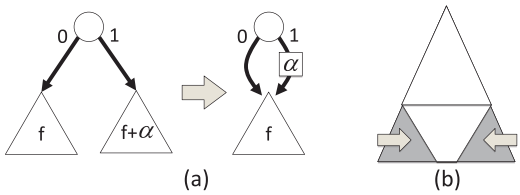

Fig. 11. Principle of reduction of width in an EVBDD.

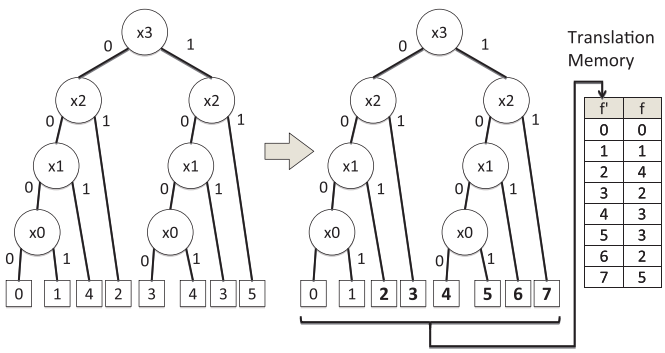

Fig. 12. Example of deriving to M_1-monotone increasing function.

A single-memory implementation of this function requires $2^4 \times 3 = 48$ bits. Thus, the LUT cascade can reduce the total memory size. ■

VI. Analysis of Memory Size

In this section, we derive an upper bound on the memory size of the LUT cascade for the EVMDD (k) for the given packet classification table.

A. Upper Bound on the Number of Segments

Example 3.2 shows that, when two interval have a common element and also neither interval is contained by the other, three segments are produced. For example, for DP shown in Fig. 4, intervals $[6, 8]$ and $[8, 9]$ produce three segments ([6:7], [8:8], and [9:9]). In contrast, for two intervals, when one contains the other or does not intersect, only two segments are produced. For example, for SA shown in Fig. 4, intervals $[0, 3]$ and $[0, 7]$ produce two segments ([0:3] and [4:7]). From above observations, we have the upper bound on the number of segments for the field function.

Theorem 6.1: A field function with p distinct intervals produces at most $2p + 1$ segments.

Proof: We prove it by mathematical induction. When $p = 1$, the number of segments is at most three. Assume that the number of segments for p intervals is $t \leq 2p + 1$. When we add an additional interval, at most two new segments increase. Thus, for $(p + 1)$ intervals, the total number of segments is at most $t + 2 \leq 2p + 1 + 2 = 2(p + 1) + 1$. (Q.E.D.)

Example 6.6: The DP shown in Fig. 4 has $p = 4$ intervals. It has nine segments. ■

Theorem 6.2 [52]: A PRT field function with p distinct intervals produces at most $p + 1$ segments.

Theorem 6.3 [52]: An IP address (SA and DA) field function with p distinct entries produces at most $p + 1$ segments.

Example 6.7: Fig. 5 decomposes the packet classification table shown in Table I by the Cartesian product method. As for the PRT field, we assigned "0" to ICMP, "1" to TCP, and "2" to UDP. Note that, we omit a part of the Cartesian product function due to the space limitations. ■

B. Upper Bound on the Memory Size

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

Theorem 6.4 [35]: Let t be the number of unique indices for the M_1-monotone increasing function. Then, there exist an LUT cascade for an EVMDD (k) with at most $\lceil \log_2 t \rceil$ rails and $\lceil \log_2 t \rceil$ Arails.

From Theorems 6.1 and 6.4, we have an upper bound on the memory size of the LUT cascade for the EVMDD (k) of the given packet classification table.

Theorem 6.5: Any n-input field function with p distinct entries can be implemented by an LUT cascade for the EVMDD (k). Each LUT has $k+r$ inputs and $2r$ outputs (rails and Arails), the total memory size is $r \cdot 2^{k+r+1} \lceil (n)/(k) \rceil$ bits, and $r = \lceil \log_2(2p + 1) \rceil$.

Proof: From Theorems 6.1 and 6.4, the number of rails (Arails) of the LUT cascade is at most $r = \lceil \log_2(2p+1) \rceil$. The number of LUTs is $\lceil (n)/(k) \rceil$. Since each LUT has $k+r$ inputs and $2r$ outputs (rails and Arails), the total memory size of the LUT cascade for the EVMDD (k) is $2^{k+r} \times 2r \times \lceil (n)/(k) \rceil$ bits. Hence, we have the Theorem. (Q.E.D.)

Theorem 6.5 shows that the memory size of the LUT cascade depends on n, p, and k. Since n and p are given by the packet classification table, we can find the best value for k.

VII. Update Method for LUT Cascades

A. Definition

An 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** of a vector. The addition is achieved by rewriting the corresponding index to non-zero, while the deletion is achieved by rewriting the corresponding index to zero. Thus, the update requires both an addition and a deletion.

Two update methods exist: **the off-line** and **the on-line**. In the off-line update, first, the HDL codes for the LUT cascade including the updated rule are re-generated. Then, the FPGA is

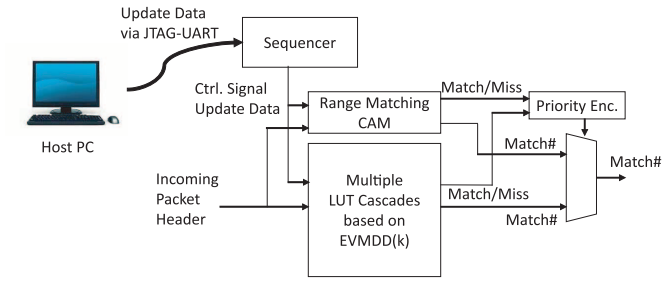

Fig. 13. Proposed system supporting on-line update.

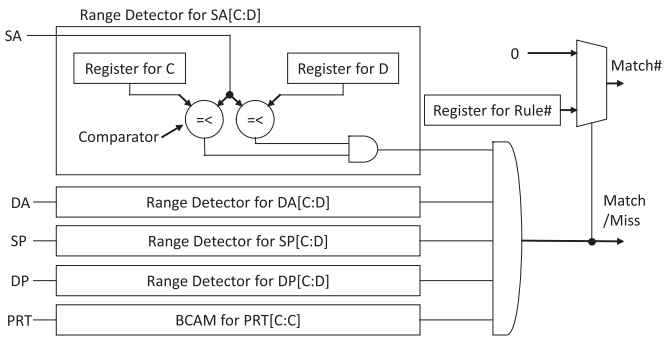

Fig. 14. Range-Matching CAM.

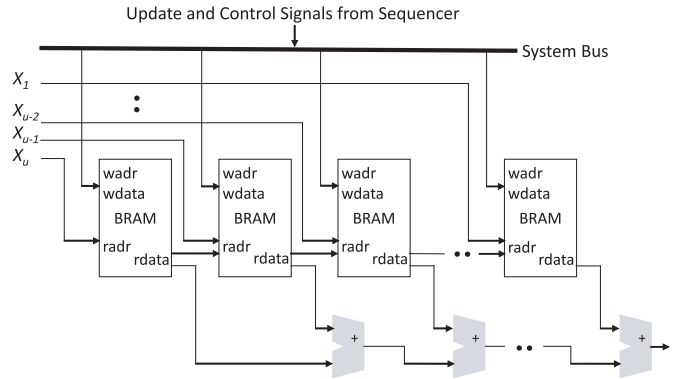

Fig. 15. LUT cascade based on the EVMDD (k) for the on-line update.

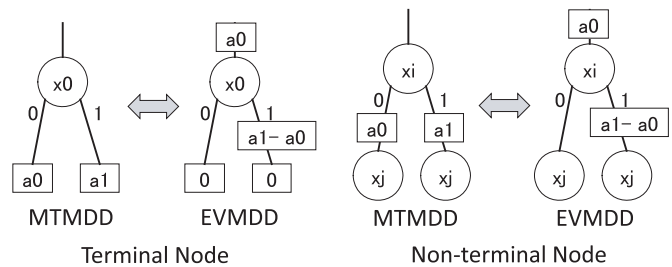

Fig. 16. Conversion rules for EVMDD (k).

re-configured. During re-configuration, the packet classification must be suspended. On the other hand, in the on-line update, rules can be updated without suspending the packet classification. In this paper, we propose an on-line update for the LUT cascades.

B. Proposed System

Fig. 13 shows the proposed system for the on-line update. It consists of LUT cascades, a small range-matching CAM, a priority encoder, a multiplexer, and the sequencer. It is similar to Luo's method [29], which uses an SRAM-based architecture using trie instead of the LUT cascades. The trie based architecture only uses the memory, while the EVMDD (k) based one uses both the memory and DSP blocks.

The Host PC synthesizes the initial bit stream to configure the FPGA, and updates the LUT cascades. It stores the EVMDD (k) on the main memory. The FPGA and the host PC are connected by the JTAG-UART (Joint test action group universal asynchronous receiver/transmitter) cable. **The range-matching CAM** stores only a single rule, and performs the range-matching defined in Section II. It generates a match signal and the corresponding rule number. It can be updated by a single clock. To update of the LUT cascades, first, the update rule is stored in the range-matching CAM. Then, the priority encoder and the multiplexer selects the output of the range-matching CAM. Next, the Host PC updates the LUT cascade using the sequencer. Finally, the sequencer clears the update rule from the range-matching CAM. Thus, the LUT cascades can be updated without suspending the classification operation. Although the proposed system requires the additional hardware, its overhead is 8.5% of the original LUT cascade. It will be shown later.

Fig. 14 shows the range-matching CAM. It consists of range detectors and a binary CAM. The range detectors perform range matching for four fields (SA, DA, SP, and DP). Each detector consists of two comparator, two registers, and an AND gate. The binary CAM performs exact matching for PRT field. When all the fields are matched, it generates the match signal (logical one) and the corresponding rule number. Otherwise, it generates the miss-match signal (logical zero) and the default rule number (zero). Thus, the range matching CAM can be quickly updated by rewriting the corresponding register.

Fig. 15 shows the LUT cascade based on the EVMDD (k) for the on-line update implemented as the pipelined BRAM architcture. The update of the pipelined memory-based lookup architecture can be done by write bubbles [2]. It injects the write data into the pipleline. Since the BRAM for the current FPGA supports a simple dual port (SDP) mode, we can perform a write-before-read operation [67]. Thus, the pipelined LUT cascade can perform update (write) before lookup (read). With this technique, we can guarantee the correct operation during update. When BRAMs on the LUT cascade are updated, the output of CAM is used to produce the correct rule number. Thus, the system produces a correct value for the matched rule during update.

C. 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. Fig. 16 shows a conversion rule between the MTMDD (k) and the EVMDD (k). To update the EVMDD (k), first, the nodes for the EVMDD (k) are expanded into nodes for the MTMDD (k) by traversing from the root node to the terminal node. Then, the terminal node is rewritten to the corresponding rule number. Finally, the nodes for the MTMDD (k) are converted into ones for the EVMDD (k). An algorithm to update the EVMDD (k) is as follows:

Algorithm 7.2: (Update EVMDD (k))

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

Example 7.8: We show an example of the update for the EVMDD (k) shown in Fig. 10. Here, the index "0" of the vector $(x_0, x_1, x_2, x_3) = (1, 1, 0, 1)$ is updated into the index "8".

First, we traverse the EVMDD (1) corresponding to the vector $(1, 1, 0, 1)$ [Fig. 17(a)] by converting to the MTMDD nodes. Then, we rewrite the terminal value into "8" [Fig. 17(b)]. Finally, we recursively convert into EVMDD nodes [Fig. 17(c) and (d)]. ■

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 7.3: (On-line update for the LUT cascade)

1. Update the range-matching CAM.
2. Generate the update data for the LUT cascade on the host PC.
   2.1. Apply Algorithm 7.2.
   2.2. 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. Send the update data to the LUT cascade on the FPGA from the host PC, and update the LUT cascade.
4. Clear the update rule in the range-matching CAM.
5. Terminate.

Algorithm 7.3 allows the update without suspension. Theorem 6.5 guarantees that an arbitrary pattern can be updated.

VIII. Experimental Results

A. Implementation Setup

We used the Virtex 7 VC707 evaluation board (FPGA: Xilinx, XC7VX485T-2FFG, 75 900 Slices, 2060 18 Kb BRAMs, and 2800 DSP48E Blocks) and the Xilinx PlanAhead version 14.7 for the synthesis. As for the LUT cascade implementation, LUT_i whose size is equal to or greater than 18 Kb was implemented by 18 Kb BRAMs, while LUT_i whose size is less than 18 Kb was implemented by distributed RAMs using Slices. To increase the system throughput, we used the dual-port mode to access the memory. By Algorithm 4.1, we partitioned the set of 9816 ACL rules generated by ClassBench [60] into two: Group 1 (9600 rules) and Group 2 (216 rules). Then, each group is decomposed into five-field functions and a Cartesian product function. Finally, each function is realized by an LUT cascade for an EVMDD (k). To reduce the widths of an EVMDD (k), we used the shifting method [49].

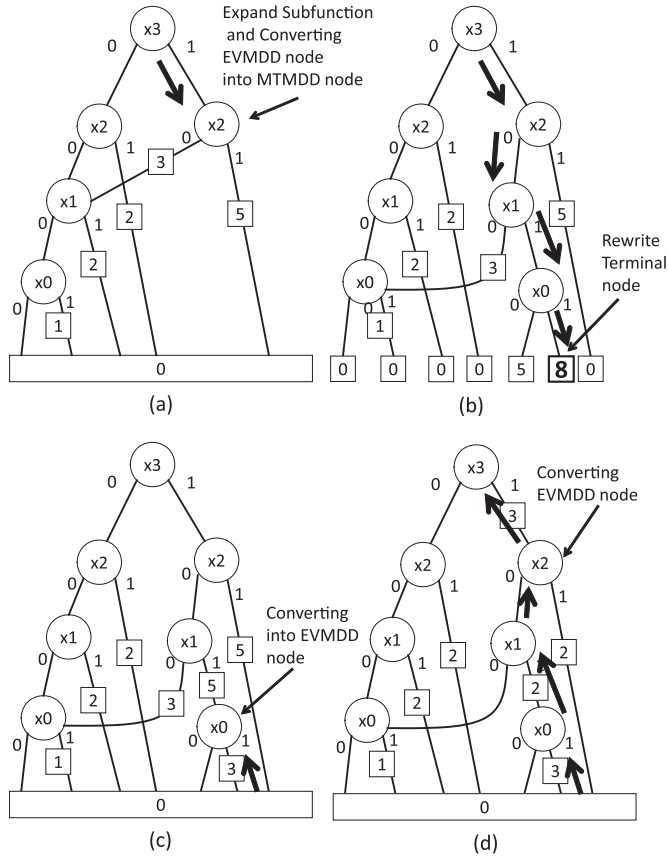

Fig. 17. Example of the update for the EVMDD (k).

By using Algorithm 4.1, we implemented the two parallel packet classifier by the multiple LUT cascades for an EVMDD (k) shown in Fig. 21, which consumes 5134 Slices (6.7%), 256 18 Kb BRAMs (12.4%), and 210 DSP48E blocks (7.5%). Since the maximum clock frequency was 511.174 MHz, we set the system clock frequency to 500 MHz. Thus, the system throughput is $2\ (\text{two parallel classifier}) \times 0.50\ (\text{GHz}) \times 2\ (\text{ports}) \times 320\ (\text{Bits}) = 640.00$ Gb/s for minimum packet size (40 Bytes).

B. Comparison of EVMDD (k) With MTMDD (k) to Implement LUT Cascade

We realized two-parallel packet classifiers by two different methods.

1) LUT cascades for the MTMDD (k).
2) LUT cascades for the EVMDD (k).

To find the smallest LUT cascade, we changed the value of k from one to four. Fig. 18 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 $k = 2$ in both methods. As for Cartesian product functions, EVMDDs (k) required smaller memory than MTMDDs (k) even if the translation memories are used. Fig. 19 shows the number of adders (DSP48Es) for EVMDD (k). Although EVMDD (k) requires DSP48Es, it requires less than 7.5% of available DSP48Es. Thus, the usage of DSP48Es is negligible.

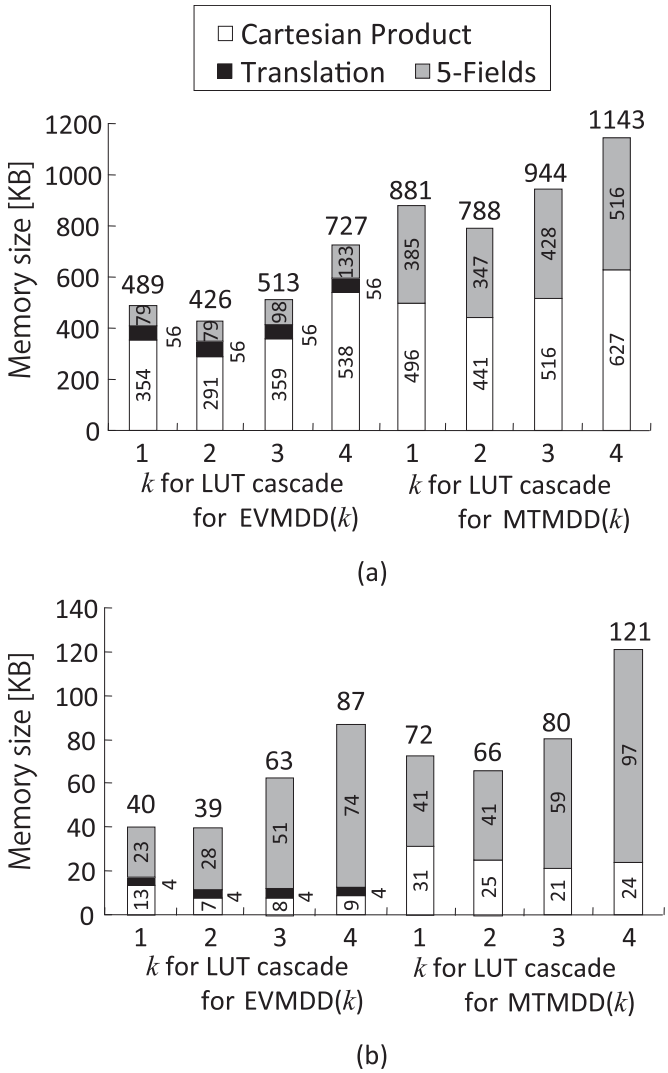

Fig. 18. Comparison of memory sizes [KB]. (a) Memory Size [KB] for Subrule 1 (9600 rules). (b) Memory Size [KB] for Subrule 2 (216 rules).

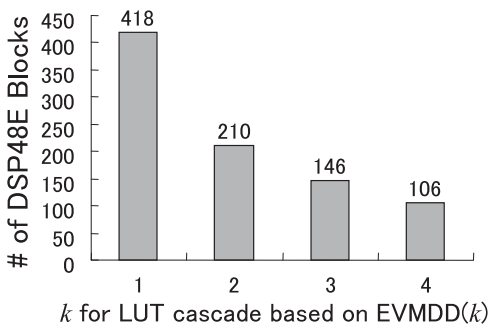

Fig. 19. Number of adders (DSP48Es) for EVMDD (k).

As shown in this part, the LUT cascade for EVMDDs (k) efficiently utilizes the resource of an FPGA.

Fig. 20 compares power consumption of the LUT cascade for the MTMDD (k) with that for the EVMDD (k). To make the comparison fair, we tried to make the temperature the same, and set the system clock frequency to 500 MHz. Fig. 20 shows that the LUT cascade for the EVMDD (2) dissipates the lowest power.

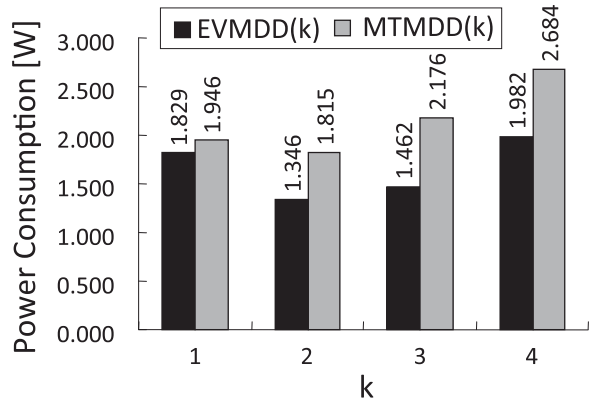

Fig. 20. Comparison of power consumption [W].

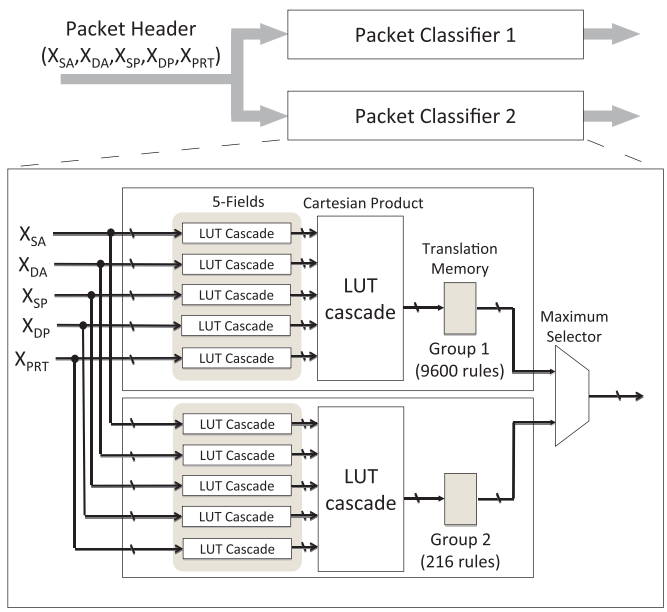

Fig. 21. Implemented two-parallel packet classifier.

C. Comparison the On-Line Update With Off-Line Update

For different number of updates k, we compared the proposed on-line update with the off-line one. We used the random pattern as the pseudo update rule.

Fig. 22 compares the update time. As for the host PC, we used the HP Z800 Workstation (Intel XeonProcessor X5672: 3.20 GHz, 12 MB cache) with 24 GB main memory, Operating system was Ubuntu 12.04 LTS. Also, we connected the workstation to the FPGA by using the USB2.0 cable as a JTAG-UART. We assume that the off-line update is the time from the beginning of the regeneration of the HDL codes until rewriting the FPGA, while the on-line update is the time from sending the update data until receiving the update done signal. As shown in Fig. 22, since the on-line update requires no resynthesis of the HDL code for the packet classifier, its update time is much shorter than that for the off-line one, and satisfies the required time (15 min) for the practical update time. Note that, the on-line update method need not suspend the system.

Table III shows the amount of hardware for LUT cascades supporting on-line update. Note that, in the architecture for the off-line update, we only implemented LUT cascades based on

TABLE II
COMPARISON OF PERFORMANCE PER MEMORY FOR VARIOUS SYSTEMS

Architecture	FPGA	Clk. Freq. [MHz]	#Rules	Memory [KB]	#DSP48E Blocks	Memory [B] /#Rule	Throughput [Gbps]	Performance/mem [Gbps·#Rules/KB]
BV-TCAM (FPGA 2005) [56]	XCV2000E	100.00	222	16	—	73.80	10.00	138.7
Memory-based DCFL (FCCM 2008) [15]	VirtexII Pro	—	128	221	—	1768.00	24.00	13.9
2sBFCE (FCCM 2008) [41]	—	—	4,000	178	—	45.56	2.06	46.3
Simplified Hyper Cuts (ANCS 2008) [24]	StratixIII	128.00	10,000	286	—	29.28	10.84	379.0
Prasanna et. al (ASAP 2009) [18]	Virtex5	143.00	9,603	432	—	46.05	91.73	2039.1
Jiang et. al (ICCCN 2009) [19]	Virtex5	125.00	9,603	245	—	54.22	80.00	3135.6
Optimized Hyper Cuts (IEEE Trans. on VLSI2012) [20]	Virtex5	125.40	9,603	612	—	65.25	80.23	1258.9
StrideBV (HPSR 2012) [7]	Virtex6	—	512	78	—	156.00	407.00	2671.6
Parasplit (HOTI 2012) [6]	Virtex5	100.23	10,000	14,364	—	1470.00	102.60	71.4
Sanny et. al (IPDPSW 2013) [50]	Virtex7	—	512	17	—	35.00	169.00	4944.5
Ganegedara et. al (IEEE Trans. on DIST 2014) [8]	Virtex7	300.00	512	26	—	52.00	135.00	2658.5
Multiple LUT cascades (Proposed)	Virtex7	500.00	9,816	576	210	60.08	640.00	10906.7

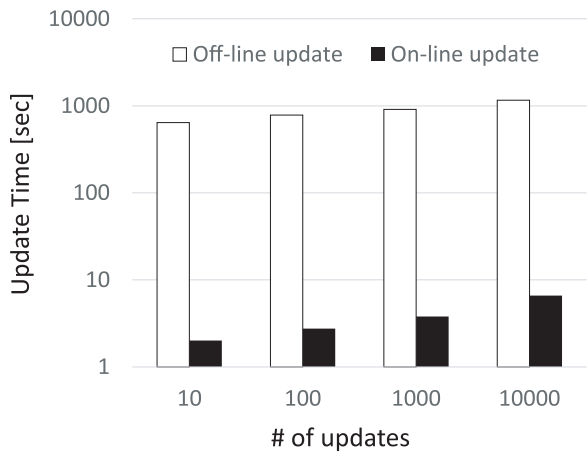

Fig. 22. Comparison of the update time.

TABLE III
HARDWARE RESOURCE FOR THE LUT CASCADES SUPPORTING ON-LINE UPDATE (10 000 RULES)

	# Slices	# 18Kb # BRAMs	# DSP48E # blocks
Two parallel LUT cascade			
	5,134	256	210
Additional hardware for on-line update			
Sequencer	322	0	0
Priority Enc.	1	0	0
Multiplexer	7	0	0
Range-Matching CAM	110	0	0

the EVMDD (k). As shown in Table III, the on-line update system requires additional 440 slices. However, the overhead is only 8.5% of the LUT cascades.

D. Comparison With Other Implementations

Table II compares the proposed implementation with other implementations with respect to **the performance per memory:** [20]

$$\frac{\text{Throughput [Gb/s]}}{\text{normalized memory (Memory [B]/\#Rules)}}.$$

The proposed architecture implemented 9 816 rules by 576 [KB] memory, and its system throughput is 640.00 [Gb/s]. Thus, the performance per memory is 10906.7 [Gb/s·#rules/KB]. Table II shows that the performance per memory of the proposed architecture is 2.21 times higher than that of StrideBV method [7] that was the highest among the existing FPGA realizations. This shows that we implemented a high-speed and memory efficient system.

We measured the power consumption of our design by using the XPower Analyzer tool available in the Xilinx PlanAhead 14.7. Table IV compares the proposed method with other methods with respect to **the power consumption per performance:** [7]

$$\frac{\text{Normalized power consumption } [\mu\text{W/\#Rules}]}{\text{Performance [Gb/s]}}.$$

The proposed architecture implemented 9816 rules by 1.346 [W] power consumption, and its system throughput was 640.00 [Gb/s]. Thus, the power consumption per performance is 0.20 [μW/(Gb/s·#Rule)]. Table IV shows that the power consumption per performance of the proposed architecture is 11.95 times lower than that of StrideBV method [7] that was the lowest among the existing FPGA realizations. This shows that we implemented a high-speed, low-power and memory efficient system.

IX. CONCLUSION

In this paper, we showed a method to implement the 5-tuple packet classifier. The design method is as follows: First, the packet classification rules are decomposed into two groups. Second, they are decomposed into five-field functions and a Cartesian product function. And finally, each function is realized by an LUT cascade for an EVMDD (2). We derived an upper bound on the memory size for the given packet classification table. In this paper, we also proposed an on-line update method. It can update the LUT cascade without suspending the operations. We showed a theorem which guarantees the possibility of update for arbitrary patterns. Although the on-line update requires an additional hardware, its overhead is 8.5% of the LUT cascades. Also, its update time is much shorter than the off-line one, and the proposed update method need not suspend the operations. We implemented the two-parallel

TABLE IV
COMPARISON OF POWER CONSUMPTION PER PERFORMANCE

Architecture	FPGA	Clk. Freq. [MHz]	#Rules	Memory /#Rule [B]	Throughput [Gbps]	Power consumption/performance [μW/(Gbps·#Rule)]
BV-TCAM (FPGA 2005) [56]	XCV2000E	100.00	222	82.90	80.00	4.58
FSBV (SPAA 2009) [17]	Virtex5	167.00	512	29.00	100.00	3.34
StrideBV (HPSR 2012) [7]	Virtex6	—	512	156.00	407.00	2.39
Sanny et. al (IPDPSW 2013) [50]	Virtex7	—	512	35.00	169.00	9.63
Ganegedara et. al (IEEE Trans. on DIST 2014) [8]	Virtex7	300.00	512	52.00	135.00	4.62
Multiple LUT cascades (Proposed)	Virtex7	500.00	9,816	60.10	691.00	0.20

packet classifier on a Virtex 7 VC707 evaluation board. Experimental result showed that, the performance per memory (throughput per normalized area) is 2.21 times higher than any of existing FPGA realizations, and the power consumption per performance is 11.95 times lower than any of existing FPGA realizations. Thus, the proposed packet classifier is a high-speed, low-power and memory efficient system.

REFERENCES

[1] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, no. 8, pp. 677–691, Aug. 1986.

[2] A. Basu and G. Narlikar, "Fast incremental updates for pipelined forwarding engines," *IEEE/ACM Trans. Netw.*, vol. 13, no. 3, pp. 690–703, 2005.

[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," in *Design Automat. Conf.*, 1993, pp. 54–60.

[4] Configuring IP ACLs [Online]. Available: http://eee.cisco.com/en/US/docs/switches/datacenter/sw/4_1/nxos/security/configuration/guide/sec_ipacls.pdf

[5] S. Dharmapurikar, H. Song, J. S. Turner, and J. W. Lockwood, "Fast packet classification using bloom filters," in *Int. Symp. Archit. Netw. Commun. Syst.*, 2006, pp. 61–70.

[6] J. Fong, X. Wang, Y. Qi, J. Li, and W. Jiang, "ParaSplit: A scalable architecture on FPGA for terabit packet classification," in *Hot Interconnects*, 2012, pp. 1–8.

[7] T. Ganegedara and V. K. Prasanna, "StrideBV: Single chip 400G+ packet classification," in *Proc. IEEE Int. Conf. High Performance Switching Routing*, 2012, pp. 1–6.

[8] T. Ganegedara, W. Jiang, and V. K. Prasanna, "A scalable and modular architecture for high-performance packet classification," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1135–1144, May 2014.

[9] K. Golnabi, R. K. Min, L. L. Khan, and E. Al-Shaer, "Analysis of firewall policy rules using data mining techniques," in *IEEE/IFIP Network Operat. Manage. Symp.*, Apr. 2006, pp. 305–315.

[10] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *Hot Interconnects*, 1999, pp. 34–41.

[11] P. Gupta and N. McKeown, "Packet classification on multiple fields," *ACM SIGCOMM*, pp. 147–160, 1999.

[12] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *IEEE Micro*, vol. 20, no. 1, pp. 34–41, 2000.

[13] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE/ACM Trans. Netw.*, vol. 15, no. 2, pp. 24–32, 2001.

[14] "IEEE 802.3: 400 Gbps ethernet study group 2013," [Online]. Available: http://www.ieee802.org/3/400GSG/index.html

[15] G. S. Jedhe, A. Ramamoorthy, and K. Varghese, "A scalable high throughput firewall in FPGA," in *Proc. IEEE Int. Symp. Field-Programmable Custom Comput. Machines*, 2008, pp. 43–52.

[16] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on FPGAs," in *ACM/SIGDA Int. Symp. Field-Programmable Gate Array* , 2009, pp. 219–228.

[17] W. Jiang and V. K. Prasanna, "Field-split parallel architecture for high performance multi-match packet classification using FPGAs," in *21st ACM Symp. Parallelism Algorithms Archit.*, 2009, pp. 188–196.

[18] W. Jiang and V. K. Prasanna, "A FPGA-based parallel architecture for scalable high-speed packet classification," in *Proc. IEEE Int. Conf. Application-Specific Syst., Archit. Processors*, 2009, pp. 24–31.

[19] W. Jiang and V. K. Prasanna, "Scalable packet classification: Cutting or merging?," in *Proc. Int. Conf. Comput. Commun. Netw.*, 2009, pp. 1–6.

[20] W. Jiang and V. K. Prasanna, "Scalable packet classification on FPGA," *IEEE Trans. Very Large Scale (VLSI) Syst.*, vol. 20, no. 9, pp. 1668–1680, Sep. 2012.

[21] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and applications," *Multiple-Valued Logic*, vol. 4, no. 1–2, pp. 9–62, 1998.

[22] Y. Kanizo, D. Hay, and I. Keslassy, "Optimal fast hashing," in *IEEE INFOCOM*, 2009, pp. 2500–2508.

[23] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Enegy efficient packet classification hardware accelerator," in *Proc. Int. Symp. Parallel Distrib. Process.*, 2008, pp. 1–8.

[24] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Low power architecture for high speed packet classification," in *ANCS*, 2008, pp. 131–140.

[25] Y.-T. Lai and S. Sastry, "Edge-valued binary decision diagrams for multi-level hierarchical verification," in *Proc. Design Automat. Conf.*, 1992, pp. 608–613.

[26] Y.-T. Lai, M. Pedram, and S. B. Vrudhula, "EVBDD-based algorithms for linear integer programming, spectral transformation and functional decomposition," *IEEE Trans. Comput.*, vol. 13, no. 8, pp. 959–975, Aug. 1994.

[27] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *SIGCOMM*, 2005, pp. 193–204.

[28] D. Liu, B. Hua, X. Hu, and X. Tang, "High-performance packet classification algorithm for many-core and multithreaded network processor," in *Proc. IEEE Int. Conf. Compilers, Archit., Synthesis Embedded Syst.*, 2006, pp. 334–344.

[29] L. Luo, G. Xie, Y. Xie, L. Mathy, and K. Salamatian, "A hybrid hardware architecture for high-speed IP lookups and fast route updates," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 957–969, 2014.

[30] P. McGeer, J. Sanghavi, R. Brayton, and A. Sangiovanni-Vincentelli, "Espresso-signature: A new exact minimizer for logic functions," *IEEE Trans. Very Large Scale (VLSI) Syst.*, vol. 1, no. 4, pp. 432–440, 1993.

[31] R. McGeer and P. Yalagandula, "Minimizing classifiers for TCAM implementation," in *INFOCOM*, 2009, pp. 1314–1322.

[32] C. R. Meiners, A. Liu, and E. Torng, "TCAM razor: A systematic approach towards minimizing packet classifiers in TCAMs," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 490–500, Apr. 2009.

[33] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD Foundations and Applications*. New York: Springer, 1998.

[34] S. Nagayama and T. Sasao, "Representations of elementary functions using edge-valued MDDs," in *Proc. IEEE Int. Symp. Multiple-Valued Logic*, 2007, pp. 1–5.

[35] 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, pp. 2752–2761, 2007.

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

[37] H. Nakahara, T. Sasao, and M. Matsuura, "Packet classifier using a parallel branching program machine," in *Euromicro Conf. Digital Syst. Design*, 2010, pp. 745–752.

[38] H. Nakahara, T. Sasao, and M. Matsuura, "A packet classifier using LUT cascades based on EVMDDs(k)," in *Proc. Int. Conf. Field-Programmable Logic Appl.*, 2013, pp. 1–6.
[39] H. Nakahara, T. Sasao, and M. Matsuura, "A packet classifier based on prefetching EVMDD (k) machines," *IEICE Trans. Info. Syst.*, vol. E97-D, no. 9, pp. 2243–2252, 2014.
[40] H. Nakahara, T. Sasao, and M. Matsuura, "An update method for a CAM emulator using an LUT cascade based on an EVMDD (k)," in *Proc. IEEE Int. Symp. Multiple-Valued Logic*, 2014, pp. 1–6.
[41] A. Nikitakis and I. Papaefstathiou, "A memory-efficient FPGA-based classification engine," in *Proc. IEEE Int. Symp. Field-Programmable Custom Comput. Mach.*, 2008, pp. 53–62.
[42] "Oinkcodes: Automatically updating the SNORT rule set," [Online]. Available: https://www.snort.org/oinkcodes/
[43] OpenFlow Foundation [Online]. Available: http://www.openflowswitch.org/
[44] M. H. Overmars and A. F. van der Stappen, "Range searching and point location among fat objects," *J. Algorithms*, vol. 21, no. 3, pp. 629–656, 1996.
[45] I. Papaefstathiou and V. Papaefstathiou, "Memory efficient 5D packet classification at 40 Gbps," in *INFOCOM*, 2007, pp. 1370–1378.
[46] Y. X. Qi *et al.*, "Towards high-performance flow-level packet processing on multi-core network processors," in *ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, 2007, pp. 17–26.
[47] Quine–McCluskey algorithm [Online]. Available: http://en.wikipedia.org/wiki/Quine-McCluskey_algorithm
[48] V. C. Ravikumar and R. N. Mahapatra, "TCAM architecture for IP lookup using prefix properties," *IEEE Micro*, vol. 24, no. 2, pp. 60–69, 2004.
[49] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *IEEE/ACM Int. Conf. Comput. Aided Design*, 1993, pp. 42–47.
[50] A. Sanny, T. Ganegedara, and V. K. Prasanna, "A comparison of ruleset feature independent packet classification engines on FPGA," in *Proc. IEEE Int. Symp. Parallel Distributed Process. Workshops Ph.D. Forum*, 2013, pp. 124–133.
[51] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," in *Proc. Int. Workshop Logic Synthesis*, 2001, pp. 225–230.
[52] T. Sasao and J. T. Butler, "Implementation of multiple-valued CAM functions by LUT cascades," in *Proc. IEEE Int. Symp. Multiple-Valued Logic*, 2006, pp. 1–11.
[53] T. Sasao, "On the complexity of classification functions," in *Proc. IEEE Int. Symp. Multiple-Valued Logic*, 2008, pp. 57–63.
[54] T. Sasao, *Memory-Based Logic Synthesis*. New York: Springer, 2011.
[55] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *SIGCOMM*, 2003, pp. 213–224.
[56] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2005, pp. 238–245.
[57] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," *Compt. Commun. Rev.*, vol. 28, pp. 191–202, 1998.
[58] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *SIGCOMM*, 1998, pp. 191–202.
[59] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.
[60] D. E. Taylor and J. S. Turner, "ClassBench: A packet classification benchmark," in *INFOCOM*, 2005, vol. 3, pp. 2068–2079.
[61] R. Tucker, "Optical packet-switched WDM networks: A cost and energy perspective," in *Optical Fiber Commun./Nat. Fiber Optic Eng. Conf.*, 2008, pp. 1–25.
[62] B. Vamanan, G. Voskuilen, and T. N. Viaykumar, "Efficuts: Optimizing packet classification for memory and throughput," in *SIGCOMM*, 2010, pp. 207–218.
[63] G. Varghese, *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Burlington, MA: Morgan Kaufmann, 2005.
[64] R. Wei, Y. Xu, and H. J. Chao, "Block permutations in boolean space to minimize TCAM for packet classification," in *INFOCOM*, 2012, pp. 2561–2565.
[65] T. Y. C. Woo, "A modular approach to packet classification: Algorithms and results," in *INFOCOM*, 2000, vol. 3, pp. 1213–1222.
[66] Xilinx Inc. [Online]. Available: http://www.xilinx.com/
[67] Xilinx Inc., "7 series FPGAs memory resources," UG473 2014.
[68] Y. Xu, Z. Liu, Z. Zhang, and H. J. Chao, "High-throughput and memory-efficient multimatch packet classification based on distributed and pipelined hash tables," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 982–995, Jun. 2014.
[69] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: From theory to practice," in *INFOCOM*, 2009, pp. 648–656.
[70] Y. R. Qu and V. K. Prasanna, "Compact hash tables for high-performance traffic classification on multi-core processors," in *Proc. Int. Symp. Comput. Archit. High Performance Comput.*, 2014, pp. 17–24.
[71] B. Xu, D. Jiang, and J. Li, "HSM: A fast packet classification algorithm," in *Proc. Int. Conf. Adv. Inf. Netw. Appl.*, 2005, vol. 1, pp. 987–992.
[72] F. Zane, G. Narlikar, and A. Basu, "CoolCAMs: Power-efficient TCAMs for forwarding engines," in *INFOCOM*, 2003, vol. 1, pp. 42–52.

Hiroki Nakahara (M'05) received the B.E., M.E., and Ph.D. degrees in computer science from Kyushu Institute of Technology, Fukuoka, Japan, in 2003, 2005, and 2007, respectively.

He has held faculty/research positions at Kyushu Institute of Technology, Iizuka, Japan and Kagoshima University, Kagoshima, Japan. Now, he is a Senior Assistant Professor at Ehime University, Ehime, Japan. His research interests include logic synthesis, reconfigurable architecture, digital signal processing and embedded systems.

Dr. Nakahara was the Workshop Chairman for the 23rd International Workshop on Post-Binary ULSI Systems (ULSIWS) held in Bremen, Germany in 2014. He received the 8th IEEE/ACM MEMOCODE Design Contest 1st Place Award in 2010, the SASIMI Outstanding Paper Award in 2010, IPSJ Yamashita SIG Research Award in 2011, the 11st FIT Funai Best Paper Award in 2012, the 7th IEEE MCSoC-13 Best Paper Award in 2013, and the ISMVL2013 Kenneth C. Smith Early Career Award in 2014, respectively. He is a member of the ACM and the IEICE.

Tsutomu Sasao (S'72–M'77–F'94–LF'16) received the B.E., M.E., and Ph.D. degrees in electronics engineering from Osaka University, Osaka Japan, in 1972, 1974, and 1977, respectively.

He has held faculty/research positions at Osaka University, Japan, IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, and the Naval Postgraduate School, Monterey, CA, USA. He has served as the Director of the Center for Microelectronic Systems at the Kyushu Institute of Technology, Iizuka, Japan. Now, he is a Professor at the Department of Computer Science, Meiji University, Kawasaki, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than 10 books on logic design including, *Logic Synthesis and Optimization*, *Representation of Discrete Functions*, *Switching Theory for Logic Synthesis*, *Logic Synthesis and Verification*, and *Memory-Based Logic Synthesis*, in 1993, 1996, 1999, 2001, and 2011, respectively.

Dr. Sasao has served Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan in 1998. He received the NIWA Memorial Award in 1979, Takeda Techno-Entrepreneurship Award in 2001, and Distinctive Contribution Awards from IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1986, 1996, 2003, and 2004. He has served an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS.

Hisashi Iwamoto received the B.S. and M.S. degrees in physics from Kwansei Gakuin University, Hyogo, Japan in 1987 and 1989, respectively, and the Ph.D. degree in information and communication engineering from Osaka City University, Osaka, Japan, in 2013.

In 1989, he joined the LSI Laboratory, Mitsubishi Electric Corporation, Hyogo, Japan, where he has been engaged in the design, development and standardization of the high-speed synchronous DRAM. He transferred to Renesas Technology Corp. and REVSONIC Corp. in 2003 and 2012, respectively. He is currently interested in system solution for high performance network.

Dr. Iwamoto is a member of the IEICE.

Munehiro Matsuura studied at the Kyushu Institute of Technology from 1983 to 1989. He received the B.E. degree in natural sciences from the University of the Air, Chiba, Japan, in 2003.

He has been working as a Technical Assistant at the Kyushu Institute of Technology, Fukuoka, Japan, since 1991. He has implemented several logic design algorithms under the direction of Prof. T. Sasao. His interests include decision diagrams and exclusive-OR based circuit design.