IEICE TRANS. INF. & SYST., VOL.E98–D, NO.2 FEBRUARY 2015

PAPER Special Section on Reconfigurable Systems

# A Memory-Based IPv6 Lookup Architecture Using Parallel Index Generation Units

**Hiroki NAKAHARA**[†a)], **Tsutomu SASAO**[††b)], **Munehiro MATSUURA**[†††c)], **Hisashi IWAMOTO**[††††d)], ***Members***, ***and*** **Yasuhiro TERAO**[††††e)], ***Nonmember***

**SUMMARY** In the era of IPv6, since the number of IPv6 addresses rapidly increases and the required speed is more than Giga lookups per second (GLPS), an area-efficient and high-speed IP lookup architecture is desired. This paper shows a parallel index generation unit (IGU) for memory-based IPv6 lookup architecture. To reduce the size of memory in the IGU, we use a linear transformation and a row-shift decomposition. A single-memory realization requires $O(2^l \log k)$ memory size, where $l$ denotes the length of prefix, while the realization using IGU requires $O(kl)$ memory size, where $k$ denotes the number of prefixes. In IPv6 prefix lookup, since $l$ is at most 64 and $k$ is about 340 K, the IGU drastically reduces the memory size. Also, to reduce the cost, we realize the parallel IGU by using both on-chip and off-chip memories. We show a design algorithm for the parallel IGU to store given off-chip and on-chip memories. The parallel IGU has a simple architecture and performs lookup by using complete pipelines those insert the pipeline registers in all the paths. We loaded more than 340 K IPv6 pseudo prefixes on the Xilinx Virtex 6 FPGA with off-chip DDRII+ Static RAMs (SRAMs). Its lookup speed is 1.100 giga lookups per second (GLPS) which is sufficient for the required speed for a next generation 400 Gbps link throughput. As for the normalized area and lookup speed, our implementation outperforms existing FPGA implementations.

***key words:*** *CAM, IP lookup, index generation unit, FPGA*

## 1. Introduction

### 1.1 Demands for Lookup Architecture in IPv6 Era

The core routers forward packets by IP-lookup using **longest prefix matching (LPM)**. On Feb. 3, 2011, IPv4 addresses maintained by Internet Assigned Numbers Authority (IANA) are depleted. Since transition from IPv4 addresses to IPv6 addresses are encouraged, IPv6 addresses are widely used in core routers. Specifications for IPv6 address are changed frequently due to the transition period. For example, IPv4-compatible IPv6 addresses are abolished, and site-local addresses would be abolished. Thus, reconfigurable architecture is necessary to accommodate the change of specifications. Since the core routers dissipate the major part of the total network power dissipation [25], we cannot use ternary content addressable memories (TCAMs) based architecture [14] which dissipate much power. Thus, the low-power lookup architecture is necessary.

To solve these problems, memory-based IP lookup architectures on the FPGA have been proposed, which opposed to the TCAM. They dissipate lower power than the TCAM [10]. Also, since the memory-based architectures can reconfigure their contents, they accommodate changes of specifications. Various memory-based IP lookup architectures have been proposed. They are roughly divided into two: Tree-based [1], [8], [9], [16] and Hash-based [5], [13], [17], [23]. Also, the hash-based compress tree [2] has been proposed. To archive a high-speed lookup, a pipelined binary-search tree (BST) has been proposed [10]. Also, the given prefixes are partitioned into disjoint groups, and they are realized by a pipelined BST (BST-IPv6) or by a binary and ternary tree (2-3-tree-IPv6) [7]. To realize the CAM function [22] with small size of memory, the index generation unit [20], [21] has been proposed. Also, its design methods have been proposed [18], [19].

Recent IP lookup architectures consist of both off-chip and on-chip memories to reduce the on-chip memory on FPGAs. However, the following conditions should be satisfied in the next generation network.

**Large-capacity at low cost:** As shown in Fig. 1, on April. 24, 2014, the number of raw IPv6 address in the border gateway protocol (BGP) was about 10 K. The num-

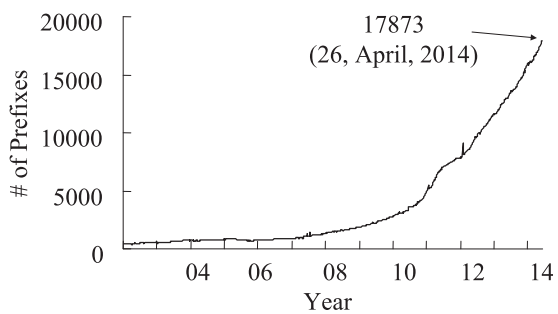



**Fig. 1** Numbers of IPv6 prefixes in the routing table for border gateway protocol (BGP) (26, April, 2014).

Manuscript received May 5, 2014.
Manuscript revised September 8, 2014.
Manuscript publicized November 19, 2014.

[†]The author is with the Department of Electrical and Electronic Engineering and Computer Science, Ehime University, Matsuyama-shi, 790–8577 Japan.
[††]The author is with the Department of Computer Science, Meiji University, Kawasaki-shi, 214–8571 Japan.
[†††]The author is with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.
[††††]The authors are with REVSONIC Corp., Yokohama-shi, 220–0012 Japan.
a) E-mail: nakahara@cs.ehime-u.ac.jp
b) E-mail: sasao@cs.meiji.ac.jp
c) E-mail: matsuura@cse.kyutech.ac.jp
d) E-mail: hisashi-iwamoto@revsonic.com
e) E-mail: yasuhiro-terao@revsonic.com
DOI: 10.1587/transinf.2014RCP0006

ber of prefixes of IPv4 addresses increased by 25–50 K per year [3]. With the rapid growth of IPv6 addresses, large-capacity core routers become necessary for the future IPv6. The advantage of FPGAs is that they can access multiple embedded memories at a time, while the disadvantage is that the sizes of embedded memories are smaller than that of off-chip memories. Thus, the conventional memory-based IP lookup architectures using on-chip memories on FPGAs cannot realize a large IP lookup table. With many FPGAs, we can realize a large IP lookup table. However, it increases costs. The off-chip memories are less expensive than FPGAs. To realize a large IP lookup table with low cost, an FPGA and off-chip memories should be combined together.

**High-speed lookup:** IEEE 802.3 working group is developing for the next standard for 400 Gbps link [6]. The minimum Ethernet frame size is 64 Bytes, which consists of IP Packets (46 Bytes), MAC address (12 Bytes), Port type (2 Bytes), and FCS (Frame check sequence) (4 Bytes). The minimum packet size for IPv6 is 46 Bytes (40 Bytes for the header and 6 Bytes for the minimum payload). In this case, the required speed is more than 1.087 giga lookup per second (GLPS) for 400 Gbps link.

### 1.2 Proposed Architecture and Contributions of the Paper

This paper proposes a memory-based architecture satisfying above two conditions. When $k$ prefixes with length $l$ for IPv6 are loaded in a single memory, the amount of memory necessary for lookup would be $O(2^l \log_2 k)$, which is too large to implement. In this paper, we use a parallel index generation unit (IGU) that reduces the total amount of memory to $O(kl)$, where $k$ denotes the number of prefixes [20]. Also, since the parallel IGU has a simpler architecture than existing ones, it performs a fast lookup. This paper is an extended version of [12]. The contributions in [12] were:

1. We loaded more than 340 K pseudo IPv6 prefixes on the parallel IGU on a single FPGA. Its performance was 1.002 GLPS (Giga lookups per second) which is slightly lower than the next generation link speed.
2. We reduced the total amount of memory for IGUs by using both a linear transformation and a row-shift decomposition.
3. We compared the parallel IGU with existing implementations on FPGAs, and showed that the parallel IGU outperforms others.

In this paper, the following new contributions are included:

1 We realized the parallel IGU using both off-chip and on-chip memories to reduce the total cost. Also, we proposed a design method for the parallel IGU. Since the proposed parallel IGU can be implemented by a smaller FPGA, the total cost is reduced drastically.

2 To archive more than 1.087 GLPS lookups, we reduced the delay time in the critical path of the parallel hardware, and used complete pipelines. Its lookup speed exceeds the next generation network link speed.

The rest of the paper is organized as follows: Sect. 2 introduces an architecture for LPM; Sect. 3 shows the IGU and its memory reduction method; Sect. 4 shows the design method for the IGU; Sect. 5 shows the parallel IGU using complete pipelines; Sect. 6 shows the experimental results; and Sect. 7 concludes the paper.

## 2. Architecture for IPv6 Prefix Lookup

### 2.1 IPv6 Prefix

The IPv6 address (128 bits) is an extension of the IPv4 address (32 bits). This extension accommodates much larger number of addresses than IPv4. An IPv6 address consists of 64 bits **network prefix (prefix)** and 64 bits interface ID. Since only network prefixes are used to make forwarding decisions, this paper considers an architecture for the network prefix lookup. Figure 2 shows the distribution of raw IPv6 prefixes (26, April, 2014) [15]. The present IPv6 uses prefixes with length 15 or more. Note that variance of the numbers of prefixes with different lengths are quite large. In this paper, we use this property to reduce the amount of memory.

### 2.2 Longest Prefix Matching (LPM) Function

**Definition 2.1:** The **LPM table** stores ternary vectors of the form $VEC_1 \cdot VEC_2$, where $VEC_1$ consists of $0's$ and $1's$, while $VEC_2$ consists of only $*'s$ (don't cares). The **length** of prefix is the number of bits in $VEC_1$. To assure that the longest prefix address is produced, entries are stored in descending prefix length. Let $n$ be the number of bits for the input, $m$ be the number of bits for the output, and $B = \{0, 1\}$. The **LPM function** [22] is a mapping $\vec{f} : B^n \to B^m$, where $\vec{f}(x)$ shows the minimum address whose $VEC_1$ corresponds to $\vec{x}$. Otherwise, $\vec{f}(\vec{x}) = 0^m$, where $m = \lceil \log_2(k+1) \rceil$.

In the TCAM, entries are represented by the ternary vectors. Let $P_i$ be a set of the prefixes with length $i$, and $\mathcal{P} = \{P_1, P_2, \ldots P_l\}$ be a set of subsets of prefixes. Each $P_l$ is represented by an **index generation function** [21], which is a mathematical model of the binary CAM.

**Definition 2.2:** [21] A mapping $F(\vec{X}) : B^l \to \{0, 1, \ldots, k\}$,

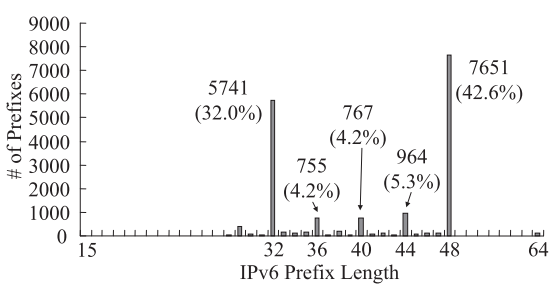



**Fig. 2** Distribution of raw IPv6 prefixes (26, April, 2014) [15].

**Table 1** Example of an index generation function $f$.

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $f$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 0 | 1 | 0 | 3 |
| 0 | 0 | 1 | 1 | 1 | 0 | 4 |
| 0 | 0 | 0 | 0 | 0 | 1 | 5 |
| 1 | 1 | 1 | 0 | 1 | 1 | 6 |
| 0 | 1 | 0 | 1 | 1 | 1 | 7 |
| Otherwise | | | | | | 0 |

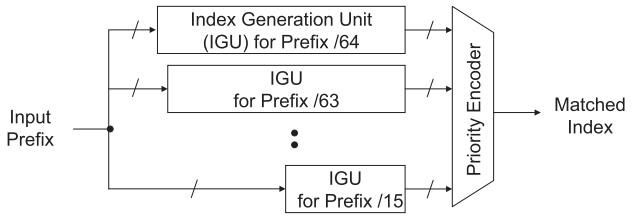



**Fig. 3** Architecture for an LPM function.

is an **index generation function with weight $k$**, where $F(\vec{a}_i) = i$ $(i = 1, 2, \ldots, k)$ for $k$ different **registered vectors**, and $F = 0$ for other $(2^l - k)$ non-registered vectors, and $\vec{a}_i \in B^l$ $(i = 1, 2, \ldots, k)$. In other words, an index generation function produces unique indices ranging from 1 to $k$ for $k$ different registered vectors, and produces 0 for other vectors.

In the TCAM, an entry IP address corresponds to a registered vector, while a non-entry one corresponds to a non-registered vector.

**Example 2.1:** Table 1 shows an index generation function with weight 7. ■

An LPM function can be decomposed into a set of index generation functions. Thus, this paper focuses on a compact realization of an index generation function.

### 2.3 Architecture for an LPM Function

Figure 3 shows an architecture for an LPM function realized by **index generation units (IGUs)** and the priority encoder. The IGU realizes the index generation function with a small size of memory. It consists of the main memory, the AUX memory, the comparator, and AND gates. When an index generation function with weight $k$ is realized by a single memory, the memory size would be $O(2^l \log k)$, which is too large for large $l$. In this paper, we use an IGU with $O(kl)$ memory size.

## 3. Index Generation Unit (IGU) [20]

Table 2 is a **decomposition chart** for the index generation function $f$ shown in Table 1. The columns labeled by $X_1 = (x_2, x_3, x_4, x_5)$ denotes the **bound variables**, while rows labeled by $X_2 = (x_1, x_6)$ denotes the **free variables**. In the IGU, the input variables are partitioned into the bound variables and the free variables. The entry denotes the function value. We can represent the non-zero elements of $f$ by

**Table 2** Decomposition chart for $f(X_1, X_2)$.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $x_5$ |
| | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | $x_4$ $X_1$ |
| | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | $x_3$ |
| | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | $x_2$ |
| 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 0 | 0 | 0 | 4 | 0 | |
| 01 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 10 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 0 | |
| $x_6, x_1$ / $X_2$ | | | | | | | | | | | | | | | | | |

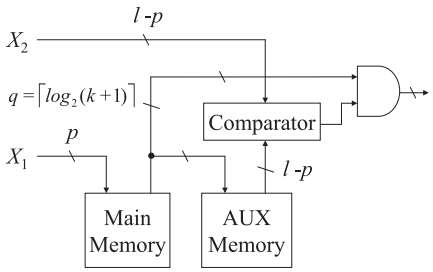



**Fig. 4** Index Generation Unit (IGU).

the **main memory** whose input is $X_1$. The main memory realizes a mapping $\hat{f}$ from a set of $2^p$ elements to a set of $k+1$ elements, where $p = |X_1|$. For example, in Table 2, $|X_1| = 4$. The output for the main memory does not always represent $f$, since $\hat{f}$ ignores $X_2$. Thus, we must check whether $\hat{f}$ is equal to $f$ or not by using the **auxiliary (AUX) memory**. To do this, we compare the input $X_2$ with the output for the AUX memory by a **comparator**. The AUX memory stores the values of $X_2$ when the value of $\hat{f}(X_1)$ is non-zero. Figure 4 shows the index generation unit (IGU). First, the main memory finds the possible index corresponding to $X_1$. Second, the AUX memory produces the corresponding inputs $X_2'$ $(l - p$ bits). Third, the comparator checks whether $X_2'$ is equal to $X_2$ or not. Finally, the AND gates produce the correct value of $f$.

**Example 3.2:** Figure 5 shows an example of the IGU realizing the index generation function shown in Table 1. When the input vector is $X = (x_1, x_2, x_3, x_4, x_5, x_6) = (1, 1, 1, 0, 1, 1)$, the corresponding index is "6". First, the main memory produces the index. Second, the AUX memory produces the corresponding value of $X_2'$. Third, the comparator checks whether $X_2$ and $X_2'$ are equal. Since the corresponding input $X_2$ is equal to $X_2'$, the AND gates produces the index. In this case, $l = 6$, $p = 4$, and $q = 3$. ■

**Example 3.3:** To realize the index generation function $f$ shown in Table 1, a single-memory realization requires $2^6 \times 3 = 192$ bits. On the other hand, in the IGU shown in Fig. 5, the amount of main memory is $2^4 \times 3 = 48$ bits, and that of the AUX memory is $2^3 \times 2 = 16$ bits. Thus, the IGU requires 64 bits in total. In this way, we can reduce the total amount of memory. ■

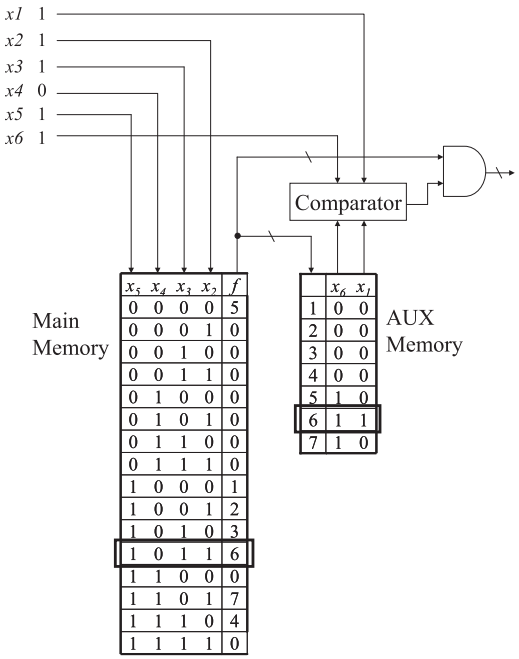



**Fig. 5** IGU for Table 1.

**Table 3** An index generation function $f'$ causing a collision.

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $f'$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 0 | 1 | 0 | 0 | 3 |
| 0 | 0 | 1 | 0 | 0 | 0 | 4 |
| 0 | 1 | 0 | 0 | 0 | 0 | 5 |
| 1 | 0 | 0 | 0 | 0 | 0 | 6 |

**Table 4** Decomposition chart for $f'(X_1, X_2)$.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $x_3$ |
| | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | $x_4$ $X_1$ |
| | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | $x_5$ |
| | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | $x_6$ |
| 00 | | 1 | 2 | | 3 | | | | 4 | | | | | | | | |
| 01 | 5 | | | | | | | | | | | | | | | | |
| 10 | 6 | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | |
| $x_1, x_2$ / $X_2$ | | | | | | | | | | | | | | | | | |

## 3.1 Linear Transformation

Example 3.2 is an ideal case. In general, a column may have two or more non-zero elements. In such a case, the column has a **collision**. When a collision occurs, the main memory cannot realize the function.

**Example 3.4:** Table 4 shows a decomposition chart for an index function $f'$ shown in Table 3. In Table 4, the first column has a collision for elements "5" and "6". ■

Let $\hat{f}(Y_1, X_2)$ be the function whose variables $X_1 = (x_1, x_2, \ldots, x_p)$ are replaced by $Y_1 = (y_1, y_2, \ldots, y_p)$, where $y_i = x_i \oplus x_j$, $x_i \in \{X_1\}$, $x_j \in \{X_2\}$, and $p \geq \lceil \log_2(k+1) \rceil$. This replacement is called a **linear transformation** [19], which reduces the number of inputs to the main memory.

**Table 5** Decomposition chart for $\hat{f}'(Y_1, X_2)$.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $y_1 = x_3 \oplus x_1$ |
| | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | $y_2 = x_4 \oplus x_1$ $Y_1$ |
| | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | $y_3 = x_5$ |
| | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | $y_4 = x_6$ |
| 00 | | 1 | 2 | | 3 | | | | 4 | | | | | | | | |
| 01 | 5 | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | 6 | | | | |
| 11 | | | | | | | | | | | | | | | | | |
| $x_1, x_2$ / $X_2$ | | | | | | | | | | | | | | | | | |

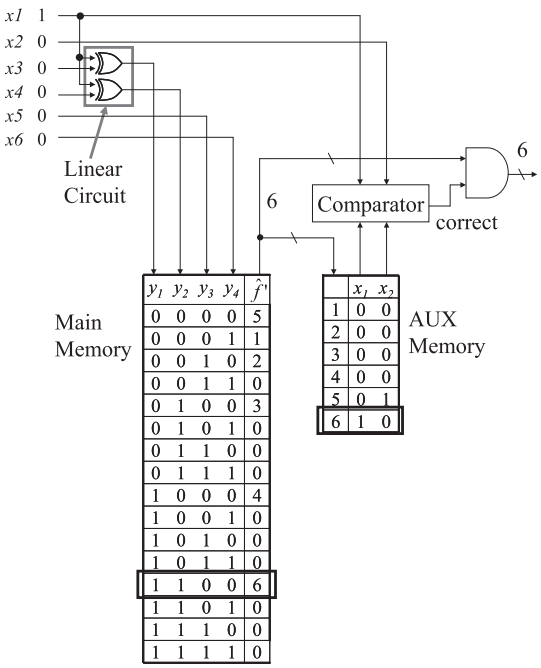



**Fig. 6** IGU using a linear transformation.

The linear transformation for $p$ variables is implemented by $p$ copies of two-input EXORs. In an FPGA, since these can be realized by $p$ LUTs, their amount of hardware is negligibly small.

As shown in Example 3.5, index generation functions can often be represented with fewer variables than original functions. By increasing the number of inputs $p$ for the main memory, we can store virtually all vectors.

**Example 3.5:** Let $f'$ be an index generation function shown in Table 3. Table 5 shows the decomposition chart for $\hat{f}'(Y_1, X_2)$, where $Y_1 = (x_3 \oplus x_1, x_4 \oplus x_1, x_5, x_6)$, and the column labels denote $Y_1$, and the row labels denote $X_2$. In Table 5, since no collision occurs, it can be realized by the IGU shown in Fig. 6. ■

**Conjecture 3.1:** [20] Consider a set of uniformly distributed index generation functions with weight $k$ ($\geq 7$). If $p \geq 2\lceil \log_2(k+1) \rceil - 3$, then, more than 95% of the functions can be represented by an IGU with the main memory having

**Table 6** Decomposition chart for $\hat{f}'(Y_1)$.

| | 0 0 0 0 1 1 1 1 | $y_1$ |
|---|---|---|
| | 0 0 1 1 0 0 1 1 | $y_2$ $Y_1$ |
| | 0 1 0 1 0 1 0 1 | $y_3$ |
| 0 | 5 2 3 4 6 | |
| 1 | 1 | |
| $y_4$ $Y_2$ | | |

**Table 7** Decomposition chart for $\hat{f}'$ after row-shift.

| | 0 0 0 0 1 1 1 1 | $y_1$ |
|---|---|---|
| | 0 0 1 1 0 0 1 1 | $y_2$ $Y_1$ |
| | 0 1 0 1 0 1 0 1 | $y_3$ |
| 0 | 5 2 3 4 6 | |
| 1 | → → → 1 | |
| $y_4$ $Y_2$ | | |

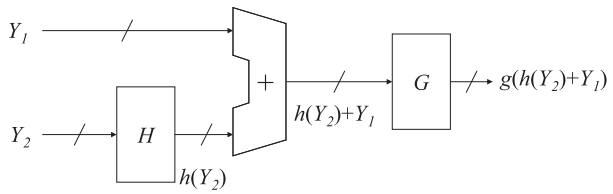



**Fig. 7** Row-shift decomposition.

$p$ inputs.

For the IPv6 prefix lookup problem, a linear transformation of $p$ variables can reduce the amount of memory $2^l\lceil\log_2(k+1)\rceil$ into $2^p\lceil\log_2(k+1)\rceil$.

### 3.2 Row-Shift Decomposition for Main Memory

In this part, we introduce a **row-shift decomposition** [18], which shifts the non-zero elements to further reduce of memory size for the IGU. Table 6 shows a decomposition chart for the index generation function $\hat{f}'(Y_1, Y_2)$, where $Y_1 = (x_3 \oplus x_1, x_4 \oplus x_1, x_5)$ and $Y_2 = (x_6)$. In Table 6, the first column has a collision for the entries "1" and "5". Consider the decomposition chart shown in Table 7 that is obtained from Table 6 by shifting the rows for $y_4 = 1$ by three bit to the right. Table 7 has at most one non-zero element in each column. Thus, the modified function can be realized by a main memory with inputs $Y_1$. In Fig. 7, assume that the memory for $H$ stores the number of bits to shift ($h(Y_2)$) for each row specified by $Y_2$, while the memory for $G$ stores the non-zero element of the column after the shift operation: $h(Y_2) + Y_1$, where "+" denotes an unsigned integer addition.

In Fig. 7, in the row-shift decomposition, the main memory is decomposed into two smaller memories. Thus, we can reduce the main memory for the IGU. Figure 8 shows the IGU using a linear transformation and a row-shift decomposition. Note that, in this case, the comparator has to check the equality of both free variables and bound variables.

**Example 3.6:** Figure 9 shows the IGU using a linear transformation and a row-shift decomposition realizing the function $f'$ in Table 3. Let $\mathcal{Y} = (Y_1, Y_2)$, where $Y_1 = (x_3 \oplus x_1, x_4 \oplus x_1, x_5)$, and $Y_2 = (x_6)$. First, EXOR gates generate $\mathcal{Y}$. Second, the memory for $H$ produces the shift value $h(Y_2)$. Third, the adder produces $h(Y_2) + Y_1$. In this implementation, since we realize both the memory for $G$ and the AUX memory by a single memory, the memory for $G$ produces the index and $(x_1, x_2, x_3, x_4, x_5, x_6)$ simultaneously. Next, the comparator checks if they are equal or not. Finally, the AND gates produces the correcting index. ■

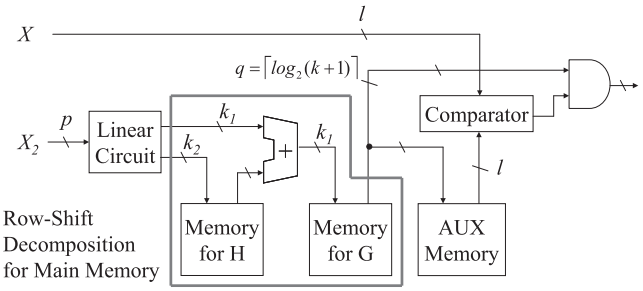



**Fig. 8** IGU using a row-shift decomposition and a linear transformation.

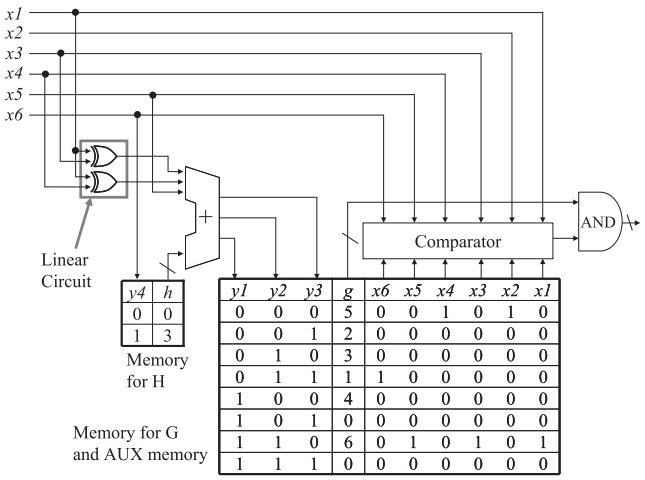



**Fig. 9** An example of IGU.

**Example 3.7:** To realize $f'$ shown in Table 3, a single-memory realization requires $2^6 \times 3 = 192$ bits. On the other hand, in the IGU shown in Fig. 9, the memory for $H$ requires $2^1 \times 3 = 6$ bits, and the memory for $G$ requires $2^3 \times (3 + 6) = 72$ bits. Thus, the IGU requires 78 bits in total. In this way, we can reduce the total amount of memory by using a linear transformation and a row-shift decomposition. ■

Experimental results [18] show that the row-shift decomposition requires $2k\lceil\log_2 k\rceil + kl$ bits to implement an index generation function with weight $k$.

## 4. Design of IGU

### 4.1 Method to Find a Linear Transformation

From here, we present a method to find a linear transformation. We assume that the prefix lookup architecture needs to update its prefix patterns frequently. In this case, it is impractical to find an optimum solution by spending much computation time. To find a reasonably good linear transformation in a short time, we use the following heuristic algorithm [21], which is simple and efficient.

**Algorithm 4.1:** Let $f(X_1, X_2)$ be the index generation function of $l$ variables with weight $k$, and $p = \lceil \log_2((k+1)/3) \rceil + 1$ be the number of the bound variables in the decomposition chart.

1. Let $X_1 = (x_1, x_2, \ldots, x_p)$ be the bound variables, and $X_2 = (x_{p+1}, x_{p+2}, \ldots, x_l)$ be the free variables.
2. While $|X_1| \leq p$, find the variable $x_i \in \{X_2\}$ that minimizes the value:

   $$|(\#\text{ of vectors with } x_i = 0) - (\#\text{ of vectors with } x_i = 1)|.$$

   Let $\{X_1\} \leftarrow \{X_1\} \cup \{x_i\}$.
3. For each pair of variables $(x_i, x_j)$, where $x_i$ is a bound variable, and $x_j$ is a free variable, if the exchange of $x_i$ with $x_j$ decreases the number of collisions, then do it, otherwise discard it.
4. For each pair of variables $(x_i, x_j)$, if the replacement of $x_i$ with $y_i = x_i \oplus x_j$ decreases the number of collisions, then do it, otherwise discard it.
5. Terminate.

### 4.2 Design of IGU Using Row-Shift Decomposition

In Table 7, we can represent the function without increasing the columns. However, in general, we must increase the columns to represent the function. Since each column has at most one non-zero element after the row-shift operations, at least $k$ columns are necessary to represent a function with weight $k$. We use the **first-fit method** [24], which is simple and efficient.

**Algorithm 4.2:** (Find row-shifts) [18]

1. Sort the rows in decreasing order by the number of non-zero elements.
2. Compute the row-shift value for each row at a time, where the row displacement $r(i)$ for row $i$ has the smallest value such that no non-zero element in row $i$ is in the same position as any non-zero element in the previous rows.
3. Terminate.

When the distribution of non-zero elements among the rows is uniform, Algorithm 4.2 reduces the memory size effectively. To reduce the total amount of memories, we use the following:

**Algorithm 4.3:** (Row-shift decomposition) [18]

1. Reduce the number of variables by the method [20]. If necessary, use a linear transformation [19] to further reduce the number of the variables. Let $l$ be the number of variables after reduction.
2. Let $q_1 \leftarrow \lceil \frac{l}{2} \rceil$. From $t = -2$ to $t = 2$, perform Steps 3 through 6.
3. Partition the inputs $X$ into $(X_1, X_2)$[†], where $X_1 = (x_p, x_{p-1}, \ldots, x_1)$ denotes the rows, and $X_2 = (x_n, x_{n-1}, \ldots, x_{p+1})$ denotes the columns.
4. $p \leftarrow q_1 + t$.
5. Obtain the row-shift value by Algorithm 4.2.
6. Obtain the maximum of the shift value, and compute the total amount of memories.
7. Find $t$ that minimizes the total amount of memories.
8. Terminate.

## 5. Parallel IGU Using Complete Pipelines

### 5.1 IP Lookup Architecture Using Complete Pipelines

To archive high throughput, we used a complete pipeline in the parallel IGU. As for the priority encoder shown in Fig. 3, we implemented a cascade of maximum selectors instead of a tree structure, which tends to be a critical path. Figure 10 shows the parallel IGU using a complete pipeline. Although the latency for the pipelined architecture is larger than the tree structure, its throughput can be higher. In the IP lookup application, the high-throughput is desired even if it increases latency.

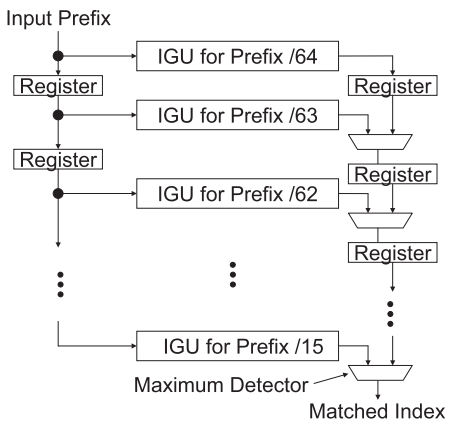
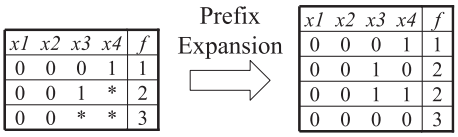



**Fig. 10** Parallel IGU using a complete pipeline.

[†]In the row-shift decomposition, we can assume that non-zero elements are uniformly distributed in the decomposition chart. In the IPv6 prefix, distribution of non-zero elements is almost uniform. Thus, unlike ordinary functional decompositions, the influence of the partition $(X_1, X_2)$ is relatively small.




| x1 | x2 | x3 | x4 | f |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | * | 2 |
| 0 | 0 | * | * | 3 |

| x1 | x2 | x3 | x4 | f |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 2 |
| 0 | 0 | 0 | 0 | 3 |

**Fig. 11** Example of prefix expansion.

### 5.2 Prefix Expansion and Merge

Let $P_i$ be the set of the prefixes with length $i$, and $\mathcal{P} = \{P_1, P_2, \ldots P_l\}$. As shown in Fig. 10, the parallel IGU consists of $l$ IGUs. Thus, the straightforward implementation requires many IGUs. To reduce the number of IGUs, we merge sets of prefixes into fewer groups. By expanding the prefixes in $P_i$ to ones with length $i+1$, we can make a group that includes both $P_{i+1}$ and $P_i$. We call this **prefix expansion and merge**. The next example illustrates this.

**Example 5.8:** The left-hand side of the table in Fig. 11 consists of three groups: $\{P_2, P_3, P_4\}$, where $P_2 = \{00{**}\}$, $P_3 = \{001*\}$, and $P_4 = \{0001\}$. By performing prefix expansion to $P_2$, we have $P'_3 = \{000*, 001*\}$. By the longest prefix matching (LPM) rule, the prefix $\{001*\}$ in $P'_3$ that is equal to $\{001*\}$ in $P_3$ is ignored. Also, by performing prefix expansion to $P'_3$, we have $P'_4$ shown in the right-hand side of the table in Fig. 11. In this case, three groups are merged into one. ■

### 5.3 Design of Parallel IGU Using Off-Chip Memories

Figure 2 shows that the variance of the prefix of lengths is quite large. When the prefix expansions with a small number of prefixes is applied, they can often be stored into a single BRAM†. On the other hand, when the prefix expansion is applied to a set with a large number of prefixes, in the worst case, the size would be too large to be stored in a BRAM. Thus, we make a **non-uniform grouping** $\mathcal{G} = \{G_1, G_2, \ldots, G_r\}$, where $G_i$ may be generated from the different number of sets. As for the large groups $G_i$, we realize them by off-chip memories. To find a good grouping, we use the following:

**Algorithm 5.4:** (Non-uniform grouping) Let $M_{off}$ be the size of off-chip memory, $M_{on}$ be the size of on-chip memory, $P_i$ be the set of the prefixes with length $i$, $\mathcal{P} = \{P_1, P_2, \ldots, P_l\}$ be the set of the prefixes, $G_j$ consists of single or several $P_i$s, $r$ be the number of reduced groups, and $\mathcal{G} = \{G_1, G_2, \ldots, G_r\}$, where $r \leq l$.

1 $Mem \leftarrow Mem_{off}$, $\mathcal{G} \leftarrow \emptyset$.
2 $G_{merge} \leftarrow \emptyset$.
3 Select $P_i$ in $\mathcal{P}$, which has maximum the number of prefixes, $g \leftarrow P_i$, $\mathcal{P} \leftarrow \mathcal{P} - \{P_i\}$.
4 Apply Algorithms 4.1 and 4.3 to $G_{merge} \cup P_i$ to generate the IGU. Let $Mem_{IGU}$ be the amount of memory to realize the IGU.
5 **if**($Mem_{IGU} \leq Mem$)**begin**
  $G_{merge} \leftarrow G_{merge} \cup \{g\}$.
  **if**($\mathcal{P} = \emptyset$)**begin**
    $\mathcal{G} \leftarrow \mathcal{G} \cup \{G_{merge}\}$, and go to Step 7.
  **end else begin**
    Go to Step 3.
  **end**
**end**
6 **if**($G_{merge} \neq \emptyset$)**begin**
  $\mathcal{G} \leftarrow \mathcal{G} \cup \{G_{merge}\}$, and go to Step 2.
**end else if**($Mem = Mem_{off}$)**begin**
  $Mem \leftarrow Mem_{on}$, and go to Step 2.
**end else begin**
  Printout "Failed to generate IGU".
**end**
7 Terminate.

†For Xilinx Virtex 6 FPGA, the BRAM stores 36Kbits.

## 6. Experimental Results

### 6.1 Implementation Environment

We designed the parallel IGU using Xilinx ISE 14.7, and implemented on the ROACH2 board (FPGA: Xilinx Virtex 6 (XC6VSX475T), 74,400 Slices, 1,064 BRAMs (36Kb)). To make a fair comparison with [12], pseudo IPv6 prefixes were generated from the present raw 340 K IPv4 prefixes (May 23, 2013) using the method in [26].

### 6.2 Comparison of Grouping Methods

Table 8 shows numbers of BRAMs in IGUs to load 340 K pseudo IPv6 prefixes generated by Algorithm 5.4. Also, it shows the computation time for Algorithm 5.4. In the experiment, we used a PC with INTEL Xeon X5570 CPU, 2.93 GHz, and 32 GB RAM, on Ubuntu 12.04 LTS Operating System. Table 9 compares proposed non-uniform grouping with previous one [12] and a direct realization (in other word, without grouping). In Table 9, *#Slices* includes the numbers of slices for both IGUs and the priority encoder. Table 9 shows that, although the proposed non-uniform grouping requires off-chip memories, the proposed grouping requires fewer BRAMs than the existing grouping.

### 6.3 Implementation of the Parallel IGU

To utilize the FPGA resources efficiently, we implemented small memory parts (marked with "*" in Table 8) by distributed RAMs [27] instead of BRAMs. Figure 12 shows the implemented parallel IGU. To increase the system throughput, we set a dual port mode to on-chip memories, and implemented a pair of IGUs. In the implementation, the proposed parallel IGU consumed 5,278 Slices and 227 BRAMs. The system used four DDRII+ Static RAMs (SRAMs) [4]. Total amount of memory usage is 25.4 Mbits. From the logic synthesis (XST), the maximum clock frequency was 571.1 MHz. We set the timing constraint (system

**Table 10** Comparison with existing FPGA implementations.

| Architecture | #prefixes | #Slices | #36Kb BRAMs | Off-chip SRAM [Mb] | Normalized FPGA area | | Speed [GLPS] |
|---|---|---|---|---|---|---|---|
| | | | | | #Slices | #BRAMs | |
| Baboescu et al. (ISCA2005) [1] | 80 K | 1,405 | 530 | — | 17.5 | 6.6 | 0.125 |
| Fadishei et al. (ANCS2005) [5] | 80 K | 14,274 | 254 | — | 178.4 | 3.1 | 0.263 |
| Le et al. (FCCM2009) [10] | 249 K | 16,617 | 473 | — | 66.7 | 1.8 | 0.340 |
| 2-3-tree-IPv6 (IEEE Trans.2012) [7] | 330 K | 15,358 | 580 | 32.5 | 46.5 | 1.8 | 0.373 |
| BST-IPv6 (IEEE Trans.2012) [7] | 330 K | 14,096 | 1,025 | 3.2 | 42.7 | 3.1 | 0.390 |
| On-chip Parallel IGUs (ARC2013) [12] | 340 K | 5,577 | 575 | — | 16.4 | 1.7 | 1.002 |
| Proposed Parallel IGUs | 340 K | 5,278 | 227 | 25.4 | **15.5** | **0.6** | **1.100** |

**Table 8** Numbers BRAMs to realize IGUs with non-uniform grouping (BRAMs marked with "*" were realized by distributed RAMs in the actual implementation).

| Group | #prefixes in a group | Memory $H$ | | Memory $G$ and AUX | | #36Kb BRAMs | DDRII+ SRAM [Mb] | Compt. Time [msec] |
|---|---|---|---|---|---|---|---|---|
| | | #In | #Out | #In | #Out | | | |
| (15,16,17,18) | 102 | 4 | 6 | 7 | 24 | 2* | | 1 |
| (19,20,21,22) | 225 | 7 | 7 | 8 | 29 | 2* | | 3 |
| (23,24,25,26) | 1,571 | 10 | 11 | 11 | 37 | 3* | | 18 |
| (27,28) | 806 | 6 | 11 | 11 | 39 | 3* | | 9 |
| (29,30) | 1,240 | 6 | 12 | 12 | 42 | 5* | | 15 |
| (31) | 2,824 | 9 | 12 | 12 | 43 | 5* | | 33 |
| (32) | 8,474 | 9 | 14 | 14 | 46 | 16 | | 98 |
| (33) | 1,469 | 8 | 11 | 11 | 44 | 3* | | 17 |
| (34) | 4,408 | 10 | 12 | 12 | 46 | 5* | | 51 |
| (35) | 2,318 | 10 | 11 | 13 | 46 | 9 | | 27 |
| (36) | 6,957 | 11 | 13 | 13 | 49 | 9 | | 80 |
| (37) | 4,079 | 13 | 12 | 12 | 49 | 8 | | 47 |
| (38) | 12,237 | 14 | 14 | 14 | 52 | 24 | | 141 |
| (39) | 6,592 | 12 | 12 | 13 | 51 | 11 | | 76 |
| (40) | 19,776 | 13 | 14 | 14 | 54 | 22 | | 228 |
| (41) | 6,874 | 13 | 13 | 13 | 54 | 12 | | 79 |
| (42) | 20,623 | 14 | 15 | 15 | 57 | 42 | | 237 |
| (43) | 9,451 | 14 | 14 | 14 | 57 | 27 | | 109 |
| (44) | 28,354 | 13 | 15 | 15 | 59 | 47 | | 326 |
| (45,46,47) | 123,110 | 15 | 17 | 17 | 64 | | 12.6 | 1,416 |
| (48) | 128,305 | 14 | 17 | 17 | 65 | | 12.8 | 1,475 |
| (49,50) | 929 | 10 | 10 | 10 | 60 | 3* | | 11 |
| (51,52) | 1,048 | 11 | 11 | 11 | 63 | 4* | | 12 |
| (53,54) | 594 | 9 | 10 | 10 | 64 | 3* | | 7 |
| (55,56) | 421 | 8 | 9 | 9 | 65 | 2* | | 5 |
| (57,58) | 530 | 9 | 9 | 9 | 67 | 2* | | 6 |
| (59,60,61,62) | 289 | 7 | 8 | 9 | 70 | 2* | | 3 |
| (63,64) | 386 | 8 | 9 | 9 | 73 | 2* | | 5 |
| **Total** | 398,880 | | | | | 274 | 25.4 | |

**Table 9** Comparison of non-uniform groupings with uniform grouping.

| Grouping | #prefixes | #groups | #BRAMs | #Slices | Off-chip Mem [Mb] |
|---|---|---|---|---|---|
| Uniform grouping (direct realization of $\mathcal{P}$) | 348,877 | 50 | 655 | 3,979 | 0 |
| Non-uniform [12] (On-chip mems only) | 347,749 | 30 | 616 | 2,299 | 0<br>0 |
| Non-uniform (Table 8) (On-chip and Off-chip mems) | 398,880 | 28 | 274 | 1,927 | 25.4 |

clock frequency) to 550 MHz which is supported by the DDRII+ SRAMs. After the place-and-root (PAR), the timing constraint met 550 MHz. Thus, the lookup speed was 1.100 GLPS which satisfies the requirement of 400 Gbps link throughput.

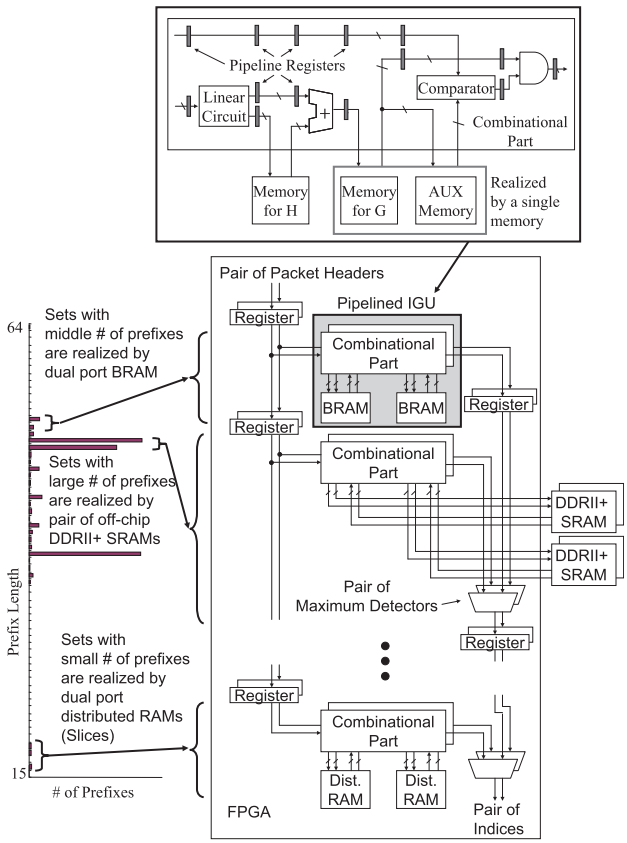



**Fig. 12** Implemented parallel IGU for IPv6 prefix lookup.

## 6.4 Comparison with Existing FPGA Implementations

Table 10 compares the proposed parallel IGU with existing FPGA implementations. Since they stored different numbers of prefixes, we used the normalized FPGA area, which shows the necessary number of FPGA primitives (# of slices or BRAMs) per 1 K prefixes. Also, it compares the amount of off-chip SRAMs. Table 10 shows that, as for the lookup speed, the proposed parallel IGU is faster than other implementations. Especially, it just satisfies the requirement of 1.087 GLPS, which is requirement of 400 GLPS link throughput. As shown in Fig. 3, the on-chip parallel IGU (previous method) [12] used a priority encoder with a tree-structure, which was a critical path. On the other hand, as shown in Fig. 12 in the proposed parallel IGU, we implemented it by a cascaded, which is suitable for a complete

pipeline architecture. Table 10 showed that, as for the normalized FPGA area, the proposed parallel IGU is the most efficient implementation. As for the off-chip SRAM, the BST-IPv6 required smaller memory than our architecture. However, since the price of the FPGA dominates that of SRAM, the system cost of proposed parallel IGU is lower than the BST-IPv6. Therefore, the parallel IGU outperforms existing FPGA realizations.

## 7. Conclusion

This paper showed that the parallel IGU using both on-chip and off-chip memories to reduce the system cost. To reduce the memory size of the IGU, we used linear transformation and row-shift decomposition. Also, this paper showed a method to store prefixes in given memories. We implemented the proposed parallel IGU using a complete pipeline on the Xilinx Virtex 6 FPGA and four off-chip DDRII+ SRAMs. Its lookup speed was 1.100 GLPS which exceeded 1.087 GLPS at 400 Gbps link throughput. Thus, our architecture fits to the next generation standard of link throughput. Experimental results showed that the proposed parallel IGU outperforms existing FPGA implementations.

Since Internet routers are updated frequently, the parallel IGU also must be updated. When the new prefix collides with existing ones, the corresponding IGU should be reconfigured. As shown in Table 8, as for the worst case, its down time is about 1.5 second. Luo et al. proposed a hybrid architecture using a memory-based lookup engine and a small TCAM [11] without down time for the update. The parallel IGU is also applicable to such architecture. The future project is an implementation of Luo's hybrid architecture using the parallel IGU.

## Acknowledgments

This research is supported in part by the Grants in Aid for Scientific Research of JSPS, and the Adaptable and Seamless Technology Transfer Program through target-driven R&D of JST. Reviewer's comments were quite useful to improve the presentation.

### References

[1] F. Baboescu, D.M. Tullsen, G. Rosu, and S. Singh, "A tree based router search engine architecture with single port memories," ISCA2005, p.123, 2005.
[2] M. Bando, L. Yi-Li, and H.J. Chao, "FlashTrie: Beyond 100-Gb/s IP route lookup using hash-based prefix-compressed trie," IEEE Trans. Netw., vol.4, no.20, pp.1262–1275, 2012.
[3] H.J. Chao and B. Liu, High performance switches and routers, John Wiley & Sons, Hoboken, NJ, USA, 2007.
[4] Cypress Inc., "DDR II+ Static RAM (SRAM)," http://www.cypress.com/
[5] H. Fadishei, M.S. Zamani, and M. Sabaei, "A novel reconfigurable hardware architecture for IP address lookup," ANCS2005, pp.81–90, 2005.
[6] IEEE 802.3 400 Gbps Ethernet study group 2013, http://www.ieee802.org/3/400GSG/index.html
[7] H. Le and V.K. Prasanna, "Scalable tree-based architectures for IPv4/v6 lookup using prefix partitioning," IEEE Trans. Comput,, vol.61, no.7, pp.1026–1039, 2012.
[8] S. Kumar, M. Becchi, P. Crowley, and J.S. Turner, "CAMP: Fast and efficient IP lookup architecture," ANCS2006, pp.51–60, 2006.
[9] H. Le, T. Ganegedara, and V.K. Prasanna, "Memory-efficient and scalable virtual routers using FPGA," FPGA2011, pp.257–266, 2011.
[10] H. Le and V.K. Prasanna, "Scalable high throughput and power efficient IP-lookup on FPGA," FCCM2009, pp.167–174, April, 2009.
[11] L. Luo, G. Xie, Y. Xie, L. Mathy, and K. Salamatian, "A hybrid IP lookup architecture with fast updates," INFOCOM2012, pp.2435–2443, 2012.
[12] H. Nakahara, T. Sasao, and M. Matsuura, "An architecture for IPv6 lookup using parallel index generation units," LNCS7806, pp.59–71, 2013.
[13] F. Pong and N.F. Tzeng, "Concise lookup tables for IPv4 and IPv6 longest prefix matching in scalable routers," IEEE Trans. Netw., vol.20, no.3, pp.729–741, 2012.
[14] 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.
[15] University of Oregon Route Views Project, http://www.routeviews.org/
[16] R. Sangireddy and A.K. Somani, "High-speed IP routing with binary decision diagrams based hardware address lookup engine," IEEE J. Sel. Areas Commun., vol.21, no.4, pp.512–521, 2006.
[17] R. Sangireddy, N. Futamura, S. Aluru, and A.K. Somani, "Scalable, memory efficient, high-speed IP lookup algorithms," IEEE Trans. Netw., vol.13, no.4, pp.802–812, 2005.
[18] T. Sasao, "Row-shift decompositions for index generation functions," DATE2012, pp.1585–1590, 2012.
[19] T. Sasao, "Linear decomposition of index generation functions," ASPDAC2012, pp.781–788, 2012.
[20] T. Sasao, Memory-Based Logic Synthesis, Springer, 2011.
[21] T. Sasao, M. Matsuura, and H. Nakahara, "A realization of index generation functions using modules of uniform sizes," IWLS2010, pp.201–208, June 2010.
[22] T. Sasao and J.T. Butler, "Implementation of multiple-valued CAM functions by LUT cascades," ISMVL2006, pp.1–7, Singapore, May 2006.
[23] H. Song, F. Hao, M. Kodialam, and T. Lakshman, "Ipv6 lookups using distributed and load balanced bloom filters for 100Gbps core router line cards," INFOCOM2009, pp.2518–2526, 2009.
[24] R.E. Tarjan and A.C-C. Yao, "Storing a sparse table," Commun. ACM, vol.22, no.11, pp.606–611, 1979.
[25] R. Tucker, "Optical packet-switched WDM networks: A cost and energy perspective," OFC/NFOEC2008, pp.1–25, 2008.
[26] M. Wang, S. Deering, T. Hain, and L. Dunn, "Non-random generator for IPv6 tables," HOTI2004, pp.35–40, 2004.
[27] Xilinx Inc., "Spartan-6 FPGA configurable logic block," http://www.xilinx.com/

**Hiroki Nakahara** 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, Japan. He 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. His research interests include logic synthesis, reconfigurable architecture, digital signal processing and embedded systems. He is a member of the IEEE, the ACM, and the IEICE.

**Tsutomu Sasao** 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 Height, NY and the Naval Postgraduate School, Monterey, CA. 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 of 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 nine 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. He 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*. He is a Fellow of the IEEE.

**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, in Japan, 2003. He has been working as a Technical Assistant at the Kyushu Institute of Technology since 1991. He has implemented several logic design algorithms under the direction of Professor Tsutomu Sasao. His interests include decision diagrams and exclusive-OR based circuit design.

**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 from 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. He is a member of IEICE.

**Yasuhiro Terao** graduated from Nagasaki technical high school in 1992. He joined the Sony LSI design Inc., Japan in 1992. He transferred to REVSONIC Corp. in 2013. He is currently interested in designs for DDR3 SRAM. SDRAMs.