Implementation of LPM Address Generators on FPGAs

Hui Qin[1], Tsutomu Sasao[1], and Jon. T. Butler[2]

[1] Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502, Japan
[2] Department of Electrical and Computer Engineering, Naval Postgraduate School
Code EC/Bu, Monterey, CA 93943-5121

Abstract. We propose the *multiple LUT cascade* as a means to configure an n-input *LPM (Longest Prefix Match) address generator* commonly used in routers to determine the output port given an address. The LPM address generator accepts n-bit addresses which it matches against k stored prefixes. We implement our design on a Xilinx Spartan-3 FPGA for $n = 32$ and $k = 504 \sim 511$. Also, we compare our design to a Xilinx proprietary TCAM (ternary content-addressable memory) design and to another design we propose as a likely solution to this problem. Our best multiple LUT cascade implementation has 5.20 times more throughput, 31.71 times more throughput/area and is 2.89 times more efficient in terms of *area-delay* product than Xilinx's proprietary design. Furthermore, its area is only 19% of Xilinx's design.

1 Introduction

The need for higher internet speeds is likely to be the subject of intense interest for many years to come. A network's speed is directly related to the speed with which a node can switch a packet from an input port to an output port. This, in turn, depends on how fast a packet's address can be accessed in memory. The **longest prefix match** (**LPM**) problem is one of determining the output port address from a list of **prefix vectors** stored in memory. For example, if the prefix vector 01001**** is stored in memory, then the packet address 010011111 matches this entry. That is, each bit in the packet address matches exactly the corresponding digit in the prefix vector or there is a * or *don't care* in that digit. If other stored prefixes match the packet address, then the prefix with the least *don't care* values determines the output port address. That is, the memory entry corresponding to the longest prefix match determines the output port.

An ideal device for this application is a **ternary content-addressable memory** (TCAM). The descriptor "ternary" refers to the three values stored, 0, 1, and *. Unfortunately, TCAM dissipates much more power than standard RAM [1].

Several authors have proposed the use of standard RAM in LPM design. Gupta, Lin, and McKeown showed a mechanism to perform LPM every memory access [2]. Dharmapurikar, Krishnamurthy, and Taylor propose the use of Bloom filters to solve the LPM problem [3]. Sasao and Butler have shown that a fast, power-efficient TCAM realization using a look-up table (LUT) cascade [4].

In this paper, we propose an extension to the LUT cascade realization: a *multiple LUT cascade* realization that consists of multiple LUT cascades connected to a special

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **MAR 2006**	2. REPORT TYPE	3. DATES COVERED
4. TITLE AND SUBTITLE **Implementation of LPM Address Generators on FPGAs**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Department of Electrical and Computer Engineering,Monterey,CA,93943**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
We propose the multiple LUT cascade as a means to configure an ninput LPM (Longest Prefix Match) address generator commonly used in routers to determine the output port given an address. The LPM address generator accepts n-bit addresses which it matches against k stored prefixes. We implement our design on a Xilinx Spartan-3 FPGA for n = 32 and k = 504 &#8764; 511. Also, we compare our design to a Xilinx proprietary TCAM (ternary content-addressable memory) design and to another design we propose as a likely solution to this problem. Our best multiple LUT cascade implementation has 5.20 times more throughput, 31.71 times more throughput/area and is 2.89 times more efficient in terms of area-delay product than Xilinx?s proprietary design. Furthermore, its area is only 19% of Xilinx?s design.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**12**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

encoder. This offers even more efficient realizations in an architecture that is more easily reconfigured when additional prefix vectors are placed in the prefix table.

We have implemented six types of LPM address generators on the Xilinx Spartan-3 FPGA (XC3S4000-5): Four different realizations using multiple LUT cascades, one using Xilinx's TCAM realization based on the Xilinx IP core, and one using registers and gates. In addition, we compare the six types of LPM address generators on the basis of delay, *delay-area* product, throughput, throughput/area, and FPGA resources used.

The rest of the paper is organized as follows: Section 2 describes the multiple LUT cascade. Section 3 shows other realizations for the LPM address generators. Section 4 presents the implementations of the LPM address generator using an FPGA. Section 5 shows the experimental results. Section 6 concludes the paper.

2 Multiple LUT Cascades

2.1 LPM Address Generators

A content-addressable memory (CAM) [5] stores 0's and 1's and produces the address of the given data. A TCAM, unlike a CAM, stores 0's, 1's, and *'s, where * is a *don't care* value that matches both 0 and 1.

TCAMs are extensively used in routing tables for the internet. A routing table specifies an interface identifier corresponding to the longest prefix that matches an incoming packet, in a process called **Longest Prefix Match (LPM)**. In the PLM table, the ternary vectors have restricted patterns: the prefix consists of only 0's and 1's, and postfix consist of only *'s (*don't cares*). In this paper, this type of vector is called a **prefix vector**.

Definition 2.1 *An n-input m-output k-entry* **LPM table** *stores k n-element prefix vectors of the form $VEC_1 \cdot VEC_2$, where VEC_1 is a string of 0's and 1's, and VEC_2 is a string of *'s. To assure that the longest prefix address is produced, TCAM entries are stored in descending prefix length, and the first match determines the LPM table's output. An address is an m-element binary vector for $m = \lceil \log_2(k+1) \rceil$, where $\lceil a \rceil$ denotes the smallest integer greater than or equal to a. The corresponding* **LPM function** *is a logic function $f : B^n \to B^m$, where $f(\vec{x})$ is the smallest address of an entry that is identical to $\vec{x}$ except possibly for don't care values. If no such entry exists, $f(\vec{x}) = 0^m$. The* **LPM address generator** *is a circuit that realizes the LPM function.*

Example 2.1 *Table 1 shows an LPM table with 5 4-element prefix vectors. Table 2 shows the corresponding LPM function. It has 16 entries, one for each 4-bit input. The output address is stored for each input corresponding to the address of the longest prefix vector that matches it. (End of Example)*

2.2 An LUT Cascade Realization of LPM Address Generators

An LPM function can be realized by a single memory. However, this often requires prohibitively large memory size. We propose functional decomposition [6, 7] to realize the LPM function with lower storage requirements. For a given LPM function $f(\vec{x})$,

Table 1. LPM table

Address	Prefix Vector
1	1000
2	010*
3	01**
4	1***
5	0***

Table 2. LPM function

Input	Output Address	Input	Output Address
0000	5	1000	1
0001	5	1001	4
0010	5	1010	4
0011	5	1011	4
0100	2	1100	4
0101	2	1101	4
0110	3	1110	4
0111	3	1111	4

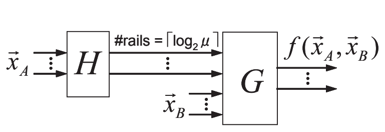
$\vec{x}_A$
H
#rails = $\lceil \log_2 \mu \rceil$
G
$\vec{x}_B$
$f(\vec{x}_A, \vec{x}_B)$

Fig. 1. Decomposition for the LPM function f

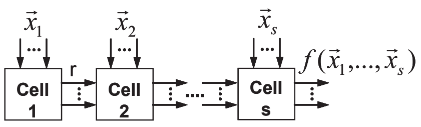

Fig. 2. LUT cascade

let $\vec{x}$ be partitioned as $(\vec{x}_A, \vec{x}_B)$. The decomposition chart of f is a table with 2^{n_A} columns and 2^{n_B} rows, where n_A and n_B are the number of variables in $\vec{x}_A$ and $\vec{x}_B$, respectively. Each column and row is labeled by a binary number, and the corresponding element in the table denotes the value of f. The column multiplicity, μ, is the number of different column patterns of the decomposition chart. Then, using functional decomposition, the function f can be decomposed as $f(\vec{x}_A, \vec{x}_B) = G(H(\vec{x}_A), \vec{x}_B)$, as shown in Fig. 1, where the number of rails (signal lines between two blocks H and G) is $\lceil \log_2 \mu \rceil$. By iterative functional decomposition, the given function can be realized by an LUT cascade, as shown in Fig. 2 [8, 9].

Theorem 2.1 *[4] An n-input LPM address generator with k prefix vectors can be realized by an LUT cascade, where each cell realizes a p-input, r-output combinational logic function. Let s be the necessary number of levels or cells. Then,*

$$s \leq \lceil \frac{n-r}{p-r} \rceil, \tag{1}$$

where $p > r$ and $r = \lceil \log_2(k+1) \rceil$.

2.3 LPM Address Generators Using the Multiple LUT Cascade

A single LUT cascade realization of an LPM function often requires many levels. Since the delay is proportional to the number of levels in a cascade, we wish to reduce the number of levels. According to (1), if we increase p, the number of inputs to each cell, then the number of levels s is reduced. For each increase by 1 of p, the memory needed to realize the cell is doubled. However, as shown in Fig. 3, we can use the multiple LUT cascade to reduce the number of levels s while keeping p fixed. For an n-input LPM function with k prefix vectors, let the number of rails of each LUT cascade be r. First, partition the set of prefix vectors into g groups of $2^r - 1$ vectors each, except the last group, which has $2^r - 1$ or fewer vectors, where $g = \lceil \frac{k}{2^r-1} \rceil$. For each group of prefix vectors, form an independent LPM function. Next, partition the set of n inputs into s groups. Then, realize each LPM function by an LUT cascade. Thus, we need a total of g LUT cascades, and each LUT cascade consists of s cells. Finally, use a **special encoder** to produce the LPM address. Let v_i $(i = 1, 2, ..., g)$ be the i-th input of the special encoder, and let v_{out} be the output value of the special encoder. That is, v_i is the output value of the i-th LUT cascade, where its binary output values are viewed as a standard binary number. Similarly, v_{out} is the output of the special encoder, where its binary output values are viewed as a standard binary number. Then, we have the relation:

$$v_{out} = \begin{cases} v_i + (i-1)(2^r - 1) & \text{if } v_i \neq 0 \text{ and } v_j = 0 \text{ for all } 1 \leq j \leq i-1 \\ 0 & \text{if } v_i = 0 \text{ for all } 1 \leq i \leq g. \end{cases}$$

Note that v_{out} is the position of a prefix vector v in the complete LPM table, while i is the index to the LUT cascade storing v. $(i-1)(2^r-1)$ is the position in the LPM table of the last entry of the previous $(i-1)$-th LUT cascade or is 0 in the case of the first LUT cascade. Adding v_i to this yields the position of v in the complete LPM table.

Example 2.2 *Consider an n-input LPM function with k prefix vectors. When $k = 1000$ and $n = 32$, by Theorem 2.1, we have $r = 10$. Let $p = r + 1 = 11$. When we use a single LUT cascade to realize the function, by Theorem 2.1, we need $\lceil \frac{n-r}{p-r} \rceil = 22$ cells, and the number of levels of the LUT cascade is also 22. Since each cell has 11 address lines and 10 outputs, the total amount of memory needed to realize the cascade is $2^{11} \times 10 \times 22 = 450,560$ bits. Note that the memory size of each cell, $2^{11} \times 10 = 20,480$ bits, is too large to be realized by a single block RAM (BRAM) of our FPGA, which stores $18,432$ bits.*

However, if we use a multiple LUT cascade to realize the function, we can reduce the number of levels and the total amount of memory. Also, the cells will fit into the BRAMs in the FPGAs. Partition the set of vectors into two groups, and realize each group independently; then, we need two LUT cascades. For each LUT cascade, the number of vectors is 500, so we have $r = 9$. Also, let $p = r + 2 = 11$. Then, we need $\lceil \frac{n-r}{p-r} \rceil = 12$ cells in each cascade. Note that the number of levels of the LUT cascades is 12, which is smaller than the 22 needed in the single LUT cascade realization. Since each cell consists of a memory with 9 outputs and at most 11 address lines, the total amount of memory is at most $2^{11} \times 9 \times 12 \times 2 = 442,368$ bits. Also, note that the size of the memory for a single cell is $2^{11} \times 9 = 18,432$ bits. This fits exactly in the BRAMs of the FPGAs.

Thus, the multiple LUT cascade not only reduces the number of levels and the total amount of memory, but also adjusts the size of cells to fit into the available memory in the FPGAs. *(End of Example)*

Fig. 3 shows the architecture of the multiple LUT cascade. The realization with this architecture is the **multiple LUT cascade realization**. It consists of a group of LUT cascades and a special encoder. The inputs of each LUT cascade are common with other LUT cascades, while the outputs of each LUT cascade are connected to the

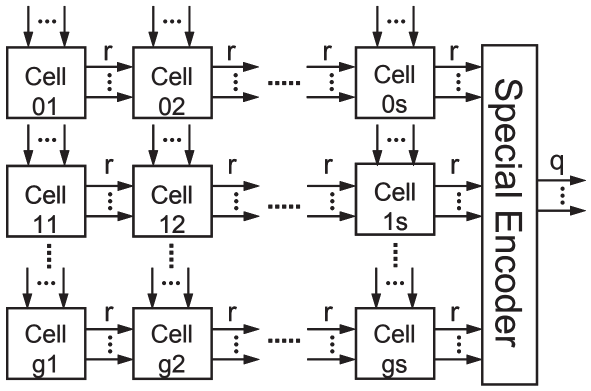

Fig. 3. Architecture of the multiple LUT cascade

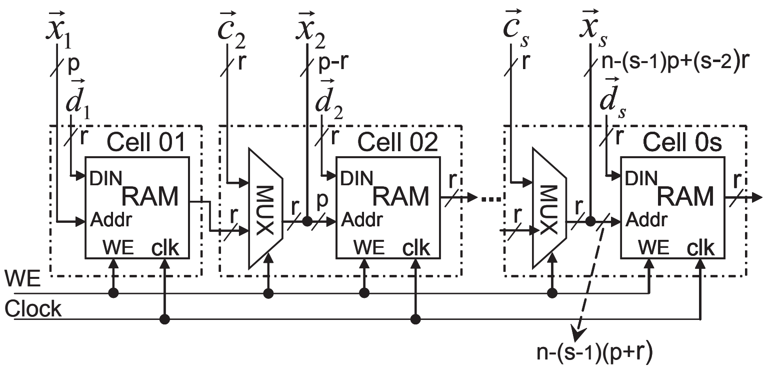

Fig. 4. Detailed design of the LUT cascade

special encoder. Each LUT cascade realizes an LPM function, while the special encoder generates the LPM address from the outputs of cascades.

For an n-input LPM function with k prefix vectors, the detailed design of the LUT cascade is shown in Fig. 4, where $\vec{x}_i$ $(i = 1, 2, ..., s)$ denotes the primary inputs to the i-th cell, $\vec{d}_i$ $(i = 1, 2, ..., s)$ denotes the data inputs to the i-th cell and provides the data value to be written in the RAM of the i-th cell, r denotes the number of rails, where $r \leq \lceil \log_2(k+1) \rceil$, $\vec{c}_j$ $(j = 2, 3, ..., s)$ denotes the additional inputs to the j-th cell and is used to select the RAM location along with $\vec{x}_j$ for write access. Note that $\vec{c}_j$ and $\vec{d}_i$ are represented by r bits. All RAMs except perhaps the last one have p address lines; the last RAM has at most p address lines. When *WE* is high, the $\vec{c}_j$ is connected to the RAM to write the data into the RAMs. When *WE* is low, the outputs of the RAMs are connected to the inputs of the succeeding RAMs, and the circuit works as a cascade to

Table 3. 6-entry LPM table

Address	Prefix Vector
1	100000
2	10010*
3	1010**
4	101***
5	10****
6	1*****

Table 4. Truth table for the corresponding LPM function

Input						Output			LUT
x_1	x_2	x_3	x_4	x_5	x_6	out_2	out_1	out_0	Cascade
1	0	0	0	0	0	0	0	1	Upper
1	0	0	1	0	*	0	1	0	Cells 1
1	0	1	0	*	*	0	1	1	and 2
1	0	1	1	*	*	1	0	0	Lower
1	0	0	*	*	*	1	0	1	Cells 3
1	1	*	*	*	*	1	1	0	and 4

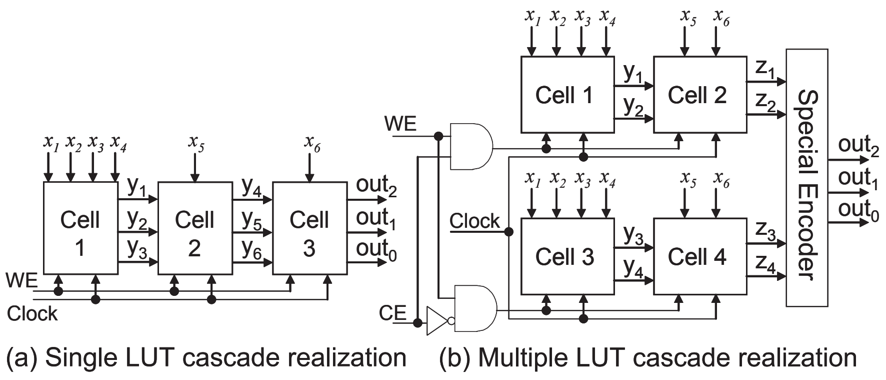

Fig. 5. Single LUT cascade realization and the multiple LUT cascade realization

realize the LPM function. Note that the RAMs are synchronous RAMs. Therefore, the LUT cascade resembles a shift register.

Example 2.3 *Table 3 shows a* 6*-input* 3*-output* 6*-entry LPM table, and the corresponding LPM function is shown in Table 4. Note that the entries in the two tables are similar. Table 4 is a compact truth table, showing only non-zero outputs. Its input combinations are disjoint. Thus, the two tables are the same except for three entries.*

Single Memory Realization: *The number of address lines is* 6*, and the number of outputs is* 3*. Thus, the total amount of memory is* $2^6 \times 3 = 192$ *bits.*

Single LUT Cascade Realization: *Since there are* $k = 6$ *prefix vectors of the function, by Theorem 2.1, the number of rails is* $r = \lceil \log_2 (6+1) \rceil = 3$*. Let the number of address lines for the memory in a cell be* $p = 4$*. By partitioning the inputs into three disjoint sets* $\{x_1, x_2, x_3, x_4\}$*,* $\{x_5\}$*, and* $\{x_6\}$*, we have the cascade in Fig. 5 (a), where only the signal lines for cascade realization are shown, and other lines such as for storing data are omitted for simplicity.*

The total amount of memory is $2^4 \times 3 \times 3 = 144$ *bits, and the number of levels is* $s = 3$*. Note that the single LUT cascade requires 75% of the memory needed in the single memory realization.*

Table 5. Truth tables for the cells in the multiple LUT cascade realization

Cell 1 and Cell 2 (upper LUT cascade)												Cell 3 and Cell 4 (lower LUT cascade)											
x_1	x_2	x_3	x_4	y_1	y_2	x_5	x_6	z_1	z_2	v_1	v_{out}	x_1	x_2	x_3	x_4	y_3	y_4	x_5	x_6	z_3	z_4	v_2	v_{out}
1	0	0	0	0	0	0	0	0	1	1	001	1	0	1	1	0	0	*	*	0	1	1	100
1	0	0	1	0	1	0	*	1	0	2	010	1	0	0	*	0	1	*	*	1	0	2	101
1	0	1	0	1	0	*	*	1	1	3	011	1	1	*	*	1	0	*	*	1	1	3	110
Other values				1	1	*	*	0	0	0	†	Other values				1	1	*	*	0	0	0	†
				Other values				0	0	0	†					Other values				0	0	0	†

† depends on values from the other LUT cascade

Multiple LUT cascade Realization: *Partition Table 3 into two parts, each with three prefix vectors. The number of rails in the LUT cascades associated with each separate LPM table is* $\lceil \log_2 (3+1) \rceil = 2$*. Let the number of address lines for the memory in a cell be* $p = 4$*. By partitioning the inputs into two disjoint sets* $\{x_1, x_2, x_3, x_4\}$ *and* $\{x_5, x_6\}$*, we obtain the realization in Fig. 5 (b). The upper LUT cascade realizes the upper part of the Table 4, while the lower LUT cascade realizes the lower part of the Table 4. The contents of each cell is shown in Table 5.*

Let v_1 *be the output value of the upper LUT cascade, let* v_2 *be the output value of the lower LUT cascade, and let* v_{out} *be the output value of the special encoder. Then, in Table 5,* (z_1, z_2) *viewed as a standard binary number, has value* v_1*, while* (z_3, z_4) *viewed as a standard binary number, has value* v_2*. The special encoder generates the LPM address from the pair of outputs,* (z_1, z_2) *and* (z_3, z_4) :

$$\begin{aligned} out_2 &= \bar{z}_1 \bar{z}_2 (z_3 \vee z_4), \\ out_1 &= z_1 \vee \bar{z}_2 z_3 z_4, \\ out_0 &= z_2 \vee \bar{z}_1 z_3 \bar{z}_4. \end{aligned}$$

Note that (out_2, out_1, out_0) *viewed as a standard binary number, has value* v_{out} *corresponding to the address in Table 3. The total amount of memory is* $2^4 \times 2 \times 4 = 128$ *bits, and the number of levels is* 2*. Note that the multiple LUT cascade realization requires 89% of the memory and one fewer levels than the single LUT cascade realization.*
(End of Example)

3 Other Realizations

3.1 Xilinx's TCAM

Xilinx [10] provides a proprietary realization of a TCAM that is produced by the Xilinx CORE Generator tool [11]. Since a TCAM can directly realize an LPM address generator, we compare our proposed multiple LUT cascade realization with Xilinx's TCAM. In the Xilinx CORE Generator 7.1i, we used the following parameters to produce TCAMs.

– *SRL16 implementation.*

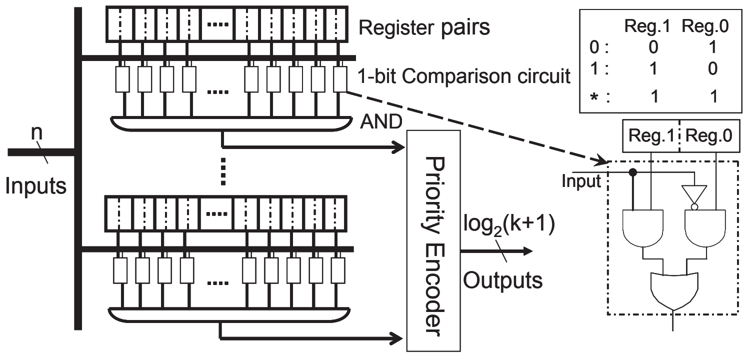

Fig. 6. Realize the address generator with registers and gates

- *Standard Ternary Mode*: Generate a standard ternary CAM.
- *Depth*- Number of words (vectors) stored in the TCAM: k.
- *Data width*- Width of the data word (vector) stored in the TCAM: n.
- *Match Address Type*- Three options: Binary Encoded, Single-match Unencoded, and Multi-match Unencoded. We used the Binary Encoded option.
- *Address Resolution*- Lowest or Highest. We used the Lowest option.

3.2 Registers and Gates

We also compare our proposed multiple LUT cascade realization with a direct realization using registers and gates, as shown in Fig 6. We use a register pair (Reg. 1 and Reg. 0) to store each digit of a ternary vector. For example, if the digit is * (*don't care*), the register pair stores (1,1). Thus, for n bit data, we need a $2n$-bit register. The comparison circuit consists of an n-input AND gate and n 1-bit comparison circuits, each of which produces a 1 if and only if the input bit matches the stored bit or the stored bit is *don't care* (* or 11).

For each prefix vector of an n-input LPM address generator, we need a $2n$-bit register, n copies of 1-bit comparison circuits, and an n-input AND gate. For an n-input address generator with k registered prefix vectors, we need k copies of $2n$-bit registers, nk copies of 1-bit comparison circuits, and k copies of n-input AND gates. In addition, we need a priority encoder with k inputs and $\lceil \log_2 (k+1) \rceil$ outputs to generate the LPM address. If the n-input AND gate is realized as a cascade of 2-input AND gates, this circuit can be considered as a special case of the multiple LUT cascade architecture, where $r = 1$, $p = 2$, and $g = k$. Note that the output encoder circuit is a standard priority encoder.

4 FPGA Implementations

We implemented the LPM address generators for 32 inputs and 504$\sim$511 registered prefix vectors on Xilinx Spartan-3 FPGAs (XC3S4000-5) [12] by using the multiple

Table 6. Four Multiple LUT Cascade Realizations

Design	**Number of prefix vectors**	$\boldsymbol{r}$	$\boldsymbol{p}$	**Group**	**Level**
$r6p11$	504	6	11	8	6
$r7p11$	508	7	11	4	7
$r8p11$	510	8	11	2	8
$r9p11$	511	9	11	1	12

$\boldsymbol{r}$: Number of rails
$\boldsymbol{p}$: Number of address lines of the RAM in a cell
Group: Number of LUT cascades

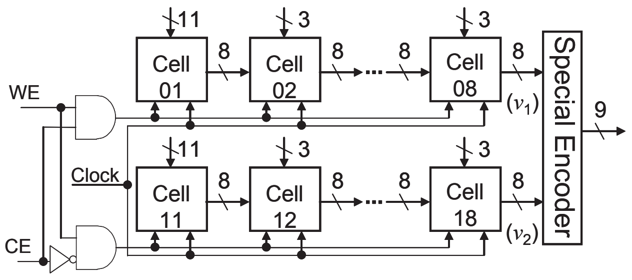

Fig. 7. Architecture of $r8p11$

LUT cascade, Xilinx CORE Generator 7.1i, and registers & gates. The FPGA device XC3S4000-5 has 96 BRAMs and 27648 slices. Each BRAM contains 18K bits, and each slice consists of two 4-input LUTs, two D-type flip-flops, and multiplexers. For each implementation, we described the circuit by Verilog HDL, and then used Xilinx ISE 7.1i to synthesize and to perform place and route.

First, we used the multiple LUT cascade to realize the LPM address generators. To use the BRAMs in the FPGA efficiently, the memory size of a cell in the LUT cascade should not exceed the BRAM size. Let p be number of address lines of the memory in the cell. Since each BRAM contains $2^{11} \times 9$ bits, we have the relation: $2^p \cdot r \leq 2^{11} \times 9$, where r is the number of rails. Thus, we have $p = \lfloor \log_2 (9/r) \rfloor$+ 11, where $\lfloor a \rfloor$ denotes the largest integer less than or equal to a.

We designed four kinds of LPM address generators $r6p11$, $r7p11$, $r8p11$, and $r9p11$, as shown in Table 6, where the column **Number of prefix vectors** denotes the number of registered prefix vectors, the column $\boldsymbol{r}$ denotes the number of rails, the column $\boldsymbol{p}$ denotes the number of address lines of the RAM in a cell, the column **Group** denotes the number of LUT cascades, and the column **Level** denotes the number of levels or cells in the LUT cascade.

To explain Table 6, consider $r8p11$ which is shown in Fig 7. For $r8p11$, since the number of rails is $r = 8$, the number of groups is $\lceil \frac{510}{2^8-1} \rceil = 2$. Thus, we need two LUT cascades. Since each LUT cascade consists of 8 cells, the number of levels of $r8p11$ is 8. To efficiently use BRAMs in the FPGA, the number of address lines of the RAM in the cell is set to $p = \lfloor \log_2 (9/8) \rfloor$+ 11 = 11. Let v_1 be the values of the outputs of the upper LUT cascade, let v_2 be the values of the outputs of the lower LUT cascade, and let v_{out} be the values of the outputs of the special encoder. Then, we have the relation:

$$v_{out} = \begin{cases} v_2 + 255 & \text{if} \quad v_1 = 0 \text{ and } v_2 \neq 0, \\ v_1 & \text{otherwise.} \end{cases}$$

This expression requires 11 slices to implement on the FPGA. After synthesizing and mapping, $r8p11$ required 16 BRAMs and 69 slices. From this table, we can see that decreasing r, increases the number of groups, but decreases the number of levels.

Next, we used the Xilinx CORE Generator 7.1i tool to produce Xilinx's TCAM. Since the Xilinx CORE Generator 7.1i does not support TCAMs with 32 inputs and 505$\sim$511 registered prefix vectors, we designed a TCAM with 32 inputs and 504 registered prefix vectors. After synthesizing and mapping, the resulting TCAM required 8,590 slices. Note that Xilinx's TCAM requires one clock cycle to find a match.

Finally, we designed the LPM address generator with $n = 32$ inputs and $k = 511$ registered prefix vectors using registers and gates, as shown in Fig 6. This design is denoted **Reg-Gates**. Note that the number of inputs is 32 and the number of outputs is 9. After synthesizing and mapping, this design required 27,646 slices.

5 Performance and Comparisons

In Table 7, we show the performance of multiple LUT cascade realizations (i.e., $r6p11$, $r7p11$, $r8p11$, and $r9p11$), and compare them with Xilinx's TCAM and Reg-Gates. In Table 7, the column **Level** denotes the number of levels or cells in the LUT cascade, the column **Slice** denotes the number of occupied slices, the column **Memory** denotes the amount of memory required, and the column **F_clk** denotes the maximum clock frequency. The column **tco** denotes maximum clock-to-output propagation delay. (It is the maximum time required to obtain a valid output at output pin that is fed by a register after a clock signal transition on an input pin that clocks the register). The column **tpd** denotes the maximum propagation time from the inputs to the outputs. The column **Th.** denotes the maximum throughput. Since the LPM address generator has 9 outputs, it is calculated by:

$$\text{Th.} = 9 \cdot \text{F\_clk}.$$

For Reg-Gates, **Delay** denotes the maximum delay from the input to the output and is equal to **tpd**. For multiple LUT cascade realizations and Xilinx's TCAM, **Delay** denotes the total delay, and is calculated by:

$$\text{Delay} = \frac{1000 \cdot \text{Level}}{\text{F\_clk}} + tco,$$

where 1000 is a unit conversion factor.

Consider the area occupied by the various realizations. From the Spartan-3 family architecture [12], we can see that the area of one BRAM is at least the area of 16 slices (a slice consists of two "4-input LUTs", two flip-flops, and miscellaneous multiplexers).

An alternative estimate shows that the area of one BRAM is equivalent to that of 96 slices, as follows. In the Xilinx Virtex-II FPGA, one "4-input LUT" occupies approximately the same area as 96 bits of BRAM (also containing 18K bits) [13]. Note

Table 7. Comparisons of FPGA implementations of the LPM address generator

Design	**Level**	**Slice**	**Memory**	**F_clk**	**tco/tpd**	**Th.**	**Area**	**Th./Area**	**Delay**	**Area-Delay**
			(BRAM)	(MHz)	(ns)	(Mbps)	(slice)	$(\frac{\text{Mbps}}{\text{slice}})$	(ns)	(slice-ns)
$r6p11$	6	178	48	103.89	24.89 (tco)	935	4786	0.195	82.64	395.53
$r7p11$	7	116	28	113.77	23.46 (tco)	1024	2804	0.365	84.99	238.31
$r8p11$	8	69	16	139.93	20.91 (tco)	**1259** $(best)$	1605	0.785	79.57	127.71
$r9p11$	12	99	12	139.08	13.72 (tco)	1252	**1251** $(best)$	**1.001** $(best)$	100.00	**125.10** $(best)$
Xilinx's TCAM	1	8590		22.52	13.48 (tco)	203	8590	0.024	**57.88** $(best)$	497.23
Reg-Gates		27646			58.67 (tpd)		27646		58.67	1621.99

Area: We assume that the area for one BRAM is equivalent to the area for 96 slices

that both "4-input LUTs" and BRAMs of the Virtex-II FPGA are similar to those of the Spartan-3 FPGA. Thus, we can deduce that one BRAM of the Spartan-3 FPGA occupies about the same area as 192 $(= 18 \times 1024/96)$ "4-input LUTs". If we view one "4-input LUT" as approximately one-half a slice according to our discussion in the previous paragraph, we conclude that one BRAM has about the same area as 96 $(= 192/2)$ slices. Thus, estimates of the area for one BRAM vary between the area for 16 to 96 slices. For this analysis a worst case of 96 slices/BRAM was used.

In Table 7, the column **Area** denotes the equivalent utilized area, where the area for one BRAM is equivalent to the area for 96 slices. The column **Th./Area** denotes the efficiency of throughput per area for one slice. The column **Area-Delay** denotes the *area-delay* product. The value denoted by *best* shows the best result.

Xilinx's TCAM has the smallest delay, but requires many slices. Reg-Gates has almost the same delay as Xilinx's TCAM, but requires about three times as many slices as Xilinx's TCAM. Note that Reg-Gates requires no clock pulses in the LPM address generation operation, while the others are sequential circuits that require clock pulses. Since the delay of Reg-Gates is 58.67 ns, the equivalent throughput is $(1000/58.67) \times 9 = 153$ (Mbps), which is lower than all others.

All multiple LUT cascade realizations have higher throughput, smaller area, higher throughput/area, and are more efficient in terms of *area-delay* than Xilinx's TCAM. $r9p11$ has the smallest area, the highest throughput/area, the most efficient in terms of *area-delay*, but has the largest delay. $r8p11$ has the highest throughput, and has the smallest delay among all multiple LUT cascade realizations. Furthermore, in terms of *area-delay*, $r8p11$ has almost the same performance as $r9p11$. Thus, $r8p11$ is the best multiple LUT cascade realization that has 5.20 times more throughput, 31.71 times more throughput/area, and is 2.89 times more efficient in terms of *area-delay* product than Xilinx's TCAM, while the area is only 19$\%$ of Xilinx's TCAM.

6 Conclusions

In this paper, we presented the multiple LUT cascade to realize LPM address generators. Although we illustrated the design method for $n = 32$ and $k = 504 \sim 511$, it can be extended to any value of n and k.

We implemented four kinds of LPM address generators (i.e. $r6p11$, $r7p11$, $r8p11$, and $r9p11$) on the Xilinx Spartan-3 FPGA (XC3S4000-5) by using the multiple LUT cascade. For comparison, we also implemented Xilinx's proprietary TCAM, and Reg-Gates by using registers and gates on the same type of FPGA. Xilinx's TCAM has the smallest delay, but requires many slices. Reg-Gates has almost the same delay as Xilinx's TCAM, but requires the largest area and requires about three times as many slices as Xilinx's TCAM. All multiple LUT cascade realizations have higher throughput, smaller area, higher throughput/area and more efficient in terms of *area-delay* product than Xilinx's TCAM.

ACKNOWLEDGMENTS

This research is partly supported by a Grant-in-Aid for Scientific Research from JSPS, MEXT, a grant from Kitakyushu Area Innovative Cluster Project, and by NSA Contract RM A-54.

References

1. Micron Technology Inc.: Harmony TCAM 1Mb and 2Mb, Datasheet, January 2003
2. Gupta, P., Lin, S., McKeown, N.: Routing lookups in hardware at memory access speeds, Proc. IEEE INFOCOM (1998) 1241-1247
3. Dharmapurikar, S., Krishnamurthy, P., Taylor, D.: Longest prefix matching using Bloom filters, Proc. ACM SIGCOMM (2003) 201-212
4. Sasao, T., Butler, J. T.: Implementation of multiple-valued CAM functions by LUT cascades, (draft)
5. Shafai, F., Schultz, K.J., Gibson, G.F.R., Bluschke, A.G., Somppi, D.E.: Fully parallel 30-MHz, 2.5-Mb CAM, IEEE Journal of Solid-State Circuits, Vol. 33, No. 11 (1998) 1690-1696
6. Ashenhurst, R. L.: The decomposition of switching functions, Proc. International Symposium on the Theory of Switching (1957) 74-116
7. Sasao, T.: Switching Theory for Logic Synthesis, Kluwer Academic Publishers (1999)
8. Sasao, T., Matsuura, M., Iguchi, Y.: A cascade realization of multiple-output function for reconfigurable hardware, Proc. International Workshop on Logic and Synthesis (2001) 225-230
9. Sasao, T., Matsuura, M.: A method to decompose multiple-output logic functions, Proc. 41st Design Automation Conference (2004) 428-433
10. http://www.xilinx.com
11. http://www.xilinx.com/products/design_resources/design_tool/grouping/design_entry.htm
12. Xilinx, Inc.: Spartan-3 FPGA family: Complete data sheet, DS099, Aug. 19, 2005
13. Sproull, T., Brebner, G., Neely, C.: Mutable codesign for embedded protocol processing, Proc. IEEE 15th International Conference on Field Programmable Logic and Applications (2005) 51-56