High Performance IP Lookup on FPGA with Combined Length-Infix Pipelined Search*

Yi-Hua E. Yang
Ming Hsieh Dept. of Electrical Eng.
University of Southern California
Email: yeyang@usc.edu

Oğuzhan Erdem
Dept. of Electrical and Electronics Eng.
Middle-East Technical University
Email: ogerdem@metu.edu.tr

Viktor K. Prasanna
Ming Hsieh Dept. of Electrical Eng.
University of Southern California
Email: prasanna@usc.edu

***Abstract*—We propose a *combined length-infix pipelined search* (CLIPS) architecture for high-performance IP lookup on FPGA. By performing binary search in prefix length, CLIPS can find the longest prefix match in $(\log L - c)$ *phases*, where L is the IP address length (32 for IPv4) and $c > 0$ is a small design constant ($c = 2$ in our prototype design). Each CLIPS phase matches one or more input infixes of the same length against a regular data structure. Various CLIPS phases can be optimized individually: (1) 16 bits of the IP address are used to direct-access a 288-kbit on-chip BRAM in phase 1; (2) 8 additional bits of the IP address are used to search a 1.5-million-entry *pipelined dynamic search forest* for a match in phase 2; (3) 1 to 8 additional bits of the IP address are used by a 2-stage TreeBitmap for storing another 1 to 8 million routing prefixes in the tail phase. Post place-and-route results show that our CLIPS prototype, utilizing 28 Mbits on-chip BRAM and 4 external SRAM channels, sustains 312 MPPS IPv4 lookup (or 160 Gbps routing thoughput with 64-byte packets) against 9.5 million prefixes on state-of-the-art FPGA.**

***Index Terms*—IP lookup, packet forwarding, longest prefix match, binary search tree, search trie, Tree Bitmap**

I. Overview & Background

IP lookup based on longest prefix match (LPM) has been the central mechanism for routing Internet packets. We propose a novel LPM architecture called *combined length-infix pipelined search* (CLIPS) suitable for implementation on FPGA. For L-bit LPM, CLIPS is constructed as a memory-based pipeline with $O(\log L)$ *phases*. Rather than searching on large numbers of different prefix lengths (Figure 1), each CLIPS phase matches only same-length *infixes* (Figure 2). As a result, CLIPS can be partitioned efficiently into $O(\log L)$ modules, one for each phase, where various modules can use hybrid architectures optimized differently across phases. In addition, since CLIPS does not expand all routing prefixes to pre-determined lengths or exclusive value ranges [1, 2, 3], dynamic updates are also much simpler in CLIPS compared to other tree-based approaches.

Definition 1: An *infix* of bitvector $\bar{v} = \langle b_1, \ldots, b_n \rangle$ is any bitvector $\bar{u} = \langle b_i, \ldots, b_j \rangle$ where $1 \leq i \leq j \leq n$.

Binary search on prefix length was proposed in [4], where $O(L)$ hash tables are used to match routing prefixes of length $\leq L$ bits. The hash tables are organized as a binary tree (Figure 1), resulting in $O(\log L)$ hash table accesses per lookup. On the other hand, a large number of different hash tables may be required to store routing prefixes of different lengths. Several techniques have been proposed to minimize the number of distinct routing prefix lengths [5, 6], which however is still a computationally expensive process and can complicate dynamic updates on the routing table.

* This work was supported by U.S. NSF under grant CCR-0702784. Equipment grant from Xilinx Inc. is gratefully acknowledged.

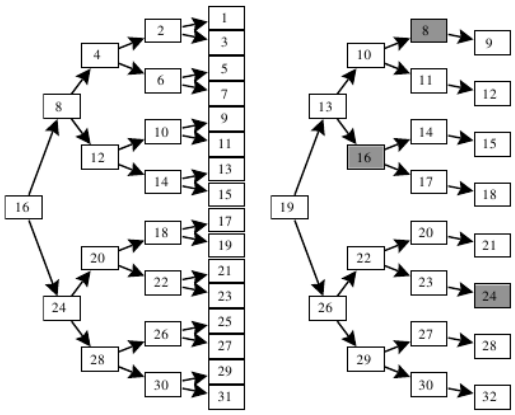


Figure 1. Binary searches on prefix length [6].

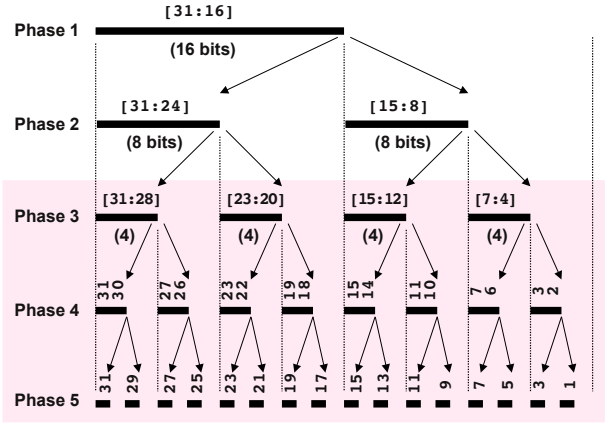


Figure 2. Overview of the CLIPS process.

II. Combined Length-Infix Pipelined Search

A. Longest Prefix Match (LPM) in CLIPS

Figure 2 shows an overview of the *combined length-infix pipelined search* (CLIPS) process. Each thick solid line represents an infix of the *input* (32-bit IP address) used by the corresponding phase. The numbers above each line show the bit offsets used for the infix; the number below shows the infix length. As shown in the figure, binary search is performed in $O(\log L)$ phases to determine the length of the longest routing prefix, where L is the length of the input address ($L = 32$ for IPv4). Efficient infix search is performed within each phase, where the infix length is $L/2^i$ in phase i, $1 \leq i \leq \log L$.

Central to CLIPS are the *local infix tables* (LITs), one per CLIPS phase (Section II-B). Subjecting to the false-negative avoidance (Section II-C), LITs can be dynamically and incrementally updated by inserting and deleting routing prefixes (Section II-D).[1]

To perform the LPM, in each Phase i, $1 \leq i \leq \log L$, an infix is extracted from the input bitvector according to the "length-path" taken to reach the phase. For example, four possible length-paths could be used to reach the Phase 3 in Figure 2, corresponding to four possible infixes [31:28], [23:20], [15:12], and [7:4], respectively. Let *B* represents the extracted infix bitvector, and *A* be the matching result up to Phase $i-1$. Then the value pair (*A*, *B*) is used to look up the LIT of Phase i. If a matching entry A' is found in the LIT of Phase i, then it becomes the new matching result from Phase i to Phase $i+1$, following a down-right arrow in Figure 2 (towards longer prefix length). If no matching entry is found, then A is kept as the matching result, following a down-left arrow in Figure 2 (towards shorter prefix length).

B. Constructing the Local Infix Tables (LITs)

To construct the LITs, each routing prefix R in the routing table is first cut into consecutive infixes, each of which is then assigned (according to the infix length) to a phase where phase i receives the infix of length $L/2^i, 1 \leq i \leq \log L$. Suppose infix B_i is assigned to phase i. Then starting from phase $i = 1$, an entry $M_i : (P_i, B_i)$ is added to LIT(i), where P_i is a unique identifier for M_{i-1}. M_0 is the "empty prefix" entry and always identified by 0. If no infix is assigned to phase i, then M_i is $null$ for R and $P_i = P_{i-1}$.

Consider an 8-bit LPM example with the following routing prefixes: $R_1 = 011*$, $R_2 = 1001*$, $R_3 = 1000011*$, and $R_4 = 10011*$. By adding R_1, R_2 and R_3 consecutively to the CLIPS solution, we can construct $\log_2 8 = 3$ LITs, one per phase, as shown in Figure 3(a). A few points worth noting:

1) The "match ID" (M) uniquely identifies any LIT entry.
2) All LIT entries in phase 1 have "prefix ID" $P = 0$, representing the zero-length (empty) prefix.
3) A LIT entry in phase $i > 1$ has its "prefix ID" either being 0 or the same as the "match ID" of some "marker" entry in phase $j < i$.

[1]Due to the length limit, we explain the operations on LITs mainly by examples. Detailed data structures and algorithms are discussed in [7].

(a) With routing prefixes $011*$, $1001*$ and $1000011*$

Phase 1

P, *B*	*m*,*r*	*M*
0, 1001	0,1	3
0, 1000	1,0	4

Phase 2

P, *B*	*m*,*r*	*M*
0, 01	1,0	1
4, 01	1,0	5

Phase 3

P, *B*	*m*,*r*	*M*
1, 1	0,1	2
5, 1	0,1	6

(b) After adding routing prefix $10011*$

Phase 1

P, *B*	*m*,*r*	*M*
0, 1001	**1**,1	3
0, 1000	1,0	4

Phase 2

P, *B*	*m*,*r*	*M*
0, 01	1,0	1
4, 01	1,0	5

Phase 3

P, *B*	*m*,*r*	*M*
1, 1	0,1	2
5, 1	0,1	6
3, 1	**0,1**	**7**

(Conventions– *P*: *PrefixID*; *B*: *InfixBits*; *m*: *IsMkr*; *r*: *IsRtp*; *M*: *MatchID*)

Figure 3. Example LITs constructed for 8-bit routing.

4) Entries with $r = 1$ represent full routing prefixes.

We will follow the insertion of R_4 to LIT entries more closely to better explain the LIT construction process. First we cut R_4 into two infixes, 1001 and 1, which are assigned to phase 1 and phase 3, respectively. Starting with $i = 1$, we search the phase 1 LIT for $P = 0$ and $B = 1001$, finding the entry with $M = 3$. This entry is set as a "marker" ($m = 1$) and $M = 3$ is used to represent the prefix 1001 of R_4. Since no length-2 infix is obtained from R_4, the phase 2 LIT is not updated. With $i = 3$, we search the phase 3 LIT for $P = 3$ and $B = 1$. Since no matching entry is found this time, a new entry is created and assigned a unique "match ID" $M = 7$. Since $B = 1$ is the last infix of R_4, the new entry indicates a routing prefix match to R_4 and has $r = 1$. Figure 3(b) shows the resulting LITs with changes in bold.

C. False-Negative Avoidance

Occasionally, two routing prefixes of different lengths can "overlap" with each other, where the shorter routing prefix is a prefix of the longer one. When this is the case, straightforward LIT construction and lookup following Sections II-B may sometimes lead to false-negative LPM results. The false negative comes from the assumption that, when a marker entry is matched in phase i, a *longer* prefix match can be found in some phase $j > i$. If that assumption turns out to be false, CLIPS will find no matching prefix even though a *shorter* prefix match may exist.[2] To ensure correct LPM, the LITs are fixed by *false-negative avoidance*. Let $M_i : (P_i, B_i)$ be a marker entry for routing prefix R. If appending a proper prefix of B_i to the prefix of R represented by P_i produces another routing prefix R', then M_i also represents R'.

Assume another routing prefix $R_5 = 100*$ is added to Figure 3(b), resulting in Figure 4(a). The marker entry with $M = 4$ in phase 1 has $B = 1000$, whose proper prefix (100), when appended to the prefix represented by its $P = 0$ (the empty prefix), results in $R_5 = 100*$. Thus the marker entry need to have its r field set to 1 to identify an LPM to R_5. The resulting LITs are shown in Figure 4(b).

[2]Similar phenomenon also occurs in previous length-based LPM [6].

(a) Without false-negative avoidance

Phase 1

P, *B*	*m*,*r*	*M*
0, 1001	1,1	3
0, 1000	1,0	4

Phase 2

P, *B*	*m*,*r*	*M*
0, 01	1,0	1
4, 01	1,0	5
0, 10	**1,0**	**8**

Phase 3

P, *B*	*m*,*r*	*M*
1, 1	0,1	2
5, 1	0,1	6
3, 1	0,1	7
8, 0	**0,1**	**9**

(b) With false-negative avoidance (set the underscored **1** in phase 1)

Phase 1

P, *B*	*m*,*r*	*M*
0, 1001	1,1	3
0, 1000	1,**1**	4

Phase 2: (same as above)

Phase 3: (same as above)

Figure 4. Example LITs after adding routing prefix $100*$ to Figure 3(c).

D. Dynamic Updates

As shown in Section II-B, LIT construction is performed by adding routing prefixes to the LITs one at a time. As a result, CLIPS naturally performs routing prefix insertion both dynamically and incrementally. To maintain false-negative avoidance, each newly inserted routing prefix, R_{ins} must be checked against the routing table to see (1) whether R_{ins} is a prefix of any existing routing prefix, and (2) whether any existing routing prefix is a prefix of R_{ins}. This can be accomplished by maintaining a "shadow trie" representation for the routing table on a separate (general-purpose) computing system, out of the critical path of IP lookup.

Similarly, deleting a routing prefix R_{del} from the LITs in CLIPS requires first finding all the LIT entries associated with R_{del} using the shadow trie, then updating or removing these entries. Note that each internal node in the shadow trie corresponds to a marker LIT entry, whereas each prefix-match or leaf node corresponds to a routing-prefix LIT entry. Thus the LIT entries to update or remove for deleting R_{del} can be found by observing the nodes modified or deleted in the shadow trie.

III. Mapping and Optimizing CLIPS on FPGA

In this section we describe the mapping and optimization of our IPv4 CLIPS prototype on state-of-the-art FPGA utilizing both on-chip BRAM and off-chip SRAM. Note that while the circuit-level designs described here are specific to 32-bit IPv4 lookup on Xilinx Virtex 6 FPGA, CLIPS is a generic LPM architecture applicable (with even stronger advantages) to longer prefix lengths and other hardware platforms.

A. Phase 1: Direct BRAM Access

Every routing prefix ≥ 16 bits long has a corresponding LIT entry in phase 1, where bits [31:16] of the routing prefix are used as the 16-bit infixes (which are really prefixes in phase 1). There can be up to $2^{16} = 64$k such entries. We use the first 13 bits ([31:19]) to address a 36-bit BRAM row, formatted as Figure 5, containing the 8 consecutive LIT entries whose prefix values have the same first 13 bits. As a result, all 64k LIT entries (one for each 16-bit value) in phase 1 can be stored in $2^{13} \times 36 = 288$ kb BRAM.

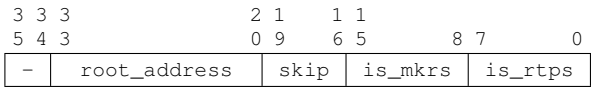


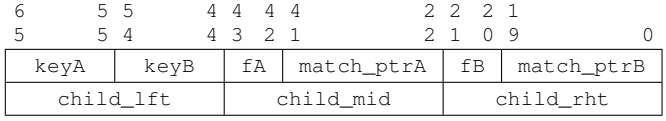
Figure 5. Format of a BRAM row in phase 1.

The 8-bit `is_rtps` and `is_mkrs` fields specify the "is marker" and "is routing prefix" flags, respectively, for the 8 consecutive LIT entries whose 16-bit prefixes have the same first 13 bits. The 4-bit `skip` and 14-bit `root_address` fields specify the "stage skip" and "root address" values for addressing a root node in phase 2 (see Section III-B). Note that all the marker entries stored in the same row in phase 1 share the same root node in phase 2. An extra 4-bit `skip0` register is maintained and accessed separately for the "stage skip" value for all the invalid (non-marker and non-routing prefix) entries in any row. An invalid entry always uses zero as its "root address."

B. Phase 2: Pipelined Dynamic Search Trees

The LIT in phase 2 is organized as a "forest" of pipelined dynamic search trees (pDST) [8], each storing up to 2048 infixes. Multiple pDSTs are stored in the same memory pipeline, each pDST headed by a different root node addressed with the "stage skip" and "root address" values passed from phase 1. A 10-stage pipeline is used to accommodate all pDSTs, with stage size ranging from 512 nodes in stage 0 to 256k nodes in stage 9, doubling every stage. The last two stages are stored in external SRAM.[3]

(a) Internal pDST node format:



(b) Leaf pDST node format:

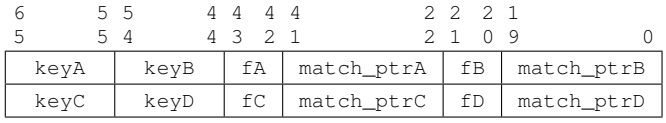


Figure 6. Formats of (a) internal and (b) leaf pDST nodes in phase 2.

Figure 6 shows the pDST node data structures. The *internal* pDST node consists of up to 2 infix keys and 3 child pointers; each infix key is also associated with two flag bits plus a match pointer. The *leaf* pDST nodes consists of up to 4 infix keys and their respective flag bits and match pointers. The `keyA` and `keyB` fields store up to 11 bits of infixes from two different routing prefixes taking the non-addressed 3-bit suffix from phase 1. Similar rules apply to `keyC` and `keyD`.

[3]While introducing extra pipeline latency (~20 cycles), this approach maximizes external SRAM usage with few (2) memory channels.

Table I
PAR RESULTS OF OPTIMIZED CLIPS ON VIRTEX 6 SX475T.

	# slices	# LUTs	# RAMB36	# IOBs	Freq. MHz
Phase 1	59	85	8	n/a	279
Phase 2	986	2,679	512		245
Tail	1,254	3,436	256		175
misc.	416	1,119	3		
Total	**2,715**	**7,319**	**779**	**539**	**156**
Usage	3.65%		73%	64%	

Table II
CAPACITY DISTRIBUTION FOR VARIOUS LENGTHS ROUTING PREFIXES.

	Phase 1	Phase 2	Tail Phase		
Length (bits)	16	24	1~15	17~23	25~31
Capacity	64k	1.5M	32k	1M	~7M
% of IPv4 max.	100%	9.4%	50%	12%	~0.2%

To simplify the circuit, instead of computing and performing dynamic updates entirely on FPGA ([8]), here we compute the required memory updates in an external computing system (see Section II-D) before sending them to the pDST.

C. Tail Phase: 2-Stage TreeBitmap

A "tail" phase of CLIPS covers the shaded parts in Figure 2, which performs 8-bit LPM on one of the following infix parts of a routing prefix: [31:24], [23:16], [15:8], and [7:0].

We implement a 2-stage TreeBitmap [9] for the tail phase of CLIPS. Each stage handles a 4-bit trie traversal by accessing a 16-bit match bitmap (stored in BRAM) and an array of 16 child pointers (stored in external SRAM) in a TreeBitmap node. Each valid child pointer in the 1st stage points a TreeBitmap node in the 2nd stage of the tail phase. Together, the 2 stages perform an 8-bit LPM and accomplishes the last 3 phases of CLIPS.

IV. PERFORMANCE EVALUATION

A. Prototype Implementation

Our CLIPS prototype is written in Verilog and synthesized, placed and routed (PAR) using Xilinx ISE 12.3 targeting the Virtex 6 SX475T FPGA. Table I shows the resource utilization and the achieved clock rates. Running at 156 MHz, the prototype achieves a lookup throughput of 312 million packets per second (MPPS), or 160 Gbps with 64-byte packets. Both of the BRAM and external SRAM are dual-ported to support the two pipelines in parallel.

Phase 1 stores up to 64k 16-bit routing prefixes. Phase 2 contains 512k pDST nodes. There are at least 256k leaf nodes each with up to 4 routing prefixes; the other 256k internal nodes can each store 2 routing prefixes. In total phase 2 can store up to $256\text{k} \times 4 + 256\text{k} \times 2 = 1.5\,\text{M}$ routing prefixes. The tail phase consist of 512k nodes accommodating at least 512k and up to 8M routing prefixes. Overall, our CLIPS prototype can hold a routing table with at least $64\text{k} + 1.5\text{M} + 512\text{k} \approx 2$ million and up to 9.5 million routing prefixes. Table II shows the distribution of capacity in various CLIPS phases.

B. Qualitative Advantages

More important than throughput and routing table size, CLIPS is optimized for the following qualitative advantages over other state-of-the-art IP lookup solutions on FPGA.

1) Non-routing table specific: Unlike trie-based IP lookup where memory balancing is a serious issue, CLIPS requires minimal table-specific memory optimizations. The only memory balancing to perform in CLIPS is to trade-off the on-chip memory usage between phase 2 and tail phase, which is simple to perform and required only when the statistics of the routing table changes significantly.

2) Easy extension to IPv6: CLIPS performs L-bit LPM in only $O(\log L)$ phases. Thus CLIPS can be extended to handle IPv6 by simply adding one or two more phases in the top of Figure 2. In contrast, other trie and tree-based approaches require up to 4 times the computation complexity.

3) Flexible implementation: Most CLIPS phases match only fixed-length infixes rather than variable-length prefixes. As a result, various exact match (*e.g.* hash-based) architectures can be used by CLIPS, potentially with simpler circuit, higher throughput and larger routing table capacity.

V. CONCLUSION AND FUTURE WORK

We proposed a novel CLIPS architecture for IP lookup on FPGA, achieving high lookup throughput against very large routing tables. Unlike trie-based approaches, CLIPS does not suffer from memory balancing issues; unlike tree-based approaches, CLIPS does not require complex pre-processing and can be dynamically and incrementally updated. CLIPS can also be extended to handle IPv6 with minor increase in circuit complexity at the same level of routing throughput.

REFERENCES

[1] K. S. Kim and S. Sahni, "Efficient construction of pipelined multibit-trie router-tables," *IEEE Trans. Comput.*, vol. 56, no. 1, pp. 32–43, 2007.

[2] H. Lu and S. Sahni, "A B-Tree Dynamic Router-Table Design," *IEEE Trans. Comput.*, vol. 54, no. 7, pp. 813–824, 2005.

[3] H. Le and V. Prasanna, "Scalable High Throughput and Power Efficient IP-Lookup on FPGA," in *Proc. of 17th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2009.

[4] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *Proc. SIGCOMM*, 1997, pp. 25–38.

[5] K. S. Kim and S. Sahni, "IP Lookup by Binary Search on Prefix Length," in *Proc. Eighth IEEE Intl. Symp. on Computers and Communication (ISCC)*, 2003.

[6] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable High-Speed Prefix Matching," *ACM Trans. Comput. Syst.*, vol. 19, pp. 440–482, 2001.

[7] Y.-H. E. Yang and V. K. Prasanna, "Hybrid Architecture for High Performance IP Lookup in $\log L$ Phases," in preparation for future publication.

[8] ——, "High Throughput and Large Capacity Pipelined Dynamic Search Tree on FPGA," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2010.

[9] W. Eatherton, G. Varghese, and Z. Dittia, "Tree bitmap: Hardware/Software IP Lookups with Incremental Updates," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 97–122, 2004.