High-performance Pipelined Architecture for Tree-based IP lookup Engine on FPGA*

Yun Qu  
Ming Hsieh Dept. of Electrical Eng.  
University of Southern California  
Email: yunqu@usc.edu

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

Abstract—IP lookup problem involves searching the input IP address for a matching IP prefix in the routing table. Hardware-accelerated IP lookup engines based on various data structures such as balanced tree structures have been proposed over the years. In tree-based approaches, as the size of the tree increases, large off-chip memory has to be used. In addition, the linear growth of wire length with respect to the number of nodes at a level adversely affects the throughput. We present a tree-based IP lookup engine on FPGA which optimizes the pipeline with respect to scalability and throughput. Our solution has the following novel features: (1) We present a 2-dimensional fine-grained layout for the Processing Elements (PEs) using distributed RAM to reduce the maximum wire length. (2) We employ “split-tree” architecture for BRAM-based PEs at each tree level to improve the clock rate. (3) We use a realistic model of off-chip memory access and guarantee high throughput for the lookup process. Post place-and-route results show that, our tree-based IP lookup engine can achieve a throughput of 400 MLPS (million lookups per second) for any routing table containing 256 ~ 512 K IPv6 prefixes, while using 59% of the logic resources and 19% of the BRAM available on a state-of-the-art FPGA device.

Index Terms—Balanced trees, IP lookup, FPGA

I. INTRODUCTION

Internet Protocol address lookup (IP lookup) has been a classic problem for the past few decades. While many sophisticated solutions have been proposed over the years [1], [2], [3], [4], designing a practical and robust IP lookup engine remains a challenging problem due to longer prefix length, larger routing table and higher lookup rate required by the future Internet architecture.

IPv4 with its 32-bit address length is running out of available addresses for the growing number of Internet-connected devices. IPv6 has been developed to overcome the long-anticipated IPv4 address exhaustion by extending the address length to 128 bits. This increased length allows for a broader range of addressing hierarchies and a much larger number of addressable nodes. In the mean time, the routing table size of Internet’s core routers continues to grow exponentially at a rate of 1.13× (IPv4) and 1.60× (IPv6) number of prefixes per year [5]. Furthermore, Internet link bandwidth is also increasing rapidly. Today’s data center switches and core routers need to handle data traffic at hundreds of gigabits per second (Gbps), which translates to lookup rates ranging from hundreds of million lookups per second (MLPS) to over a billion lookups per second (BLIPS). The higher IP lookup rate in turn requires a higher (random) memory access rate to the routing table.

Both the growing size of the routing tables and the increasing bandwidth of the network links make memory access a critical bottleneck for scalable IP lookup. State-of-the-art VLSI chips can be built with massive amount of on-chip computation and memory resources, as well as large number of I/O pins for off-chip memory accesses; Field-Programmable Gate Arrays (FPGAs), with their flexibility and reconfigurability, are especially suitable for accelerating network operations. On the other hand, it is still challenging to accelerate IP lookup on VLSI/FPGA due to many design-time and run-time issues. First, network operations, especially packet forwarding, usually have stringent requirements on scalability, throughput and energy consumption. Second, since future networks are expected to evolve and vary drastically in their structures and compositions, the performance of an IP lookup solution should not depend on any network-specific statistics; hence an IP lookup engine should be evaluated with its guaranteed worst-case performance. Third, the massive on-chip and off-chip resources provide many choices in designing the IP lookup engine, which can be optimized for diverse performance requirements with complex tradeoffs. Last but not the least, accurate and realistic models must be used during design time for fair estimation of the achievable performance, especially when off-chip resources such as DRAM or Reduced-Latency DRAM (RLDRAM) are utilized.

With these challenges in mind, we present a FPGA-based architecture for balanced tree structures which achieves both scalability and high throughput. In summary:

• We present a deeply pipelined architecture for tree-based IP lookup by efficiently mapping the tree structure level-by-level onto various types of PEs.
• We map the first few tree levels onto PEs based on distributed RAM (distRAM) and propose a fine-grained 2-dimensional layout for each PE to reduce the maximum wire length.
• We map the following several tree levels onto PEs using Block RAM (BRAM) and employ “split-tree” architecture to improve the clock rate.

* This work is supported by the U.S. National Science Foundation under grant CCF-1116781; Equipment grant from Xilinx Inc. is gratefully acknowledged.
We use off-chip DRAM/RLDRAM to construct the PEs for the last few tree levels and guarantee high access rate independent of the access pattern based on realistic models.

We present a high-throughput IP lookup engine (400 MLPS per dual-pipeline) whose performance does not depend on the features of the routing table. This engine sustains a routing table containing 256 ~ 512 K IPv6 prefixes, using 59% of the logic resources and 19% of the BRAM on FPGA; it achieves 6.6× power efficiency compared with TCAM [6].

The rest of the paper is organized as follows: Section II introduces the tree-based lookup engine and summarizes the related work. We present our architecture in Section III. We also detail the optimization techniques for PEs based on various types of memory in this section. Section IV provides the performance evaluation for tree structures on FPGA. Section V covers a specific implementation for tree-based IP lookup and Section VI concludes the paper.

II. BACKGROUND AND RELATED WORK

A. IPv4/v6 Lookup

IP Lookup involves retrieving the routing information by finding the longest prefix match (LPM) of the destination IP address in the routing table. With IPv4, the prefix length can be up to 32 bits; with IPv6, the prefix length can be up to 64 bits, and in some cases even 128 bits.\(^1\)

A TCAM-based architecture for IPv4/v6 lookup is proposed in [7], [8]. The TCAM-based IP lookup architecture does not depend on prefix distribution, but it is expensive and power-hungry as well. As an alternative solution, algorithms for IPv6 lookup on FPGA have involved trie-based, hash-based and tree-based approaches.

In trie-based approaches [9], [2], [10], the path from the trie root to each “match” node in the trie represents the bit string of a prefix. In [2], compression is employed to remove redundancy in trie nodes. In [10], only the next-hop bit map and the offset value are stored in each node. In general, trie-based lookup requires memory bandwidth proportional to the prefix length. In hash-based approaches [11], [12], [13], [1], hash functions or bloom filters are used to improve memory efficiency. In particular, the Concise Lookup Tables (CoLT) [3] is a unified hash table-based architecture which handles both IPv4 and IPv6 lookups simultaneously. The FlashTrie [1] proposes a hash-based trie architecture for both high lookup rate and large IPv4 and IPv6 routing tables. However, these architectures provide solutions under various high-level assumptions on hardware resource and prefix distribution; it is difficult to scale up their performance for larger routing table or longer prefix length (e.g. 64 or 128-bit IPv6) given more available resources.

B. Tree Structures

A classic search problem is to find or insert a key into an ordered list [14]. This problem can be described as follows: given an ordered set of \( N \) keys \( a_0, a_1, ..., a_{N-1} \), where \( a_0 \leq a_1 \leq ... \leq a_{N-1} \), an input \( b \) is to be inserted into the correct position of the sequence such that \( a_{k-1} \leq b \leq a_k \), where \( k = 1, 2, ..., N - 1 \).

A general search tree structure keeps single or multiple keys in each node, where each node can be linked to multiple children. In the previous example, \( a_0, a_1, ..., a_{N-1} \) can be used as keys to construct a search tree. The input \( b \) is compared against the values of keys inside a tree node, and any node in the data structure can be reached by starting at root node and repeatedly following the results of key comparisons. As a result, the input can be efficiently searched inside the tree level-by-level.

The search tree is balanced when the heights of the left and right subtrees of each node never differ by more than 1. For any set of \( N \) nodes, the height of a balanced tree is at most \( \log_m N \), where \( m \) is the degree of the tree [15], [16]. This leads to a search time of \( O(\log_m N) \).

C. Tree-based IP Lookup on FPGA

A dual-pipeline architecture on FPGA for IP lookup problem using binary search tree is introduced in [17]. In their architecture, each tree level constructed on the prefix set is mapped onto a single pipeline stage on FPGA with 2\(^n\) PEs, where \( n \) is the stage number. Dual-port on-chip memory access is used to achieve a throughput of 324 MLPS (million lookups per second) or a clock rate of 162 MHz on Virtex-4 devices. Although off-chip SRAM is explored to improve the scalability, the long wires connecting a large number of PEs and I/O pins become the throughput bottleneck when the tree becomes large.

Range trees have been proposed as efficient and robust data structures for storing and searching routing tables [18]. To construct a range tree from the routing table, each routing prefix is first represented as a “prefix range” in the address space. Then, all pairs of overlapping prefix ranges are “flattened” to produce a set of non-intersecting subranges. The range tree is constructed of the subrange boundaries. In [19], IP lookup is performed using a multi-way range tree algorithm, which achieves a worst-case search and update time of \( O(\log N) \). In [20], the subranges are organized in a B-tree data structure for efficient lookup and dynamic update. The range tree data-structure is extended in [21] to support both LPM and incremental updates. [22] introduces a recursive balanced multi-way range tree. Although range-trees have been widely used for the classic IP lookup problem, achieving high throughput and fast clock rate on hardware is still challenging for large routing tables due to long wires connecting large memory.

Besides range trees, another tree-based IP lookup architecture is developed in [4], where specialized prefix matching and prefix partitioning are employed to minimize the total memory requirement. Their architecture can support a large

\(^1\)The IPv6 Addressing Architecture (RFC 4291) specifies IPv6 unicast address to consist of a (64-bit) subnet prefix followed by a (64-bit) interface ID. However, an IPv4-embedded IPv6 address can have a longer-than-64 bits routing prefix (RFC 6052).
routing table of 330 K IPv6 prefixes on FPGA; it can achieve a throughput of 390 MLPS and 195 MHz clock rate.

As we can see, dual-pipeline designs using dual-port memory are commonly used on FPGA to double the throughput performance with respect to the maximum achievable clock rate [17], [4]. Since the memory and I/O banks are usually linearly aligned on FPGA [23], for large IP routing table using large amount of on-chip and off-chip memory, the length of the wires connecting a large number of components grows rapidly, which in turn limits the maximum clock rate [24].

III. ARCHITECTURE

We adopt the range search-based approach discussed in Section II-C as the basis of our IP lookup architecture on FPGA [18], [19], [20], [22]. All prefixes are first translated into non-overlapping subranges; the subrange boundaries are used to construct a balanced range tree (BRTree). On average, each routing prefix can be defined by at least one and at most two subrange boundaries (SB), plus one next-hop information (NHI). For each input IP address, the lookup engine searches the BRTree for the subrange where the input address is located. Each subrange is identified by an NHI stored in a leaf node. A large (leaf) node may store multiple subrange boundaries and NHIs. We choose BRTree for implementation on FPGA because for a given routing table size, the lookup latency and the throughput performance of the BRTree do not depend on the routing table features such as the number of overlapping prefixes or the average length of prefixes.

The main challenges of mapping a large BRTree onto FPGA are as follows:

- Scalability: The number of BRTree nodes increases exponentially with each additional tree level. For limited on-chip resources, only a limited number of tree levels can use on-chip resources, which restricts the scalability of the architecture.

- High throughput: When a large amount of memory are to be utilized, long wires connecting memory modules become the clock rate bottleneck, which in turn limits the throughput.

When the routing table size becomes larger, we use off-chip memory to support large routing tables. We choose DRAM/RLDRAM as the off-chip memory due to its low power and price per bit.

In this section, we first present the overall pipeline architecture in Section III-A. Then we present optimization techniques to improve throughput for distRAM-based PEs and BRAM-based PEs in Section III-B and Section III-C, respectively.

To achieve scalability for the pipeline architecture, we use off-chip DRAM/RLDRAM in Section III-D and show how to sustain high throughput for the pipeline architecture.

A. Overall Architecture

The overall pipeline architecture is shown in Figure 1. Each tree level is mapped onto a single or multiple PEs; each PE has its own designated memory modules. We configure the memory module to be dual-port, therefore in each clock cycle two inputs are fed into the dual-pipeline architecture. As the tree grows, we use various types of memory in different PEs as shown in Table I:

- The top \( l_1 \) tree levels are mapped onto \( p_1 \) PEs, while each one of the \( p_1 \) PEs uses distRAM.
- The middle \( l_2 \) tree levels are mapped onto \( p_2 \) PEs, each PE using BRAM.
- The bottom \( l_3 \) tree levels are mapped onto \( p_3 \) PEs, each PE using off-chip DRAM to achieve better scalability.

The underlying reason of this organization is that: distRAM is based on a limited number of slice Look Up Tables (LUTs), and it has the smallest total size but the lowest delay, while off-chip DRAM has the largest size but the longest delay and limited bandwidth; compared with distRAM and DRAM, BRAM offers the largest bandwidth and moderate size. As the number of tree levels increases, the memory size required for each tree level also increases; a scalable architecture should accommodate this requirement.

In the case of a binary BRTree, the pipelined architecture in Figure 1 can support \( 2 \times \) the routing table size with each additional off-chip memory stage, while taking only a constant number of additional I/O pins per off-chip memory stage\(^2\). By utilizing multiple off-chip memory stages per pipeline and multiple memory buses per stage, the routing table size can be scaled up significantly. Therefore the pipelined BRTree architecture in Figure 1 provides a scalable solution to IPv4/IPv6 lookup.

The main difference between our architecture and the architectures introduced in [17], [24] is that in our architecture, each tree level can be mapped onto several PEs, each PE having multiple pipeline stages. Search in each tree level can take multiple clock cycles to complete in our architecture; the architectures mentioned in [17], [24], however, strictly enforce the delay per tree level to be exactly one clock cycle. As a result, our design achieves better clock rate and throughput.

\(^2\)For BRTree with a larger degree, the pipelined architecture can support even larger routing table size with each additional off-chip memory stage.
B. distRAM-based PE

1) 2-dimensional layout: Each distRAM-based PE consists of a multi-stage Comparison Unit (CU) and a single or multiple distRAM modules. At the top \((k_1 - 1)\) BRTree levels \((0 < k_1 \leq l_1)\), since the memory size required for each level is relatively small and can be fit in a single distRAM module, each BRTree level is mapped onto a CU along with one distRAM module.

When the tree grows larger \((\geq k_1\) levels), more memory modules need to be used for a single BRTree level, where the wire length connecting those modules also increases. We show a 2-dimensional layout in Figure 2 to improve the clock rate:

- The distRAM modules are arranged in a 2-dimensional array.
- The CU is located in the center of the 2-dimensional array.

As shown in Figure 2, each circle represents a tree node. The black solid arrows indicate the node access order in the tree structure. All the tree nodes inside a specific dotted block are stored in a single distRAM module. Specifically, Figure 2 shows an example of a lookup process; the grey nodes signify the nodes used for comparison with the input, while the distRAM modules in grey indicate the corresponding activated distRAM modules. The advantages include:

- A tree level using a large amount of distRAM is mapped onto a 2-dimensional array that can be easily fit on FPGA.
- Initially the wire length grows at a rate of 2 in the BRTree structure, while with the 2-dimensional layout, the maximum wire length grows at a rate of \(\sqrt{2}\) for each additional BRTree level.
- For each of the tree levels from level \(k_1\) to level \((l_2 - 1)\) with the 2-dimensional layout, only one distRAM module is accessed by the CU during run-time, while others can be deactivated to save energy.

The 2-dimensional layout is only done when the tree grows beyond a sufficient depth \((k_1)\); the value of \(k_1\) will be decided later in Section IV.

2) CU organization: To achieve high clock rate, we divide each CU further into multiple stages. As a result, a 3-stage dual-pipeline architecture consisting of 3 register groups is constructed for each CU as shown in Figure 3. The memory addresses for the two inputs are provided to the memory after the first pipeline register group; the keys are fed into the comparators in the next stage, where the comparison results are combined with the memory address. The data paths of pass-by signals are also included; all control signals have been omitted for simplicity.

C. BRAM-based PE

Level \(l_1\) to level \((l_1 + l_2 - 1)\) of the BRTree are mapped onto BRAM-based PEs. When the tree grows larger, more BRAM modules need to be used for a single BRTree level; the wire length also increases. We propose a split-tree architecture to improve the clock rate:

- When a large amount of BRAM blocks is to be used (starting from level \(k_2, l_1 < k_2 \leq l_2\)), instead of mapping each BRTree level onto a single PE, we split the set of nodes in this level onto multiple subsets and use one PE for each subset. In total, \(x_i\) PEs are used for level \(i\) of the BRTree, where \(i = k_2, k_2 + 1, ...,\) and \(\sum x_i = p_2\).

The number of PEs in a specific tree level \((x_i)\) is to be determined later.

- In a specific tree level that has been split, each PE is designed to have its own designated memory module. All the PEs in a specific tree level are locally connected and linearly aligned to be efficiently mapped onto FPGA. Only one PE performs key comparison instructed by the previous tree levels, while other PEs in the same level forward data on to the next PE without key comparison.

We identify three types of split-tree architecture as shown in Table II. For instance, let \(M\) denote the total memory required.

![Figure 2: 2-dimensional layout starting at level \(k_1\)](image)

![Figure 3: Organization of distRAM/BRAM-based PE](image)

<table>
<thead>
<tr>
<th>Type</th>
<th>Wire length</th>
<th>No. of PEs in each tree level</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>(O(N))</td>
<td>(O(1))</td>
<td>1, 1, 1, 1,...</td>
</tr>
<tr>
<td>II</td>
<td>(O\left(\frac{N}{\log N}\right))</td>
<td>(O(\log N))</td>
<td>1, 2, 3, 4,...</td>
</tr>
<tr>
<td>III</td>
<td>(O(1))</td>
<td>(O(N))</td>
<td>1, 2, 4, 8,...</td>
</tr>
</tbody>
</table>
for level $k_2$ of the BRTree. In Type-II split-tree architecture, level $k_2$ is split into $x_{k_2} = 2$ PEs, each having a memory size of $\frac{M}{2}$. Level $(k_2+1)$ is split into $x_{k_2+1} = 3$ PEs. For a specific tree level, one of the PEs performs key comparison; the other simply receives data from its previous PE and passes data directly to the next PE.

We have the following observations:

- Type-I split-tree architecture is exactly the original mapping without any optimization, while Type-III split-tree architecture results in a linear array of PEs.
- At any particular time, all PEs except those who need to perform key comparisons can turn off their memory modules to save energy.

The split is only done when the tree grows to a sufficient depth ($k_2$); the value of $k_2$ which gives the best timing performance will be shown later in Section IV. We will also study the tradeoffs for various types of split-tree architecture in Section IV.

For BRAM-based PEs, the internal organization of each CU is similar to that of distRAM-based PEs as shown in Figure 3, except that dual-port BRAM is utilized.

### D. DRAM-based PEs

1) **Timing constraints:** We use DDR3 DRAM as an example to illustrate our ideas. The same methodology and analysis presented in this section can be easily extended to other types of DRAM including RLDRAM.

For the sake of completeness, we briefly discuss DRAM timing issue, which leads to our data layout optimization. The memory interface on FPGA for a single DRAM bus takes $\leq 60$ FPGA I/O pins; two DRAM chips can be connected to this DRAM bus (dual-rank). A DRAM chip is organized in multiple (usually 8) banks, each composed of a large array of (usually 8K) rows and (usually 1K) columns. A fixed number of data bits (usually 16; the bus width) are located at any valid [bank, row, column] address. A sense amplifier is connected to each row via long wires. Every DRAM read or write can initiate a burst of 8 transfers of sequential data, which leads to 128 bits per access. The following major parameters define the DDR3 DRAM access performance with various activation scenarios [25], [26]:

- $t_{CCD}$: Minimum time between successive accesses to the same bank and row, $\approx 5$ ns.
- $t_{RRD}$: Minimum time between successive row activations to different banks, $\approx 8$ ns.
- $t_{RC}$: Minimum time between issuing two successive row activations to the same bank, $\approx 50$ ns.
- $t_{FAW}$: Minimum time to perform 4 row activations to the same bank, $\approx 50$ ns.
- $t_{RCD}$: Minimum time to open/activate a row in a bank, $\approx 14$ ns.
- $CL$: Delay from when the column address is provided to when the data appears on the data pins, $\approx 10$ ns.

2) **Data replication:** As shown in Figure 4 and Figure 5, we use the data replication technique to improve read access rate:

- We replicate the same data in all the 8 banks of a DRAM chip, which allows successive accesses always to be performed to a different bank in a round-robin fashion. This technique can remove $t_{RC}$ from being a timing constraint and increase the access rate to 100 Maccs/s (or 95 Maccs/s considering refresh operations).

In addition to the timing constraints mentioned above, the DDR3 controller must issue at least 8192 refresh commands every 32 ms in the worst case [25]. Since all accesses to the memory bank are delayed during the refresh process, this leads to an access rate loss of 5% [26].

Usually a single read command is issued by the memory controller for each row access.\(^3\) Therefore the number of successive accesses to the same row of a bank is assumed to be 1; $t_{CCD}$ can be ignored due to the random access pattern. For successive row activations to the same bank, $t_{FAW}$ dominates the term $t_{RRD}$. Hence the read access rate is mainly limited by $t_{RC}$ and $t_{FAW}$. We will use $t_{RCD}$ and $CL$ later to optimize the internal organization of CUs.

---

\(^3\)Since we only read data from DRAM, we only focus on the read access model in this paper.

\(^4\)More reads per row access enforces the DRAM to work in Fast Page Mode (FPM), which leads to even better access rate.
Table III: Guaranteed access rate for a DRAM-based PE

<table>
<thead>
<tr>
<th>Pipeline clock rate (MHz)</th>
<th>≤ 95</th>
<th>95 ~ 190</th>
<th>190 ~ 285</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access rate (Maccs/sec)</td>
<td>≤ 190</td>
<td>190 ~ 380</td>
<td>380 ~ 570</td>
</tr>
<tr>
<td>No. of DRAM buses</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>No. of used I/O pins</td>
<td>60</td>
<td>120</td>
<td>180</td>
</tr>
</tbody>
</table>

Table IV: Virtex 6 XC6VLX760 FFG1760-2 FPGA [23]

<table>
<thead>
<tr>
<th>Configurable Logic Blocks (CLBs)</th>
<th>BRAM (Kb)</th>
<th>User I/O pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>Distributed RAM</td>
<td></td>
</tr>
<tr>
<td>118,560</td>
<td>8,280</td>
<td>25,920</td>
</tr>
<tr>
<td></td>
<td>1,200</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7: Achievable clock rate of distRAM-based PE for various key widths (FL: 2-dimensional fine-grained layout)

1) First we only implement distRAM-based PEs and decide the number of tree levels to be mapped onto distRAM-based PEs. We use 2-dimensional fine-grained layout when a large number of distRAM modules are used.
2) With the distRAM-based PEs already implemented, we decide the number of tree levels to be mapped onto BRAM-based PEs; we use split-level architecture for BRAM-based PEs.
3) With the guaranteed off-chip DRAM access rate as discussed in Section III-D2, we map the last few levels of the tree onto DRAM-based PEs.

B. Parameter Optimization

The parameters \(l_1, l_2, l_3, p_1, p_2, p_3, k_1, k_2\) are optimized with respect to the following metrics:

- Achievable clock rate: maximum clock rate a specific design can run at after place-and-route process.
- Resource consumption: on-chip and off-chip resources consumed including logic, memory and I/O pins.\(^5\)

Now we optimize distRAM-based PEs, BRAM-based PEs and DRAM-based PEs individually.

1) \textit{distRAM-based PE}: We show the achievable clock rate in Figure 7; specifically, we use solid lines for 2-dimensional fine-grained layout implementation, compared with the dotted lines where no optimization is explored. The resource consumption of all distRAM-based PEs is shown in Figure 8. In these two figures, we vary both the key width and the number of tree levels to be mapped to distRAM-based PEs \((l_1)\). We have the following observations:

- Internally, distRAM is based on 6-input LUTs, which can support efficient access to a 64-entry table which stores the keys. Therefore level 6 with a table of \(2^6\) entries, and

\(5\)For logic resources, we mainly report the number of used slice registers and slice LUTs, since these two components are the most important resources in a logic slice.
all levels above it can be mapped onto distRAM-based PEs without adversely affecting the clock rate. We are still able to sustain high clock rate (> 300 MHz) when \( l_1 = p_1 \leq 7 \) in Figure 7 even when wider keys are used in the BRTree.

- For tree levels under the seventh level, the table that stores keys is deeper than 64; in this case, multiple 6-input LUT-based distRAM modules need to be used instead of only one. Without 2-dimensional fine-grained layout, due to complex address decoding and wire routing, the clock rate deteriorates dramatically (< 200 MHz) when \( l_1 = p_1 > 7 \) in Figure 7.
- Using the 2-dimensional layout starting from \( k_1 = 6 \), we improve the clock rate to > 200 MHz for \( l_1 = p_1 > 7 \). In order to achieve very high clock rate (∼ 250 MHz), the value of \( l_1 \) and \( p_1 \) is chosen to be 9.
- In Figure 8, the resource consumption increases linearly with respect to the key width. The distRAM-based PEs for \( l_1 = p_1 = 9 \) consume under 52% of total logic resources for all key widths.
- Logic resource consumption grows exponentially with respect to the number of tree levels mapped onto distRAM-based PEs.

2) BRAM-based PE: Given \( l_1 = p_1 = 9 \) above, we first choose the value of \( k_2 \). Then we aim to choose the type of the split-tree architecture and determine the values of \( l_2 \) and \( p_2 \).

As shown in Table V, each BRAM module can be arranged in various key widths with various depths but fixed maximum size (36 K\( b \)). For instance, a data width of 64 bit limits each BRAM to have a maximum depth of 512 entries, while the 10th tree level (level 9) requires a memory depth of exactly 512 entries. Using the same approach discussed in Section IV-B1 where the value of \( l_1 \) and \( p_1 \) is determined, we split level 10 and all tree levels below level 10, before the clock rate can drop dramatically. We come to the following conclusion: the value of \( k_2 \), as shown in Table V, depends on the data width of the keys in the search tree.

Using the values of \( k_2 \) in Table V to split tree levels for various key widths, we show the achievable clock rate and processing delay in Figure 9. The tree levels above and including level 6 are all implemented using distRAM-based PEs. When calculating the processing delay of the pipeline consisting of both distRAM-based PEs and BRAM-based PEs, we assume all distRAM-based PEs can run at the same clock rate as the BRAM-based PEs. We also show the resource consumption in Figure 10 for BRAM-based PEs with respect to the number of tree levels. We have the following observations:

- A pipeline architecture without split (Type-I) can run at most 120 MHz clock rate, but the processing delay is also the minimum among all three types of split-tree architecture.
- We can sustain > 270 MHz clock rate using Type-III split-tree architecture, which guarantees a very high throughput for BRTree levels up to 15, while the processing delay is increasing exponentially with respect to the number of tree levels.
- For key widths of 32 and 64, Type-II split-tree architecture supports > 200 MHz clock rate with a little increase in processing delay. Therefore: We choose Type-II split-tree architecture when designing the entire pipeline.
- Since the BRAM is aligned linearly on FPGA in multiple lines, the single dual-pipeline architecture should also be placed next to BRAM lines linearly. For instance, since the FPGA shown in Table IV has 10 BRAM lines, one dual-pipeline architecture can use at most 2 lines of the total BRAM available on-chip without significantly affecting the achievable clock rate. Using 36 K\( b \) BRAM, 16 BRTree levels for key width of 32 and 15 BRTree levels for key width of 64, both consume only < 19% of total BRAM on-chip (two lines of BRAM). This analysis leads to the following conclusion: For key widths of 32 and 64, the values of \( l_2 \) are chosen to be under 16 and 15, respectively, while the value of \( p_2 \) is under 21 in Type-II split-tree architecture.
- The resource consumption increases linearly with respect to the number of PEs that are used as shown in Figure 10.

3) DRAM-based PE: We conservatively estimate the clock rate of the entire pipeline architecture implementation on FPGA to be around 280 MHz. Therefore, in order to remove the memory access rate as a performance bottleneck, we use 3 dual-rank DRAM buses for the same data copy in a bank (in total 3 \times 2 \times 8 = 48 data copies) in our design for one DRAM-based PE. In addition, as discussed in Section IV-B3, 7 latency-hiding stages have to be used in one PE to compensate the extra latency \( (R_{CD} + CL \simeq 24 \text{ ns}) \) introduced by each off-chip DRAM access.

Considering the large amount of resource overhead introduced by the off-chip DRAM access, we choose not to map the last few levels individually onto DRAM-based PEs; instead, we map multiple levels onto a small number of DRAM-based

![Figure 8: Resource consumption of distRAM-based PE for various key widths](image)

<table>
<thead>
<tr>
<th>Key width</th>
<th>BRAM configuration</th>
<th>( k_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4</td>
<td>32</td>
<td>1K \times 32</td>
</tr>
<tr>
<td>IPv6</td>
<td>64</td>
<td>31.2 \times 64</td>
</tr>
</tbody>
</table>
Figure 9: Clock rate and delay for BRAM-based PEs at various key widths (e.g. 32-I means Type-I split-tree architecture for 32-bit key width, etc.)

Table VI: Performance of 3 DRAM-based PEs at various key widths

<table>
<thead>
<tr>
<th>Key width</th>
<th>Clock rate (MHz)</th>
<th>Slice reg.</th>
<th>Slice LUTs</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4</td>
<td>32</td>
<td>291.85</td>
<td>804</td>
</tr>
<tr>
<td>IPv6</td>
<td>64</td>
<td>248.68</td>
<td>1476</td>
</tr>
</tbody>
</table>

This can be done by merging the last few levels onto new levels with a higher node degree. We have the following observations:

- Virtex 6 XC6VLX760 FFG1760-2 FPGA has 4 I/O bank lines aligned in the same direction as BRAM lines, each having 6, 9, 9, 6 40-pin I/O banks. Since the I/O pins to be utilized should be allocated locally close to the occupied BRAM line to achieve better clock rate, at most 15 banks (600 pins) can be used for a single dual-pipeline design. Therefore the maximum number of DRAM-based PEs is limited by \( p_3 \leq \left\lfloor \frac{600}{180} \right\rfloor = 3 \). Considering the pins used for other inputs and outputs, we choose \( p_3 = 3 \).

- Since each DRAM access brings in 128 bits, we choose the values of \( l_3 \) to be 4 and 2 for key widths of 32 and 64, respectively. For instance, one 128-bit DRAM access can provide data for 4 BRTree nodes of 32-bit key width, meaning at least 2 levels can be merged together.\(^6\)

The post place-and-route results in Table VI show that a pipeline consisting of 3 DRAM-based PEs can run at > 240 MHz clock rate, and utilize under 50% of total I/O pins for DRAM buses.

V. TREE-BASED IP LOOKUP

After we optimized the design parameters in the previous section, now we combine the designs of distRAM-based PEs, BRAM-based PEs and PEs based on off-chip memory together using Xilinx ISE Design Suite 14.1 PlanAhead Tool and measure the overall performance of the entire design.

The BRTree is constructed with key width of 64 bits to accommodate IPv6 lookup. We map the tree structure onto Virtex 6 XC6VLX760 FFG1760-2 FPGA [23], following the design procedure discussed in Section IV. A summary of the optimized design parameters is shown in Table VII. The following four metrics are used to measure the performance of our pipeline architecture:

- Throughput (MLPS): product of achievable clock rate and the number of pipelines.
- Routing table size: minimum number of IPv6 prefixes the pipeline architecture can support.
- Resource consumption: sum of the resources used for all PEs, including both on-chip and off-chip resources.
- Energy efficiency: total energy consumed by the lookup engine normalized by routing table size and throughput.

\(^6\) Generally it is not easy to convert several BRTree levels into a multi-way tree level, unless the degree of the multi-way tree is the power of 2.
A. Throughput

We show the achievable throughput with respect to number of tree levels for a single dual-pipeline architecture in Figure 11. As can be seen, a throughput of 400 MLPS (200 MHz clock rate) can be sustained for BRTree with 18 levels. The throughput bottleneck is still from BRAM-based PEs.

B. Routing table size

We also show the ranges of achievable routing table size for various number of BRTree levels in Figure 11. As can be seen, the routing table size that a single dual-pipeline architecture can support increases exponentially with respect to the number of BRTree levels.

Since each subrange boundary corresponds to at least one and at most two IP prefixes, and the balanced BRTree consists of 18 levels, our pipeline architecture can support any routing table containing 256 K~512 K IPv6 prefixes.

C. Resource Consumption

The BRAM consumption for the BRTree of 18 levels is 4680 Kb (19% of total BRAM), while 637 I/O pins (53% of total pins) are used in the entire dual-pipeline design. After careful floorplanning, the entire dual-pipeline design only consumes two lines of BRAM and three lines of I/O pins. The logic resource consumption with respect to the number of BRTree levels is shown in Figure 12. As can be seen:

- Only a small amount of logic resources (19% of total slice registers and 59% total slice LUTs) are used in a dual-pipeline design for one BRTree of 18 levels.
- Logic resource consumption increases exponentially with respect to the number of tree levels due to the split-level architecture, until DRAM-based PEs are used.

D. Power Dissipation and Energy Efficiency

As we maintain a high clock rate for the pipeline architecture, one concern is that the power consumption can be relatively high, compared with a pipeline design running at a low clock rate. To save on-chip power, we shut off the memory module when it is not being accessed as discussed in Section III. Xilinx ISE Design Suite 14.1 XPower Analyzer is used to estimate the worst-case on-chip power of one dual-pipeline design specified in Table VII. The off-chip power is calculated by using [27], assuming 9 dual-rank DDR3 DRAM buses are connected to the I/O pins of the FPGA device. Since one 800 × 16 dual-rank DDR3 DRAM bus consumes a power of 677 mW, the total off-chip power for a single dual-pipeline architecture is estimated to be around 6.09 W.

In Figure 13, we show the power dissipation of the pipeline architecture; the entire design of the single dual-pipeline is based on the design parameters shown in Table VII. Mapped from a 18-level BRTree, this pipeline architecture can support a routing table with at least 256 K IPv6 prefixes. We also show the power dissipation of PEs using distRAM, BRAM and off-chip DRAM separately, excluding leakage power. As can be seen:

- The total power consumption for a BRTree with 18 levels is under 20 W.
- Excluding leakage power, which is a static power that we have no control on, the off-chip dual-rank DRAM buses along with the off-chip DDR3 chips consume the most power (33% of total power). 14% of total power is spent on I/O interface; 17% of total power is spent on BRAM.

To make a fair comparison of our energy consumption with other hardware-based approaches for IPv6 lookup, we define the energy efficiency per lookup operation as:

$$\eta \triangleq \frac{\text{Energy Consumption per Lookup}}{\text{Throughput} \times \text{Routing Table Size}}$$

This metric captures the energy consumption normalized by throughput and routing table size; a small value of $\eta$ is desirable. We show the power efficiency of our dual-pipeline architecture for 18-level BRTree in Table VIII. Note that
most of the prior designs [17], [1], [28] do not focus on optimizing power efficiency. Also, our approach does not exploit statistics of the routing table such as average length of the prefixes, while the trie-based and hash-based approaches optimize performance based on the features of the routing table. Thus, we compare our design with the TCAM solution only. The TCAM results are based on a state-of-the-art 20 Mb TCAM [29], assuming 4 chips are used in parallel to boost the size and bandwidth performance [8]. As can be seen, we achieve better power efficiency (6.6×) than TCAM. For our pipeline architecture, we have considered the power spent on FPGA, off-chip DRAM and I/O interfaces.

VI. CONCLUSION AND FUTURE WORK

In this paper we presented a high-performance pipeline architecture on FPGA as a scalable solution to IP lookup problem. We optimized the pipeline architecture using various hardware resources and an accurate model of off-chip memory access. We studied the tradeoffs between design parameters and system requirements in detail.

We have used careful floorplanning and considered the actual layout on FPGA for tree structures. In the future, we will explore efficient mapping from other data structures onto hardware for network applications.

REFERENCES


Table VIII: Power efficiency

<table>
<thead>
<tr>
<th>Solution</th>
<th>Power (W)</th>
<th>Delay per Lookup (ns)</th>
<th>Energy Consumption (µJ)</th>
<th>Throughput (MLPS)</th>
<th>IPv6 prefixes (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our design</td>
<td>18.74</td>
<td>282.0</td>
<td>5.28</td>
<td>360</td>
<td>256</td>
</tr>
</tbody>
</table>

TCAM solution also does not depend on the routing table features.