International Journal of Electrical and Electronics Engineers
Vol. No.8 Issue 01, January-June 2016
www.arresearchpublication.com
IJEEE
ISSN 2321 - 2055

BLOOM FILTER WITH NOVEL PARALLEL PREFIX TREE COMPARATOR

R.Logaprabhavathy[1], B.Priyanka[2]

[1,2]P.G.Scholar, Department of Electronics and Communication,
Ultra College of Engineering and Technology for Women, Madurai TN (India)

ABSTRACT

Bloom filters (BFs) provide a simple and effective way to check whether an element belongs to a set. BFs are implemented using electronic circuits .The contents of a BF are commonly stored in a high speed memory and required processing is done in a processor or in dedicated circuitry. In some cases, the performance of those systems is critical, and fastest comparison filter implementations are needed. In order to improve comparison capability of bloom filter, parallel prefix comparator used to faster data insertion and deletion . The results show that the proposed scheme can effectively compare in the associated set with minimum switching activity of circuits . The proposed scheme can be of interest in practical designs to effectively mitigate delay of filter with a reduced overhead in terms of circuit area and power.

Index Terms: Bloom Filter, Parallel Prefix Comparator, Switching Activity.

I. INTRODUCTION

Bloom filters are compact data structures for probabilistic representation of a set in order to support membership queries (i.e. queries that ask: “Is element X in set Y?”). This compact representation is the payoff for allowing a small rate of false positives in membership queries; that is, queries might incorrectly recognize an element as member of the set. In this Section we describe Bloom filters in detail. We briefly review Bloom filters. A Bloom filter represents a set S of m elements from a universe U using an array of n bits, denoted by $B_1, \ldots, B_n$, initially all set to 0. The filter uses a group H of k independent hash functions $h_1, \ldots, h_k$ with range $\{1, \ldots, n\}$ that independently map each element in the universe to a random number uniformly over the range. (This optimistic assumption is standard and convenient for Bloom filter analyses.) Deleting elements from a Bloom filter cannot be done simply by changing ones back to zeros, as a single bit may correspond to multiple elements. To allow for deletions, a counting Bloom filter (CBF) uses an array of n counters instead of bits; the counters track the number of elements currently hashed to that location.

II. RELATED WORK

Bloom filters have an essential role in network services and consequently the growing importance of operations such as information retrieval, distributed databases, packet content inspection, and cooperative caching results in the wide applications of Bloom filters that provide set-membership queries based on a relatively easy hardware implementation. The main design tradeoffs are the number of hash functions used (driving the computational

overhead), the size of the filter and the error (collision) rate.[1]counting Bloom filter (CBF) generalizes a Bloom filter data structure so as to allow membership queries on a set that can be changing dynamically via insertions and deletions. [2]a pipelined Bloom filter consists of two groups of hash functions. The first stage always computes the hash values. By contrast, the second stage of hash functions only compute the hash values if in the first stage there is a match between the input and the signature sought.[4] compressing Bloom filters might lead to significant bandwidth savings at the cost of higher memory requirements (larger uncompressed filters) and some additional computation time to compress the filter that is sent across the network. We do not detail here all theoretical and practical issues analyzed.

III. PROPOSED WORK

In order to improve the comparison time of bloom filter, we use parallel prefix comparator to compare incoming data. The comparator's design is elaborated which is based on using a novel parallel prefix tree .Each set or group of cells produces outputs that serve as inputs to the next set in the hierarchy, with the exception of set 1, whose outputs serve as inputs to several sets.

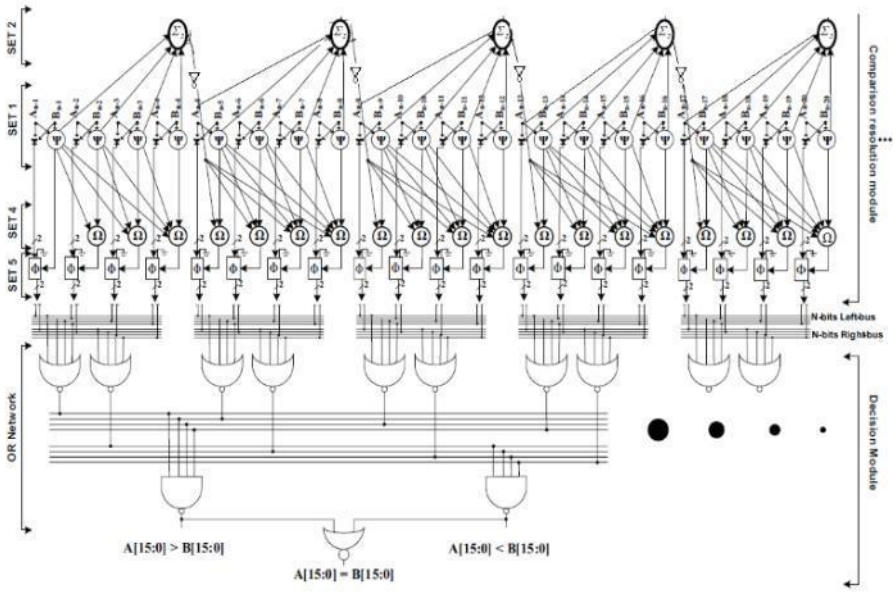

Figure 1: Parallel prefix tree architecture

Set 1 compares the *N*-bit operands *A* and *B* bit-by-bit, using a single level of *N* ψ type cells. The ψ type cells provide a termination flag D_k to cells in sets 2 and 4, indicating whether the computation should terminate. These cells compute (where $0 \le k \le N - 1$).

$$= \quad = \quad \oplus$$

Set 2 consists of Σ_2 type cells, which combine the termination flags for each of the four ψ type cells from set 1 (each 2-type cell combines the termination flags of one 4-b partition) using NOR-logic to limit the fan-in and fan-out to a maximum of four. The Σ_2 type cells either continue the comparison for bits of lesser significance if all four inputs are 0s, or terminate the comparison if a final decision can be made. For $0 \le m \le N/4–1$, there is a total of $N/4$ Σ_2 type cells, all functioning in parallel

International Journal of Electrical and Electronics Engineers
Vol. No.8 Issue 01, January-June 2016
www.arresearchpublication.com
IJEEE
ISSN 2321 - 2055

$4m+3$

: C 2, m = COMP(Di)

Set 3 consists of Σ_3 type cells, which are similar to Σ_2-type cells, but can have more logic levels, different inputs, and carry different triggering points. A Σ_3-type cell provides no comparison functionality; the cell's sole purpose is to limit the fan-in and fan-out regardless of operand bit-width.

$$\sum = , \quad = \quad ,$$

Set 4 consists of Ω-type cells, whose outputs control the select inputs of ϕ-type cells (two-input multiplexors) in set 5, which in turn drive both the left bus and the right bus. For an Ω-type cell and the 4-b partition to which the cell belongs, bitwise comparison outcomes from set 1 provide information about the more significant bits in the cell's Ω type cells. The number of inputs in the Ω-type cells increases from left to right in each partition, ending with a fan-in of five. Thus, the Ω type cells in set 4 determine whether set 5 propagates the bitwise comparison codes.

Set 5 consists of N ϕ-type cells (two-input, 2-b-wide multiplexers). One input is (A_K, B_K) and the other is hardwired to "00." The select control input is based on the Ω-type cell output from set 4. They define the 2-b as the left-bit code (A_K) and the right-bit code (B_K), where all left-bit codes and all right-bit codes combine to form the left bus and the right bus, respectively. The output Fk1,0 denotes the "greater-than," "less-than," or "equal to" final comparison decision Essentially, the 2-b code F1,0 k can be realized by OR-ing all left bits and all right bits separately.

IV. FPGA IMPLEMENTATION

Our proposed bloom filter using novel comparator has been coded in Verilog HDL and implemented in Spartan 3 FPGA board. The implementation and Xilinx simulation results listed below . The comparison between those two devices has been done based on the parameters like number of slices, number of IO's, number of bonded IOBs, number of slice flip -flops and time consumption.

s.no	Parameter	Existing	Proposed
1	Slice	69	59
2	Lut	120	104

After performing the synthesize process, the RTL schematic has been created automatically based on the functionality. The routing between the different cells can be viewed clearly by this schematic in figure 2.

International Journal of Electrical and Electronics Engineers
Vol. No.8 Issue 01, January-June 2016
www.arresearchpublication.com
IJEEE
ISSN 2321 - 2055

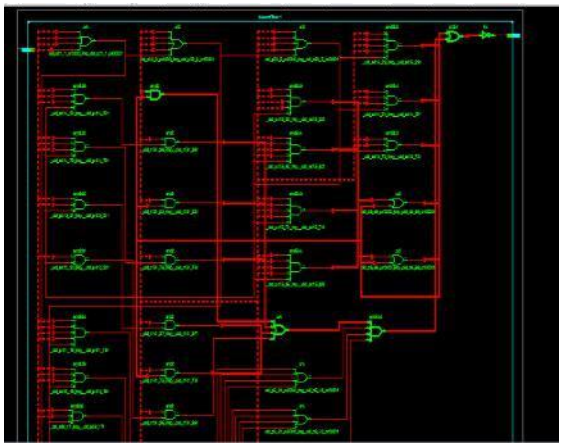

Figure 2: RTL schematic

V. SIMULATION RESULTS

Figure 3 and 4 shows behavioural simulation of filter .and corresponding data rejection and insertion into the memory

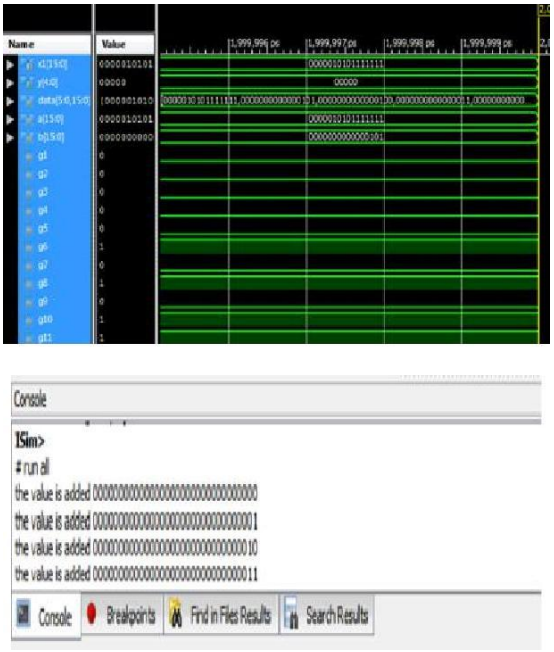

Figure 3: Data Insertion

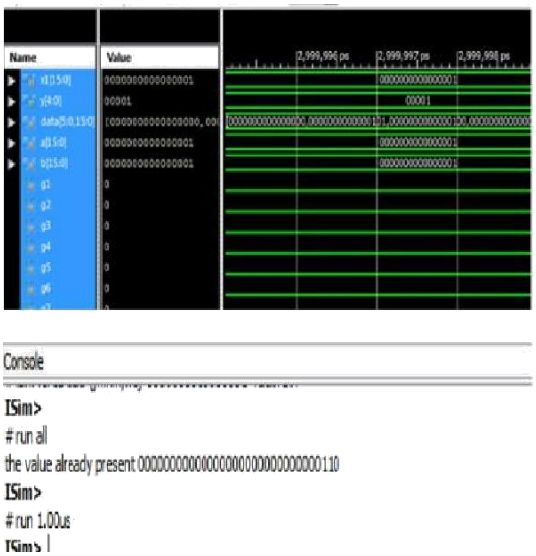

Figure 4: Data rejection

VI. PERFORMANCE ANALYSIS

The Figure 5 given below is shown that there is a considerable reduction in time and area based on the implementation results which have been done by using Spartan-3 processor. The proposed algorithm significantly reduces area consumption when compared to the existing system

International Journal of Electrical and Electronics Engineers
Vol. No.8 Issue 01, January-June 2016
www.arresearchpublication.com
IJEEE
ISSN 2321 - 2055

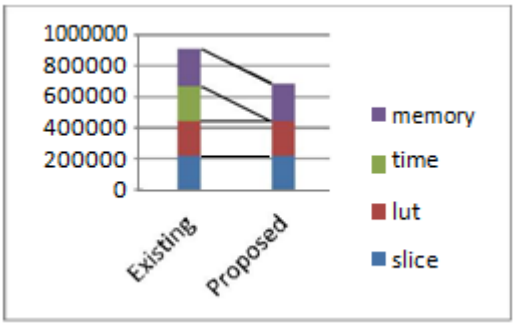

Figure 5: comparison analysis

VII. CONCLUSION

In this brief, a new application of BFs has been proposed. The idea is to use high-speed low-power comparator in BFs to compare element set. In particular comparator structured as parallel prefix trees with repeated cells in the form of simple stages that are one gate level deep with a maximum fan-in of five and fan out of four, independent of the input bit width. simulation results shows our proposed bloom filter has improved performance in terms of both comparison time and memory protection .

REFERENCES

[1] F.Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese,"An improved construction for counting bloom filters," in Proc. 14th Annu. ESA,2006, pp. 1–12.

[2] T.Kocak and I. Kaya, "Low-power bloom filter architecture for deep packet inspection," IEEE Commun. Lett., vol. 10, no. 3, pp. 210–212,Mar. 2006.

[3] A.Broder and M. Mitzenmacher, "Network applications of bloom filters:A survey," in Proc. 40th Annu. Allerton Conf., Oct. 2002, pp. 636–646.

[4] M. Mitzenmacher, "Compressed bloom filters," in Proc. 12th Annu. ACM Symp. PODC, 2001, pp. 144–150. [5]C.Fay et al., "Bigtable: A distributed storage system for structured data,"ACM TOCS, vol. 26, no. 2, pp. 1–4, 2008.

[6] B.Bloom, "Space/time tradeoffs in hash coding with allowable errors,"Commun. ACM, vol. 13, no. 7, pp. 422–426, 1970.