International Journal of Computer and Communication Technology

Volume 6 | Issue 2 Article 1

April 2015

COUNTING BLOOM FILTER ARCHITECTURE IN VLSI NETWORK SYSTEMS

NAGAMALLI. A
Department of E.C.E, A.I.E.T, Narsipatnam,Visakhapatnam , AndhraPradesh ,India, mallika.arasavalli@gmail.com

KEDARESWARARAO. M
Department of E C E,A.I.E.T, Narsipatnam, J. N.T.U.K, Kakinada, India, Kedhar_mutyala@yahoo.com

COUNTING BLOOM FILTER ARCHITECTURE IN VLSI NETWORK SYSTEMS

NAGAMALLI. A[1]& KEDARESWARARAO. M[2]

[1]Department of E.C.E, A.I.E.T, Narsipatnam,Visakhapatnam , AndhraPradesh ,India
[2]Department of E C E,A.I.E.T, Narsipatnam, J. N.T.U.K, Kakinada, India
Email: mallika.arasavalli@gmail.com, Kedhar_mutyala@yahoo.com

Abstract— the Counting Bloom Filter (CBF) is useful for real time applications where the time and space efficiency is the main consideration in performing a set membership tests. The CBF estimates whether an element is present in a large array or not by allowing false positives and by not permitting false negatives. In this paper CBF architecture is analyzed and has been implemented. There are two approaches of CBF, SRAM based approach using up/down counters and the LCBF using up/down LFSR unit. In this paper the LCBF architecture discussed and analyzed. In the latest VLSI technology it is easy to fabricate memories that hold a few million bits of data and addresses. But in the recent embedded memory technologies rather than mapping of addresses of 5000 bits of data using hashing functions we can concise in to single contiguous memory.

Keywords- *Bloom filter, Hashing mechanism, SRAM, LCBF, false positives, false negatives, look –ups, VLSI network systems*

I. INTRODUCTION

In 1970 BURTON HOWARD BLOOM proposed a elegant probabilistic data structure for testing set-membership tests, in the process of finding automatic hyphenation using an English dictionary. As there is limited core memory RAM that time, therefore the entire dictionary of the English language could not be held there. It is somewhat paradoxical to have to wait for an era of ever increasing core memory for Bloom filters to broadly take hold. Fig 1 shows the Bloom filter architecture.

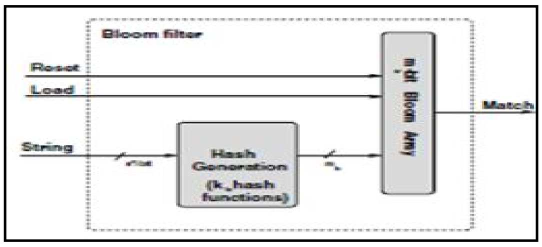


Fig .1 The Bloom Filter Architecture

Simply stated, a Bloom filters achieves space efficiency, less than O (mn) space, by allowing for a small probability of false positives, but no false negatives, to the set-membership question. Bloom Filter is a space efficient probabilistic data structure which is used for testing the presence of an element[9].

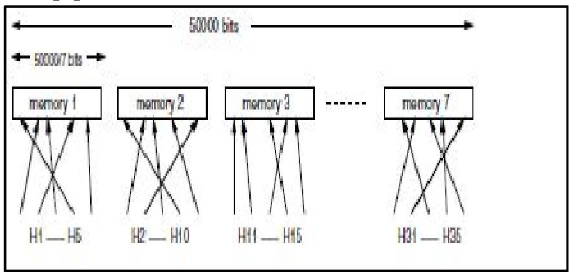


Fig.2. Bloom filter using multiple smaller memories with smaller lookup capacity

II. PROPERTIES AND TYPES OF THE BLOOM FILTER

A. *PROPERTIES*

Two useful properties of the Bloom filter for access control are:

1. The union of two Bloom filters of the same size and using the same hash functions can be obtained by bitwise ORing the two filters.

2. The intersection of two Bloom filters of the same size and using the same hash functions can be obtained by bitwise ANDing the two filters.

B. TYPES

Bloomier filters

In the case of "Bloomer filters [6]", a false positive is defined as returning a result when the key is not in the map. The map will never return the wrong value for a key that is in the map.

Stable Bloom filters

The Stable Bloom filter introduces false negatives, which do not appear in traditional bloom filters.

Scalable Bloom filters

The technique is based on sequences of standard bloom filters with increasing capacity and tighter false positive probabilities, so as to ensure that a maximum false positive probability can be set beforehand, regardless of the number of elements to be inserted[12].

Attenuated Bloom filters

The attenuated filter of level i indicates which services can be found on nodes that are i-hops away from the current node.

Counting Bloom Filter (CBF)

An increasing number of architectural techniques rely on hardware counting bloom filters (CBFs) to improve upon the power, latency and complexity of various key processor structures. CBF dynamically bypasses the conventional mechanism as frequently as possible. Accordingly, the benefits obtained through the use of a CBF depend on how frequently it can be utilized and on the CBF's energy and latency characteristics. The more tests are serviced by the CBF alone and the lower the power and latency of the CBF, the higher the benefits. Architectural techniques and application behavior determine how many tests can be serviced by the CBF.

In this paper CBF architecture is analyzed and applications are discussed for network systems.

III. CBF ARCHITECTURE

Counting Bloom Filter (CBF) consists of a small element addresses array that is connected to the array through hashing mechanism, i.e. multiple addresses mapped to single array [1], as shown in fig. 3

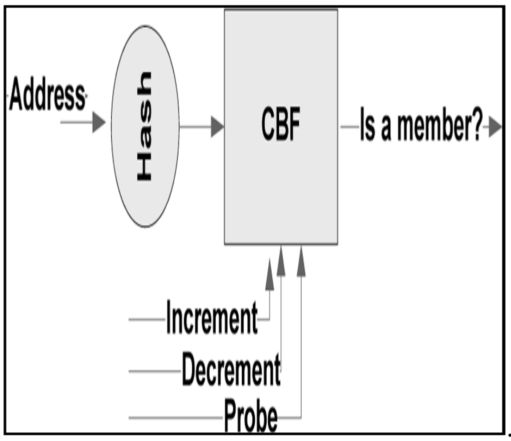


Fig.3 Counting bloom filter basic block diagram

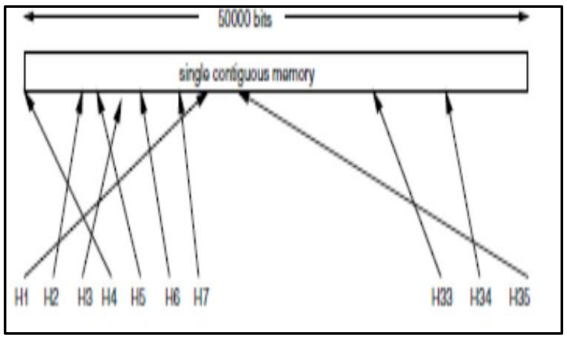


Fig. 4 shows CBF with single memory vector for a typical 5000 bits.

Fig.4 Bloom filter with single memory vector

CBF has three operations increment count (inc), decrement count (dec) and test if count is zero (probing) as shown in fig [3]. CBF is characterized by its number of entries and width of the count per entry. CBF can be accessed faster than any other data structure and needs very less energy when compared to that of accessing a large set of data, most of the membership tests are serviced by the CBF.

In the latest VLSI technology it is easy to fabricate memories that hold a few million bits of data and addresses. But in the recent embedded memory technologies rather than mapping of addresses of 5000 bits of data using hashing functions we can concise in to a single contiguous memory [3].

IV. CBF ALGORITHM

We want to test for membership in a set S = of n elements. The universe U of elements is typically very large. Let h1, h2, h3, hk be k independent hash functions with range .

Bloom filter consists of a bit array of m – bits with all set to 0.There must be k- hash functions defined each one hashes some set element to one of the m-array positions in a uniform random distribution format.

Adding an *element*: feed to each of the k hash functions to get k array positions. Set the bits at all these positions to 1.

Query for an *element: To* find whether the element is in it or not feed it to each of the k hash functions to get k array positions. If any of the bits at these positions are 0, the element is definitely not in the *set;* if it were, then all the bits would have been set to 1 when it was inserted. If all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, this is called as false positives.

False positives are acceptable but false negatives are not accepted , because removing an element from the simple bloom is not possible ,once the element is removed from the composite filter re-adding is also impractical[4].This problem can overcome in counting bloom filter architectures.

Basic steps followed by bloom filters with initial assumed state as empty,

1. Empty bloom filter is a bit array of m '0' bits.
2. Introduce 'k' hash functions, each maps key value to one of m array positions.
3. Insert element by feeding it to each hash function, to obtain k array positions. Set these bits to '1'.
4. Query an element by re-feeding it in to each hash function, and checking corresponding bit positions. If all bits are set 1, then either the element is in the filter or a false positive.
5. If bit positions of hashes of an element contains '0', then that element is definitely not in the filter; no false negatives.

This can be represented in an array as shown in below fig [5]

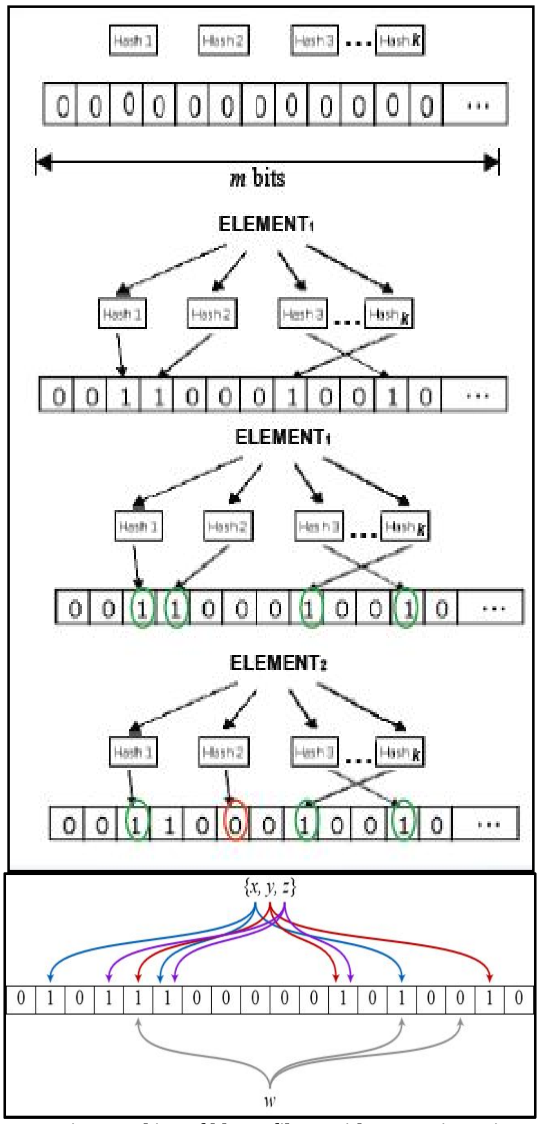


Fig.5 working of bloom filters with empty, insertion and query operations

V. ESTIMATION OF FALSE POSITIVE PROBABILITY (P)

The false positive probability P is a function of n number of elements in the filter of size m. Assuming an optimal number of hash functions, k=m/n ln 2.

Assume that a hash function selects each array position with equal probability. If *m* is the number of bits in the array, the probability that a certain bit is not set to one by a certain hash function during the insertion of an element is then - — .

The probability that it is not set by any of the hash functions is. - —

If we have inserted *n* elements, the probability that a certain bit is still 0 is

—

The probability that it is 1 is therefore

—

Now test membership of an element that is not in the set. Each of the *k* array positions computed by the hash functions is 1 with a probability as above. The probability of all of them being 1, which would cause the algorithm to erroneously claim that the element is in the set, is often given as

$$\left(1-\left[1-\frac{1}{m}\right]^{kn}\right)^{k} \approx \left(1-e^{-kn/m}\right)^{k}$$

This is not strictly correct as it assumes independence for the probabilities of each bit being set. However, assuming it is a close approximation we have that the probability of false positives decreases as *m* (the number of bits in the array) increases, and increases as *n* (the number of inserted elements) increases. For a given *m* and *n*, the value of *k* (the number of hash functions) that minimizes the probability is

$$\frac{m}{n}\ln 2 \approx 0.7\frac{m}{n},$$

This gives the false positive probability of

$$2^{-k} \approx 0.6185^{m/n}$$

The required number of bits *m*, given *n* (the number of inserted elements) and a desired false positive probability *p* (and assuming the optimal value of *k* is used) can be computed by substituting the optimal value of *k* in the probability expression above:

$$p=\left(1-e^{-(m/n\ln 2)n/m}\right)^{(m/n\ln 2)} \qquad (1)$$

And can be simplified to

$$\ln p = -\frac{m}{n}(\ln 2)^2 \qquad (2)$$

This results in:

$$m = -\frac{n\ln p}{(\ln 2)^2} \qquad (3)$$

This means that for a given false positive probability *p*, the length of a Bloom filter *m* is proportionate to the number of elements being filtered *n*[2]. While the above formula is asymptotic (i.e. applicable as *m*, *n* → ∞), the agreement with finite values of *m*, *n* is also quite good; the false positive probability for a finite bloom filter with *m* bits, *n* elements, and *k* hash functions is at most

$$\left(1-e^{-k(n+0.5)/(m-1)}\right)^{k} \qquad (4)$$

So we can use the asymptotic formula if we pay a penalty for at most half an extra element and at most one fewer bit [1].

VI. COUNTING BLOOM FILTER IMPLEMENTATION

CBF implementation can be handled in two methods

1. SCBF –SRAM BASED CBF
2. LCBF- LOWPOWER CBF

In earlier work SRAM based CBF (SCBF) was implemented, where up/down counters were used in place of up/down LFSR unit in each partition.

In LCBF architecture input is a*n* address given to a pre decoder. Through global decoder it is transferred to local decoder and gated clock *circuitry*, where the clock is generated according to the input address and local decoded data.

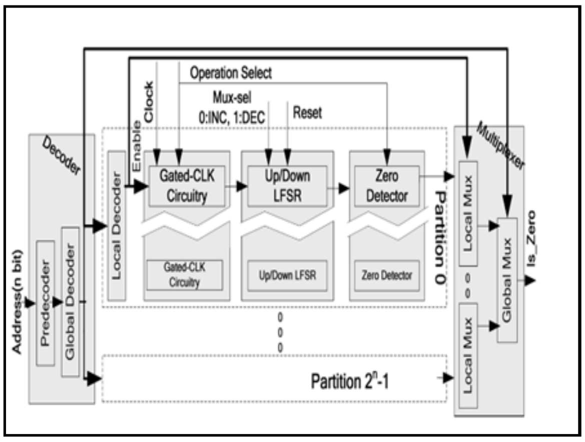


Fig .6 Basic architecture of LCBF: low power fast counting bloom filter [1]

Basic cells consist of DWL based decoder structure, up/down LFSR unit and a zero detector with local and global multiplexer units. For a 3 level decoder DWL STRUCTURE can be represented as shown below in fig. 7

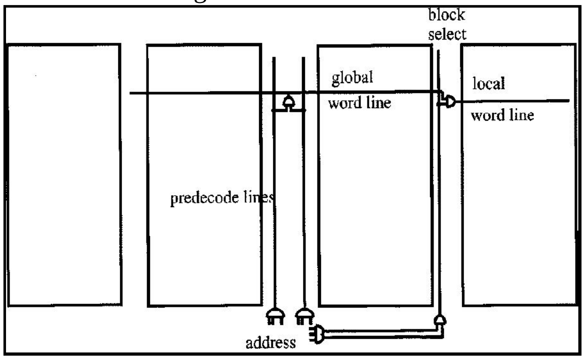


Fig .7 DWL architecture using a three level decoder [4]

The critical path for a typical decoder implemented using the DWL architecture as shown in fig. 8

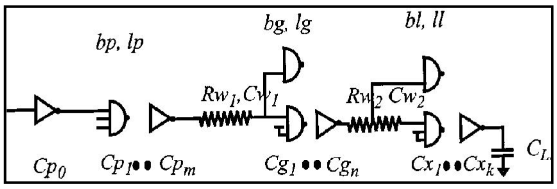


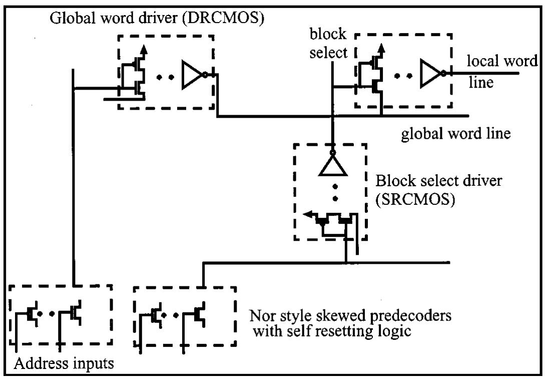
Fig. 8 Critical path for a 3 level decoder

Decoders encompasses the circuits from the address input to the word line as shown in fig .6

Design of each decoder consists of combinational modes:

DR CMOS – delayed reset logic uses delayed version of one of the inputs to conditionally reset the gate

SR CMOS – self resetting logic uses output to reset the gate



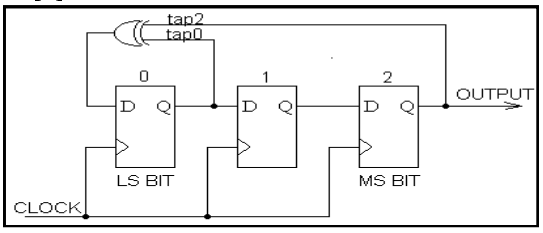
Global word driver (DRCMOS)
block select
local word line
global word line
Block select driver (SRCMOS)
Nor style skewed predecoders with self resetting logic
Address inputs

Fig .9 Schematic of fast- low power three-level decoder structure.

DRCMOS techniques will have the least logical effort and hence the lowest delay.

Next block in internal structure of LCBF is up/ down LFSR unit with Galois implementation

LFSR will produce a pseudorandom sequence of maximal length 2^{n-1} states, where n is the number of stages The sequence will then repeat from the initial state for as long as the LFSR is clocked[5]. An example of a 3 bit LFSR and truth table are shown in fig. 10[7]

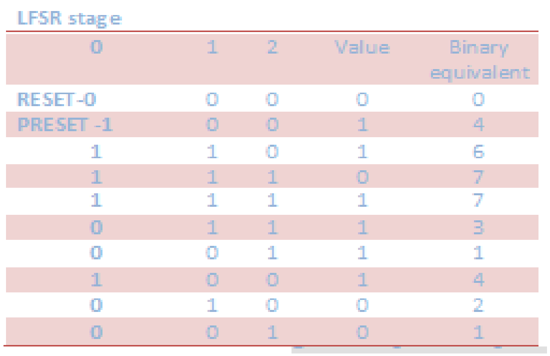


LFSR stage			Value
0	1	2	
0	0	1	1
1	0	0	4
1	1	0	6
1	1	1	7
0	1	1	3
1	0	1	5
0	1	0	2

Fig.10 3-bit LFSR and truth table

Thus changing the tap points and setting the pre-set value many pseudo random sequences will be generated; following truth table is an example

Table (1) Truth table for 3-bit LFSR making tap 0 as 1



LFSR stage				
0	1	2	Value	Binary equivalent
RESET-0	0	0	0	0
PRESET -1	0	0	1	4
1	1	0	1	6
1	1	1	0	7
1	1	1	1	7
0	1	1	1	3
0	0	1	1	1
1	0	0	1	4
0	1	0	0	2
0	0	1	0	1

Output of each LFSR given to a zero detector which is selected according to the reset logic and the corresponding result transferred through local multiplexer, global multiplexer, as a result output of global multiplexer combinely gives set member ship test result as " is-zero" as Shown in fig. 6

CONCLUSION

Main advantage of using LFSR in the CBF architecture is switching noise can be reduced as there is no "rollover" unlike a binary counter. By and then various types of bloom filters were studied. CBF architecture is analyzed to obtain probability of minimum false positive.

Thus by using 3 level decoder structure with "DRCMOS logic" delay can be reduced as the decoders designed with lowest delay and critical path delay also reduced for a 3 level decoder of LCBF.

Thus LCBF estimates and speeds the set membership test for an element with minimum false positives, and reduced path delay structures in DWL decoders.

For efficient tradeoff between delay and energy in a large range RAM structures in VLSI can be designed by the simple mechanism of varying the sizes of the word driver inputs and the total logical effort can be reduced significantly by skewing the gates in the inner structures of DWL decoders of CBF.

A technique where Constant time computation of the algorithm along with the scalability is needed, such as applications of network intrusion detections, which require real time processing then LCBF is a best method of implementing the hardware architecture.

ACKNOWLEDGMENT

We acknowledge with thanks our respective college colleagues who have given support for successful presentation of this paper.

We also thank Sri G.TirumalaRao, Professor, Department of E.C.E, G.V.P. College of Engineering, for his valuable suggestions in completing this paper.

REFERENCES

[1] E. Safi, A. Moshovos, and A. Veneris, "L-CBF: A fast, low-power counting bloom filter architecture," in *Proc. Ann. Int. Symp. Low Power Electron. Des.*, Oct. 2006, pp. 250–255.

[2] http:/ en.wikipedia.org/wiki/BLOOM FILTER

[3] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd S. Sproull, John W. Lockwood, Deep Packet Inspection using Parallel Bloom Filters, IEEE Micro, vol. 24, no. 1, pp. 52-61, 2004

[4] Bharadwaj S. Amrutur and Mark A. Horowitz, *Fellow*,IEEE "Fast Low-Power Decoders for RAMs" journal of solid-state circuits, vol. 36, no. 10, october 2001

[5] P. Alfke, "Efficient shift registers, LFSR counters, and long pseudorandom sequence generators," Xilinx, San Jose, CA, Appl. Note 052, Jul. 1996.

[6] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal, "The Bloomier filter: an efficient data structure for static support lookup tables", Proceedings of the Fifteenth Annual ACM-SIAM symposium on discrete algorithms, pp. 30-39, 2004

[7] http:/www.markharvy.info/fpga/lfsr/lfsr.

[8] Parvez Anandam, Network Access Control using Bloom filters, March 12, 2007.

[9] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal, The Bloomier filter: An efficient data structure for static support lookup tables, Proceedings of the Fifteenth Annual ACM-SIAM symposium on discrete algorithms, pp. 30-39, 2004.

[10] M.Abramovili, M.A.Breues, A.D.Friedman-"Digital Systems Testing and Testable Design" Jaico publications.

[11] Parag K.Lala- "Fault Tolerant& Fault Testable Hardware Design" -PHI Publications.

[12] Simha Sethumadhavan, Rajagopalan Desikan Doug Burger Charles R. Moore Stephen W. Keckler Department of Computer Sciences &University of Texas "Scalable Hardware Memory Disambiguation for High ILP Processors" Proceedings of the 36th International Symposium on Micro architecture (MICRO-36 2003) 0-7695-2043-X/03 2003 IEEE .

[13] Abhishek Kumar Jun (Jim) Xu College of Computing Georgia Institute of Technology Space Code Bloom Filter for Efficient Traffic Flow Measurement-Li Li ,Bell Labs

❖❖❖