Reconfigurable Finite State Machine based IP Lookup Engine for High Speed Router

Madhav Desai, Ritu Gupta, Abhay Karandikar, Kshitiz Saxena and Vinayak Samant

Information Networks Laboratory

Department of Electrical Engineering

Indian Institute of Technology, Bombay

Mumbai 400 076, India

Ritu Gupta is now with University of Illinois, Urbana-Champaign. Kshitiz Saxena is now with Sun Microsystems, Bangalore, India and Vinayak Samant is with Indian Institute of Management, Lucknow. This work was performed while these authors were at Indian Institute of Technology (IIT), Bombay.

Abhay Karandikar is the corresponding author. email:karandi@ee.iitb.ac.in. Tel:+91-22-25767439. Fax:+91-22-25723707

Abstract

IP address lookup is one of the major performance bottlenecks in high–end routers. This paper presents an architecture for an IP address lookup engine based on programmable finite state machines (FSMs). The IP address lookup problem can be translated into the implementation of a large FSM. Our hardware engine is then used to implement this FSM using a structured approach, in which the large FSM is broken down into a set of smaller FSMs which are then mapped into reconfigurable hardware blocks. The design of our hardware engine is based on a regular and well structured architecture, which is easy to scale. Our simulation results demonstrate that the FSM based architecture can easily scale to wire speed performance at OC-192 rates. Unlike previous approaches, the performance of our architecture is not constrained by memory bandwidth and is therefore in principle scalable with VLSI technology.

Index Terms

High Speed Internet, IP Address Lookup, Finite State Machine, Wire Speed Performance, Routing.

I. Introduction

Over the past several years, the Internet has witnessed remarkable growth both in terms of the number and the bandwidth requirements of applications. The high bandwidth need requires faster communication links and faster packet processing capability in routers. IP address lookup remains one of the major performance bottlenecks for faster packet processing in routers. The IP Lookup Engine must have the ability to process every packet at 'wire speed', *ie*, it must be able to forward minimum size IP packets at line rates. This would mean that the lookup engine must have the capability to process millions of packets per seconds at line rates of the order of Gigabits per second.

The primary reason for the complexity of IP address lookup is that the lookup requires longest prefix match computation which arises due to Classless Interdomain Routing (CIDR). CIDR is usually employed in the Internet to address the problem of IP address space inefficiency. Instead of classful address aggregation (based on Class A, B and C type addresses), CIDR allows arbitrary aggregation of addresses at various points within the Internet and the address prefix bits common to all the IP addresses at an aggregation point is used to denote the aggregate. The address prefix used to represent an aggregate of networks can vary in length from 1 to 32 bits depending upon the aggregation. Each routing table entry is represented by a (address prefix, prefix length) pair. When an IP packet is received, a lookup is performed in the routing table to determine which of the prefixes match the destination IP address. If multiple prefixes match, then the output interface corresponding to the longest match is selected and the packet is forwarded on this interface.

In this paper, we denote an address prefix by a bit string of 0 and 1 followed by $*$ and up to a maximum length of 32 bits. For example prefixes could be of the form $110*$ and $11*$. If the destination IP address is 11011, it matches both the entries. The one with the longest match, *ie*, $110*$ is selected.

IP address lookup has been an active area of research in the recent past. Several innovative solutions [1], [2], [3], [4], [5], [6], [7], [8], [9], [10] have been proposed in the literature for faster IP lookup algorithms. An exhaustive survey of these techniques can be found in [11]. Most of these approaches organize the routing database of address prefixes in a clever manner so as to reduce the required memory accesses for faster search times. The performance is, however, still limited by memory bandwidth and the worst case performance cannot always be guaranteed. We, therefore, believe that despite several interesting solutions proposed by earlier researchers, we do not have a 'truly' scalable architecture, *ie*, an architecture that would scale with VLSI technology.

In this paper, we take a fundamentally different approach and present a new paradigm for the problem of address lookup. The primary contribution of the paper is an architecture for a lookup engine in the form of a finite state machine (FSM) that can be implemented using reconfigurable hardware. The FSM is generated using the routing database. Backbone routers may have routing database with more than 50000 prefixes. For such a large number of prefixes, the number of states in the corresponding FSM would be very large. In the design of such Very Large FSMs (VLFSM), we adopt a structured approach wherein the size and performance of the implementation is predictable given the knowledge of the graph of the FSM. We achieve this using decomposition of FSM graphs. The large FSM is decomposed into several small FSMs to realize an efficient architecture. One of the advantages of such an architecture is that it is not constrained by memory bandwidth.

The rest of the paper is organized as follows. Section II briefly discusses various solutions to IP address lookup schemes that have been proposed in the past. Section III proposes our lookup engine architecture. We also illustrate the generation of the route lookup FSM. Section

IV discusses how a VLFSM can be decomposed into small FSMs. We describe the results of decomposition of a VLFSM generated from the Mae-East routing database [12] and the database of the Finnish University and Research Network (FUNET). Most research works in this area have primarily tested their algorithms on these databases. Section V illustrates the simulation results of IP address lookup performance. Section VI briefly discusses the VLSI architecture. In Section VII, we highlight some issues related to scalability and area optimizations of our approach. We finally conclude the paper in Section VIII.

II. IP Address Lookup Schemes

The performance of an IP address lookup algorithm is characterized by two parameters. One is the lookup time, *ie*, the time required to determine the output interface corresponding to a destination IP address. Since routing table entries may change due to route updates, the time required by an IP lookup algorithm to respond to the changes in the routing database is another parameter used to characterize the IP address lookup. This is termed as update time.

IP address lookup engines can be broadly classified into two categories: one based on Content Addressable Memories (CAM) and the other Processor-Memory combination. Our scheme actually creates a third category, which is based on Programmable FSMs.

CAM based Solutions

In this model the address lookup can be performed using Ternary CAM (TCAM) [13]. In a TCAM, a mask of bits can be specified per word. The routing table entries are stored in the order of decreasing prefix lengths. The longest prefix match thus corresponds to the first entry among all the entries that match the destination IP address. A TCAM is an attractive solution for high speed IP address lookup, however, TCAMs with large sizes are typically very expensive. Historically, the CAM technology has also not kept pace with the Dynamic Random Access Memory (DRAM) technology in terms of storage density. TCAMs are also very poor in terms of update time, though, recently some progress [14] has been made in this direction.

Processor-Memory based Solutions

In this model, the routing table entries are present in memory and the lookup algorithm runs on a processor. The objective of an IP lookup algorithm is to organize the routing database in an intelligent manner such that during actual lookup operation as few memory accesses are required as possible.

For backbone routers with a large routing database, architectures that use off–chip DRAMs are usually employed. One measure of the lookup speed of an algorithm is the number of DRAM accesses that are required to be made. New memory technologies such as Synchronous DRAM (SDRAM), RAMBUS, Double Data rate DRAM (DDR-DRAM) employ some form of parallel banks of memory and interleaving can be performed to hide memory access latency. As pointed out in [6], each memory technology introduces some tradeoffs and IP lookup algorithms need to be carefully tuned across memory architectures to extract the best performance.

One of the simplest ways to store the routing database of address prefixes in memory is in the form of a one bit trie. A trie is a tree like data structure where the prefix bits are used to create tree branches. Several modifications to the basic one bit trie have been proposed in the literature. Path Compression techniques [15] can be used to remove those nodes from the tree that have only one child. The missing nodes are denoted by a skip value that indicates how many nodes have been skipped on the one way path.

Instead of one bit tries, multibit tries [3] can also be used. Unlike in a one bit trie, where each node branches to its children depending upon the value of a binary bit, in multibit tries, the branching occurs depending upon the value of several bits taken together. The search also proceeds by inspecting several bits simultaneously. The number of bits to be examined is called the stride length. The strides can be of fixed length or of variable lengths at different levels of the tree. The address prefixes need to be converted into prefixes with lengths equal to the stride. The length of the strides offers a tradeoff between memory and search speed. The optimal strides can be computed using the prefix length distribution [3].

In LC Tries [16], each complete subtree of height l is converted into a subtree of height 1 with 2^l children. Thus a one bit trie gets converted into a multibit trie. In [5], a multibit trie with fixed stride lengths is implemented using memory banks. By appropriate pipelining, the authors claim that lookup can be performed in one memory access. This is, however, achieved at the expense of large memory size.

Though the above algorithms have provided very novel techniques to arrange the prefixes in an intelligent manner, we believe that the scalability of Processor-Memory solutions is limited by the fact that the lookup operation requires DRAM accesses. Despite the considerable progress, the DRAM technology has not kept pace with the processor technology. We attempt to address this issue by proposing an architecture for lookup engine that is not constrained by memory bandwidth. The solution presented in [6] also belongs to Processor-Memory model but uses a very novel technique based on multibit tries and bit maps to compress the data structure so that it can be fit into a processor's cache. Since the accesses are now performed on cache, the lookup performance improves considerably. This is an attractive solution and we compare our results with this scheme later in the paper.

III. Proposed Approach

A. Lookup Engine Architecture

Our basic architecture for the lookup engine is shown in Figure 1. The reconfigurable hardware shown in the figure performs the address lookup. A reconfigurable hardware is essentially a circuit whose behavior can be modified on the fly. The hardware implementation is in the form of a programmable FSM. The state transition table can be loaded onto it by the processor. The processor computes the finite state machine for a given routing database of address prefixes and then compiles it in a format appropriate for programming the reconfigurable hardware. Due to a routing update, if there is a change in the routing database, then the state machine is recomputed again. In case of changes either the entire FSM may have to be reprogrammed or changes to some part of the FSM graph may have to be made.

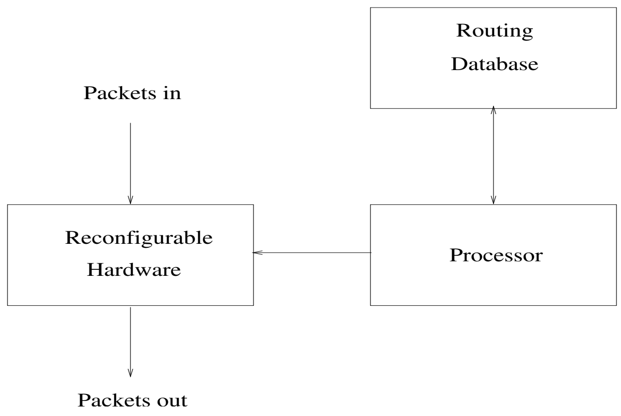

Fig. 1. Lookup Engine Architecture

All the approaches discussed in Section II (except CAM based solutions) require several memory accesses and thus the memory bandwidth is one of the major performance bottlenecks. The

FSM based architecture can be efficiently implemented using flip flops (FF) and all the memory accesses can be reduced to accessing the high speed registers. The implementation can thus scale with VLSI technology.

We now present ways to generate an efficient FSM for the routing database and evaluate the lookup speed of such an approach.

B. Finite State Machine for Lookup Engine

To illustrate our basic approach, we first consider generating an FSM from the one bit trie structure. Consider the one bit trie for the following prefix table. The procedure for generating

Prefix	Output Interface
001*	1
000*	2
11*	3
110*	2

TABLE I

EXAMPLE PREFIX TABLE

the one bit trie begins at the root node for each prefix. The bits in the prefix are examined one by one. If the bit is 0, then the left node is formed (if not already present) otherwise if the bit is 1, then the right node is formed. To generate an FSM, each node in the resulting one bit trie can be associated with a state in a finite state machine. The one bit trie and the corresponding FSM for the prefix database of Table I are illustrated in Figure 2. We call this FSM a one bit FSM. The state transition table for this FSM is given in Table II. The state corresponding to an address prefix stores the corresponding output interface.

To perform a lookup, the destination IP address bits are applied in serial order and the state machine makes a transition from one state to another depending upon the bit. If a state representing a valid interface is encountered, the state number is stored. The IP address bits are applied until a node whose next state is FINAL is encountered. The search is terminated and the output interface number corresponding to the last stored state is retrieved. In the given example, if the destination IP address is 1100, the states that would be traversed are S1, S3, S5 and S8 and

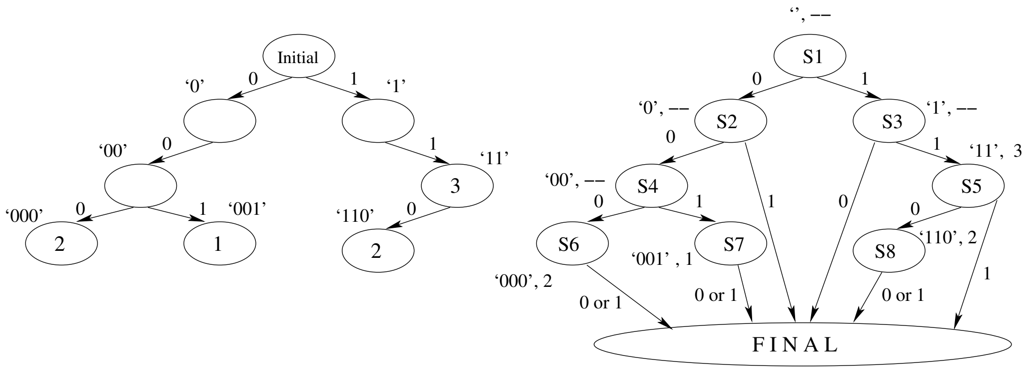

Fig. 2. One Bit Trie and the Corresponding FSM for Example Prefix Table

Current State	Input bit	Next State	Output
S1	0	S2	-
S1	1	S3	-
S2	0	S4	-
S3	1	S5	3
S4	0	S6	2
S4	1	S7	1
S5	0	S8	2
S2	1	FINAL	-
S3	0	FINAL	-
S5	1	FINAL	-
S6	0	FINAL	-
S6	1	FINAL	-
S7	0	FINAL	-
S7	1	FINAL	-
S8	0	FINAL	-
S8	1	FINAL	-

TABLE II

STATE TRANSITION TABLE

the output would be 2. In the worst case, 32 states might have to be traversed for IP lookup, but note that these are not memory accesses and hence can be quite fast.

Database	Number of Entries	Number of States
FUNET	41625	125669
Mae-East	38367	172826
Ripe	55532	88961

TABLE III

NUMBER OF STATES IN FSM

For practical routing databases, the number of states present in the state machine would be large. We have calculated the number of states in the FSM generated for Mae-East, FUNET and Ripe routing databases. The results are summarized in Table III. The large number of states may result in inefficient hardware implementation and higher delays. We, therefore, follow a structured approach where the FSM graph is partitioned into smaller machines each containing some maximum number of states say 1024. These smaller machines are then connected together as shown in Figure 3 to obtain the overall functionality of the FSM. The partitioning of FSM graph is done with a view to minimize the area of the chip and make the performance of the chip predictable. Each machine is made reconfigurable by introducing memory cells. When one machine completes the processing of a packet, the packet is handed over to an appropriate machine by the central block. We now investigate methods for decomposition of state machine into smaller state machines by exploiting the structure of FSM graphs.

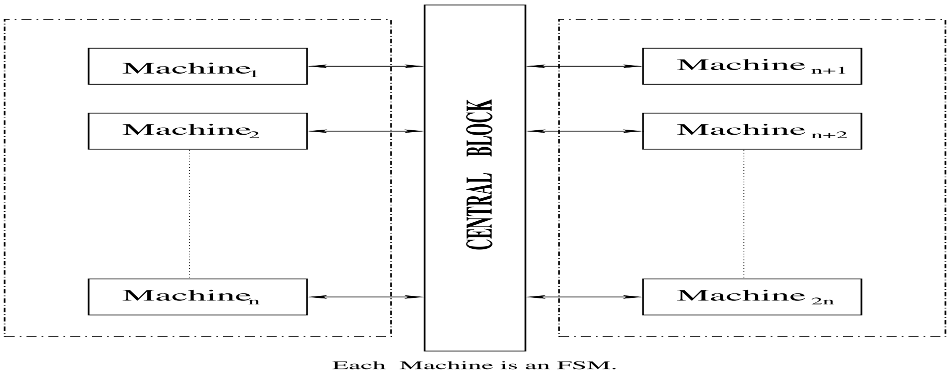

Fig. 3. Partitioned FSM Architecture for Reconfigurable Hardware of Lookup Engine

IV. Decomposition of the Finite State Machine for IP Lookup Engine

We first illustrate through an example how a state machine can be decomposed into two inter-dependent state machines. Figure 4 illustrates a simple FSM. The state transition table is given in Table IV.

The state machine can be decomposed into two state machines A and B. The decomposition is illustrated in Table V. Note that no two states of the original FSM are associated with the same states in FSM A and B. These partitions are called orthogonal. The state transition tables for these FSMs are given in Table VI and VII. The FSMs A and B are inter dependent in the sense that the next state of the FSM depends not only on its state but also on the state of the other machine.

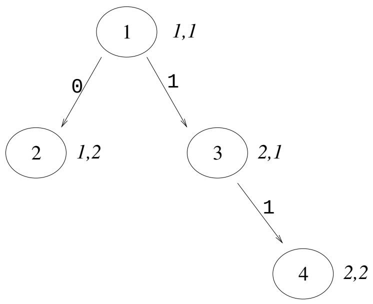

Fig. 4. One Bit FSM

Current State	Input	Next State
1	0	2
1	1	3
3	1	4

TABLE IV

STATE TRANSITION TABLE FOR ORIGINAL FSM

This example has illustrated the concept of decomposition of a state machine. However, we could not achieve any reduction in the number of states for this particular example. We now investigate ways to efficiently decompose a large finite state machine generated from the one bit trie of address prefixes.

Original FSM State	State in FSM A, B
1	1,1
2	1,2
3	2,1
4	2,2

TABLE V

DECOMPOSITION OF ORIGINAL FSM

Current State	Input	FSM B State	Next State
1	0	1	1
1	1	1	2
2	1	1	2

TABLE VI

STATE TRANSITION TABLE FOR FSM A

Current State	Input	FSM A State	Next State
1	0	1	2
1	1	1	1
1	1	2	2

TABLE VII

STATE TRANSITION TABLE FOR FSM B

There are several standard structures like completely balanced trees, paths, *etc* that can be efficiently decomposed into smaller state machines at the cost of slightly more complex transitions [18]. We have explored the decomposition of an FSM that exploits these structures and their applicability in the state machine of a routing database. Specifically, we have investigated the presence of these structures in Mae-East and FUNET routing database in order to ascertain the decomposition of FSM generated for these databases. We have observed from the results that the number of such structures is very small and hence efficient decomposition can not be obtained for minimizing the number of states present in FSM. These results and observations have been discussed in [20]. Since the applicability of these results in an IP lookup engine is limited, we do not pursue them here. Instead we attempt a two level hierarchical decomposition of the FSM. The first level of decomposition is the additive decomposition which is done using we what call as the topographical breakdown. Each of the smaller FSMs obtained by topographical breakdown is orthogonally decomposed to realize an efficient hardware architecture.

A. Topographical Breakdown

In this section, we illustrate how the finite state machine can be decomposed into smaller FSMs using topographical breakdown. In topographical breakdown, the large FSM is decomposed into smaller FSMs such that the number of states in each smaller FSM does not exceed some upper limit. These smaller FSMs are interconnected to function as a one bit FSM. For example, Figure 5 illustrates how a one bit FSM can be decomposed into smaller FSMs. In this case, the number of states in each small FSM is not more than 3. Note also that each small FSM has only one incoming edge. The topographical breakdown may also be performed to optimize the number of smaller machines. The algorithm for topographical breakdown employed by us is explained in **Algorithm** 1.

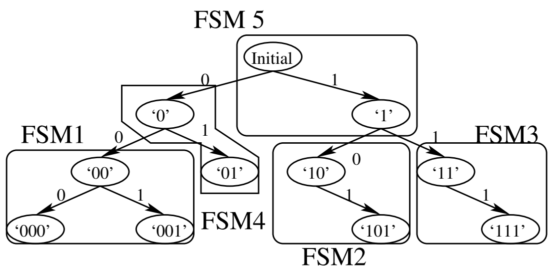

Fig. 5. Topographically Breaking FSM into smaller FSM

Note that the flow of control is always directed downwards in a one bit FSM, *ie*, after visiting a node the control is transferred to its child node. Thus the smaller FSMs are not interactive. After having traversed through one FSM, the control is transferred to its child FSM and the current FSM can start processing the next packet. These FSMs can work in a pipelined fashion and increase the throughput. The time spent in one FSM is dependent on its depth. To operate make a $link$-$list$ of all leaf nodes of the FSM

$L \leftarrow 1$

while $link$-$list$ is not empty **do**

$node \leftarrow$ first element of $link$-$list$ (and remove the element from the $link - list$)

if $node$ is included in $[FSM_1$ to $FSM_{L-1}]$ **then**

goto 3

end if

$FSM_L \leftarrow \phi$

$size \leftarrow 0$

$FSM_L \leftarrow FSM_L \cup node$

$increment \leftarrow$ number of children of $parent(node)$ not included in $[FSM_1$ to $FSM_{L-1}]$ + 1 // $parent(node)$ implies parent of the node // 1 is added for $parent(node)$

if ($size + increment < MAX\_SIZE$) **then**

$FSM_L \leftarrow FSM_L \cup$ children of $parent(node)$ not included in $FSM_L \cup parent(node)$

$size \leftarrow size + increment$

$node \leftarrow parent(node)$

goto 11

else

Add the deepest nodes among the children of $parent(node)$ not included in FSM_L to the $linked - list$.

$L \leftarrow L + 1$

goto 3

end if

end while

Algorithm 1: Topographical Breakdown these smaller FSMs efficiently, all of them should ideally have the same depth so that the time spent in all FSMs is same.

In the rest of the paper, we refer to each of such smaller FSMs obtained by topographical breakdown as *machine*. Each machine is now partitioned into two sub FSMs using orthogonal decomposition. These sub FSMs are referred to as *Partitions* in the rest of this paper. The approach followed by us is a factoring of the original machine (with N states) into two partitions (each with $\sqrt{N}$ or more states) and this factoring can be viewed as the meet of the two orthogonal partitions of the set of states of the original machine. For example, a machine with 100 states may be decomposed into two orthogonal partitions each with 10 states.

B. Orthogonal Partitioning

Let $S = \{s_1, s_2 \dots, s_N\}$ represent the set of states of the machine. Let the machine M itself be represented by a 6 tuple

$$M = (\Sigma, \Delta, S, \delta, \lambda, s_{reset})$$

where Σ is the input alphabet, Δ is the output alphabet, $\delta : \Sigma \times S \rightarrow S$ is the next state function and $\lambda : \Sigma \times S \rightarrow \Delta$ is the output function. The state s_{reset} is the reset state.

A partition of S is a set of disjoint and nonempty subsets of S whose union is S. Let $A = \{A_1, A_2, \dots, A_{n_1}\}$ and $B = \{B_1, B_2 \dots, B_{n_2}\}$ be the partitions of the set of states S. The partitions A and B are called orthogonal if the following conditions are met

1) $n_1 \times n_2 \geq N$.
2) For $i \leq n_1$ and $j \leq n_2$ either $A_i \cap B_j = \phi$ or $A_i \cap B_j = \{s_k\}$ for some $k \leq N$.

The finite state machines corresponding to the partitions A and B are interacting in the following way. The states in finite state machine of Partition A (Partition B) are the elements of A (B). The inputs are drawn from $\Sigma \times B$ ($\Sigma \times A$) while the outputs are the states themselves. Thus the finite state machine corresponding to, say, partition A is represented by $(\Sigma \times B, A, A, \delta_A, \lambda_A, A_{reset})$ where A_{reset} is an element of A that contains s_{reset}.

The next state function

$$\delta_A : \Sigma \times B \times A \rightarrow A$$

is defined such that $\delta_A((\sigma, B_j), A_i) = A_k$ if and only if $\delta(\sigma, s_m) = s_n$ where $s_m \in A_i \cap B_j$, $s_n \in A_k$ and $\sigma \in \Sigma$. The function λ_A simply outputs the current state of A.

A combinational circuit is used to generate the outputs corresponding to the original machine. The combinational circuit implements the function

$$f : \sigma \times A \times B \rightarrow \Delta$$

with $f(\sigma, A_i, B_j) = \lambda(\sigma, s_k)$, where s_k is the unique state in $A_i \cap B_j$. It can be easily proved that the terminal behavior of the machine is identical to that of the partitions along with the combinational logic.

The orthogonally decomposed machine as above belongs to *General Decomposition* category discussed in the classical literature [17]. The FSM graph of a decomposed machine may contain parallel edges. If all the parallel edges emanating from a state s_1 and terminating on the state s_2 are replaced by a single directed edge (called multi-edge) from s_1 to s_2 then we get a diagraph $G(S, E_s)$ where S is a set of states and E_s is the set of edges. This diagraph may, however, contain self loops.

We would like to decompose each machine into two orthogonal partitions: Partition A and Partition B such that the number of multi–edges in diagraphs corresponding to the partitioned machines is minimized. It has been shown in the previous work of one of the authors of this paper (Desai) [18] that this reduces the area of the chip as the number of multi–edges has a direct correlation with the area of the chip. It is also shown in [18] that this decomposition helps in reducing the delays as well. The Greedy Algorithm of [18] has been found to give 4-8% less area and about 80-100% improvement in delays than conventional state assignment approaches considered in the literature [19]. This provides us the motivation to apply the Greedy Algorithm for decomposition of FSM of routing database. The Greedy Algorithm builds Partition A by forcing tightly connected states into the same block so that the edges between them are replaced by self-loop. While building the second Partition B, the states are added one at a time by doing a local search (on assignments of states to vertices in the partition) to determine which assignment creates the minimum number of additional edges in the partition.

The pseudocode is explained in **Algorithm** 2. The algorithm generates the partitions A and B. The algorithm creates partitions such that A will consist of n_1 blocks each containing at most n_2 states and B will consist of n_2 blocks each containing at most n_1 states.

The function `Find_most_adjacent_state` returns the state with the maximum number of fan-in edges from the states already in a given block of the Partition A.

Inputs:

G(S,E_s) = State Transition Graph

S = Set of states ($\|$ S $\|$ = N)

E_s = Set of edges

$n_1 > 0, n_2 > 0, n_1 \times n_2 \geq N$

Outputs:

$PartitionA, \| PartitionA \| \leq n_1, \| A_i \| \leq n_2$

$PartitionB, \| PartitionB \| \leq n_2, \| B_j \| \leq n_1$

Steps:

G_B = Graph corresponding to Partition B

$V_{left} \leftarrow S$

$A_i \leftarrow \emptyset, 0 \leq i < n_1$

$B_j \leftarrow \emptyset, 0 \leq j < n_2$

$i \leftarrow 0$

while $(i < n_1)$ **do**

  while ($\| A_i \| < n_2$) **do**

    $v \leftarrow$ Find_most_adjacent_state(A_i, V_{left}, G)

    $A_i \leftarrow A_i \bigcup v$

    $V_{left} \leftarrow V_{left} - v$

    Put_in_most_suitable_block($v, G, PartitionB, G_B$)

  end while

  $i \leftarrow i + 1$

end while

Algorithm 2: Greedy Algorithm for FSM Decomposition

The function `Put_in_most_suitable_block` chooses one block among all the available blocks of Partition B such that when a state is put in that block, the additional number of edges created in the diagraph corresponding to partition B is minimized.

C. Decomposition of FSM for IP Lookup Engine

In our implementation, the FSM is generated from the one bit trie obtained after pre-processing the routing database. This one bit FSM is topographically decomposed into machines. The maximum possible number of states in each machine is set to some upper limit. We have performed our simulations with different maximum limits of 256, 512, 1024 and 2048 states and evaluated the performance of our approach in each case. The machines are decomposed into inter-dependent orthogonal partitions using the above mentioned Greedy Algorithm.

The number of states in the one bit FSM for Mae-East, FUNET and Ripe database have already been given in Table III. The results of the topographical breakdown are given in Table VIII for the cases when the maximum number of states in each machine is restricted to 256, 512, 1024 and 2048. These are called Case 1, 2, 3 and 4 respectively in the table. The results after each machine is decomposed into orthogonal partitions are given in Table IX. Note that we can achieve a substantial reduction in the total number of states. The results in the second and third column of Table IX indicate that we can achieve an effective partitioning of a very large FSM generated from routing database. The simulations performed with a C model of the IP lookup engine based on such a partitioned FSM architecture are discussed in the next section.

Cases	Number of Machines		Minimum Number of States in Machine		Maximum Number of States in Machine	
	Mae-East	FUNET	Mae-East	FUNET	Mae-East	FUNET
Case 1	1043	775	1	1	256	256
Case 2	520	370	56	51	509	511
Case 3	260	186	82	88	1019	1013
Case 4	125	88	361	88	2019	2023

TABLE VIII

RESULTS OF TOPOGRAPHICAL BREAKDOWN FOR MAE-EAST AND FUNET DATABASES

V. SOFTWARE MODEL AND PERFORMANCE RESULTS

We have developed a C model of IP address lookup engine based on FSM to analyze the performance of such a lookup chip in the network. In the software model, a VLFSM is generated

Cases	Number of States after Orthogonal Decomposition		Average Number of States in Partition A		Average Number of States in Partition B	
	Mae-East	FUNET	Mae-East	FUNET	Mae-East	FUNET
Case 1	27878	20413	10.8	10.5	15.9	15.7
Case 2	19718	14148	14.9	15.2	23	23
Case 3	13844	9970	21.2	21.6	32	32
Case 4	9568	6823	30.5	31.5	46	46

TABLE IX

RESULTS OF ORTHOGONAL DECOMPOSITION FOR MAE EAST AND FUNET DATABASES

using the routing database. The FSMs are controlled by FSM controller. For the simulation, the packet trace is generated from the database. For this, the prefixes are chosen at random from the database and these become addresses (prefix with trailing zeros) that need to be looked up.

The function of the FSM controller in our software model is to input the IP address bits one by one to an appropriate FSM. It also passes control from one FSM to another when necessary. We also assume pipelining of the packets. When one FSM completes its processing of an IP address, it can start processing another packet. In the model, we have assumed that 2 clock cycles are required for the transfer of control from one FSM to another FSM. This assumption is reasonable. Even if we assume that 5 clock cycles are required to transfer control, our simulations indicate that the average number of cycles for lookup without pipelining increases by about 10 cycles while with pipelining, it increases only by 2-3 cycles.

The results of our simulations for Mae-East and FUNET routing database are given in Table X. From the results, we observe that the average number of cycles required for lookup without pipelining is of the order of 30 cycles for Mae-East and 20 cycles for FUNET. The pipelining reduces the number of cycles to about 5-8 cycles for lookup. Note that the average number of cycles without pipelining in case of Mae-East is more than that of FUNET, however, with pipelining Mae East requires actually smaller number of cycles. This is possible as the depth of the FSM graph is not same for every machine and the effect of pipelining is dependent on the FSM graph of a routing database and its partitioning.

If we assume that each machine runs at about 100 MHz (as has been explained in the next

Cases	Avg number of cycles without pipelining		Avg number of cycles with pipelining	
	Mae-East	FUNET	Mae-East	FUNET
Case 1	34.4	23.2	4.5	6.7
Case 2	32.9	22.1	4.6	7.5
Case 3	31.6	21.4	5.1	7.9
Case 4	30.1	20.8	5.8	8.3

TABLE X

LOOKUP TIME FOR MAE-EAST AND FUNET DATABASES

section, it is possible to achieve this clock speed in the actual realization of the chip), then the average lookup time is of the order of 200-300 ns without pipelining and 50- 80 ns with pipelining. These roughly translate to lookup capability of the order of about 10 million lookups per second. We have compared these results with that of [6]. The authors of [6] have carefully tuned their scheme to give performance that can scale to OC-192 line rates. Our scheme gives comparable and with pipelining even superior performance to Tree Bit map scheme. Moreover, as pointed out earlier, our approach is not constrained by memory bandwidth.

In the next section, we briefly discuss the VLSI architecture of the lookup chip. For lack of space, we discuss only the salient features. The details of the VLSI implementation including the layout of the lookup engine have been discussed at length in [21].

VI. VLSI ARCHITECTURE

Our basic lookup chip has been illustrated in Figure 3. In an actual implementation, the number of machines and the maximum number of states in the machine will be fixed. Each machine shown in Figure 3 is drawn in Figure 6 with input and output ports. These machines are controlled by the central control logic. The central block is also responsible for transferring control from one machine to another. Bit is the port where IP address bits are applied. The signal Select is low when the machine is operated in lookup mode. For update purposes, the signal Select is high. The other input signals shown in the figure are also required for programming the state machine. The output Done is high when the lookup operation terminates.

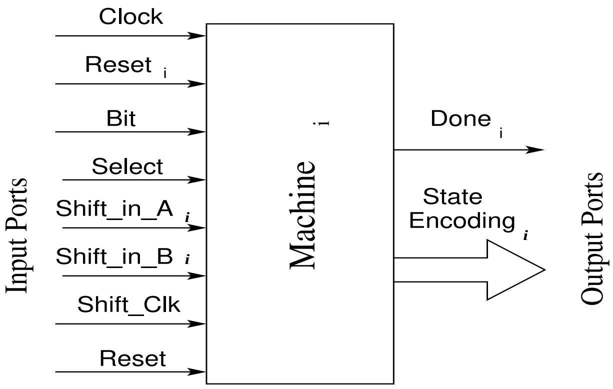

Fig. 6. Machine with Input and Output Ports

The FSM generated from the routing database is topographically decomposed into smaller FSMs that are mapped onto these machines. Each of these machines is partitioned into two orthogonal partitions as explained above. The block diagram of the machine with orthogonal partitions is shown in Figure 7. Each combinational logic block computes the next state or

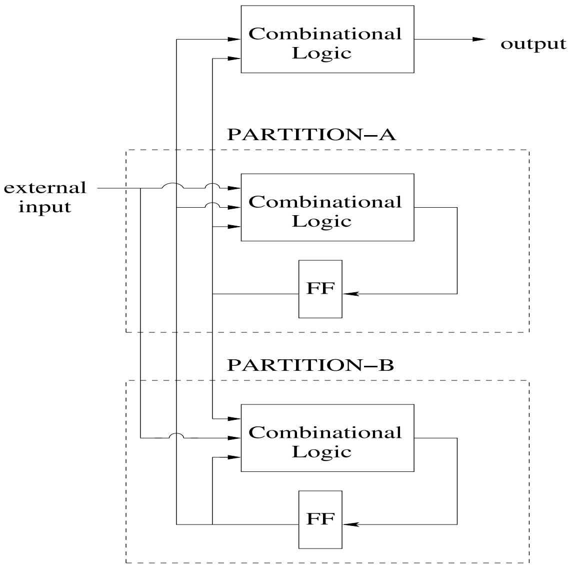

Fig. 7. Block Diagram of Machine with Orthogonal Partitions

the output function using the present states of the two partitions and the external input. The complete VLSI architecture is depicted in Figure 8. Currently, we have not performed any Boolean optimization while implementing each machine. The optimization can be performed to achieve more number of states per machine within the same area.

Each partition has been implemented with a Double PLA (DPLA) as shown in Figure 9. Any

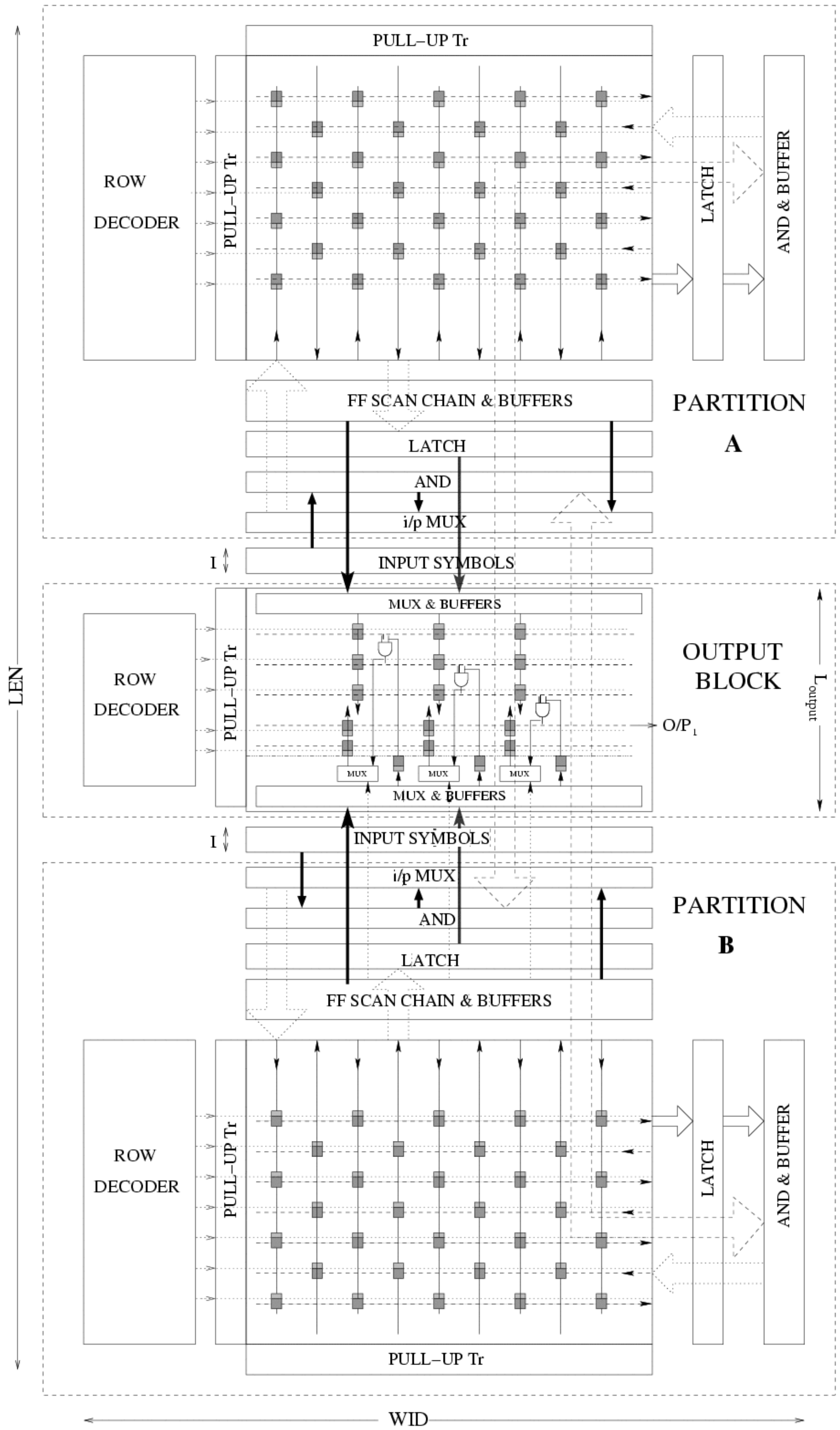

Fig. 8. Architecture of Each Machine with Partitions

FSM graph can be realized by programming the DPLA structure. The programmability of the

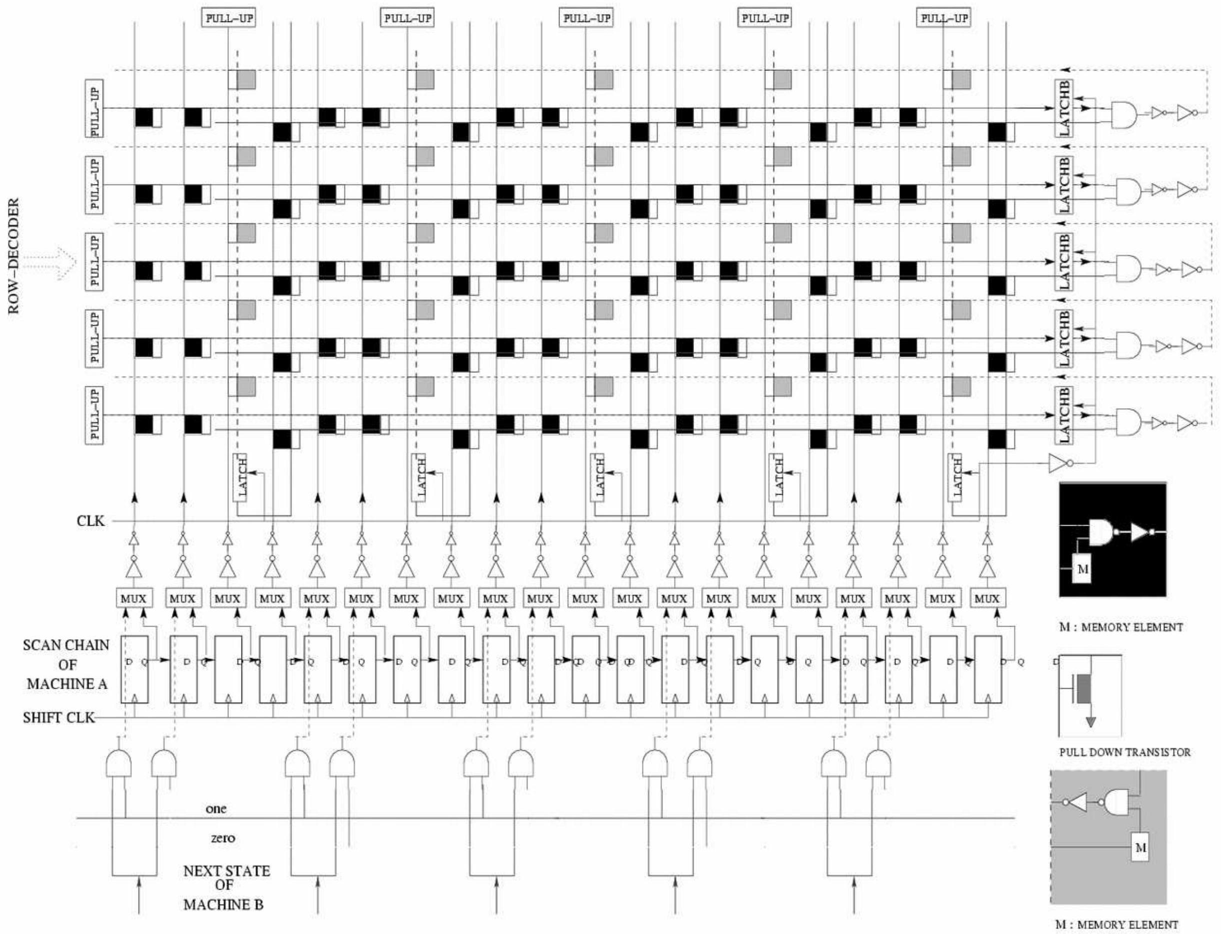

Fig. 9. Architecture of Partition with Multiplexed PLA

architecture is obtained by introducing memory elements. When the FSM is to be updated, the input signal Select is set high. The values to be stored in the memory elements of a particular row of the PLA structure are scanned in the Flip-Flop scan chain (FF Scan Chain shown in Figure 8) using the input ports Shift_in_A for Partition A and Shift_in_B for Partition B.

For lack of space, we do not discuss here the clock distribution scheme and the design of Central Block (See Figure 3). The central block is responsible for controlling the FSM machines and transferring the control from one machine to another. The performance of the chip has been predicted through simulations. MAGIC version 7.1 is used to do the layout for 0.25μ technology N-well process. The circuit layouts are extracted and simulated using SPICE3f5 [22] and IRSIM, Version 9.5. The transistor model BSIM3, Version 3.1 [23], level 8 is used. The power supply used is 2.5 Volts. A plot of clock period (T_c) of operation of the machine as a function of N (where N is the total number of states in the machine) is shown in Figure 10. These simulation

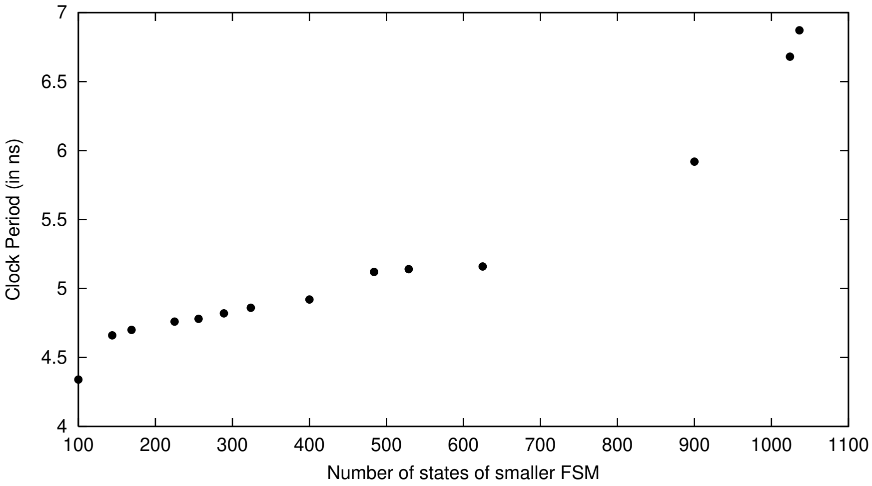

Fig. 10. Clock Period (T_c) of Operation (in ns) as a function of Number of States in the Machine

results indicate that each machine can easily work at 100-150 MHz. Thus we can achieve the lookup performance predicted by the C model of the chip.

VII. Discussion

Apart from the lookup speed, other key issues that need to be considered in judging a route lookup solution are: area of the silicon, the problem of applying updates, and the scalability of the solution.

A. Area Considerations and Optimizations

The estimated area for a 1024 state programmable block is $1mm^2$ in the reference 0.25μ technology [21]. Thus, assuming a 50% overhead for the central block, we estimate that a $1cm \times 1cm$ VLSI chip in 0.25μ technology can accommodate $50,000$ states, that is, a table with up to $10,000$ prefix entries. This packing will improve as technology scales, and in 0.13μ technology which is currently available, a $1cm \times 1cm$ chip should accommodate a routing database with $40,000$ entries.

The area efficiency can be improved further if we take advantage of the following observation: in the pipelined architecture, very few machines are active simultaneously (at most 5 machines are active in the cases we have studied). Thus, there is no need to have 50 machines to accommodate a $50,000$ state FSM. Instead, we can make do with a smaller number of machines, and use dynamic reconfiguration to map several sub FSMs onto the same hardware block. One possible architecture is indicated in Figure 11. In this figure, the individual machines $M1, M2, \ldots$ have attached configuration memories $C1, C2, \ldots$ which store the possible sub FSM that can be mapped to the corresponding machine. At run–time, a machine is programmed with a sub FSM depending on the state of the lookup. Essentially, this architecture trades off machine area for memory area. We are presently exploring this theme further, and initial results indicate that a considerable saving in area can result. The key technical difficulties that need to be addressed here are improvement of the reprogramming time of each machine, and the effective scheduling of the individual machines.

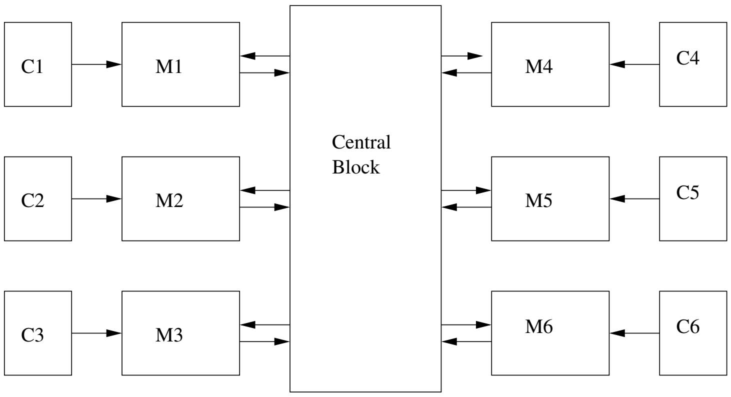

Fig. 11. Dynamically Reconfigurable Architecture

B. The Update Problem

Whenever the routing database changes, this change needs to be applied to the hardware. This update problem has two parts: first, the lookup trie needs to be modified; second, the modified update needs to be applied to the hardware. We will concentrate on the second aspect of the update problem. Addition or deletion of an entry typically leads to a small change in the trie. When the trie changes, the change is typically localized within a single sub FSM. Hence the number of machines to be updated is small. Thus, for such local changes the update is not a serious issue.

If the entire database changes, then all the sub FSMs need to be updated. In the worst case, assuming a sub FSM size of 1024 states, $15K$ bits of configuration memory need to be transferred for each machine [21]. Thus, for a $1cm \times 1cm$ circuit consisting of 50 machines, $750K$ bits need to be transferred into the system. Assuming a 32 bit access bus operating at 100 MHz, this much data can enter the chip in 25 μsec. Thus, we can safely claim that the physical update for a $1cm \times 1cm$ chip with $50K$ states can be performed in less than 100 μsec.

C. Scaling Issues

FSM based lookup engine is scalable with respect to the following properties.

1) As process technology improves, and feature sizes shrink, there is a direct benefit in packing and speed. This benefit will track the technology scaling exactly (as opposed to memory speeds which do not track as well).
2) As the prefix lengths scale (towards IPV6), the lookup FSM size is determined mainly by the size of the routing table, and not as much by the length of the tag being looked up. On the other hand, CAM based approaches will have a direct penalty here.
3) As the key size increases, the key stored with the tag can be moved to memory outside the FSM. Thus, a lookup in the FSM can be followed by a (guaranteed) single lookup in memory.

VIII. Conclusions

In this paper, we have presented a new approach for an IP address lookup engine based on a programmable FSM. Apart from CAM based solutions, the previous research has concentrated mainly on Processor-Memory solutions. The database is arranged in various forms of trie data structures. The researchers have carefully tuned their algorithms across various memory architectures and exploited some new DRAM architectures to extract the best performance. In contrast to these approaches, we have presented an approach that is not constrained by memory bandwidth. The partitioning approach followed by us results in a regular and well structured FSM. Our simulation results demonstrate that we can achieve a wire speed performance at OC-192 rates. We believe that our approach has the potential to scale with VLSI technology.

We have thus far considered an FSM generated from a one bit trie. Multibit tries and its variants can also be considered within our framework. As indicated earlier, the VLSI architecture can be optimized for area by using dynamic reconfiguration. This can improve the packing properties of the architecture. Our current partitioning algorithm is not optimal. Indeed, we have developed some heuristics (See [21]) that have shown some promise for better performance.

ACKNOWLEDGEMENTS

The authors would like to acknowledge Prof H Narayanan for stimulating discussions and the annonymous reviewers for their comments that have improved the quality of the paper.

REFERENCES

[1] M Waldvogel, G Varghese, J Turner and B Plattner, "Scalable High Speed IP Routing Lookups", *Proc ACM SIGCOMM'97*, 1997.

[2] M Degermark, A Brodnik, S Carlsson and S Pink, "Small Forwarding Tables for Fast Routing Lookups", *Proc ACM SIGCOMM'97*, 1997.

[3] V Srinivasan and G Varghese, "Fast Address Lookup using Controlled Prefix Expansion" *Proc ACM SIGMETRICS'98*, June 1998, pp 1-11.

[4] B Lampson, V Srinivasan and G Varghese, "IP Lookup using Multiway and Multicolumn Search", *Proc IEEE INFOCOM'98*, 1998, pp. 1248-1256.

[5] P Gupta and N McKeown, "Routing Lookups in Hardware at Memory Access Speeds", *Proc IEEE INFOCOM'98*, 1998, pp 1240-47.

[6] W Eatherton, "Full Tree Bit map", Master's Thesis, Washington University, 1999

[7] G Cheung and S McCanne, "Optimal Routing Table Design for IP Address Lookups under Memory Constraints", *Proc IEEE INFOCOM'99*, 1999, pp. 1437-44.

[8] N Huang and S Zhao, "A Novel IP Routing Lookup Scheme and Hardware Architecture for Multigigabit Switching Routers", *IEEE Journal on Selected Areas in Communications*, Vol 17, Nu 6, June 1999, pp. 1093-1104.

[9] M Kobayashi, T Murase and A Kuriyama, "A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing", *Proc ICC*, 2000, pp. 1360-1363.

[10] P Wang, C Chan and Y Chen, "A Fast IP Routing Lookup Scheme", *Proc ICC*, 2000, pp. 1140-1144.

[11] M Ruiz-Sanchez, E Biersack and W Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", *IEEE Network Magazine, March/April 2001*, pp 8-23.

[12] Prefix database Mae East available at http://www.merit.edu/ipma.

[13] NetLogic Microsystems, http://www.netlogicmicro.com/, 2001

[14] P Gupta and D Shah, "Fast Updates on Ternary-CAMs for Packet Lookups and Classification", *Proc of Hot Interconnects VIII*, August 2000.

[15] D Morrison, "PATRICIA-A Practical Algorithm to Retrieve Information Coded in Alphanumeric", *Journal of ACM*, Vol 15, No 4, October 1968, pp 514-34.

[16] S Nilsson and G Karlsson, "IP Address Lookup using LC Tries", *IEEE Journal on Selected Areas in Communications*, Vol 17, No 6, June 1999, pp 1083-92.

[17] Z Kohavi, *Switching and Finite Automata Theory*, Tata McGraw Hill Reprint, 1996.

[18] R Shelar, M Desai and H Narayanan, "Decomposition of Finite State Machines for Area and Delay Minimization", *Proc IEEE International Conference on Computer Design*, 1999, pp 620-625.

[19] T Villa and A Sangiovanni Vincentenlli, "NOVA: State Assignment for Optimal Two Level Logic Implementation", *IEEE Transactions on Computer Aided Design*, Vol 9, No 9, September 1990, pp 905-24.

[20] K Saxena, *"Finite State Machine based Design for Packet Forwarding Engine"*, Master's Thesis (under Dual Degree), IIT Bombay, June 2001.

[21] R Gupta, *"Reconfigurable Hardware Design for Packet Router"* Master's thesis (under Dual Degree), IIT Bombay, June 2002.

[22] T Quarles, AR Newton, DO Peterson and AS Vincentelli, *"SPICE3f5 User's Manual"*, University of California, Berkeley, March 1994.

[23] http://www-device.eecs.berkeley.edu/bsim3

List of Figures

List of Tables

AUTHORS' BIOGRAPHIES

Madhav Desai received his BTech degree in Electrical Engineering from IIT, Bombay in 1984 and MS and PhD degrees from University of Illinois at Urbana-Champaign in 1986 and 1991 respectively. During the period 1992-96, he worked in Semiconductor Engineering Group of Digital Equipment Corporation (DEC) at Hudson, MA where he was Principal Engineer. Since 1996, he is working in IIT Bombay where currently he is an Associate Professor. Dr Desai has consulted extensively for industries in the area of VLSI design. His primary research interests include VLSI Design, Design Automation, Graph Theory and Circuits and Systems.

Ritu Gupta received her BTech and MTech (under Dual Degree) in Electrical Engineering with specialization in Microelectronics from IIT Bombay in June 2002. She is now a graduate student in University of Illinois at Urbana-Champaign. Her research interests include VLSI design and computer architecture.

Abhay Karandikar received his MTech and PhD degrees from IIT Kanpur in 1988 and 1994 respectively. During 1988-89, he worked in Indian Space Research Organization. During 1994-97, he worked in Center for Development of Advanced Computing, Pune as Team Coordinator in High Speed Communications Group. Since 1997, he is working in IIT Bombay where currently he is an Associate Professor in the department of Electrical Engineering. Dr Karandikar has consulted extensively for industries in the area of communications network. His research interests include Quality of Service in Internet, VLSI in Communications Systems and Statistical Communications Theory.

Kshitiz Saxena received his BTech and MTech degrees (under Dual Degree) in Electrical Engineering with specialization in Communications and Signal processing from IIT, Bombay in June 2001. He is currently working in Sun Microsystems, Bangalore, India as Member, Technical Staff. His research interests include high speed router design and optical networks.

Vinayak Samant received his BTech degree in Electrical Engineering from IIT, Bombay in 2002. He is currently working towards his MBA degree from Indian Institute of Management at Lucknow, India.