

# ***Novel Hardware Architecture for Fast Address Lookups***

**Pronita Mehrotra**  
**Paul D. Franzon**

Department of Electrical and Computer Engineering  
North Carolina State University  
{pmehrot,paulf}@eos.ncsu.edu

This research is supported by



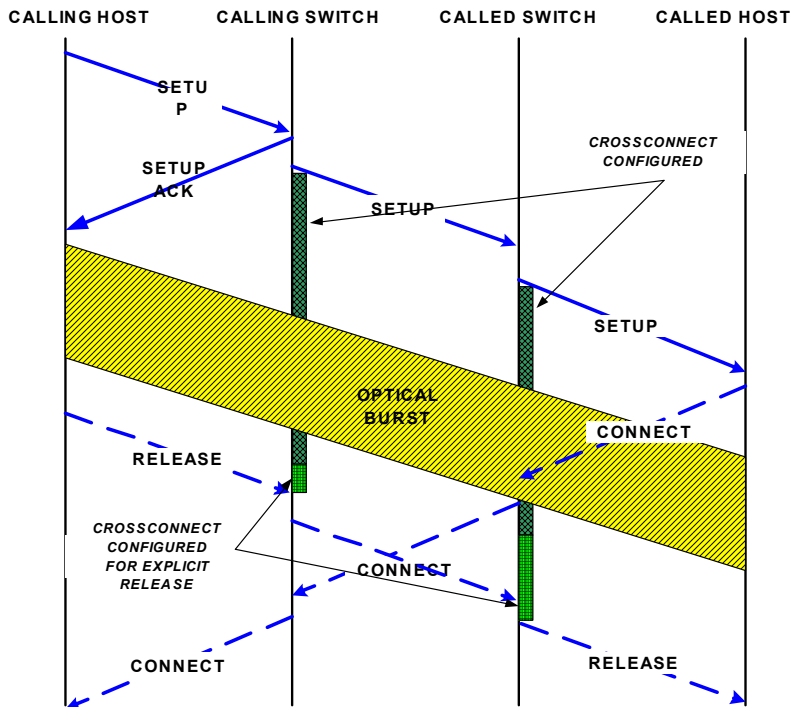
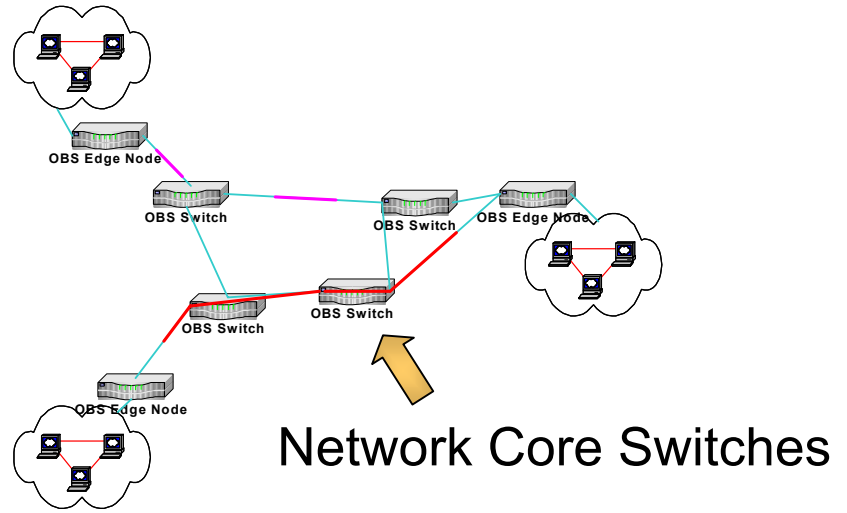
# Outline



- ❖ Background & Motivation
- ❖ Description of the Scheme
- ❖ Example for a 16-way Implementation
  - Building SRAM/DRAM data
  - Searching an address
- ❖ Hardware Implementation of the Scheme
- ❖ Performance
  - Memory Consumption
- ❖ Conclusions

# Background

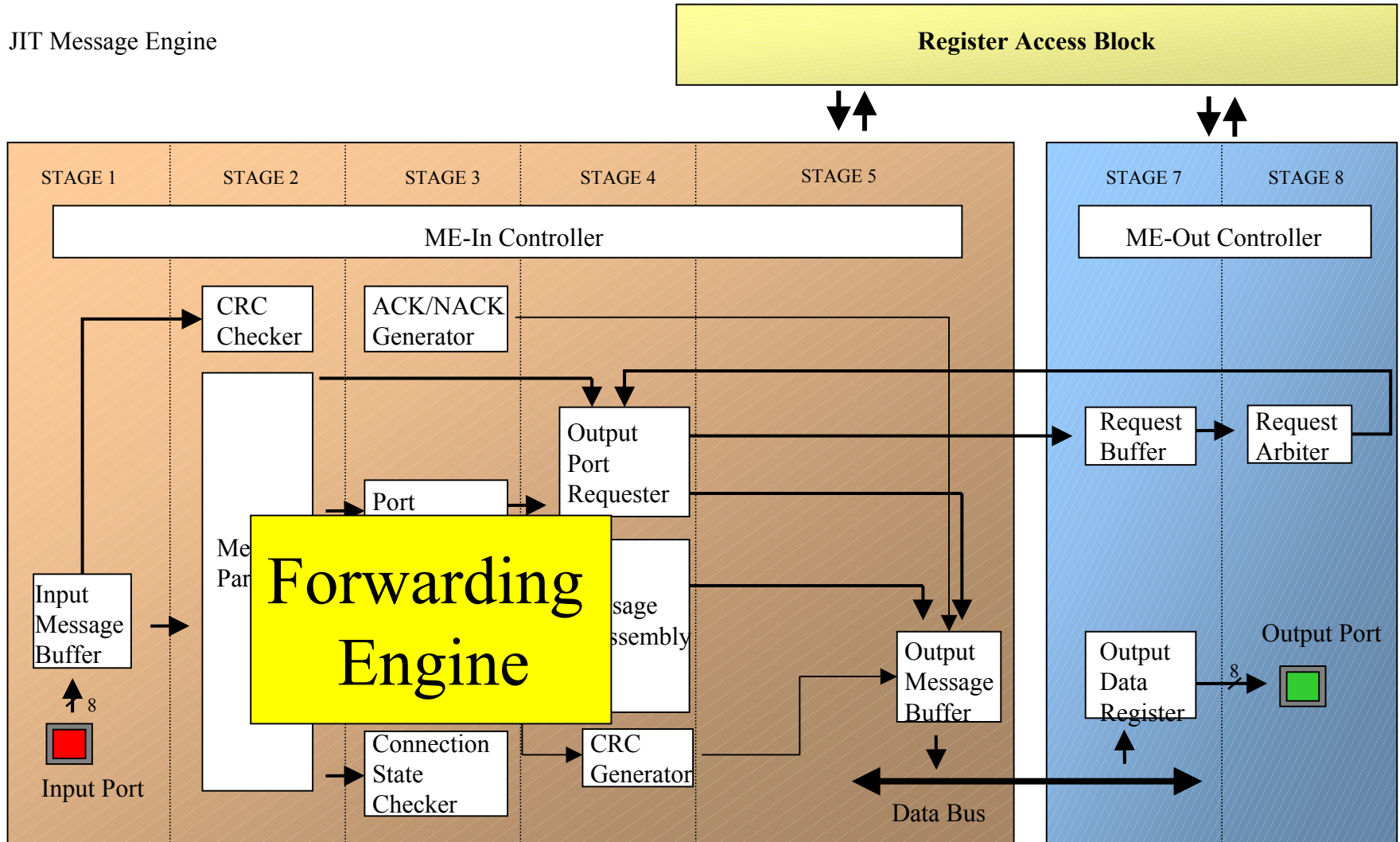
- ❖ Optical Burst Switching
  - Using Just In Time protocol
  - MCNC/NCSU project



- ❖ My group
  - Network Processor for OBS .....

# Network Processor Architecture

JIT Message Engine



# Motivation



The bottleneck of the forwarding engine is the route lookup

- ❖ Speed
  - Reduce the number of lookups esp. in main memory
    - Current routing tables have over 100,000 entries
      - ◆ Store in DRAMs
    - DRAM random accesses are slow ( $\approx 60-65\text{ns}$ )
  
- ❖ Scalability
  - Reduce the amount of memory required to store data
    - Tree Based Schemes most efficient
  
- ❖ Applies to all routers, not just OBS

## ...Motivation



- ❖ Content Addressable Memories (CAMs) are still not large enough for large routing tables
  - Current CAM sizes  $\approx$ 1MB
  - Multiple CAMs need to be used in the design
- ❖ Cost of CAMs is still high compared to DRAMs
  - 256 MB of DRAM costs  $\approx$ \$50
  - 1MB of T-CAM costs  $\approx$ \$300

# Problem Definition

- ❖ Routers store prefixes and not IP addresses
  - To determine the next hop, the *longest matching prefix* needs to be determined

- ❖ Example:

- Destination address = 10110101
- 10\*, 1011\*, 101101\* all match
- Longest prefix match is 101101\*
- Next hop address = 2

Prefix	Next Hop
10*	3
1011*	9
011*	1
010110*	5
001*	4
101101*	2
011010*	6
011100*	1
10111*	8
00101*	7

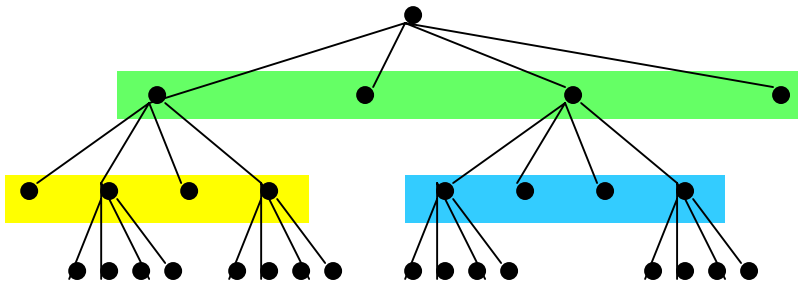
# *Proposed Scheme using Compaction*



- ❖ Since DRAM accesses are fairly expensive, limit the number of DRAM accesses
- ❖ The next hop information is needed only once the trie has been traversed fully
  - Separate the data (next hops) from the trie-path information
- ❖ An on-chip SRAM can store a representation of the trie to allow fast traversals
- ❖ The off-chip DRAM contains the next hop addresses which is required only once at the end

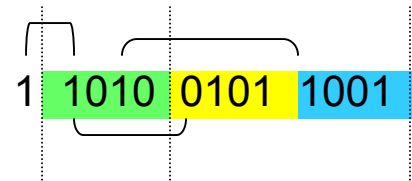
# Proposed Scheme Using Compaction

- ❖ Store path information in a smaller ( $\approx 250x$  than forwarding table), faster, wide on-chip SRAM
- ❖ Few SRAM and one DRAM lookups



Compact Trie node/path information:

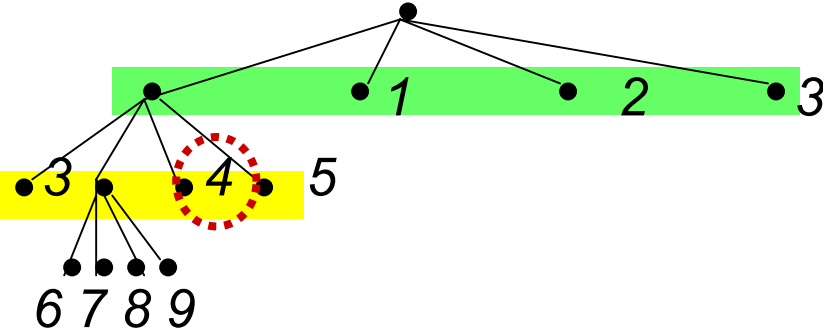
- 1  $\rightarrow$  node has child
- 0  $\rightarrow$  leaf node



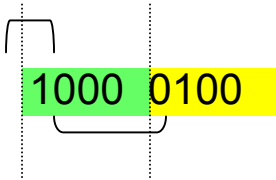
$\rightarrow$  A lookup can be done every 60-65ns  
(14-15 million lookups per second)

# Simplified Example

❖ Tree



❖ SRAM data

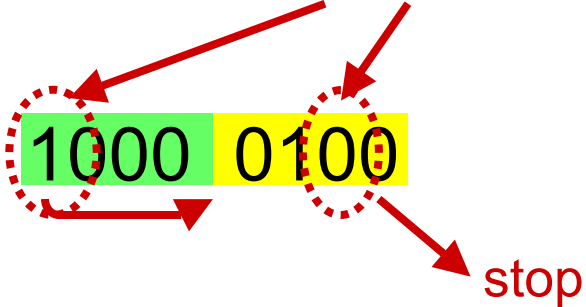


❖ DRAM data

0	*	1	2	3
1	3	*	4	5
2	6	7	8	9

❖ Example:

➤ Address = 00 10 01



- DRAM row = # 1s encountered (row 1)
- DRAM col = 2
- Output port = 4

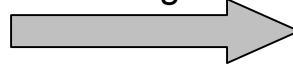
# ...Proposed Scheme Using Compaction



- ❖ On-chip SRAM and Off-chip DRAM
  - A wide on-chip SRAM
  - For 40,000 prefixes in the routing table, the required SRAM size is less than 50kB
  - 2 sets of these memories can be used to hide the update operations
    - Updates performed via embedded CPU
- ❖ Pipelined SRAM and DRAM operation
  - Only 1 DRAM lookup in all cases
  - One lookup can be done every 60-65ns
  - 14-15 million lookups per second
  - Multiple DRAMs can be used to increase the lookup rate

# Example for a 16-way Implementation

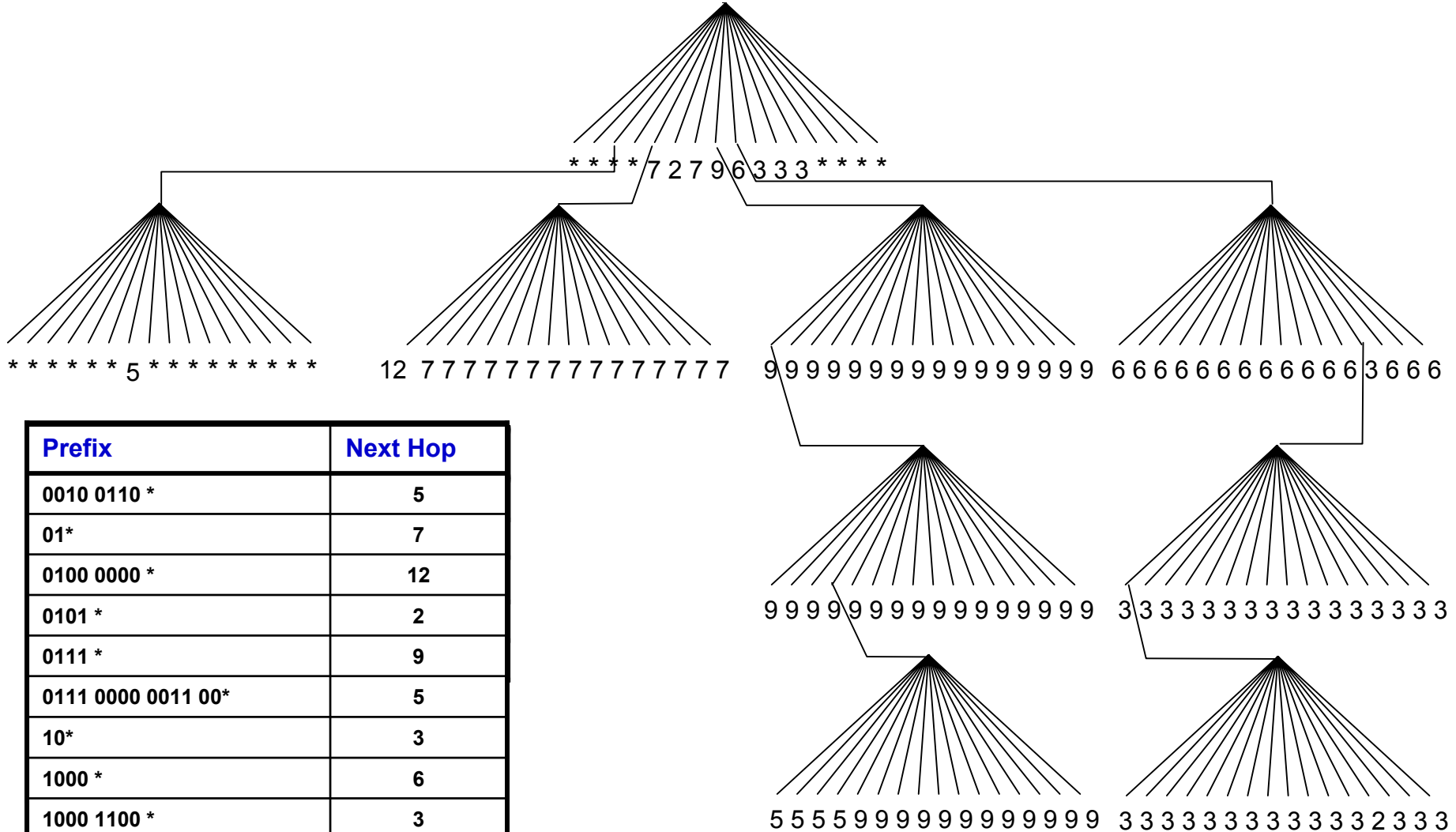
Prefix	Length	Next Hop
128.0.0.0	2	3
128.0.0.0	4	6
140.0.0.0	8	3
140.12.0.0	16	2
64.0.0.0	2	7
64.0.0.0	8	12
38.0.0.0	8	5
112.0.0.0	4	9
112.48.0.0	14	5
80.0.0.0	4	2

Sorting 

Prefix	Length	Next Hop
38.0.0.0	8	5
64.0.0.0	2	7
64.0.0.0	8	12
80.0.0.0	4	2
112.0.0.0	4	9
112.48.0.0	14	5
128.0.0.0	2	3
128.0.0.0	4	6
140.0.0.0	8	3
140.12.0.0	16	2

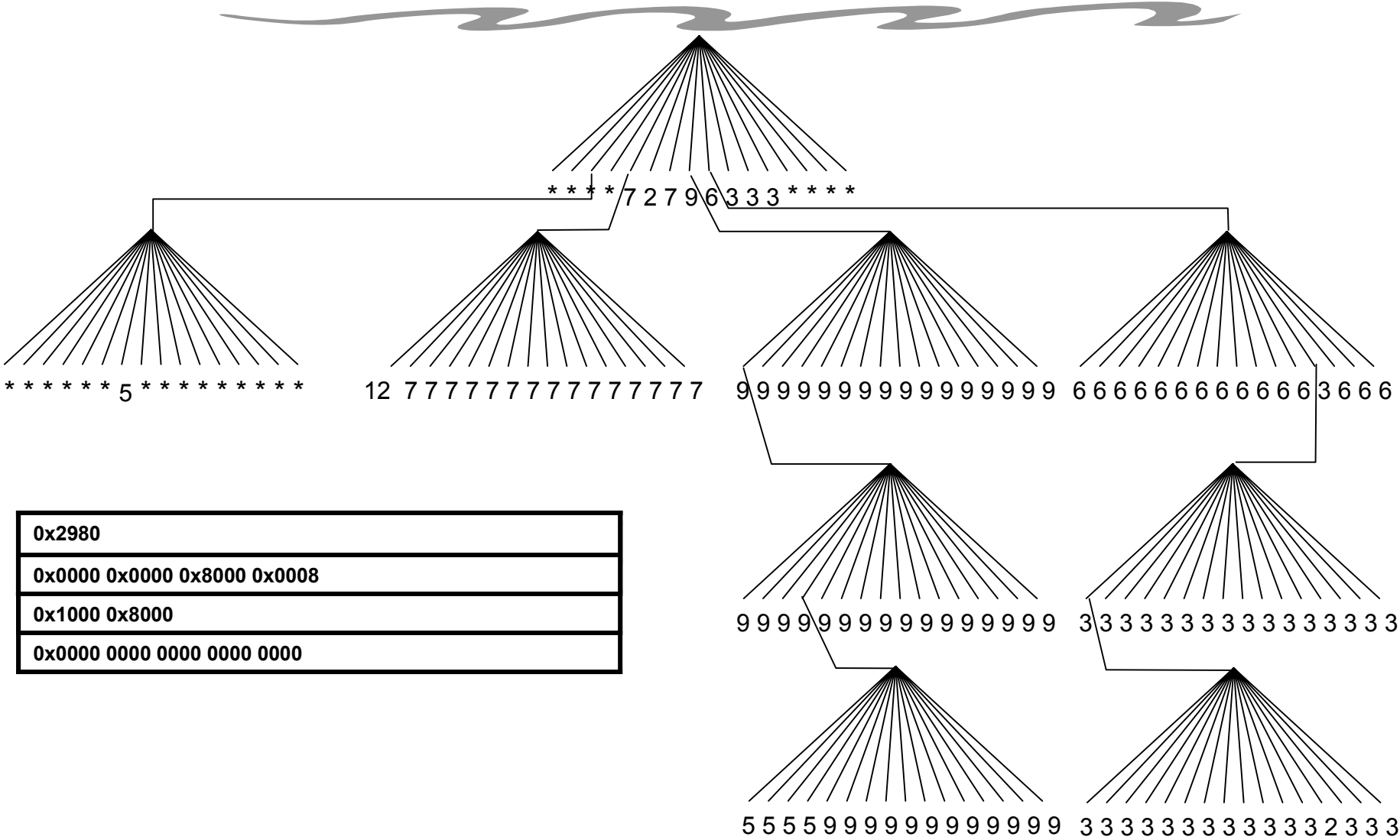
- ❖ The entries are first sorted in ascending order
- ❖ Each entry is read and expanded (for trie completion) and added to the trie structure
- ❖ Breadth first search of the trie structure is done to generate the SRAM data

# Example: Building the Trie



Prefix	Next Hop
0010 0110 *	5
01*	7
0100 0000 *	12
0101 *	2
0111 *	9
0111 0000 0011 00*	5
10*	3
1000 *	6
1000 1100 *	3
1000 1100 0000 1100 *	2

# Example: Building SRAM data



# SRAM and DRAM contents



Sum	Bit Pattern
0x0000	0x2980
0x0000	0x0000 0x0000 0x8000 0x0008
0x0000	0x1000 0x8000
0x0000	0x0000 0x0000

Level 0  
Level 1  
Level 2  
Level 3

```

* * * * 7 2 7 9 6 3 3 3 * * * *
* * * * 5 * * * * * * * * * *
12 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
6 6 6 6 6 6 6 6 6 6 6 6 3 6 6 6
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
5 5 5 5 9 9 9 9 9 9 9 9 9 9 9 9
3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3
    
```

## SRAM Contents

- ❖ The 1 corresponding to the root node is neglected while building the SRAM data
- ❖ Breadth-first search is performed to generate the bit pattern

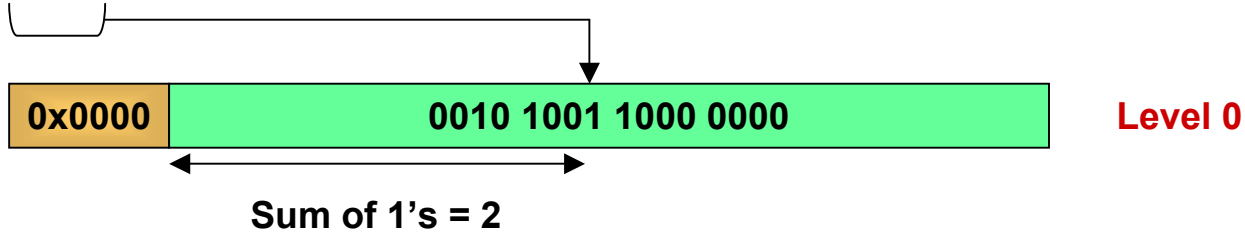
## DRAM Contents

# Example: Searching the Trie

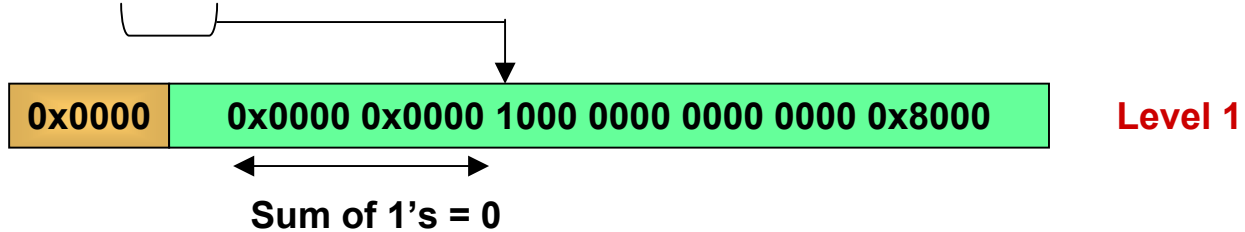
Input Address: 128.48.32.248

(0111 0000 0011 0000 0010 0000 1111 1000)

(0111 0000 0011 0000 0010 0000 1111 1000)

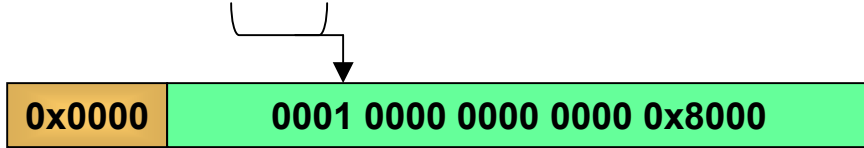


(0111 0000 0011 0000 0010 0000 1111 1000)



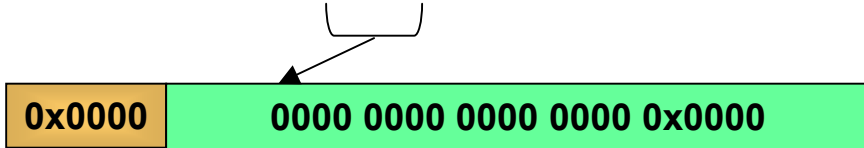
# Example: Searching the Trie

(0111 0000 0011 0000 0010 0000 1111 1000)



Level 2

(0111 0000 0011 0000 0010 0000 1111 1000)



Level 3

*	*	*	*	7	2	7	9	6	3	3	3	*	*	*	*
*	*	*	*	*	5	*	*	*	*	*	*	*	*	*	*
12	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
5	5	5	5	9	9	9	9	9	9	9	9	9	9	9	9
3	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3

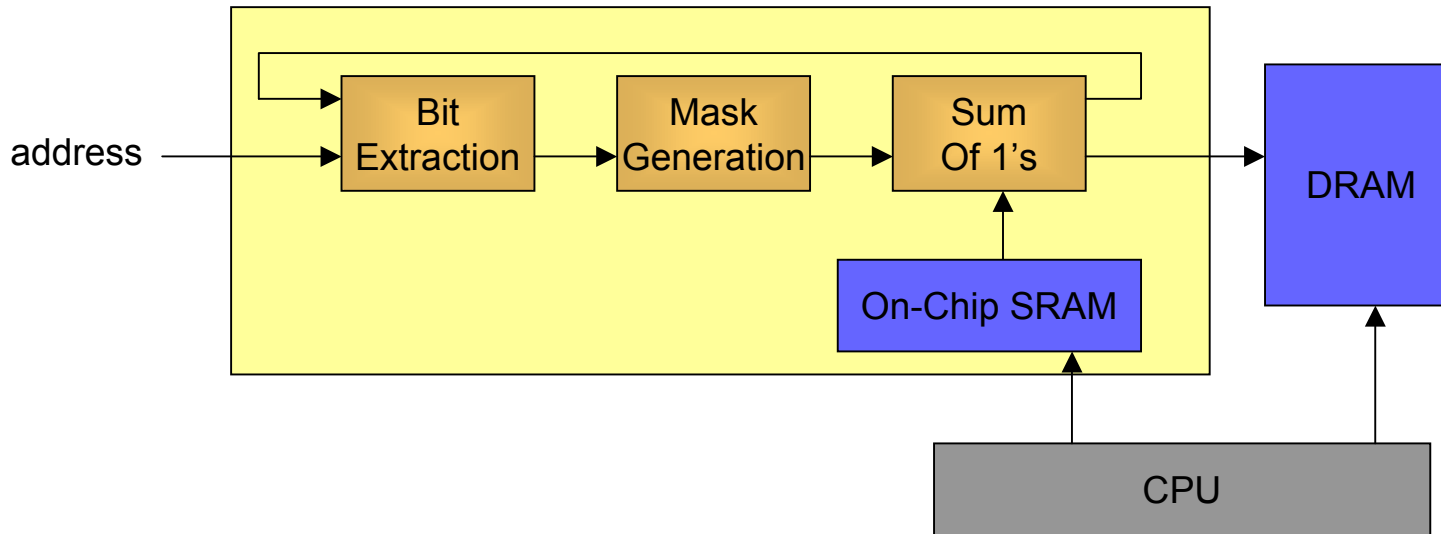
DRAM data

❖ Total number of 1's before and including the 1 in level 2 gives the DRAM row number where the next hop addresses are stored

- In our case row = 7

- Column = 0000

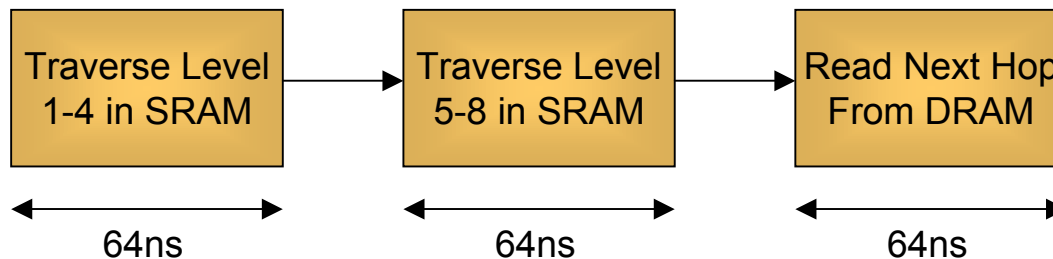
# Hardware Implementation



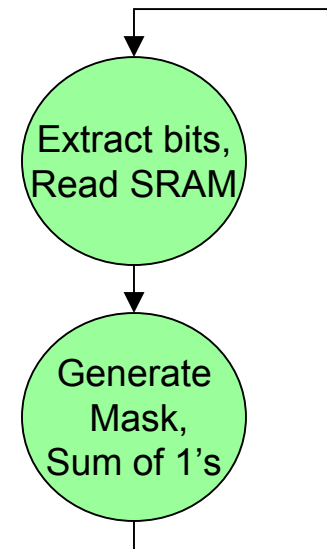
- ❖ To generate the mask, a decoder followed by a cross-bar like circuit is used
  - E.g. Address Field = 0010 Mask = 1110 0000 0000 0000
  - E.g. Address Field = 0011 Mask = 1111 0000 0000 0000
- ❖ Sum of 1's is done using a bank of adders

# ...Hardware Implementation

- ❖ Traversal in SRAM is further pipelined into two stages (for a 16-way implementation)



- ❖ Each traversal in SRAM goes through two stages
  - Extract bits and read the SRAM row
  - Generate Mask and Compute the sum of 1's
- ❖ Each stage takes under 8ns to give a total traversal time of 16ns



# Memory Requirement

Site	Entries	SRAM (KB)	DRAM (MB)	Trie Memory (MB)	Bytes per entry
MaeEast	23,113	24.4	11.43	24.28	1.08
MaeWest	35,752	34.75	16.32	34.68	1.99
PacBell	27,491	29.08	13.66	29.03	1.08
Paix	17,641	20.5	9.63	20.46	1.19
AADS	31,958	32.25	15.15	32.18	1.03

- ❖ For a 16-way implementation:
  - Amount of SRAM is about 1 byte per entry for all routing tables
  - About 500x reduction in memory from SRAM to DRAM
- ❖ Average Bytes/Entry =  $M/\ln(M)$ , where M is the degree of the trie

# Memory Requirement for different degrees

Site	Degree = 16		Degree = 8		Degree = 4		Degree = 2	
	SRAM (KB)	DRAM (MB)	SRAM (kB)	DRAM (MB)	SRAM (kB)	DRAM (MB)	SRAM kB)	DRAM (MB)
MaeEast	24.4	11.43	16	3.87	8.09	1	6.57	0.41
MaeWest	34.75	16.32	23.03	5.58	11.41	1.4	9.23	0.57
PacBell	29.08	13.66	19.24	4.66	9.76	1.2	7.99	0.5
Paix	20.5	9.63	13.09	3.17	6.86	0.84	5.6	0.35
AADS	32.25	15.15	21.33	5.17	10.67	1.31	8.67	0.54

- ❖ Efficiency of memory consumption increases as the degree of the trie decreases
  - Less wastage during trie completion
- ❖ Total latency would increase with smaller degree as more SRAM accesses required

# Conclusions



- ❖ Proposed a trie-based routing scheme using compaction
  - SRAM stores a representation of the trie and is used only for trie traversal
    - Small on-chip SRAM ( $\approx 35\text{KB}$  for  $>30,000$  entries)
  - Off chip DRAM stores next hop information
    - Only one DRAM access is required
  - The throughput is limited by the random access time of the DRAM
    - 14-15 million lookups per second
  - Multiple/Multibank DRAMs allow further improvement
- ❖ Implemented and tested in hardware
  - FPGA & ASIC
    - E.g. 0.25 sq.mm. of logic in 0.25  $\mu\text{m}$  process