A Fast Ternary CAM Design for IP Networking Applications

Bruce Gamache‡ (gamacheb@colorado.edu)
Zachary Pfeffer‡ (pfefferz@colorado.edu)
Sunil P Khatri‡* (spkhatri@colorado.edu)

‡Department of Electrical and Computer Engineering,
UCB 425, University of Colorado, Boulder CO 80309.
Telephone/fax: 303-735-1962
*Corresponding author

Abstract

In this paper we describe a VLSI implementation and complete circuit design of a fast Ternary CAM (TCAM). TCAMs are commonly used to perform routing lookups in the backbone of IP networks and small gateways. Our TCAM is designed to have a greater capacity and speed than any commercial offering at this time.

In contrast with existing TCAM approaches, our TCAM allows complete flexibility in the location where any new entry is inserted. This is achieved by a novel Longest Prefix Match (LPM) determination circuit, whose delay increases logarithmically with the number of bits to be looked up. We have implemented our TCAM with 512 bits of prefix entry with 512 bits of destination information, allowing it to implement large address lookups as well as quality of service mechanisms. This would make our TCAM design particularly suitable for IPv6 routing lookup applications.

The speed improvement of our TCAM over currently available TCAMs results from various carefully selected VLSI architectural and implementation choices. The TCAM size is 21 Mb and is broken up into a regular grid of 13×13 smaller TCAM blocks for improved speed characteristics. Routing lookup operations use a heavily pipelined approach for maximum throughput, while ensuring a lookup latency of 3 clock cycles. Individual match lines in these blocks are split into 4 sections to reduce RC delay in the lookup process. Our LPM determination circuit is implemented using an efficient Wired-NOR circuit for further reduced delay. Sense amplifiers are utilized in the LPM and SRAM sections of the TCAM and are located in the center of each TCAM sub-block in order to improve lookup speed.

We have implemented and validated our design using state-of-the-art circuit analysis and design tools. We have also generated mask layouts of the entire TCAM design using current layout tools. The complete TCAM circuit design is approximately 18mm on a side, with a total capacity of 21Mb. Our TCAM has an ability to perform routing lookups at a line rate of 76.8Gb/s which is twice as fast as the fastest commercially available TCAM today.

1. Introduction

Internet Routers have become a major determining factor in the performance of the Internet backbone. Routing networks have become increasingly fast with Gigabit ethernet (1.0Gb/s), OC-48 (2.4Gb/s), and OC-192 (9.6Gb/s) standards becoming prevalent in current designs. Optic communication standards with speeds up to 40Gb/s are around the corner. For IP networks operating at these speeds, fast routing lookup is a critical requirement.

When an IP packet is being routed from its source to its destination, we perform a *routing lookup* at every *router* that is encountered in the path of the packet. The routing lookup operation at each router involves performing a comparison of the incoming packet destination address against every entry in the *routing (forwarding) table* associated with the router. The forwarding table entry for the incoming packet records the *destination address* or *port* for the packet. The packet is forwarded to this port (also referred to as its *next hop address*) on the way to its final destination.

The process of IP routing is illustrated using the simplified IP routing network shown in Figure 1. Suppose A sends a data packet intended for B. Typically, rather than burden A with the apriori knowledge of the path to utilize in order to reach B, the network computes a route from A to B in a localized fashion. A simply sends the data to router R1, allowing R1 to decide how to route the data to B. R1 may now forward the data to R2, while R2 would forward the data to B. Note that in this figure, each router has several incident edges, which are referred to as *interfaces*. So the task of each router is to forward an incoming packet on interface *i* to an interface *j* such that the packet makes "progress" in reaching its destination. A router achieves this by means of a *routing table*, which records the outgoing interface for each incoming packet. Of course, since it would be prohibitively expensive to record the outgoing interface for each destination address, routers store entries in a compact manner. This is achieved by storing destination addresses as *subnet prefixes* (henceforth referred to as prefixes) along with a corresponding *subnet mask* (henceforth referred to as a mask). A mask records the bits of the address that need to be considered while performing the lookup.

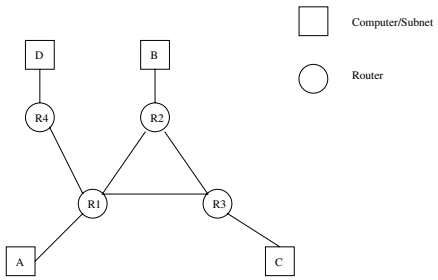


Figure 1: IP Routing Network Model

Table 1 is a small IP routing table which illustrates this idea. In this table, every entry contains a subnet prefix, a subnet mask, and the next hop address (i.e. the outgoing interface for entries that match the corresponding entry).

Number	Prefix	Mask	Next Hop
1	128.96.34.0	255.255.255.128	3
2	128.96.34.128	255.255.255.128	4
3	128.96.35.0	255.255.255.0	2

Table 1: Routing Table Example

Each time the router receives a data packet, it extracts its destination IP address and bitwise-ANDs the destination IP with the mask of each table entry. The resulting data is compared with the prefix of the corresponding table entry. If a match is detected, the data packet is forwarded to the interface (next hop) stored against the matching table entry. In practice, this is performed in parallel.

For example, if a data packet with destination IP address 128.96.35.22 is received by the router, it is ANDed with each mask. The resulting data for the third entry matches the prefix of the third entry, and therefore the data is forwarded to interface 2.

However in some cases, a destination IP address may match two or more entries in the routing table. In this case, the router forwards the packet to the interface which has the longest mask. This is referred to as *Longest Prefix Matching* (LPM). This is illustrated in Table 2.

For the routing table shown in Table 2, the destination IP of 128.96.34.55 matches both entry 1 and entry 2. In this case, the router compares the mask lengths of each entry, finds the longest mask, and forwards the packet to the corresponding interface. Longer masks indicate that more specific forward-

Entries	Subnet Prefix	Subnet Mask	Next Hop
1	128.96.0.0	255.255.0.0	3
2	128.96.34.0	255.255.255.0	4

Table 2: Example of Routing Table with Overlap

ing information is available for matching addresses, and therefore the router forwards the packet to the next hop associated with the matching entry with the longest mask. In this example, entry 1 has a mask of length 16 bits, while entry 2 has a mask of length 24 bits. So the packet will be sent to interface 4.

IP addresses were originally partitioned using a Class-based scheme. Class A, B and C addresses utilized 8, 16 and 24 bits respectively. Because the number of such IP addresses was rapidly being exhausted, there was a compelling need to utilize IP addresses more efficiently. This motivated the introduction of Class-less Inter-Domain Routing (CIDR) [1] in 1993. In this method, "classes" were permitted to have an arbitrary number of network bits, allowing a more flexible IP allocation. The downside of this choice was that it resulted in an increase in the size of IP routing tables, due to the fine granularity of addressing that results from the choice of the CIDR addressing methodology. In the CIDR scheme, routing table entries could have arbitrary length prefixes, allowing for a more efficient method of assigning addresses and for route aggregation. The CIDR scheme does not allow for two entries in the routing table to have the same prefix.

The routing lookup operation is preferably done in hardware instead of software due to the need for extremely fast lookup speeds. One hardware implementation of this operation is a Ternary Content Addressable Memory (TCAM) [2, 3, 4]. TCAMs are designed to store an entire routing table, and allow the simultaneous comparison for *all* routing entries against the destination address of the packet being routed. A good overview of existing TCAM approaches can be found in [2].

A TCAM is similar to a cache (also referred to as a CAM [5]) with the additional ability to disregard a subset of address bits while performing the lookup. The address bits that are considered during a routing lookup operation correspond to the non-zero mask bits of the routing table entry. As illustrated in the above example, the TCAM must be able to perform a Longest Prefix Match (LPM) operation as well, in order to resolve situations where multiple entries match a certain address.

Typical TCAM implementations store routing entries in clusters [6, 4], where each cluster contains routing entries of a particular mask length. This allows for fast lookups, but results in a worst-case insertion penalty of $n/2$ clock cycles (where n is the size of the network address). For IPv6, a clustered TCAM scheme would require 64 clock cycles for insertion in the worst case. Such techniques utilize a *priority encoder* [7] circuit to perform the LPM. Such a circuit is quite complex [4], often limiting overall performance. Alternate implementations [8] allow routing table entries to be stored in arbitrary locations in the TCAM, but require two cycles to perform a lookup.

In our implementation, we allow routing table entries to be stored in any order, thus eliminating the large worst-case insertion cost of typical TCAM implementations, as described in [6]. In addition, we utilize an efficient Wired-NOR based LPM circuit, whose delay scales logarithmically with n, thus improving over the linear complexity (in the size of the TCAM) of priority encoder based circuits. As a result, our method simultaneously achieves both fast updates to the routing table by allowing arbitrary insertion of the entries and high speed search throughput as well.

The architecture of our TCAM is pipelined, and provides 1 lookup per clock cycle with a latency of 3 clock cycles. This was achieved by careful VLSI architecture and implementation to ensure a fast and flexible design.

The remainder of this paper is organized as follows: Section 2 discusses some previous work in this area. In Section 3 we describe the implementation and motivations for the various modules of the TCAM design. In Section 4 we present experimental results comparing our TCAM design with products on the market today. Conclusions are discussed in Section 5.

2. Previous Work

Each entry in a TCAM has a prefix and a mask, along with the associated next hop information. Prefixes and masks can be of any length (not to exceed the length of the network address) and together specify a portion of the network address. When a mask bit is a '1', the corresponding prefix bit ('0' or '1') is considered in determining a match. Conversely, when a mask bit is a '0', the corresponding prefix bit is disregarded. In this manner, a TCAM cell (which consists of a prefix and mask bit pair) can logically take on one of three states: 0, 1, or X (or don't-care). Note that in a typical implementation, all 'don't-cares' (X's) are limited to successive right-hand bits for any entry.

The LPM computation is typically done in hardware, either using dedicated hardware [8], or by arranging the routing table entries in a specific order as described in [6]. Typical TCAM-based hardware routing approaches store the entries in groups increasing order of their mask lengths [6]. This is illustrated in Figure 2 a). In case of multiple matches, the LPM computation simply requires that we find the match from the group with the largest prefix length. The major drawback in these approaches is that insertion of a new entry may require $O(n)$ entries to be re-arranged (to create the space required to add the new entry and ensure that the groups are maintained in increasing order of prefix lengths), where n is the length of the network address. For IPv6, this could result in worst case insertion delays of 128 clock cycles, which is undesirable in large backbone routers.

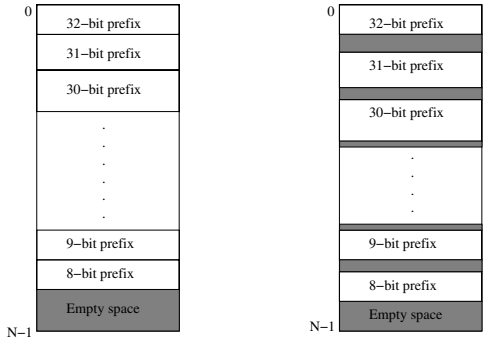


Figure 2: TCAM Memory Pool Organization

An alternate LPM implementation keeps some entries unused for each group, for possible use at a later time [6]. This is illustrated in Figure 2 b). When a new entry is inserted, it is placed in the free space of the group corresponding to its prefix length. The drawback of this scheme is that portions of the TCAM memory remain unused, and the worst case insertion still requires n clock cycles.

Often, the free space pool shown at the bottom of Figures 2 a) and 2 b) is located in the center of the TCAM, resulting in a halving of the worst-case insertion delay. Even with such an implementation, the worst-case insertion cost of such schemes is linear in n [6].

In the scheme of [6], memory management was performed external to the TCAM. Although this resulted in a reduction of the worst-case insertion delay, the proposed memory management was performed in software, reducing the effectiveness of the technique.

Some commercial TCAM solutions allow the insertion of new routing entries in arbitrary locations within the TCAM. In such approaches, the LPM operation requires the use of a priority encoder [7, 9]. The drawback of this technique is that the priority encoder circuit required for the LPM task has an implementation whose complexity grows linearly with n.

An alternative approach [8] also allows entries to be stored in arbitrary locations. In this scheme, routing lookup is a non-pipelined, two stage operation. The TCAM performs the lookup in the first phase and bitwise ORs the matching entries' masks to produce the longest mask. This 'longest mask' is fed back to the TCAM and further constrains the original matching entries to produce the entry with the longest prefix. The main drawback of this approach is that in lowering the cost of insertion, the cost of each lookup is doubled.

In order to appeal to network system architects, current commercial offerings are focusing on flexible lookup table configuration, increasing table capacity and lookup frequency. This push is intended to support CIDR with a longer prefix length for Quality of Service (QoS) extensions as well as more flexible policing schemes. Our implementation enables such flexibility by allowing 512 bits of ternary lookup and 512 bits of forwarding data per routing table entry.

3. Our Approach

3.1 Overview

Our design stores routing entries in an arbitrary locations within the TCAM. Its salient features are:

- Aggressive VLSI architecture and implementation techniques to minimize TCAM lookup delays.
- A unique Wired-NOR based LPM circuit. At the time of inserting a routing entry, the number of '1's in its mask are added together, and the result is stored in *LPM cells* (9 for each routing entry). While performing a routing table lookup, the Wired-NOR circuit finds the matching entry with the largest prefix by using the information stored in the LPM cells of all the matching entries.

As a consequence of the above design decisions, our TCAM performs routing lookups at twice the rate of the fastest commercially available offering [10]. The size of our TCAM is also slightly larger than that of [10]. Further, the delay of our LPM circuit grows logarithmically with n (where n is the length of the network address), a significant improvement over existing techniques where this delay grows linearly with n.

3.2 TCAM Architecture

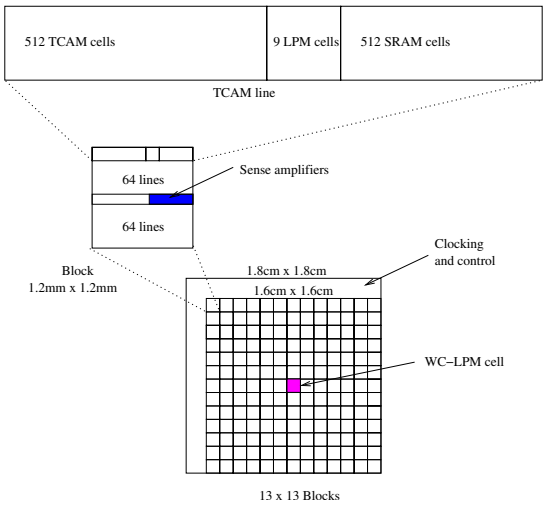


Figure 3: Floorplan and cell arrangement of our TCAM

The chip level floorplan of the memory portion of our TCAM design is shown in Figure 3. The entire TCAM design is made up of (13×13) - 1 **blocks**, each containing 128 routing entries[1]

Each block performs a parallel search on the applied destination address, independent of all other blocks. Once the match with the longest prefix is determined *for each individual block*, we need to find the match with the longest prefix *among all blocks*. Therefore the LPM cell contents of this entry is sent to a second LPM circuit, which is located in the middle of the chip. We call this LPM circuit the *winner's circle LPM* (WC-LPM) circuit. The task of the WC-LPM circuit is to find the matching entry with the longest prefix *among all 168 blocks*. The WC-LPM circuit is similar to the block-level LPM circuit, except that it operates on 168 entries instead of 128. Once the WC-LPM circuit has computed the entry with the longest prefix among all blocks, it transmits this information to the appropriate block, and the routing data (next hop address) is read out from the appropriate entry. The entire operation is performed in a pipelined fashion as illustrated in Figure 4. As a result, the delay incurred during WC-LPM computation is *"hidden"* from the user.

3.3 TCAM Implementation

Our TCAM design consists of 168 TCAM blocks, implemented in a 13×13 grid as described in Figure 3. Each block consists of 128 entries, implemented

[1]With 128 entries per block, the layout of each block is approximately square, resulting in the lowest circuit delay.

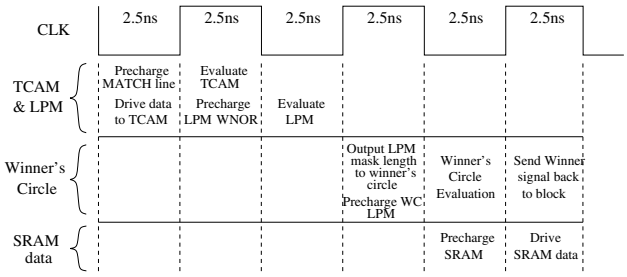


Figure 4: Pipelined Implementation of Lookup Functionality

in separate rows. A block is a square of size 1.2mm×1.2mm. Each row of a block contains 512 TCAM cells, 9 LPM cells, and 512 Static Random Access Memory (SRAM) data cells that are laid out side by side, as illustrated in Figure 3. The TCAM, LPM and SRAM cells are pitch-matched (i.e. they have the same height). The LPM and SRAM portions of a block have sense amplifiers placed between rows 64 and 65, in order to speed up read operations from the LPM or SRAM sections.

Breaking the TCAM into multiple blocks reduces the overall delay of the TCAM by speeding up match detection. The specific choice of 128 lines per TCAM block was dictated by the goal of having a 1:1 aspect ratio for each block and the overall TCAM die.

Each row consists of 512 **TCAM cells**, 9 **LPM cells** and 512 **SRAM cells**, allowing our design to support IPv6 as well as QoS extensions. By disabling all but 32 TCAM cells, backward compatibility with IPv4 is also supported by our TCAM.

Each TCAM cell consists of two memory cells which store the prefix and the mask values.

Each row contains 9 LPM cells, which store the sum of the number of mask bits with non-zero values. Additionally, the LPM cells contain the circuitry required to determine the longest prefix among the rows of each block.

SRAM cells store the next hop address for their row entry. Additional SRAM cells per row allow the user to store additional information for the corresponding row entry. SRAM cells are implemented in clusters of 4 bits, resulting in the final layout being pitch-matched with the TCAM and LPM designs. Layouts of these cells are shown in Figure 8.

We next describe the TCAM, LPM and SRAM cells in further detail.

TCAM Cell Design:

Each of the 512 TCAM cells consist of two memory cells which store the prefix and the mask bits. The applied input is compared to the entry bit stored in the TCAM cell. If the mask bit has a stored value of '1', the TCAM cell will perform the comparison between the applied input bit (for that particular cell) and the stored value.

The TCAM cell is made up of two SRAM memory cells to store the prefix and mask values, as shown in Figure 6. The top portion of the cell stores the prefix entry while the bottom portion stores mask data. The prefix (mask) is written by asserting $W1$ ($W0$) while placing the data to be stored on B and $BBar$.

A match is performed by first pre-charging the $MATCH$ line using the PCH signal. Next, the data being looked up is asserted on $BBar$ and B. If the data mis-matches with the contents of the cell *and* the stored mask value is '1', then $MATCH$ is pulled low. In all other cases, $MATCH$ stays pre-charged. The $MATCH$ line is shared among all 512 TCAM cells in a row.

In other words, whenever the mask value stored in a cell is '0', the cell no longer affects the final value of the $MATCH$ signal, as we require.

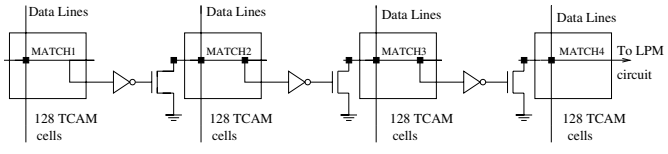


Figure 5: TCAM row broken into four sections

In any row of a block, there are 512 TCAM cells laid out side by side. These cells share the $MATCH$, $W0$ and $W1$ signals. However, this results in a fairly long $MATCH$ line, resulting in a large match computation delay for any row. The parasitic resistance and capacitance of the $MATCH$ line

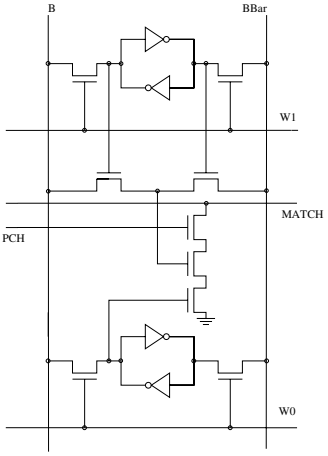


Figure 6: TCAM cell circuit design

are proportional to its length, and therefore its RC time constant increases quadratically with length. In order to minimize this delay the *MATCH* line is split into p sections.

We experimentally determined that the the optimal value of p, which results in the minimum match computation delay, is 4. In other words, we split the *MATCH* line into 4 sections, as shown in Figure 5. Section q of the *MATCH* line is connected to section $q+1$ with an inverter/pull-down structure shown in Figure 5. If section q ($1 < q < 3$) was pulled low, then sections $q+i$ ($1 < i < 3$) are automatically pulled low by the inverter/pull-down structures between adjacent row sections.

The timing of a write cycle is not critical, and as a result, the $W0$ and $W1$ lines are not split in this fashion.

LPM Cell Design:

Each row in any block includes 9 LPM cells associated with it, as shown in Figure 3. These LPM cells each contain a SRAM cell. At the time a row entry is written into the TCAM, the number of '1' bits in the mask are added and stored in the 9 LPM cells for that row entry. In other words, the 9 LPM cells of a row store the length of the mask for that row.

During the routing lookup process, the 9 LPM cells perform the task of determining the matching entry in the block with the longest prefix, using our Wired-NOR circuit structure.

The circuit of our LPM cell is shown in Figure 7. The SRAM cell stores a particular bit of the length of the row mask. The 9 LPM cells are laid out side by side. The left-most LPM cell is the 9^{th} and most significant bit (MSB) of the length of the mask. We refer to an arbitrary bit of the mask length as the i^{th} bit.

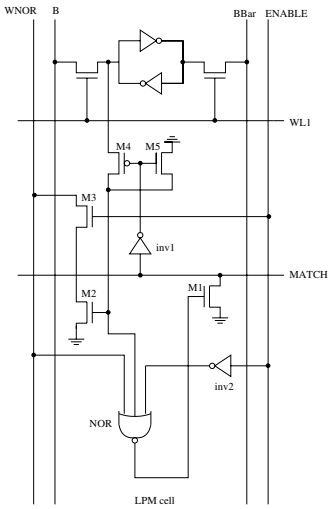


Figure 7: LPM cell circuit design

The LPM cell is written by asserting the data to be written on the B and *BBar* lines along with the $WL1$ line.

The *WNOR* and *ENABLE* lines run vertically in our design, and are shared by 128 LPM cells, one per row. For example, the *WNOR* and *ENABLE* lines corresponding to the MSB are shared by the left-most LPM cell of each of the 128 rows in a block. There are 9 pairs of *WNOR* and *ENABLE* lines per block, which we refer to $WNOR_i$ and $ENABLE_i$.

The LPM operation in our TCAM is performed in 9 phases, since the TCAM consists of 512 cells. In the first phase, we pulse $ENABLE_9$, which allows the MSB LPM computation to proceed. In the k^{th} phase, $ENABLE_{9-k}$ is pulsed. Before each phase, the corresponding *WNOR* line is pre-charged.

For any bit i, when $ENABLE_i$ is pulsed, the $WNOR_i$ line performs the wired-NOR of the values in the i^{th} LPM cell for each of the 128 rows in the block *which match the applied data*. If one of the rows does not match the applied data ($MATCH = 0$), then M5 is turned on, and the gate of M2 is low. This ensures that this row never pulls down any $WNOR_i$ line. On the other hand, if one of the rows matches the applied data ($MATCH = 1$), then M4 turns on, and hence $WNOR_i$ is pulled low if the i^{th} LPM cell has a '1' value stored in it. If the i^{th} LPM cell has a '0' value stored in it, it does not pull $WNOR_i$ low.

Consider rows j and k for which the *MATCH* line is high. In other words, these rows match the applied data. In case there exists a matching entry k with a longer mask than the mask of row j, the row j should be removed from consideration. We next describe how this is done.

Consider the i^{th} phase. If there is a longer mask than the mask of row j, there must be some row k with a '1' value stored in its i^{th} LPM cell, while the i^{th} LPM bit of row j stores a '0' value. In this case, the $WNOR_i$ line will be pulled low in the i^{th} phase, due to the influence of the k^{th} row. Now consider the i^{th} LPM cell of row j. The $WNOR_i$ line is low, and the stored value in the i^{th} LPM cell of row j is '0'. Since the j^{th} *MATCH* line is high, M4 is turned on and the gate of transistor M2 is pulled low. Also, $ENABLE_i$ is high in the i^{th} phase. As a result, all the inputs to the NOR gate in Figure 7 are low. Therefore, the gate of M1 is pulled high and the j^{th} *MATCH* line is pulled low, removing the j^{th} row from further consideration. On the other hand, if the i^{th} LPM cell of row j had a '1' valued stored in it, the j^{th} *MATCH* line does not pull low, allowing row j to continue participating in the LPM computation.

In this way, a matching entry is removed from consideration whenever there exists a matching entry with a longer mask. After all 9 phases have completed, a single row (with the largest mask length) remains with its *MATCH* signal held high.

The entire operation incurs a delay which grows logarithmically in n, the length of the network address.

The worst-case delay for the LPM computation occurs when the LPM cell in the k^{th} row pulls down the *WNOR* line *and* a "0" is stored in the LPM memory of the j^{th} row (resulting in the *MATCH* line of row j being pulled low). In this case, the total delay is the time it takes to pull the *WNOR* line down *plus* the time for an LPM cell to pull down its *MATCH* line. This is the minimum amount of time that we would have to wait until we could assert the *ENABLE* line for the next phase. In order to minimize this delay, we tried to break the *WNOR* line into p sections just as we did for the row of 512 TCAM cells. For all values of p, this resulted in an increased LPM delay. This is because the delay overhead associated with breaking up the WNOR line into p sections was greater than the total RC delay for a single *WNOR* line.

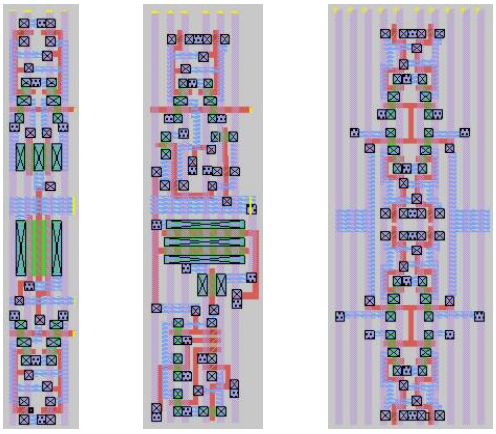
Our design has a maximum mask length of 512, and hence requires 9 LPM cells. The value stored in the LPM cells is computed using a ripple-carry-adder and written into the LPM memory cells during the cycle in which the TCAM cell mask is written.

We illustrate our LPM computation scheme using a small example. Consider the routing entries in Table 3. Suppose we looked up the value '101' in this table. All rows would match the query. The LPM cells would then begin to perform a Longest Prefix Match computation.

TCAM prefix entry	LPM entry
1XX	01
10X	10
XXX	00
101	11

Table 3: LPM Mask Entry

After the first phase, $WNOR_2$ will be pulled low due to the LPM cell en-



(a) TCAM (b) LPM (c) SRAM

Figure 8: TCAM, LPM & SRAM cell layouts

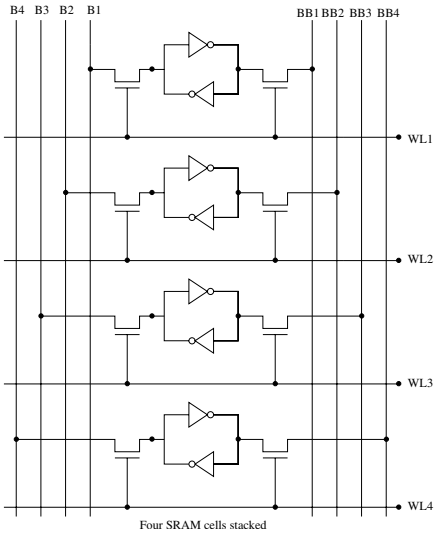


Figure 9: Four SRAM cells stacked circuit design

tries in the second and fourth rows which contain '1' values. Since the most significant LPM cell entry of the first and third rows is 0, the *MATCH* lines of the first and third rows is pulled low, removing the first and third rows from further consideration. As a result, the remaining rows with a high *MATCH* value are the second and fourth rows.

After the second phase, $WNOR_1$ is pulled low due to the '1' LPM cell entry of the fourth row. Since the least significant LPM cell entry of the second row is a 0, the *MATCH* line of the second row is now pulled low, removing it from further consideration. As a consequence, the only remaining row with a high *MATCH* signal is the fourth row, which has the longest mask.

SRAM Cell Design:

The SRAM cells of each row contain the next-hop/address information for that row entry. We use a 6-transistor SRAM cell in our design due to its simplicity and robustness. These cells are also laid out side by side like the TCAM cells and the LPM cells, as shown in Figure 3. Since a single SRAM cell is significantly simpler than the LPM and TCAM cells, and since we desire to ensure that all cells have the same height, we implement the SRAM cells in clusters of 4. The circuit showing a cluster of 4 SRAM cells is shown in Figure 9.

Sense amplifiers are inserted in the SRAM section, between rows 64 and 65 of each block. These sense amplifiers are used to sense the value driven out of the SRAM during a read operation, and reduce the delay of the read operation.

The layouts of the SRAM, TCAM and LPM cells are shown in Figure 8. Note that the heights of all cells are identical, allowing an efficient row layout.

4. Experimental Results

Our TCAM was designed and simulated using state-of-the-art IC design tools in a 0.1um CMOS process technology. Predictive SPICE model cards were obtained from [11]. After designing our TCAM at the architectural level, we performed functional verification using the Verilog hardware description language.

We next sized our circuits and performed layout generation of all the constituent blocks using MAGIC [12]. From the layouts we were able to do 3-D parasitic extraction on the circuits using Space-3D [13]. The resulting parasitic resistances and capacitances were inserted into our simulation, which was performed in SPECTRE [14].

Based on our timing simulations, our TCAM has a clock period of 5ns. In other words, it operates with a clock frequency of 200MHz, with a throughput of one lookup per clock cycle. This is 2× faster than the fastest TCAM commercially available at this time [10]. The lookup latency is 3 cycles, as Figure 4 indicates.

To compute the highest serial line rate supported by our TCAM, we used a procedure similar to [8]. Assuming a 48 byte packet, our TCAM supports link speeds of 76.8 Gb/sec which is an undefined standard as of yet. This allows our TCAM to do two OC-768 lookups in a clock cycle.

Our methodology in determining the speed of our TCAM is described next. Based on Figure 4, we found the delay T_L (T_H) of the slowest task among all tasks computed in the low (high) phase of the *CLK* signal. The duration of the low and high phases of the *CLK* are greater than T_L and T_H respectively.

The TCAM design requires pre-charging of various sections of the design on different phases of the clock. As seen in Figure 4 there are a total of four different pre-charge operations performed per pipelined lookup. We simulated the delays for each such pre-charge operation. It turned out that these delays were inconsequential in determining the speed of our TCAM.

One time-critical portion of our TCAM design is the lookup/evaluation for the 512 TCAM cell row. This operation is labeled *Evaluate TCAM* in Figure 4. Initial simulation results revealed long delays, since the *MATCH* line for any row is long and highly capacitive. The worst-case delay for this operation occurs when a single TCAM cell does not match the data being looked up. In this case, the single TCAM cell pulls down the *MATCH* line. Initial simulations indicated that this delay was over 3ns. In order to reduce this delay the MATCH line was broken into 2, 4, and 8 sections as illustrated in Figure 5. The smallest *MATCH* delay was obtained when the MATCH line was broken into 4 sections. This delay was 1.6ns, as shown in Figure 10. In this figure, the *MATCH*1 signal of the first match section in Figure 5 is pulled low by a single TCAM cell, which results in the remaining sections being pulled low as well.

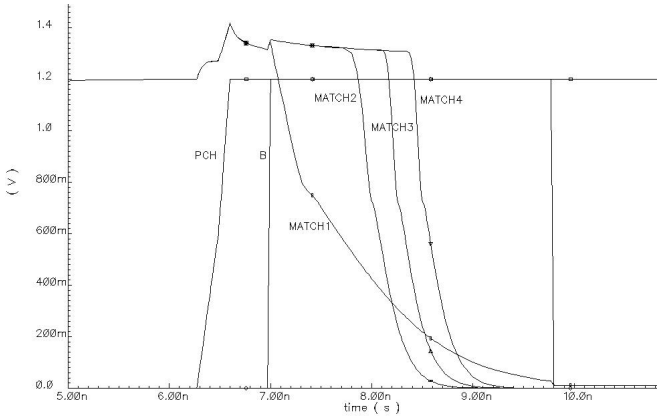


Figure 10: MATCH line broken into four sections getting pulled low

The other time-critical section of our design is the LPM evaluation stage (labeled *Evaluate LPM* in Figure 4). In this section of the design, we perform the LPM computation for a block in 9 phases as described in Section 3.3. As discussed earlier, the worst-case delay for any LPM phase i corresponds to the situation in which a single LPM cell pulls down $WNOR_i$, after which the

MATCH line of another row is pulled down. We simulated this condition, and by carefully sizing the LPM cell devices, we obtained a delay less than 225ps. In other words, this is the minimum duration we must wait before asserting the *ENABLE* line of the next LPM phase. Thus the total delay for LPM computation is 9×225ps = 2.025ns. Figure 11 illustrates the LPM computation waveforms. Note the assertion of the 9 *ENABLE* lines ($E1$ through $E9$) for the 9 LPM phases. In the figure, a particular *MATCH* signal drops out of LPM contention in the third phase.

We also tried to improve the delay of the *WNOR* line by splitting the *WNOR* line into multiple sections (just as we did for the TCAM row *MATCH* line). Unfortunately, this proved to be ineffective due to the low wiring parasitics of the *WNOR* line.

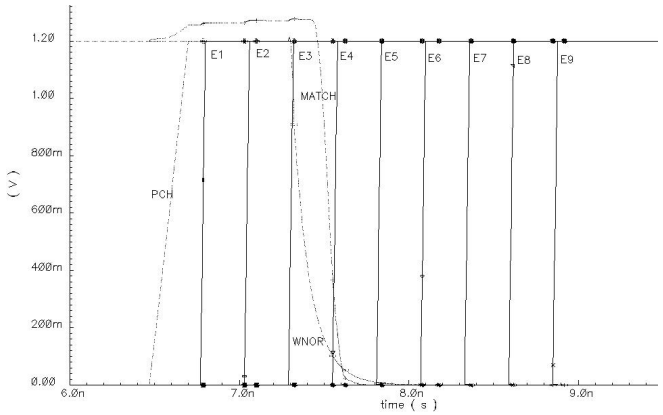


Figure 11: Timing for 9 LPM cells' ENABLE lines

After each block has completed its LPM evaluation there will be one "winner" for each block. Each block then sends the winner's LPM cell contents to the WC-LPM circuit, to be compared against winners from remaining blocks. The delay incurred in driving the LPM data of the winner of each block to the WC-LPM circuit, and writing this data into a pre-defined row of the WC-LPM circuit was accounted for. The actual WC-LPM circuit is structurally and functionally similar to a block LPM circuit, with 168 rows instead of 128 rows. This resulted in a slightly larger delay in computing the WC-LPM result. The delay for one phase of WC-LPM computation was 235ps, hence the total delay for WC-LPM evaluation would be 9×235ps = 2.115ns.

Our simulations indicate that the LPM output delay was 950ps, the delay in sending the LPM value to the WC-LPM cell was 600ps. Writing the LPM value to the WC-LPM block required 300ps. So the total delay for these operations (which occur in the high phase of *CLK*) is 1.85ns.

The *Drive SRAM data* stage seen in Figure 4, was simulated to have a delay of 250ps. The delay incurred in the *Send winner signal back to block* operation is 600ps, for a total delay of 850ps.

From the above simulations, we can determine T_H and T_L as follows

T_H = max(1.6ns (TCAM eval), 200ps (LPM pchg), 1.85ns (WC-LPM operations), 200ps (WC-LPM pchg), 850ps (SRAM data drive)) = 1.85ns

T_L = max(600ps (MATCH pchg), 300ps (Data drive), 2.025ns (LPM eval), 2.2ns (WC-LPM eval), 250ps (SRAM pchg)) = 2.2ns

This indicates that our TCAM can operate with a clock of period 4.1ns. To account for process, voltage and temperature variations, we specify the operation of our design with a 5ns clock.

We simulated the power consumption of our design using SPECTRE [14] simulations. We computed the maximum power consumption in the high phase (P_H) and low phase (P_L) of the system clock. The chip power consumption is the average of these two values. From these power simulations, we determine P_H and P_L as follows:

P_H = Σ (8.28W (TCAM eval), 0.41W (LPM pchg), 0.93W (WC-LPM operations), 0.004W (WC-LPM pchg), 0.31W (SRAM data drive)) = 9.93W

P_L = Σ (9.56W (MATCH pchg), 5.35W (Data drive), 0.99W (LPM eval), 0.004W (WC-LPM eval), 0.13W (SRAM pchg)) = 16.04W

Our total power consumption is therefore about 13W. This compares favorably with [15], where the authors report a power consumption of 4W (with *all* power management features enabled) for a TCAM that is 57% smaller than our design. The same authors state that traditional TCAM implementations consume 3 to 4 times more power than their design.

Since our technique allows new entries to be placed in any location in the TCAM, it would be easy to reduce our power consumption further, by utilizing the fewest number of blocks that are sufficient to store all the routing information. All other blocks would be disabled, by statically leaving their clocks in the pre-charge phase, thereby potentially reducing the power consumption dramatically. For example, if we had a total of 224 routing entries, we would need 2 blocks, and therefore the remaining 167 blocks can be disabled in the manner described, reducing power consumption significantly. Our current design does not implement any power management features.

Entry insertion and deletion operations are performed in a similar manner. The pipelining diagram for these operations are not shown for brevity. Insertion and deletion operations can be completed with a clock period of 5ns. A write operation is always preceded by a pipeline flush, followed by a write data cycle, which requires a single clock cycle. A deletion operation requires a lookup to be performed, followed by a write operation.

5. Conclusions

In this paper, we have presented a VLSI implementation of a 21 Mb TCAM circuit. The fastest commercially available TCAM product [10] operates at serial line rates of 38.4 Gb/s, with a 19 Mb capacity. In contrast, our TCAM has a clock period of 5ns, and can perform sustained routing lookups at line rates of 76.8 Gb/s. Routing lookups have a 3 clock cycle latency, with a 1 clock cycle throughput. Unlike many existing TCAM implementations, our TCAM imposes no constraints on where a new entry is inserted.

The speed of our TCAM is achieved by a carefully trading off architectural and implementation choices. By pipelining the individual operations of the TCAM, we are able to obtain extremely fast sustained lookup speeds. By carefully partitioning the TCAM into blocks, we achieve a fast design and a square aspect ratio of the IC. Individual blocks have their *MATCH* lines implemented in 4 decoupled sections, resulting in significant speed-up. A special LPM circuit based on a pre-charged Wired-NOR computation ensures that the determination of the longest prefix for each block is done efficiently. This LPM circuit has delays which grow logarithmically in n, the length of the network address. This makes our design especially suitable for IPv6 applications. Finally, by locating the sense amplifiers of the LPM and SRAM sections in the center of each block, we are able to achieve faster read accesses to these sections.

We have validated and implemented our design using state-of-the-art circuit design and analysis tools, and generated the mask layout for our design. Our final die size is approximately 18mm on a side, not including I/O logic.

References

[1] V. Fuller *et al.*, "Classless Inter-Domain Routing (CIDR): an address assignment and aggregation strategy," *RFC 1519*, 1993.

[2] A. J. McAuley and P. Francis, "Fast Routing Table Lookup Using CAMs," *Proc.IEEE INFOCOM*, pp. 1382–1391, March-April 1993.

[3] T. B. Pei and C. Zukowski, "VLSI Implementation of Routing Tables: Tries and CAMs," *Proc. IEEE INFOCOM*, vol. 2, pp. 515–524, 1991.

[4] J. Wade and C. Sodini, "A ternary content addressable search engine," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1003–1013, Aug 1989.

[5] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*. Prentice Hall, 2nd ed., 2003.

[6] D. Shah and P. Gupta, "Fast Updating Algorithms for TCAMs," *IEEE Micro*, vol. 21, pp. 36–47, Jan/Feb 2001.

[7] J. Wakerly, *Digital Design Principles and Practices*. Prentice Hall, 1990.

[8] M. Kobayashi, T. Murase, and A. Kuriyama, "A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing," in *Proc. IEEE Intnl Conference on Communications*, vol. 3, pp. 1360–1364, 2000.

[9] M. Ciletti, *Advanced digital design with the Verilog HDL*. Prentice-Hall, 2002.

[10] "SiberCore Technologies, Ultra-18M Product Family." `http://www.sibercore.com/products_siberCAM.htm`.

[11] "BSIM3 Homepage." `http://www-device.eecs.berkeley.edu/~bsim3/intro.html`.

[12] G. T. Hamachi, R. N. Mayo, and J. K. Ousterhout, "Magic: A VLSI Layout system," in *21st Design Automation Conference Proceedings*, 1984.

[13] "Physical Design Modelling and Verification Project (SPACE Project)." `http://cas.et.tudelft.nl/research/space/html`.

[14] Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA, *Affirma™ Spectre® Circuit Simulator User Guide*, June 2000.

[15] S. Technologies, "9 MBit 100 MHz ternary CAM delivers wire speed packet forwarding and classification at OC-48, OC-192 and OC-768 speeds," Oct 2001.