FTCAM: An Area-efficient Flash-based Ternary CAM Design

Viacheslav V. Fedorov[‡], Monther Abusultan[‡], Sunil P. Khatri[‡]
[‡]ECEN Department, Texas A&M University, College Station, TX 77843, USA.

Abstract—This paper presents a Ternary Content-addressable Memory (TCAM) design which is based on the use of floating-gate (flash) transistors. TCAMs are extensively used in high speed IP networking, and are commonly found in routers in the internet core. Traditional TCAM ICs are built using CMOS devices, and a single TCAM cell utilizes 17 transistors. In contrast, our TCAM cell utilizes only 2 flash transistors, thereby significantly reducing circuit area. We cover the chip-level architecture of the TCAM IC briefly, focusing mainly on the TCAM block which does fast parallel IP routing table lookup. Our flash-based TCAM (FTCAM) block is simulated in SPICE, and we show that it has a significantly lowered area compared to a CMOS based TCAM block, with a speed that can meet current (∼400 Gb/s) data rates that are found in the internet core.

Index terms: flash, TCAM, circuit design, routing, internet.

1 Introduction

For routers that serve the internet core, the routing lookup operation needs to be performed on data that arrives on optical interfaces operating at a few 100 Gb/s. As a result, it is imperative that the routing table lookups for these routers be done in hardware, in parallel, rather than in software. The circuit of choice for hardware routers is Ternary Content Addressable Memory (TCAM) [1], [2], [3]. TCAMs ideally store an entire routing table, and perform a simultaneous comparison for *all* routing entries against the destination address of the packet being routed. A TCAM is a variant of a cache (which is also referred to as a CAM [4]), with the added ability to disregard a subset of address bits while performing the lookup. The address bits that are disregarded during a routing table lookup operation correspond to the mask bits of the routing table entry (which have 0 values).

Traditionally, TCAMs are implemented using CMOS ICs, in which each TCAM cell requires 17 transistors, and each SRAM cell (which stores the interface or next hop port) requires 6 transistors. In our paper, we realize each TCAM cell using 2 flash transistors, and each ”SRAM” cell (we refer to it as a *port cell*) requires 1 flash transistor. As a consequence, our flash-based TCAM (FTCAM) block [5] is significantly more dense than the traditional CMOS-based TCAM block (by a factor of about 7.9×), with a lookup delay which is 2.5× larger, and a power consumption which is 1.64× lower than the CMOS-based TCAM.

The key contributions of this paper are:

- To the best of the authors' knowledge, this is the first paper that utilizes flash technology to realize a Ternary CAM design.
- We implemented the flash-based TCAM block in HSPICE [6], using a 45nm PTM [7] CMOS technology, and a 45nm flash technology, yielding accurate delay and power, compared to a CMOS-based TCAM block [8], which was also implemented.
- The layout of the flash-based TCAM was generated. The TCAM block area of the flash-based design was compared with that of a CMOS based design [8], and we demonstrate that the flash-based design is about 7.9× more dense.
- Our FTCAM was simulated using a real trace of a backbone internet router, to evaluate the system lifetime and to ensure no packets were lost during real-time operation.

The remainder of this paper is organized as follows. Section 2 discusses some previous work in this area. In Section 3 we briefly describe the architecture of our envisioned flash-based TCAM, and provide details of the flash-based TCAM block we employ. In Section 4 we present experimental results comparing our flash-based TCAM block with an implementation of an existing CMOS TCAM block. Conclusions are discussed in Section 5. A brief background on internet routing and TCAM operation, as well as a discussion of Flash transistor basics, erase and program operations of the proposed cells, and the cell layouts of our design can be found in the Appendix.

2 Previous Work

A good overview of existing TCAM approaches can be found in [1], [9]. Most TCAM implementations store routing entries in blocks [3], [10], where each block contains routing entries of a particular mask length. This allows for fast lookups, since the IP address would be looked up in all blocks, and only the match from the block with the highest match length would be selected. Another implementation [11] allows routing table entries to be stored at any locations in the TCAM, but require two cycles to perform a lookup. Routing table lookup in this approach is done in a non-pipelined, two stage manner. In the first phase, the TCAM performs the lookup and performs a bitwise OR of the matching entries' masks. This produces the longest mask, which 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.

Most TCAM designs utilize a *priority encoder* [12] circuit to perform the Longest Prefix Match (LPM). The LPM computation is usually done in hardware, either using dedicated hardware [11], or by arranging the routing table entries in a specific order as described in [10].

In [13], the authors discuss techniques for reducing power in a TCAM, including 3D stacking and the use of programmable vias to save area in the port memory. As such, the techniques described are orthogonal to the ideas we present in this paper. STT-based TCAM circuits were proposed [14], however due to the resistance variation of the magnetic junctions their design utilizes three memory elements per cell, as well as 3 + 11 CMOS devices, whereas our FTCAM cell utilizes only two flash transistors. Our FTCAM uses 0.6 fJ/bit/search while the TCAM in [14] requires 7.1 fJ/bit/search. The lookup energy ratio roughly tracks the number of devices per TCAM cell. Memristor-based CAM utilizes two memory elements and three CMOS transistors [15]. However, the focus of their paper is not on TCAMs, but rather on the cell design of a memristor-based CAM. The authors of [16] present a resistive TCAM cell, to be integrated with the virtual memory, making the physical address space content-addressable. In general, the focus of this paper is at a higher level of abstraction, unlike our work.

In [8], the authors implemented an efficient TCAM, in which routing table entries are stored in any order, thus eliminating the large worst-case insertion cost of typical TCAM implementations, as described in [10]. In addition, they used 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. All the above approaches utilize a CMOS implementation of the TCAM.

There have been several research efforts which study the flash devices and their use in memory. A shortened list includes [17], [18], [19], [20], [21]. These papers report details of flash devices and their characterization. However, they do not describe the use of flash transistors for TCAM like circuits. To the best of our knowledge, there has been no prior work that uses flash devices to realize TCAM structures.

In our approach, we utilize flash transistors to realize the TCAM cells. We assume that the TCAM is realized in blocks (with 256 entries per block). Each entry consists of 256 TCAM bits (thereby supporting IPv6 routing tables), and 512 data bits. Although the focus of our work is on the design of a TCAM block, we also discuss the architecture of the entire TCAM, discussing how routing updates and route flaps would be handled. The flash-based TCAM block is compared in terms of layout area, delay and power with an efficient CMOS TCAM design [8], which was re-implemented in the 45nm PTM [7] technology for a fair comparison with our flash-based TCAM block.

3 Our Approach

3.1 Definitions

We first provide the definitions of terms used in this paper. The proposed design consists of TCAM blocks, each block containing a number of Flash TCAM cells and Port cells.

Definition 1. **TCAM block:** A block consisting of 256 rows of 256 flash TCAM cells and 512 flash port cells per row.

Definition 2. **Shadow TCAM block:** A CMOS TCAM block consisting of 512 rows of 256 CMOS TCAM cells and 512 SRAM (port) cells per row.

Definition 3. **LPM block:** A special block performing the Longest Prefix Match operation among multiple matching flash TCAM blocks.

Definition 4. **Flash TCAM cell:** A circuit consisting of two flash transistors, capable of storing a ternary value and comparing against the stored value.

Definition 5. **Flash Port (Memory) cell:** A single flash transistor circuit holding one bit of port (next hop) information.

3.2 Overview

In the next two sections, we discuss the design of out FTCAM router following a top-down approach.

In Section 3.3, we discuss the architecture of the TCAM at the chip level, along with a discussion of how insertions and deletions are handled. In the proposed design, the routing entries are stored in *blocks* of a fixed size.

The focus of this paper, however, is the design of the TCAM block. The design of each block is discussed in Section 3.4. This section covers the design of the TCAM cell and the port cell. The FTCAM block in our approach was simulated in HSPICE [6] , with wiring parasitics extracted via Raphael [22]. The layouts of the FTCAM and port cells were generated as well. The delay, area and power of the FTCAM block were compared with those of a CMOS TCAM block [8].

In existing flash memory ICs (which are used in SD-Cards, memory sticks, and SSDs), both CMOS and Flash transistors co-exist on the same die. The CMOS transistors are used for control operations, pulse generation, and read-out management. Similarly, our FTCAM based router also uses CMOS and flash devices together, on the same die.

Whenever a TCAM based router comes online, a protocol called the BGP (Border Gateway Protocol) is run to announce its routes to neighboring routers, and, conversely, to discover new routes from them.

3.3 TCAM Architecture

Our design stores routing entries in blocks, with each block used for the storage of entries with a specific mask length M. For each mask length M, a fixed number of blocks N_M are employed, based on the mask length statistics of the routing table entries being stored in the router.

The floorplan of our proposed TCAM design is shown in Figure 1. The total number of flash TCAM blocks in this design is N^2-1, and these are divided among mask lengths,

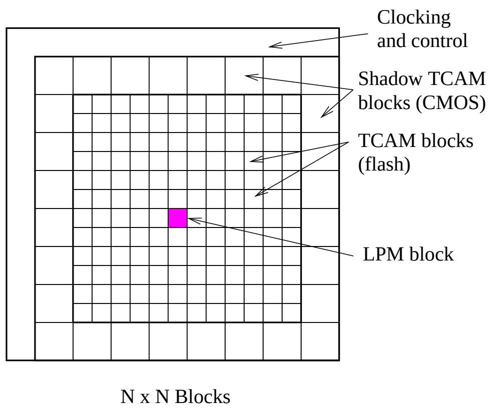

Fig. 1. Floorplan and Block Arrangement of our TCAM

such that $\Sigma_{M=1}^{M=128} N_M = N^2 - 1$. The largest mask length is 128 (assuming IPv6 IP addresses). A single central block is used as an LPM block, to select one entry with the longest prefix, from all the matching entries from several TCAM blocks. The LPM block can be implemented as a priority encoder [12], or a wired-NOR based circuit of [8], which exhibits a logarithmic delay instead of a delay that is linear in IP address length.

In our TCAM, as in the scheme of [10], memory management was performed external to the TCAM, in software. The flash TCAM entries can be thought of as NAND flash memory, but with extremely short device stacks (we utilize two devices in series, unlike a NAND flash memory in which there are typically 100s of series devices in a NAND stack). Each short stack corresponds to a TCAM entry, and can be erased and programmed independently. However, flash devices have long erase and program times (in the 100s of microseconds [23]. Therefore, our flash-based TCAM has *shadow* blocks which are implemented using CMOS cells (see Figure 1). These CMOS TCAM shadow blocks are used to perform lookups while flash TCAM blocks are being modified (and are consequently off-line). The CMOS TCAM shadow blocks are implemented in a manner similar to [8], with each block being able to store entries of variable mask lengths. As we show later, 48 512-entry CMOS shadow blocks are sufficient to support the operation of our TCAM.

The router firmware contains the *golden* state of each block of the flash-based TCAM, in its DRAM memory. The DRAM is not used for lookup (this would be prohibitively slow), but rather as a means to ensure consistency of TCAM entries. We next discuss how route additions and deletions are performed, using the DRAM, shadow TCAM blocks and flash-based TCAM blocks. The following discussion assumes that each TCAM block has a pointer indicating an unused entry (a *hole*), and each TCAM entry has a flag to indicate whether it's contents are valid or invalid (this flag is implemented in CMOS, and can be written at the speed of the TCAM clock).

The hole pointer would require a small amount of memory per TCAM block. Each pointer requires one byte as there are 256 potential hole positions. This translates into a total of 6 KB of memory (since we can accommodate 6000 FTCAM blocks in a 1.5 cm $\times$ 1.5 cm die). This amount of memory would fit into an area corresponding to one FTCAM block. Invalid/valid information is stored as a single bit in the FTCAM array, and utilizes 2% of the FTCAM block area.

Flash transistors typically have a finite (10k - 100k) number of times they can be written [24]. In traditional flash memory, *wear leveling* is performed at the architectural level to spread the wear of the cells. In our approach, the same wear leveling techniques would be used for blocks of a particular prefix size.

3.3.1 Route Addition

When a new route R with mask length M is to be added, it is first written by the firmware to the appropriate location in DRAM. This location corresponds to a vacant position in a flash-based TCAM block which stores entries with mask length M. The entry R is then copied to a CMOS shadow block which is on-line and available to do lookups.

This technique filters route flaps and reduces write-stress and wearout of the flash-based blocks. Once the CMOS block is full, the process of copying to flash-based TCAM is initiated. The corresponding flash-based TCAM blocks are now taken off-line, the contents of the DRAM blocks with the new route entries from the CMOS shadows are copied to the corresponding flash-based TCAM blocks, and the flash-based blocks are then again brought on-line. At this time, the CMOS shadow block is reset.

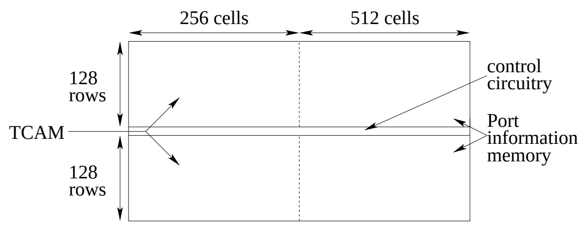

Fig. 2. TCAM Block Organization

3.3.2 Route Deletion

When a route with a mask length M is to be deleted, we first do a lookup. Assuming that the route is present in the TCAM, there is a match from one of the flash TCAM blocks which store routes of mask length M. Note that this route may be present in a CMOS shadow TCAM block as well, and the next steps are identical in that case. The row number of the route entry is hard-wired in the port memory, and is recorded along with the match. Now, we invalidate this entry, and update the hole pointer to this invalid row in the flash-based (or CMOS shadow) TCAM block. Simultaneously, the deleted route is removed from the appropriate location(s) in the DRAM as well.

In this way, the latency of writes to the flash-based TCAM blocks is hidden, and consistent operation is ensured at the router level. Note that the number of CMOS shadow TCAM blocks is significantly smaller than the number of flash-based TCAM blocks.

3.4 TCAM Block Implementation

In existing flash memory designs, flash as well as CMOS transistors are used on the same die [25]. Our work assumes that both flash and CMOS devices are present on the same die.

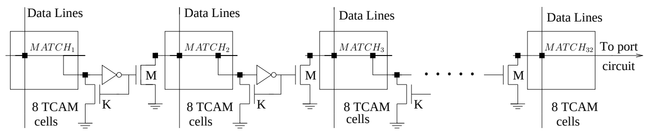

Fig. 3. TCAM Row Split into 32 Sections

As discussed in the previous section, the flash-based TCAM consists of $N^2 - 1$ TCAM blocks. Each block (illustrated in Figure 2) consists of 256 routing table entries (split into an upper group of 128 entries, and a lower group of 128 entries). In order to minimize wire lengths, the control circuitry (bitline drivers and keepers) are situated between the upper and lower groups of entries, a technique employed in most high-speed memories. The number of TCAM cells per routing table entry was chosen to be 256, which is twice the length of an IPv6 address. By disabling all but 32 TCAM cells, backward compatibility with IPv4 is also supported. The number of port cells per routing table entry is 512. Eight hard-wired bits are used to store the row number of each entry (required during route deletion, as previously described). Other bits can be used to store port (next hop) information, QoS (quality of service) data, etc.

The number of entries per block was arrived at after significant circuit level optimizations, using HSPICE [6]. For the TCAM block design, we generated the layout of the flash-based TCAM and port cells, and based on layout dimensions, extracted accurate 3-D wiring parasitics (R and C) using Raphael [22], for the HSPICE simulations. We held the number of FTCAM cells fixed at 256 to support IPv6 address lookups (with 2× over-provisioning), and varied the number of rows. Beyond 256 rows, the lookup delay (our metric for optimality) gets drastically higher. This is because in the worst case, one port cell has to pull down an entire column in the TCAM block. We found that 256 rows present a reasonable compromise in size versus speed.

Each TCAM block performs a parallel lookup of the applied destination address among the 256 routing entries stored in it. Because all the routing entries have the same mask length, it is guaranteed that exactly 0 or 1 entry will match the applied destination address. The TCAM block has 256 match lines, with each precharged match line being pulled down if one or more TCAM cells mismatch the destination address. As a consequence, each match line spans 256 TCAM cells horizontally. 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 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 smaller *sections*, as illustrated in Figure 3. In this figure, the match line is shown as 32 smaller lines. If any section i determines a mismatch condition, it pulls down its $MATCH_i$ signal, which turns on device M for section $i+1$, which pulls down the $MATCH_{i+1}$ signal for section $i+1$. This effect cascades, until $MATCH_{32}$ is pulled down. There is also a keeper device K in each section, and it serves to speed up the pulldown of the $MATCH_i$ signal once a mismatch condition is detected. In other words, if the $MATCH_q$ signal of section q $(1 < q < 32)$ is pulled low, then sections $r > q$ $(q < r \leq 32)$ are automatically pulled low by the NMOS devices labeled M, in each of the sections with index r.

We performed several SPICE simulation sweeps to determine the optimal number of sections p for the match line, and found this number to be 32 (with 8 TCAM cells per section). For the CMOS TCAM block we use for comparisons, the value of p was found to be 4.

Each TCAM block performs a parallel search on the applied destination address, independent of all other blocks. The pipelining of the operation is illustrated in Figure 4. The duration for the TCAM lookup (T_2) is larger than that for port memory lookup (T_1), and the total cycle time $T = T_1 + T_2$ is one of the figures of merit of our design.

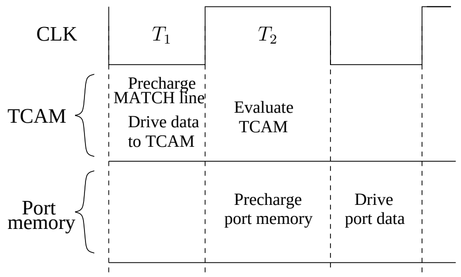

Fig. 4. Pipelined Implementation of Lookup Functionality

3.4.1 Flash-based TCAM Cell

For a primer on Flash transistor operation the reader is referred to the Appendix.

Our flash-based TCAM cell is illustrated in Figure 5. This figure refers to a cluster of four TCAM cells. Cells in row j $(j+1)$ are all connected to the $match(j)$ $(match(j+1))$ match line as shown. Each match line illustrates two TCAM cells connected to it. Each TCAM cell consists of two flash FETs connected in series to the match line. The match line is precharged, and the control gates of the two flash FETs are connected to signals a_i and b_i, respectively.

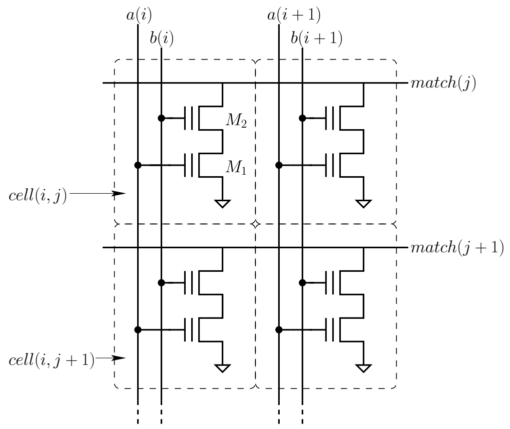

Fig. 5. Flash-based TCAM Cell

We now describe the encoding of the signals a_i and b_i that are used to perform a routing table lookup, along with the corresponding threshold voltage settings for the two flash FETs in each TCAM cell. Consider $cell(i,j)$ in Figure 5. The control gate of transistor M_1 (M_2) is connected to a_i (b_i).

Consider the Karnaugh-map [12] for the state of M_1 in Figure 6 a). When R_H is applied on a_i, transistor M_1 conducts regardless of the threshold voltage of M_1, since $R_H > T_H$ and $R_H > T_L$. When R_L is applied to a_i, M_1 conducts only when the threshold voltage of M_1 is T_L, since $T_H > R_L > T_L$.

In a similar manner, we construct the Karnaugh-map for the state of M_2 in Figure 6 b), as a function of b_i and the threshold voltage of M_2. Note that the rows (columns) of Figure 6 b) have values in the reverse order as compared to the rows (columns) of Figure 6 a).

Now consider Figure 6 c). This figure is the logical intersection of Figures 6 a) and b). The row (column) labels of Figure 6 c) are the concatenation of the row (column) labels of Figures 6 a) and b). In other words, the top left box in Figure 6 c) corresponds to the situation when R_L is applied on a_i *and* R_H is applied on b_i, when M_1's threshold value is T_L *and* M_2's threshold value is T_H. The meaning of the other 3 boxes in Figure 6 c) can be derived similarly.

Since M_1 and M_2 are connected in series, the entry in any box of Figure 6 c) is ON iff the entries in *both* the corresponding boxes of Figures 6 a) and b) are ON. This yields the logic function of the cell state in Figure 6 c).

Now let us correlate the above discussion to the TCAM cell operation. Consider the threshold voltage of the $\{M_1, M_2\}$ pair. Assume that a value of $\{T_L, T_H\}$ refers to a "1" value being stored in the TCAM, $\{T_H, T_L\}$ refers to a "0" value being stored in the TCAM and $\{T_H, T_H\}$ refers to a "X" value being stored in the TCAM. Now assume that $\{a_i, b_i\}$ value of $\{R_L, R_H\}$ refers to the "lookup 0" value of the applied destination IP address, and $\{R_H, R_L\}$ refers to the "lookup 1" value of the applied destination IP address.

Before a lookup is performed, the matchline is precharged to (R_L). When a "lookup 0" value is applied to the TCAM cell, the cell state is ON only when a "1" value is stored in the TCAM (thereby pulling down the precharged match line, and declaring a mismatch condition). When the "0" or "X" value is stored in the TCAM, the cell state is OFF, and the match line is not pulled down, as required. Similarly, when a "lookup 1" value is applied to the TCAM cell, the cell state is ON only when a "0" value is stored in the TCAM (thereby pulling down the precharged match line, and declaring a mismatch condition). When the "1" or "X" value is stored in the TCAM, the cell state is OFF, and the match line is not pulled down. This discussion describes the construction of the TCAM cell, and simultaneously serves as a proof-by-construction of correct operation of the TCAM cell.

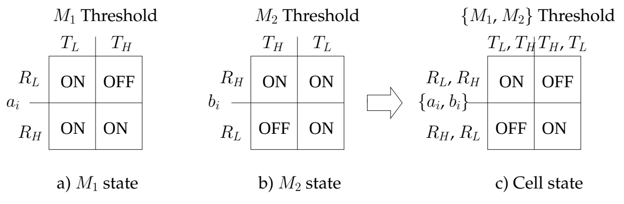

Fig. 6. TCAM Cell Logical Construction

Theorem 3.1. The proposed flash-based TCAM cell correctly performs a ternary lookup

Proof: Follows from the construction of the TCAM cell described above □

The traditional CMOS TCAM cell utilizes 17 transistors (instead of 2 flash transistors in our design) [8]. This CMOS TCAM cell consists of 2 6-T SRAM cells. One SRAM cell stores the prefix bit, and the other SRAM cell stores the mask bit.

3.4.2 Flash-based Port Cell

Our flash-based port cell is illustrated in Figure 7. This figure refers to a cluster of four port cells. Each port cell consists of a single flash FET. The control gate terminals of the cells in row j $(j+1)$ are all connected to the $match(j)$ $(match(j+1))$ match line. Each match line in the figure illustrates two port cells connected to it. Hence, when the match signal of any row is high, the corresponding port cells for that row are read out on the bitlines of the port memory.

The flash FET of each port cell is programmed with one of two threshold voltages T_H or T_L (the same threshold voltages were used for the flash FETs of the TCAM cell) and driven with one of two voltages 0 or R_L depending on whether there is a match (R_L) or no match (0). To perform a read, all bitlines are first precharged to VDD. Now, assume that *match(i)* is driven to a value R_L, indicating that the i^{th} row of the port memory is to be read out. If *cell(i,j)* has a threshold of T_H, the flash FET in this cell will not turn on, and *bitline(i)* will stay precharged. If *cell(i,j)*, on the other hand, has a threshold of T_L, the flash FET in this cell will turn on, and *bitline(i)* will be discharged to ground.

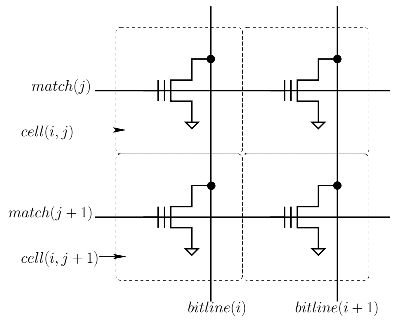

Fig. 7. Flash-based Port Cell

Our HSPICE simulations of the port memory indicated that it was faster than the TCAM cells of the TCAM block, and hence we did not split the bitlines of the port memory into sections. Also, to assist in the discharge of the port memory, each bitline had a single inverter which drove a NMOS keeper transistor. The flash-based port memory does not use a traditional sense amplifier. The structure utilized is identical to the keeper circuit employed in each section of the matchline (shown in Figure 3), and is located in the control circuitry bay between rows 128 and 129 of the TCAM block (see Figure 2).

The corresponding port cell for the CMOS TCAM block is a standard 6T SRAM cell, and is not shown for brevity. Note that the port cell of our flash-based TCAM block uses just one flash FET, as opposed to 6 CMOS FETs for the CMOS TCAM block.

For the CMOS port memory, sense amplifiers [26] are inserted at the ends of the bitlines, between rows 128 and 129 of each block.

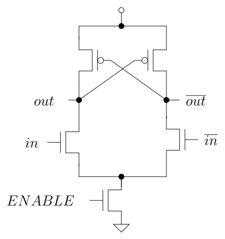

Fig. 8. Sense Amplifier used in CMOS Port Array

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 sense amplifier used is shown in Figure 8. The in and $\overline{in}$ terminals of the sense amplifier are connected to $bitline$ and $\overline{bitline}$ of the port memory array respectively. The bitlines of the CMOS port memory are precharged to VDD/2 before a read.

We refer the readers to the Appendix for erase and program operations on the proposed Flash-based cells.

4 Evaluation

We implemented the FTCAM block in HSPICE, using accurate resistive and capacitive parasitics obtained from a 3-D parasitic extraction tool, Raphael [22]. A 45nm process was used for our simulations. We started by generating a layout of the TCAM and port cells, and used these layouts to estimate the wire lengths and spacing for the bitlines and matchlines, which were then fed into Raphael for 3-D parasitic extraction.

For CMOS devices, we used a 45nm PTM process [7], while for flash devices, we derived our model card from the device-level measurements presented in [17], [18]. The basic idea is to emulate the states of a floating-gate device with two separate PTM model cards, one that models the flash FET in the low V_T state (we call this value T_L), and the other for the flash FET in the high V_T state (we call this value T_H). We used the gate and oxide thicknesses, and doping levels from [17], [18]. We then took a base 45nm PTM CMOS model card and modified it so that the threshold voltages of the two derived model cards would be T_H and T_L, respectively, and the I_{ds}-V_{gs} curve slopes matched in [17], [18].

In our experiments, VDD was set to 1.1V, and T_H and T_L were 760 mV and 210 mV respectively. Our R_H was VDD, while R_L was selected to be 600 mV. The values of R_L, T_H and T_L were chosen based on several HSPICE sweeps which aimed at minimizing TCAM lookup delay primarily, and overall power consumption secondarily.

The number of sections per match line was chosen to be 32. The output of the 32^{nd} section was buffered, registered and then driven to the port memory array with a buffer chain. The port memory bitlines were not split into sections. The bitlines of the port memory, as well as the match lines of the TCAM cells were precharged to VDD.

The FTCAM block occupied an area of 190.72 μ × 187.44 μ. About 46% of this area was used by the port memory. We refer the reader to the Appendix for flash-based TCAM and port cell layouts.

For the CMOS TCAM block, we used the same specifications as for the FTCAM block (256 entries, 256-bit wide TCAM entries and 512 bit wide port entries). We implemented the CMOS TCAM block in HSPICE, with resistive and capacitive parasitics obtained from Raphael [22]. A 45nm PTM process [7] was used for our simulations. The cell layouts for the TCAM and SRAM blocks were obtained from [8], and were scaled to obtain an estimate of the wire lengths for the matchlines and bitlines. These wire dimensions were used for 3-D parasitic extraction for the HSPICE simulations of the CMOS TCAM block.

The number of sections per match line was chosen to be 4, based on HSPICE experiments which were aimed at minimizing TCAM lookup delay. The output of the fourth section was buffered, registered and then driven to the port memory array with a buffer chain. The match lines of the TCAM cells were precharged to VDD. The port memory transitions were sped up by using a sense amplifier for each port memory cell. The port memory bitlines were not split into sections. The bitlines of the port memory were precharged to VDD/2 and equilibrated before the port memory read operation.

The CMOS TCAM block occupied an area of 311.04 μ × 921.6 μ, with ∼55% used by the port memory. These numbers were obtained from the layouts in [8], after scaling to a 45nm fabrication process.

The results of our comparison of the FTCAM block with the CMOS TCAM block are presented in Table 1. We note that our FTCAM block occupies about 8× less area than the CMOS TCAM block, with a power reduction of about 1.64×. The speed of the FTCAM block is about 2.5× lower than the CMOS TCAM block, but it can sustain the line rates of the fastest current routers in the internet core, as we will see next. The FTCAM cells delay (679 ps) shown in the table constitutes the critical part of the total delay, since it is 69% of the total (985 ps).

The increase in FTCAM block delay compared to a CMOS TCAM does not pose a problem at the system level since our FTCAM can support ∼400 Gb/s link speeds, as we show next in the paper. The current backbone routers need to operate at ∼100 Gb/s. We conclude that, despite the 2.5× larger delay, the proposed design is fast enough to be utilized in modern systems.

To compute the highest serial line rate supported by our TCAM, we assume a 48 byte packet, and that the FTCAM operates at ∼1 GHz clock speed (based on the total delay of 985 ps from Table 1). The system is able to perform 1 billion lookups per second, which translates to a $10^9 \times 48 \times 8 = 384$ Gb/sec link speed. Further, if the FTCAM lookups were pipelined, a clock speed of ∼1.4 GHz is achievable (based on the cycle time of 679 ps, the larger of the FTCAM cells and the Port memory delays in Table 1), allowing a $1.4 \times 10^9 \times 48 \times 8 = 537$ Gb/sec link speed.

Thus, our FTCAM supports link speeds of 384–537 Gb/sec (which is an undefined standard as of yet). This allows our FTCAM to do about 10–13 OC-768 (whose data rate is 38.4 Gb/sec) lookups in a clock cycle.

We also computed the sensitivity of the delay and power to variations in the T_H and T_L values. Both T_H and T_L were

	TCAM cells			Port memory			Total		
	Delay	Power	Area	Delay	Power	Area	Delay	Power	Area
CMOS TCAM block	218 ps	96 mW	127402.0 μ^2	174 ps	33 mW	159252.5 μ^2	393 ps	129 mW	286654.5 μ^2
FTCAM block	679 ps	65.2 mW	19398.6 μ^2	306 ps	14.1 mW	16731.3 μ^2	985 ps	79.3 mW	36129.9 μ^2
Ratio (Flash-based/CMOS)							2.51×	0.61×	0.126×

TABLE 1
Comparing Delay, Area and Power of CMOS TCAM and FTCAM blocks

varied by ± 20 mV independently, to simulate variations in the floating gate charge that occur during the flash programming step. With these variations in T_H and T_L, the delay numbers reduced (increased) by 2.8% (3.8%). The power consumption varied by ± 0.2%.

4.1 Lifetime Estimation

We estimate the flash-based part of our chip to have the area of 1.5cm×1.5cm. This allows us to have 6000 FTCAM blocks, for a total FTCAM size of 1.5 million entries. We assume that there are 32 + 15 CMOS shadow blocks (32 for each prefix, and 15 for double/triple buffering). The area of shadow CMOS blocks is estimated at 0.52cm×0.52cm, using the area numbers reported in Table 1. (By increasing the CMOS shadow block capacity further, it is possible to boost the FTCAM lifetime, but at the expense of the total chip area.) Hence the total chip area is about 4cm^2, including the area of the control circuitry. The interconnect is accounted for by arranging extra chip area to realize wiring. We assume that 37% of the total die area (4 cm^2) is used by the interconnect and control logic. There is no performance impact of the interconnect, since the pipelined nature of TCAM lookup and Port memory lookup allows us to hide the interconnect delay.

We conducted an experiment to estimate the lifetime of our TCAM under real workloads. For this purpose, we developed and used an in-house TCAM-based router simulator based on the architecture described in Section 3. The contents of the routing table were populated using the Routing Information Base (RIB) snapshots [27] of a real internet router on July 1, 2014. The size of the internet routing table was about 500K entries. We "replayed" the UPDATE traces for one full day following the RIB dump and recorded the number of writes to each FTCAM block. We also tracked the CMOS shadow block utilization.

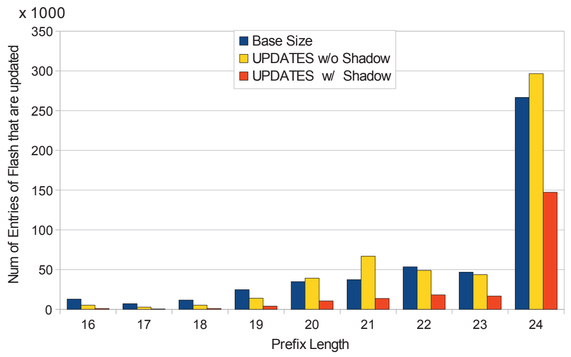

Fig. 9. FTCAM Utilization over One Day (with and without CMOS Shadow Blocks)

Figure 9 shows the number of flash entries that are used over the duration of one day, as a function of their prefix length. "Base Size" refers to the number of entries in the FTCAM at the start of the day. "UPDATES" refers to the number of entries updated (with and without shadow CMOS blocks). Prefixes with less than 2000 entries are not shown.

The results show that CMOS shadow blocks (without double/ triple buffering) filter about 61% of all the UPDATEs to the FTCAM, which increases its lifetime. There were 45 cases when a CMOS shadow block was flushed twice within one second, and for the prefix length of 24, there were two cases when the CMOS shadow block was flushed 7 times in one second. The average time between consecutive flushes was 290 seconds. We estimate the erase/write delay of the FTCAM blocks to be similar to the traditional flash storage, meaning that they would support up to 3-5 writes per second [28]. Thus, in case we did not use double buffering, prefix 24 would not be possible to update correctly. Hence we use double/triple buffering.

Since some prefixes are written more frequently than 3 times a second (prefix 24 in our example), we need a pool of extra CMOS shadow blocks to double-buffer (or triple-buffer) the UPDATEs to such prefixes. From our simulation, the total number of CMOS shadow blocks required is about 32 + 15 (as explained before). With this choice, no UPDATEs are lost. The minimum number of shadow blocks necessary to sustain the FTCAM operation from our simulation was 32 + 5. With less than 37 shadow blocks the system experienced packet loss due to insufficient flash write bandwidth. We over-provision the double-buffering blocks by 3× (with a total of 32 + 15 shadow blocks) to provide a safety margin.

The observed number of UPDATEs to FTCAM blocks was 210K per day (out of a total of 500K per day). To estimate the lifetime of the proposed scheme, we made the following assumptions: (1) The FTCAM has 1.5M entries, with 500K of them occupied by the routing table. (2) In the worst case, rewriting each entry needs to erase and copy the whole 256-entry block. (3) Flash endurance is 100K write cycles [29]. (4) Randomized wear leveling techniques are used in flash-based blocks [30], [31].

In our FTCAM, 500K out of 1.5M entries are occupied. The remaining 1M entries correspond to 4000 FTCAM blocks of 256 entries. The FTCAM experiences 210K UPDATEs, which in the worst case requires 210K block writes per day. Assume any block out of the pool of unoccupied blocks can be used for (uniform) wear leveling, thus the free blocks will be fully overwritten $\frac{210K}{4000} = 52.5$ times a day. With the flash endurance of 100K rewrite cycles, the FTCAM lifetime is therefore $\frac{100K}{52.5} = 5.2$ years until wear-out. If we now assume that FTCAM blocks are statically allocated for the life of a router, it is easy to notice (see Figure 9) that the prefix 24 will be the most stressed. Hence it is reasonable to allocate more entries to that prefix. If we were to allocate 500K free entries (2000 blocks) to prefix 24 (i.e. about 750K entries total), then the estimated lifetime would be computed as follows. For prefix 24, there are 150K UPDATEs per day (see Figure 9), requiring 150K block writes per day in the worst case. There are 2000 free blocks for prefix 24. Hence the free blocks will be overwritten $\frac{150K}{2000} = 75$ times a day. Assuming a flash endurance of 100K cycles, the FTCAM lifetime can be estimated as $\frac{100K}{75} = 3.65$ years until wear-out. By adding uncommitted spare replacement FTCAM blocks, this lifetime can be further increased.

5 Conclusions

Ternary Content-addressable Memories are commonly used for high-speed IP packet routing applications in the internet core. We present a flash transistor based design of a TCAM block. Our FTCAM cell utilizes only 2 flash transistors, while our flash-based port memory cell utilizes a single flash transistor. In comparison, the traditional CMOS TCAM cell requires 17 transistors, while the CMOS port memory cell requires 6 transistors. The paper implements a TCAM block using both approaches in a 45nm process technology. Based on our layout and circuit simulation experiments, we conclude that our FTCAM block achieves an area improvement of 7.9× and a power improvement of 1.64× compared to a CMOS approach. The estimated lifetime of such FTCAM implementation is about 5 years. The speeds accomplished by our flash-based TCAM can meet current (∼400 Gb/s) data rates that are encountered in the internet core.

References

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

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

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

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

[5] V. Fedorov, M. Abusultan, and S. Khatri, "An area-efficient ternary cam design using floating gate transistors," in *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pp. 55–60, Oct 2014.

[6] I. Meta-Software, "HSPICE user's manual," Campbell, CA.

[7] "PTM website." `http://ptm.asu.edu/`.

[8] B. Gamache, Z. Pfeffer, and S. Khatri, "A fast ternary CAM design for IP networking applications," in *IEEE International Conference on Computer Communications and Networks (ICCCN)*, pp. 434–439, 2003.

[9] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, pp. 712–727, March 2006.

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

[11] M. Kobayashi, T. Murase, and A. Kuriyama, "A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing," in *Proc. IEEE ICC*, vol. 3, 2000.

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

[13] M. Lin, J. Luo, and Y. Ma, "A low-power monolithically stacked 3D-TCAM," in *IEEE International Symposium on Circuits And Systems*, IEEE, 2008.

[14] W. Xu, T. Zhang, and Y. Chen, "Design of spin-torque transfer magnetoresistive ram and cam/tcam with high sensing and search speed," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, pp. 66–74, Jan 2010.

[15] K. Eshraghian, K.-R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang, "Memristor mos content addressable memory (mcam): Hybrid architecture for future high performance search engines," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, pp. 1407–1417, Aug 2011.

[16] Q. Guo, X. Guo, Y. Bai, and E. Ipek, "A resistive TCAM accelerator for data-intensive computing," in *MICRO'11*, pp. 339–350, 2011.

[17] K.-W. Lee, K.-S. Kim, S.-W. Shin, S.-S. L. adn J.-C. Om, G.-H. Bae, and J.-H. Lee, "Modeling of Vth shift in NAND flash-memory cell device considering crosstalk and short-channel effects," *IEEE Transactions on Electron Devices*, vol. 55, pp. 1020–1026, Apr 2008.

[18] H. An, K. Kim, S. Jung, H. Yang, K. Kim, and Y. Song, "The threshold voltage fluctuation of one memory cell for the scaling-down NOR flash," in *2nd IEEE International Conference on Network Infrastructure and Digital Content*, Sept 2010.

[19] B. Park, J. Song, E. Cho, S. Hong, J. Kim, Y. Choi, Y. Kim, S. Lee, C. Lee, D. Kang, D. Lee, B. Kim, Y. Choi, W. Lee, J. Choi, K. Suh, and T. Jung, "32nm 3-bit 32gb NAND flash memory with DPT (double patterning technology) process for mass production," in *IEEE Symposium on VLSI Technology*, pp. 125–126, June 2010.

[20] E. Choi and S. Park, "Device considerations for high density and highly reliable 3D NAND flash cell in near future," in *IEEE International Electron Devices Meeting (IEDM)*, (San Francisco, CA), pp. 9.4.1 – 9.4.4, Dec 2012.

[21] H. Shim, S. Lee, B. Kim, N. Lee, D. Kim, H. Kim, B. Ahn, Y. Hwang, H. Lee, J. Kim, Y. Lee, H. Lee, J. Lee, S. Chang, J. Yang, S. Paark, S. Aritome, S. Lee, K. Ahn, G. Bae, and Y. Yang, "Highly reliable 26nm 64Gb MLC E2NAND (embedded-ECC and enhanced-efficiency) flash memory with MSP (memory signal processing) controller," in *IEEE Symposium on VLSI Technology*, pp. 216–217, June 2011.

[22] "Synopsys Raphael Interconnect Analysis Tool." `http://www.synopsys.com/Tools/TCAD/-InterconnectSimulation/Pages/Raphael.aspx`.

[23] J. DeVos, L. Haspeslagha, M. Demand, K. Devriendt, D. Wellekens, S. Beckx, and J. Houdt, "A scalable stacked gate NOR/NAND flash technology compatible with high-k and metal gates for sub 45nm generations," in *IEEE International Conference on Integrated Circuit Design and Technology (ICICDT)*, pp. 1–4, 2006.

[24] D. Jung, Y.-H. Chae, H. Jo, J.-S. Kim, and J. Lee, "A group-based wear-leveling algorithm for large-capacity flash memory storage systems," in *Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '07, pp. 160–164, ACM, 2007.

[25] K. Takeuchi, "Novel co-design of nand flash memory and nand flash controller circuits for sub-30 nm low-power high-speed solid-state drives (ssd)," *Solid-State Circuits, IEEE Journal of*, vol. 44, pp. 1227–1234, April 2009.

[26] B. Wicht, T. Nirschl, and D. Schmitt-Landsiedel, "A yield-optimized latch-type sram sense amplifier," in *Solid-State Circuits Conference, 2003. ESSCIRC '03. Proceedings of the 29th European*, pp. 409–412, Sept 2003.

[27] "University of Oregon Route Views Project." `http://www.routeviews.org`.

[28] H.-L. Li, C.-L. Yang, and H.-W. Tseng, "Energy-aware flash memory management in virtual memory system," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, pp. 952–964, Aug 2008.

[29] S. Boboila and P. Desnoyers, "Write endurance in flash drives: Measurements and analysis," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, (Berkeley, CA, USA), pp. 9–9, USENIX Association, 2010.

[30] A. Ben-aroya and S. Toledo, "Competitive analysis of flash-memory algorithms," tech. rep., Tel Aviv University, 2006.

[31] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, (New York, NY, USA), pp. 14–23, ACM, 2009.

[32] R. Fowler and L. Nordheim, "Electron emission in intense electric fields," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 119, pp. 173–181, May 1928.

[33] "ITRS website." `http://www.itrs.net/`.

[34] J. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. Davis, P. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *IEEE International Conference on Microelectronic Systems Education (MSE)*, pp. 173–174, June 2007.