A Low-Power CAM Design for LZ Data Compression

Kun-Jin Lin and
Cheng-Wen Wu, *Senior Member*, *IEEE*

Abstract—Low-power and high-performance data compressors play an increasingly important role in the portable mobile computing and wireless communication markets. Among lossless data compression algorithms for hardware implementation, LZ77 is one of the most widely used. For real-time communication, some hardware LZ compressors/decompressors have been proposed in the past. Content addressable memory (CAM) is widely considered as the most efficient architecture for pattern matching required by the LZ77 compression process. In this paper, we propose a low-power CAM-based LZ77 data compressor. By shutting down the power for unnecessary comparisons between the CAM words and the input symbol, the proposed CAM architecture consumes much lower power than the conventional ones without noticeable performance penalty. Moreover, using the proposed conditional comparison mechanism and the novel CAM cell with the NAND-type matching logic, on average we have close to two orders of improvement on power consumption, i.e., a reduction of more than 98 percent for 8-bit words. Speed is sacrificed if we use the NAND-type matching logic, but the NAND-type logic and the NOR-type logic can be combined to provide the best solution that balances power and delay. Our approach also can be applied to general-purpose CAMs which use the valid bits, so far as the proposed design techniques are adopted.

Index Terms—Associative memory, content addressable memory, data compression, low power design, LZ77 algorithm, semiconductor memory.

1 Introduction

DATA compression is an economical way to increasing the effective volume of a storage device and the effective bandwidth of a data communication channel. With the advent of VLSI technologies, more and more wireless and portable products come into our life. As an increasing number of functions are being built into these products, large-volume data transmission and intensive computation are becoming inevitable. Real-time, low-power transmission/computation is now the main design objective. Therefore, low-power and high-performance data compressors will play an increasingly important role in the growing portable computing and wireless communication markets.

Among many proposed lossless data compression algorithms, the LZ77 algorithm [1] is the most widely used one. The term *lossless* means that the original data must be identical to the decompressed result from the compressed original data. Variants of LZ77, such as compress (a typical compression program adopted in many UNIX systems), arj, lha, zip, and zoo, have become popular software tools. To fulfill real-time requirements, several works on hardware realization of LZ77 or its variants have been presented in the literature. The core computation in the LZ77 compression algorithm, which is the most time-, area-, and power-consuming task, is searching for a given string in a rather large buffer that stores previously processed input data. Some hardware architectures, including content addressable memory (CAM) [2], [3], [4], systolic array [5], [6], [7], [8], [9], and embedded processor [10], have been proposed in the past.

CAM has been considered the fastest architecture among all proposed hardware solutions for searching for a given string, as required in LZ77. A CAM-based LZ77 data compressor can process one input symbol per clock cycle, no matter what the buffer size and string length are. Even if a CAM-based compressor's clock rate is only about a half of a systolic-array-based one [4], [8], [9], the former is still much faster than the latter so far as the overall compression efficiency is concerned. It is because the total number of clock cycles required for a CAM-based compressor is much smaller than that for a systolic-array-based one. However, CAM's major drawbacks are its high hardware complexity and high power consumption. As the IC fabrication technology continues to advance into the deep submicron age, large and complex functional blocks or cores with hundreds of thousands of logic gates or more (such as the CAM block discussed here) become feasible and popular. Silicon area is becoming less a concern, but power consumption (heat removal) is becoming a major obstacle. For the CAM-based LZ77 data compressor, power consumption needs to be investigated and reduced.

In this paper, we will stress the design of a low-power CAM-based data compressor. The decompressor can be realized in a much simpler way—it consists of a RAM, an address decoder, and a simple control logic [1]. We have examined every execution step of the CAM while it is searching for a match string in its contents, and found that many comparisons between the input symbol and the CAM words actually are redundant—these comparisons are wasting power. Based on the observation, we modify a CAM word's matching mechanism without noticeable performance degradation so that the power supply of the matching mechanism will be cut off if a compare operation is unnecessary. Experimental results show that about 78.77 percent of the power consumed on the comparison mechanism can be saved—the power consumed on the comparison mechanism is one of the two main sources of the power consumed by the CAM. To further reduce the power, we also propose a novel CAM cell with the NAND-type matching logic. On average, we have close to two orders of improvement on power consumption as a whole. Speed is sacrificed if we use the NAND-type matching logic, but the NAND-type and NOR-type implementations can be combined to provide the best solution that balances power and delay. This low-power technique not only is very efficient for the CAM used in the LZ77 compressor, but it also can be easily applied to other types of CAM, such as those using the valid bits.

The paper is organized as follows: In Section 2, we present a typical procedure describing how CAM processes input symbols in an LZ77 compressor. From this procedure, we identify the compare operations that are redundant and can be removed. The CAM cell and word structure adopted in this paper is presented in Section 3. Under the structure, the bit lines (which are the major consumers of power in the CAM) have the lowest switching activities. In Section 4, we propose an approach to further reducing power by turning off the comparison mechanism for those words that do not need to be compared with the input symbol. A detailed analysis of the redundant comparisons is given. Other possible implementations considering the trade-offs between power and performance are also discussed. Finally, conclusions are given in Section 5.

2 CAM Operation in LZ77

We assume the data to be compressed is composed of *symbols*. To increase the CAM's data-processing speed and simplify the complexity of the logic blocks associated with the CAM, the CAM word length is selected to be equal to the symbol size

- *K.-J. Lin is with the Consumer Electronics Hardware Design Department, Winbond Electronics Corp., Hsinchu, Taiwan 300, ROC. E-mail: kclin0@winbond.com.tw.*
- *C.-W. Wu is with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan 300, ROC. E-mail: cww@ee.nthu.edu.tw.*

Manuscript received 15 Sept. 1997; revised 28 May 1999; accepted 7 Aug. 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105673.

(number of bits). For example, for characters using the ASCII code, the CAM word length will be eight bits. The address space of the CAM is considered cyclic. Let the CAM block have N words, i.e., $word_0, word_1, \ldots, word_{N-1}$. For $1 \leq i \leq N-2$, the neighboring predecessor and successor of $word_i$ are $word_{i-1}$ and $word_{i+1}$, respectively. Furthermore, $word_{N-1}$ is the predecessor of $word_0$ ($word_0$ is the successor of $word_{N-1}$). Every word contains an additional *flag* bit, which indicates whether the word and its predecessors are still candidates for a match string.

All the CAM's activities required for processing an incoming string of input symbols according to the LZ77 compression algorithm are shown in the following procedure:

```
Procedure LZ-CAM()
1. initialize all words;
   set the pointer P to 0;
2. set all flags to 1;
   set the match length ML to 0;
3. compare the next input symbol with all words;
4. set a word's flag to 0 if any of the following two
   conditions hold:
   a. (the word does not match the input symbol)
   b. (the flag of the word's neighboring
       predecessor is 0 just before Step 3);
5. write the input symbol into the word indexed by P;
6. P = P+1;
   ML = ML+1;
7. go the Step 3 if all the following three conditions
   hold:
   a. (there is at least one word whose flag is 1)
   b. (ML is less than a pre-defined maximal value)
   c. (there are input symbols to be processed);
8. produce an output codeword according to the
   current state of CAM;
9. go to Step 2 if there are input symbols to be
   processed;
10. end;
```

Even if there are many LZ77 variants, the central task—string searching—for the compression process still can be performed efficiently by a CAM, with possible slight modification to the above procedure. The total number of CAM words and the maximal value for *ML* are two important LZ77 parameters which must be carefully determined according to the characteristics of the input data to achieve the best compression ratio. This can be easily done by simulation [9].

A CAM can simultaneously execute N comparisons between the input symbol and each of the N words stored in the CAM in Step 3 in one clock cycle. However, we found that many comparisons are redundant and, thus, can be removed (see Condition b of Step 4). In fact, Step 3 of LZ-CAM() can be rewritten as follows without affecting its correctness:

```
3. Compare the next input symbol with only the words
   whose neighboring predecessors' flags are 1.
```

That is, if the flag of a word's neighboring predecessor is 0 before Step 3, the flag of this word must be 0 after Step 4, no matter whether the compare result between the word and the input symbol is "matched" or "unmatched." Therefore, the comparison is redundant and we can save power by not doing it.

In the original LZ77 algorithm, the buffer is treated as a FIFO (first-in first-out) queue. Whenever an input symbol has been processed, it is shifted in the buffer from one end and an unused symbol is shifted out from the other end. From the software point of view, however, direct shifting of an input symbol is an inefficient task which requires N read-and-write commands for a buffer of N words. A more efficient way is to treat the buffer as a circular queue. A pointer `P` is used to point to the word that should be shifted out in the next cycle. Note that `P` also points to the location into which an input symbol should be inserted. The "shifting" that involves all elements in the FIFO does not really take place. The strategy is adopted in LZ-CAM().

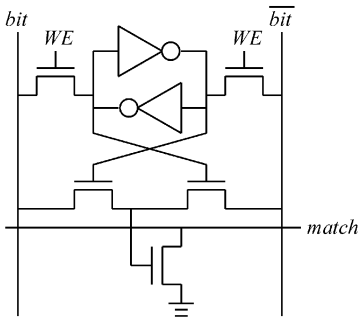

Fig. 1. A typical CAM cell.

There are other possible hardware solutions for the shift operation. The most straightforward method is to use a shift register, which performs the shifting in constant time. The operating speed is basically independent of the buffer size if we neglect the delay increase due to the clock load growth. In [2], a CAM block with an embedded shifting function was proposed to meet this requirement. Although shifting is not a time-critical operation, it is a very power-consuming task. For random bit patterns, about one half of the flip-flops (FFs) in the shift register will change their states and consume power in each clock cycle. If a buffer of 2,048 eight-bit words are considered, at least 2,048 shift register stages will be required. Each of the register stages stores eight bits of data, i.e., is composed of eight flip-flops. Furthermore, power consumption of the clock tree must be taken into consideration in such a large circuit. For a low-power LZ compressor, shift registers or the shiftable CAM design proposed in [2] are not suitable for the buffer implementation.

Note that, when the pointer `P` is pointing to the last location, it will point back to the first location in Step 6. Such a pointer can be easily implemented by a binary counter with reset capability. Generally, LZ77 requires a buffer with a capacity of $N = 2^k$ symbols, which is equal to the number of words contained in the CAM. When a k-bit binary counter reaches the maximal value $2^k - 1$, it will automatically go back to 0 in the next cycle. A combination of a counter and a k-to-2^k decoder can generate the necessary write-enable (WE) signals for all CAM words.

3 CAM Structure

A typical one-bit CAM cell adopted in the LZ77 data compressor is shown in Fig. 1 [11], [4]. The cell consists of a traditional SRAM cell, a cross-coupled XOR (exclusive-or) gate, and a pull-down transistor (PDT, shown in the bottom of the figure). The PDT is gate-controlled by the output of the XOR gate. If an input bit, represented by the two complementary voltage values on bit and $\overline{bit}$ lines, is not identical to the stored bit in the cell, the PDT is turned on and the node *match* is shorted to ground; otherwise, the PDT is turned off and *match* is in the high-impedance (HZ) state. The input *WE* is the write-enable signal. The input bit is written into the SRAM cell when *WE* is high. A CAM word is composed of eight cells with all their *match* and *WE* signals connected together. The connected PDTs thus form an open-drain eight-input NOR gate. The common *match* node is in the HZ state if and only if every

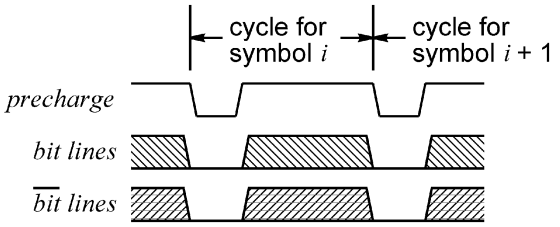

Fig. 2. Waveform to cut down static current during precharge phase.

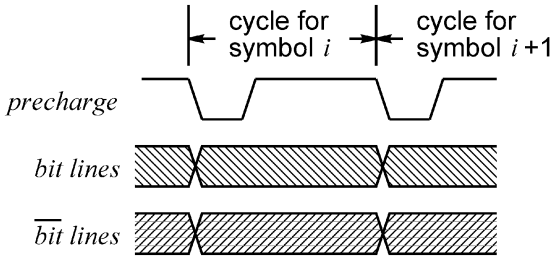

Fig. 3. Waveform to lower switching activities of bit lines.

cell in the word matches its corresponding input bit. If at least one cell does not match the input bit, *match* will be low.

Determining the size of the PDT is a dilemma. A large PDT can speed up the compare operation, since the *match* node can be quickly discharged when the input symbol is not identical to the word. However, a large PDT also increases the load of the bit lines and power and speed penalties must be paid. A compromise solution is to partition a long CAM word into "bytes," each of which contains a small number of cells. Every byte has an individual *match* node. All *match* nodes from the bytes are then ANDed together to form the final *match* signal.

Two widely used logic families capable of accomplishing the desired NOR function for the *match* signal are the pseudo-nMOS logic [3], [4] and the dynamic-CMOS logic [12]. Both logic families have their own drawbacks so far as low-power CAM design is concerned. For the pseudo-nMOS logic gate, there is always a static current in the shorted path from V_{DD} to ground if the input symbol does not match the word. Assume that the input data is random. The probability that two symbols are identical is only $1/2^n$, where n is the number of bits in a symbol. That is, if $n = 8$ (which is assumed for most data-compression applications), about $255/256 = 99.61\%$ of the total CAM words will conduct this static current in every compare operation. One solution to lower down the power consumption is to increase the resistance of the pull-up's. However, it also increases the access time of the *match* line.

The dynamic-CMOS structure is a more appropriate choice since the static current is blocked by the pMOS precharge transistor. There is additional power consumption due to the precharge clock signal, which is a global line spanning all the CAM words and requires large driving buffers. However, the power saving due to the elimination of static currents is still very obvious.

To further lower the power consumption, the waveform depicted in Fig. 2 can be adopted. All bit lines are pulled low during the precharge phase so that all PDTs can be turned off to push *match* to the high state and the transient short-circuit current through the precharge transistor and PDTs can be avoided. Unfortunately, such waveform still has a drawback of increasing the switching activities of bit lines. For every pair of bit lines, one line must transit once, no matter what input patterns are.

The power dissipated on the bit lines is also one of the dominant factors for the total power consumption. Similar to the precharge clock, the bit lines are globally distributed over the whole CAM. Large driving buffers are required. The stray capacitance and the MOS drain/source junction capacitance associated with every CAM cell impose very heavy loading onto the bit lines. The bit lines also need to drive the PDTs and the SRAM cells. Obviously, the waveform shown in Fig. 2 will cause the bit lines to switch more frequently and, thus, to consume more power. In this paper, therefore, we use the waveform shown in Fig. 3. If random input data is to be compressed, only half of the bit lines will transit on average in each clock cycle.

Besides the bit lines, the comparison mechanism is also an important power consumer. To reduce the power consumed there, we will propose an improved NOR structure by which most of the static currents will be cut off during the compression process. This will be discussed later.

4 Removal of Redundant Comparisons

4.1 Conditional Comparison Mechanism

In LZ77, the CAM compares every input symbol with all the stored words. It writes the current symbol into a specific location in the memory core before reading the next input symbol for subsequent comparison. According to LZ-CAM() as discussed in Section 2, the CAM only needs to compare the input symbol with those words whose neighboring predecessors' flags are 1 after the previous cycle. The heuristics is that if the word located at address i does not match the current input symbol, it is unnecessary for the word at address $i + 1$ to be compared with the next input symbol. This is derived directly from the LZ77 algorithm. A typical match logic for a pair of neighboring words, i.e., words i and $i + 1$, is depicted in Fig. 4. In the figure, we use the AND gates to perform the masking of unnecessary comparison results on the *match* nodes. Specifically, the match logic consists of an AND gate and a flip-flop (FF). In the beginning of the search for a match string, all FFs are preset to high

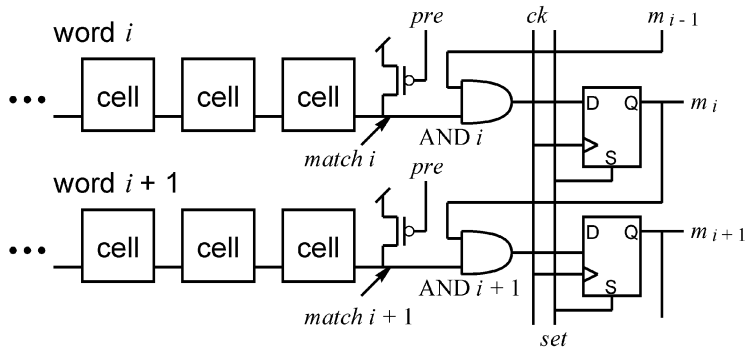

Fig. 4. A typical match logic.

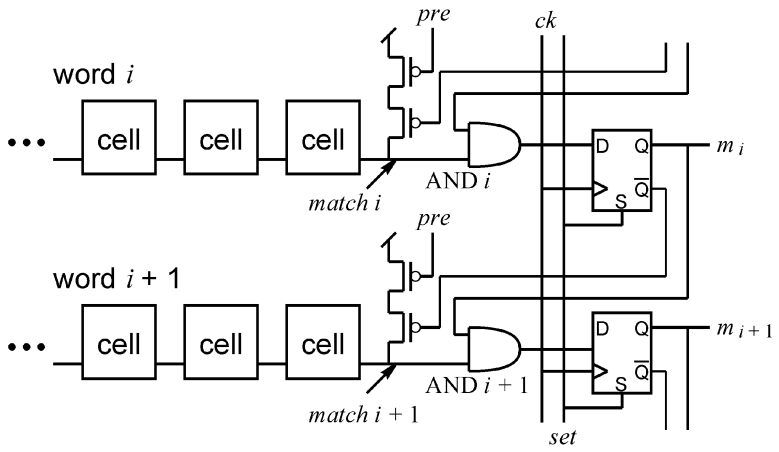

Fig. 5. Conditional comparison mechanism.

to enable all AND gates. This corresponds to Step 2 in LZ-CAM(). If the CAM word at address i does not match the input symbol, the signal $match_i$ will be low. When the FF is triggered by the system clock to store the output value of AND_i (which is low), m_i becomes low and forces the output value of AND_{i+1} to low. Thus, m_{i+1} also becomes low in the next cycle no matter what $match_{i+1}$ is. The m_i outputs from all words are sent to an address arbiter to generate an appropriate starting address of a match string for codeword generation.

Whenever m_i is low, $match_{i+1}$ is always masked by the gate AND_{i+1}. The compare operations carried out at all $word_j$, $j > i$, become redundant. It does not matter whether the $match_j$ values are correct or not. To reduce the power consumption, we can turn off the comparison mechanism at all address j, $j > i$. A straightforward method for such a purpose is to break the connection between the comparison mechanism of $word_j$ and V_{DD}. In Fig. 5, we show a novel *conditional comparison mechanism* (CCM), in which the gate of the p-type active load is connected to the inverse of the predecessor's match output (i.e., $\bar{m}_i$). If m_i is low, the path from V_{DD} to $match_{i+1}$ is open. No switching activity can occur at $match_{i+1}$ for the subsequent compare operation and the static current is totally blocked. Therefore, no power will be consumed for the redundant comparison on all $word_j$, $j > i$.

We design an 8-bit CAM by a typical $0.35\mu\mathrm{m}$ CMOS technology. The operating frequency is 50 MHz under 3.3 volts. To make a fair power comparison between the traditional and conditional match logic, all operating conditions are the same for both. For example, to achieve the same *match* precharge period, the cascaded pMOS transistors in the CCM need to be slightly enlarged to shorten the delay. The SPICE simulation discloses that both structures consume almost the same power when CCM is turned on during a compare cycle. If CCM is turned off, however, the power can be neglected.

For a general-purpose CAM or some other application-specific CAMs, there is no match logic as discussed here. However, the CCM is still applicable to them. Usually, there is a valid bit associated with every word in such CAMs. Whenever an input symbol is written into a word, the valid bit is set to indicate that the data stored in this word is effective and should participate in the compare operations. If the valid bit is implemented by an FF, the gate of the p-type active load can be connected to the complemented output of the FF. Therefore, there will be no power consumed by the pseudo-nMOS structure in each word, unless the word becomes valid.

4.2 Redundancy Analysis

We now analyze the efficiency of the approach by calculating the number of words that can benefit from the proposed CCM.

Consider a typical data compressor using a CAM of 2,048 8-bit words as the buffer. In the original CAM, a significant power level is required for a word when the word does not match the input symbol. If the CCM is adopted, however, the significant power level exists only when the CCM is also turned on. Therefore, the average number of words whose CCMs are turned on will be approximately proportional to the power consumption during a compare cycle.

Assume that the input symbols are randomly distributed. On average, only $2,048 \times \frac{1}{256} = 8$ words can match any input symbol. That is, there are $2,048 - 8 = 2,040$ words that will require the power if the original CAM structure is used. We implement the LZ77 algorithm on several benchmark files to calculate the more realistic data. Our experiments were performed on real files from the Calgary/Canterbury text compression corpus [13]. The second column of Table 1 shows the average number of words, denoted as H_{CM}, that does not match the input symbol in a compare cycle. These words require significant power while being compared with a given input symbol. We can see that on average 89.81 percent of the CAM words will dissipate substantial power for this set of files.

If the proposed CCM is adopted, all CAM words still must participate in the comparison with the first input symbol during the first cycle of the search for a new match string. However, the active loads of the match logic in most of the words will be open in the next cycle since these words do not need to be compared with the second input symbol during the next cycle. All subsequent compare operations involve only a very small portion of the CAM words.

Assume that a given n-symbol match string $S = s_0 s_1 \cdots s_{n-1}$ can be found in the buffer. The number of CAM words which will conduct static current while being compared with a specific input symbol, s_i, is denoted as h_i, where $0 \le i < n$. Obviously, using the CCM, we have

$$h_0 \ge h_1 \ge \cdots \ge h_{n-1}. \tag{1}$$

The total number of comparison operations resulting in static current in the search for the match string S is

$$H = h_0 + h_1 + \cdots + h_{n-1}. \tag{2}$$

Assume that there are k match strings for an input file after we have performed the LZ77 compression algorithm, i.e., there are k codewords in the final compressed result. The match strings are referred to as S_i, $1 \le i \le k$. Also, for each S_i, there is a

TABLE 1
Experimental Results on the Files from the Calgary Corpus

File	H_{CM}	$H_{CM}/2,048$	H_{CCM}	$H_{CCM}/2,048$	H_{CCM}/H_{CM}	L_{avg}
bib	1962.67	95.83%	406.12	19.83%	20.69%	4.04
book1	1916.19	93.56%	473.68	23.13%	24.72%	3.32
book2	1931.15	94.29%	408.79	19.96%	21.17%	4.01
geo	1846.87	90.18%	588.10	28.72%	31.84%	2.48
news	1946.83	95.06%	440.97	21.53%	22.65%	3.64
obj1	1785.72	87.19%	456.00	22.27%	25.54%	3.45
obj2	1952.55	95.34%	339.15	16.56%	17.37%	5.04
paper1	1940.00	94.73%	403.93	19.72%	20.82%	4.07
paper2	1925.44	94.02%	422.71	20.64%	21.95%	3.84
paper3	1932.63	94.37%	442.03	21.58%	22.87%	3.63
paper4	1933.12	94.39%	435.93	21.29%	22.55%	3.70
paper5	1943.19	94.88%	427.45	20.87%	22.00%	3.79
paper6	1931.94	94.33%	396.81	19.38%	20.54%	4.16
pic	454.03	22.17%	114.11	5.57%	25.13%	7.81
progc	1938.61	94.66%	385.18	18.81%	19.87%	4.32
progl	1921.35	93.82%	275.74	13.46%	14.35%	6.43
progp	1876.40	91.62%	283.68	13.85%	15.12%	6.22
trans	1967.38	96.06%	329.29	16.08%	16.74%	5.22
Avg		89.81%		19.07%	21.23%	

corresponding H_i (number of comparisons resulting in static current in the search for a match of S_i). The average number of comparisons resulting in static current for each input symbol is

$$H_{CCM} = \frac{H_1 + H_2 + \cdots + H_k}{N}, \tag{3}$$

where N is the total number of symbols contained in the input data. That is, the value of H_{CCM} refers to the average number of CAM words whose CCM will dissipate static power during a compare operation for a given input symbol. The fourth column of Table 1 shows the H_{CCM} values from our experiments on the benchmark files. The fifth column of the table shows that the average number of words that really participate in the comparison is reduced to 19.07 percent. If we compare H_{CCM} with H_{CM}, the average ratio is about 0.2123, as shown in the sixth column of the table. The average match length (L_{avg}) shown in the last column of the table is defined as

$$L_{\text{avg}} = \frac{|S_1| + |S_2| + \cdots + |S_k|}{k}, \tag{4}$$

where $|S|$ is the string length of S. If longer match strings are often encountered while compressing a specific input file, L_{avg} will be larger. From Table 1, we find that the ratio of H_{CCM} as compared with H_{CM} will decrease, while L_{avg} increases. This is also clear from the relation between H_{CCM} and L_{avg} (see (1)-(4)). Again, on average only 19.07 percent of the total CAM words can lead to static current under the CCM scheme. This number is only about 21.23 percent of H_{CM} (the original comparison mechanism). In other words, about 78.77 percent of the power consumed by the comparison mechanism of the original CAM can be saved.

4.3 NAND Implementation

In the beginning of the search for a new match string, all flip-flops must be preset to high when we use the CCM. It means that all CAM words must participate in the compare operation during this cycle. Therefore, h_0 in (1) is usually a rather large value, which is equivalent to H_{CM}. To further reduce the power consumption, we propose the cell shown in Fig. 6. The two complementary inputs from the SRAM cell to the cross-coupled XOR gate are reversed, so the PDT is turned on if the input bit is identical to the stored bit and is turned off otherwise. In addition, all the PDTs of the same word are cascaded to form an open-drain NAND gate—one end of the PDT-chain is connected to ground, and the other end is the *match* node.

The *match* node will be discharged to ground when all the PDTs of the same word are turned on. Therefore, only those words identical to the input symbol can pull their own *match* nodes to ground and consume power during this cycle. The CCM discussed above is also applicable to this structure, as illustrated in Fig. 7. The match logic consists of an OR gate and an FF. All FFs are reset to 0 to enable the OR gates before a new string search begins. If $word_i$ does not match the current input symbol, the output of its FF will turn off the p-type active load of $word_{i+1}$ to block the possible static current during the next cycle.

Table 2 shows the average numbers of CAM words that will conduct static currents. The numbers, denoted as $\hat{H}_{CCM}$, were obtained from the experiments done on the same set of benchmark files. From the table, we can see that the number of words that consume power during the compare operation has been greatly

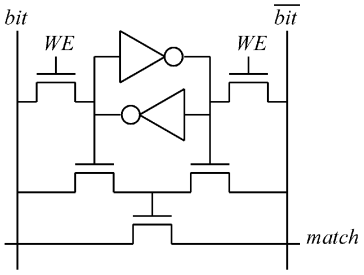

Fig. 6. Proposed CAM cell for cascaded PDTs.

TABLE 2
Experimental Results by Using the Proposed CAM Cell

File	$\hat{H}_{CCM}$	$\hat{H}_{CCM}$/2,048
bib	21.40	1.04%
book1	33.83	1.65%
book2	26.29	1.28%
geo	71.47	3.49%
news	24.43	1.19%
obj1	181.06	8.84%
obj2	21.81	1.06%
paper1	23.83	1.16%
paper2	28.19	1.38%
paper3	27.41	1.34%
paper4	25.92	1.27%
paper5	22.97	1.12%
paper6	24.92	1.22%
pic	1045.93	51.07%
progc	23.23	1.13%
progl	24.48	1.20%
progp	26.95	1.32%
trans	18.18	0.89%

reduced. On average, we have close to two orders of improvement, as shown in the third column of the table. There is an exception—the file pic, which is a monochrome bitmap picture and consists of large amounts of white space. The white space is represented by very long runs of 0s in its graphic format. Thus, the CAM is frequently filled with 0s during the compression process. Whenever there is a long run of 0s in the CAM, each 0 will be compared with almost all of the CAM words. Therefore, pic suffers from a high $\hat{H}_{CCM}$ value.

The main drawback of the pseudo-nMOS NAND gate is its long delay. The critical path is a cascade of n PDTs, where n is the number of cells in a word. If a word does not match an input symbol due to the difference between the leftmost cell (see Fig. 7) and the input bit, and the word matches the next input symbol, then it will take the longest time to discharge the *match* line. In this case, the period of a compare operation increases as compared with the NOR-type implementation. The NAND-type implementation is suitable when power is more of a concern than speed. Otherwise, the NOR-type implementation should be used. If the fanin of the NAND gate is large (e.g., greater than eight, which is rare in real applications), the chain can be partitioned and a multilevel NAND implementation can be used. For example, a high-fanin NAND function has a two-level AND-NAND implementation, which is equivalent to the NAND-OR implementation, as shown in Fig. 8. This, in fact, is a trade-off between the NOR-type implementation (for delay) and the NAND-type implementation (for power). From the figure, we see that to speed up the discharge of the *match* node through the long PDT-chain, the long word is partitioned into segments (two segments are shown in the figure). Each segment has its own pseudo-nMOS NAND gate and an individual *match* node. The CCM is the same as that shown in Fig. 7, except that all the *match* nodes in the same word and the FF output of the preceding word should be ORed together.

5 Conclusion

In this paper, we present a low-power application-specific CAM design, which is the fundamental functional block of a high-performance LZ77-based data compressor. The proposed CAM consumes much less power than the conventional CAM. We found that the bit lines of the memory core and the comparison mechanism associated with every CAM word are the major power consumer of the CAM. An appropriate CAM cell and word structure has been proposed to reduce the power consumed by the bit lines. We showed that the redundant comparisons in the compression process can be removed by turning off the power supply to those words that do not need to participate, saving about 80 percent of the power consumption of the comparison mechanism as compared with the conventional CAM. Moreover, using the proposed conditional comparison mechanism and the novel CAM cell with the NAND-type matching logic, on average we have close

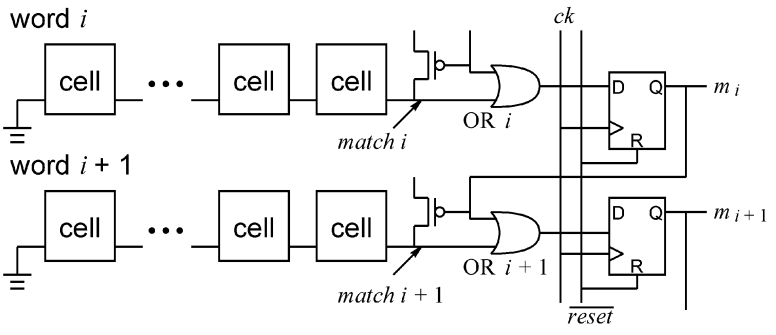

Fig. 7. Conditional match logic for the proposed CAM cell.

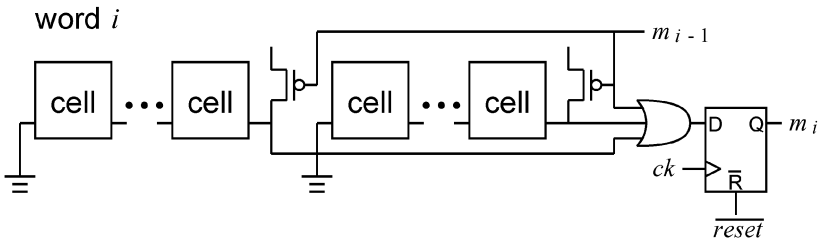

Fig. 8. A long word using two-level NAND-type implementation.

to two orders of improvement on power consumption, i.e., a reduction of more than 98 percent for 8-bit words. Speed is sacrificed if we use the NAND-type matching logic. We showed that the NAND-type logic and the NOR-type logic can be combined to provide the best solution that balances power and delay. Our approach can be applied to general-purpose CAMs so far as the design techniques proposed here are adopted. In that case, the power will be approximately proportional to the number of valid words.

REFERENCES

[1] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Information Theory*, vol. 23, pp. 337-343, May 1977.

[2] S. Jones, "100 Mbit/s Adaptive Data Compressor Design Using Selectively Shiftable Content-Addressable Memory," *IEE Proc. Pt. G*, vol. 139, pp. 498-502, Aug. 1992.

[3] B.W.Y. Wei, R. Tarver, J.-S. Kim, and K. Ng, "A Single Chip Lempel-Ziv Data Compressor," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS)*, pp. 1,953-1,955, May 1993.

[4] C.-Y. Lee and R.-Y. Yang, "High-Throughput Data Compressor Designs Using Content Addressable Memory," *IEE Proc.-Circuits Devices Systems*, vol. 142, pp. 69-73, Feb. 1995.

[5] S. Henriques and N. Ranganathan, "A Parallel Architecture for Data Compression," *Proc. IEEE Int'l Symp. Parallel and Distributed Processing*, pp. 260-266, Dec. 1990.

[6] N. Ranganathan and S. Henriques, "High-Speed VLSI Designs for Lempel-Ziv-Based Data Compression," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 40, pp. 96-106, Feb. 1993.

[7] B. Jung and W. Burleson, "A VLSI Systolic Array Architecture for Lempel-Ziv-Based Data Compression," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS)*, pp. 65-68, May 1994.

[8] S.-A. Hwang and C.-W. Wu, "Area-Efficient High-Speed Systolic Arrays for Lempel-Ziv Data Compression," *Proc. Sixth VLSI Design/CAD Symp.*, pp. 212-215, Aug. 1995.

[9] S.-A. Hwang and C.-W. Wu, "C-Testable Systolic Array Design for LZ Data Compression," *Proc. Eighth VLSI Design/CAD Symp.*, pp. 81-84, Aug. 1997.

[10] J. Chang, H.J. Jih, and J.W. Liu, "A Lossless Data Compression Processor," *Proc. Fourth VLSI Design/CAD Workshop*, pp. 134-137, Aug. 1994.

[11] W.R. Daasch, "Inexact Match Associative Memory Cell," *Electronics Letters*, vol. 27, pp. 1,623-1,625, 1991.

[12] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, second ed. Reading, Mass.: Addison-Wesley, 1993.

[13] T.C. Bell, J.G. Cleary, and I.H. Witten, *Text Compression*. Englewood Cliffs, N.J.: Prentice Hall, 1990.