High-throughput data compressor designs using content addressable memory

C.-Y. Lee
R.-Y. Yang

Indexing terms: Data storage, Content addressable memory

Abstract: The paper presents a novel VLSI architecture for high-speed data compressor designs which implement the well known LZ77 algorithm. The architecture mainly consists of three units, namely, content addressable memory, match-logic unit, and output-stage unit. The content-address-memory unit generates a set of hits signals which identify those positions whose symbols in a specified window are the same as the input symbol. These hits signals are then passed to the match-logic unit which determines both match length and location to form the kernel of compressed data. These two items are then passed to the output-stage unit for packetisation before being sent out. Simulation results show that, based on a 0.8 μm CMOS process technology, a clock speed of up to 50 MHz can be achieved for a VLSI design containing a 2K buffer size. This implies that the developing data compressor chip can handle many real-life applications, such as in high-speed data storage and networking systems.

1 Introduction

Since Lempel and Ziv [1] published the well-known LZ77 lossless data compression algorithm in 1977, many different versions have been developed. A good survey of these compression algorithms can be found in Reference 2. In principle, compression ratio, instead of algorithm complexity, is the major issue of these algorithms in the development phase. However, when real-time requirements are demanded, a trade-off between algorithm complexity and achievable compression ratio has to be taken into account. Fortunately, state-of-the-art VLSI technology offers great advantages in system integration and can be used to overcome such complexity. Several research reports on hardware implementation of the LZ77 algorithm can be found in the literature [3, 4, 5, 6]. Among these hardware solutions, different realisation approaches have been exploited such as the content addressable memory or CAM approach [3, 5], the array-processor approach [4], and the RISC approach [6]. However, these hardware solutions are not suitable for high-speed applications because of low throughput or too much hardware overhead.

© IEE, 1995

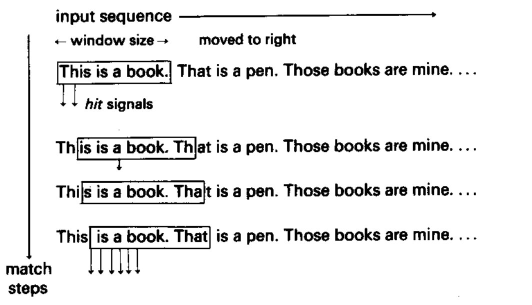
Paper 1443G (E10), first received 28th September 1993 and in revised form 24th June 1994

The authors are with the Dept. of Electronics Engineering of Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, Republic of China

We propose a VLSI architecture for single chip implementation of the LZ77 algorithm. The architecture is achieved by exploiting partitioning and pipelining techniques based on the CAM approach. Using a 0.8 μm CMOS double-metal technology, a clock rate of up to 50 MHz can be achieved.

2 The LZ77 compression algorithm

The LZ77 algorithm can be briefly illustrated as in Fig. 1*a* which contains a window to buffer a certain amount of continuous symbols. For each input symbol, the hit signal will be propagated to the next symbol for the purpose of stream matching. The input stream will be


input sequence
← window size →
moved to right
This is a book. That is a pen. Those books are mine. ...
hit signals
This is a book. That is a pen. Those books are mine. ...
This is a book. That is a pen. Those books are mine. ...
This is a book. That is a pen. Those books are mine. ...
match steps

a

matched stream	codeword	
	modified version	original version
"Th"	(0,2,0)	(0,2,'a')
"a"	(1,'a')	
no hit	(1,'t')	(0,0,'t')
"is a"	(0,6,0)	(0,6,'p')

* codeword for modified version is: (0,start address,match length) or (1,source symbol) while for original version is: (start address, match length, unmatched symbol).

b

Fig. 1 *Illustration of the LZ77 algorithm*
a LZ77 compression algorithm
b Codeword assignment

The authors would like to thank their colleagues within the SI2 group of NCTU for many discussions and fruitful suggestions.

The work was supported by the National Science Council of Taiwan, ROC under Grant NSC 83-0404-E009-031.

continuously processed in this way until no hit signal or maximum match length is detected. Then a codeword consisting of match length and start position will be sent out. However, in some cases, when the matched stream is less than two symbols, we do not gain compression ratio. Thus, in this modified version of the LZ77 algorithm, we define that the input stream will be replaced by a codeword only when its match length is more than one. Otherwise, the source symbol together with an identification (ID) code will be sent out. This is shown in Fig. 1*b*. With this method, the compression ratio is improved on average by 15% [7].

Since the window size represents the required memory space and computation complexity in hardware realisation, one has to trade off algorithm complexity and achievable compression ratio. To find an optimal solution, we defined two parameters, window size S_w and maximum match length L_m. By adjusting these two parameters and simulating several test files, we can obtain an optimum set of (S_w, L_m) under some hardware constraints. Fig. 2 shows the simulation results for one

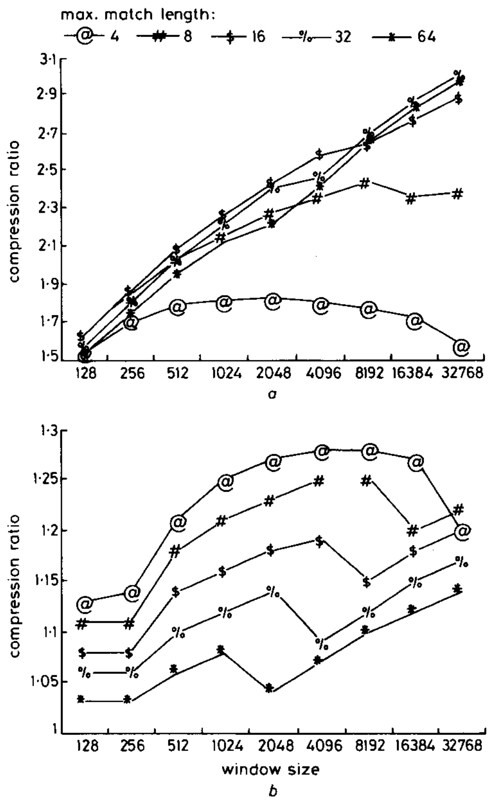


Fig. 2 *Compression ratio obtained from two test files*
a Text file of 200 Kbytes
b Image file with frame size of 256 × 256

text file and one image file with different window sizes and maximum match lengths. These results point out that the value of maximum match length should be more than four when text files are concerned. However, for image files, the maximum match length is limited to four due to their statistics. By adjusting these two parameters and simulating several test files, as well as trading off hardware complexity and application requirements, the output codeword is defined as 16-bit and the pair of (S_w, L_m) is assigned to (2K, 32) [7].

3 High-throughput VLSI architecture

A block diagram of the proposed architecture is shown in Fig. 3. It mainly consists of three blocks. The content

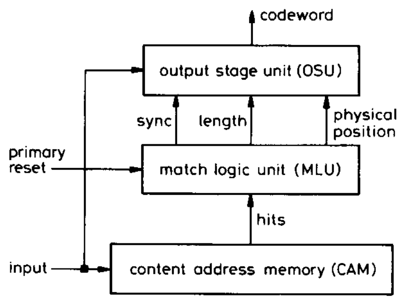


Fig. 3 *Block diagram of the LZ77 encoder*

addressable memory (CAM) acts as a dynamic moving window to partially store previous symbols and, in the meantime, to output hit signals indicating those locations whose symbols are identical to the current-input symbol. These hits signals are then passed to the match-logic unit (MLU) to produce three items of output information such as match length, physical position and synchronisation. These information items, together with the current-input symbol, are then processed at the output-stage unit (OSU) to produce a codeword which will be sent out. In the following, we first discuss the details of each block, and then present some strategy to overcome the critical path so that the clock speed can be enhanced.

3.1 Content addressable memory design

The basic structure of this unit is given in Fig. 4. Since only the hit signal is needed, each CAM bit-cell can be

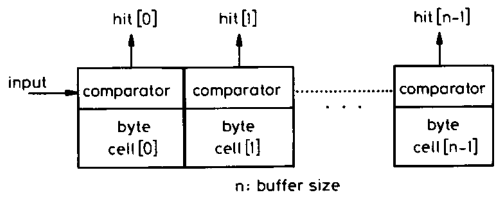


Fig. 4 *Structure of the content addressable memory*

realised on a 9-transistor cell [8]. However, the address generation for the CAM should be taken into account to optimise area and timing. For encoding purposes, input symbols are cyclically stored and then accessed. This implies that the ring counter can be exploited. For decoding purposes, start position should be first determined from received compressed data, and cannot be produced efficiently by a ring counter. Thus the random access address generator is exploited here.

3.2 Match-logic unit design

The MLU can further be partitioned into 3 subunits as shown in Fig. 5. These three subunits are designed to produce the compressed data according to the hit signals generated by the CAM.

Match cells: These cells are designed to: (1) detect the hit signals between input sequences and buffered symbols,

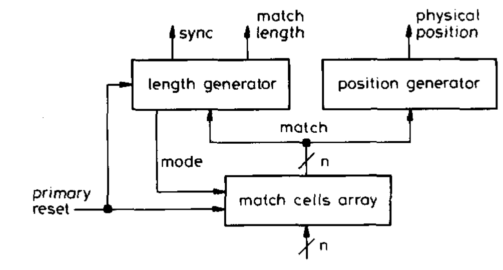


Fig. 5 *Structure of the match-logic unit*

and (2) conditionally propagate hits signals for stream matching. As shown in Fig. 6A, the input hit signals are processed by the match cells to produce the match signals, indicating the matching status of input streams.

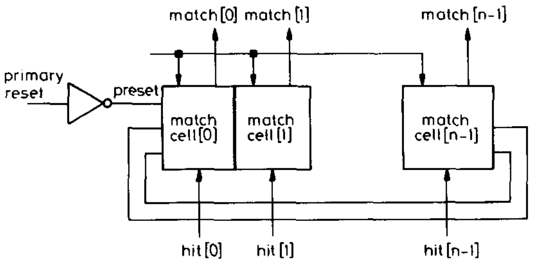


Fig. 6A *Match-cell array*

The internal structure of each match cell is given in Fig. 6B. It consists of two delay elements, one multiplexer and one AND gate. The top delay element can be preset in

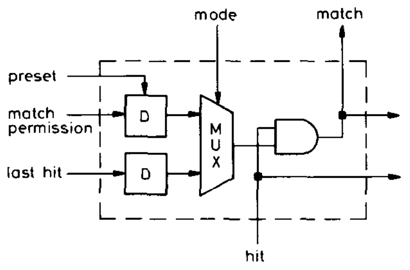


Fig. 6B *Circuit for the match cell*

the initial phase to indicate that all buffered symbols are candidates. Then this delay element continuously reports the match status of the left match cell, acting as the match permission for the current match cell. The bottom delay element stores the hit signal from the left cell and is needed only when the maximum match length is detected, or no global match signal is produced. The selection is governed by the mode control signal. The output from the multiplexer is then ANDed with the hit signal to produce a match signal.

Length generator: This subunit is designed to: (1) calculate the match length, (2) limit the maximum match length in stream matching, and (3) generate a sync signal to inform the output stage unit to produce a codeword.

The match signals are first OR-ed to detect if there is a global match signal found for the current input symbols. Output from the OR-gate is then connected to the counter to calculate the length. The accumulated length is also compared with a predefined maximum length to generate the sync signal. This sync is delayed one cycle to become the mode control signal needed in the match cell. The detailed structure of this subunit is given in Fig. 7.

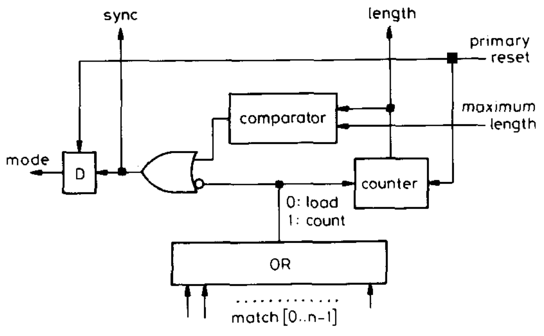


Fig. 7 *Structure of the length generator*

Priority generator: This subunit generates the end address of the matched stream. Since multiple match signals may appear simultaneously at match cells, and only one match signal is needed to produce the corresponding address in the CAM, we use the priority generation scheme [9] to reach this goal. That is, the match signal from the lower address has higher priority than those from the higher addresses.

3.3 Output-stage unit design

This unit is used to packetise the compressed data according to the sync signal. To improve the compression ratio, the match length and start position will be assembled only when the match length is greater than one. Otherwise, input symbols together with an ID code will be assembled. In our design, we assume that the start address is to be sent out, and can be obtained by subtracting the length from the physical position.

3.4 Strategy to improve clock speed

We first locate the critical path of the architecture design, and then use partitioning and pipelining strategies to improve speed. The critical path can be identified from CAM, MLU, and to OSU. Since a 2K buffer size is selected to store symbols, it is necessary to partition the memory into a 64 by 32 structure as shown in Fig. 8*a*. Here each row contains 32 symbols whose output, i.e. match signals, are then OR-ed to generate a row-match signal. These 64 row-match signals are again OR-ed to generate the final match signal exploited by other units. In the meantime, these 64 row-match signals are sent to a row priority generator and encoder to produce the row address of the matched position. Also, the output from the row priority generator is sent to a tristate buffer, which selects a matched row whose 32 match signals are again sent to the column priority generator to generate the column address as shown in Fig. 8*b*. After this row–column partitioning, we find that the speed can be improved. In addition, the layout becomes more feasible in physical design.

To further enhance clock speed, we then consider the iterative bound in order to insert the pipeline register. The recursive loop is only detected from CAM to MLU which continuously determines the match signals. Thus we can insert pipeline registers at both row and column priority generation blocks as shown in Fig. 8*a*. Here pipeline registers are inserted at the output of the row priority generator. In addition, a set of tri-state buffer controls are

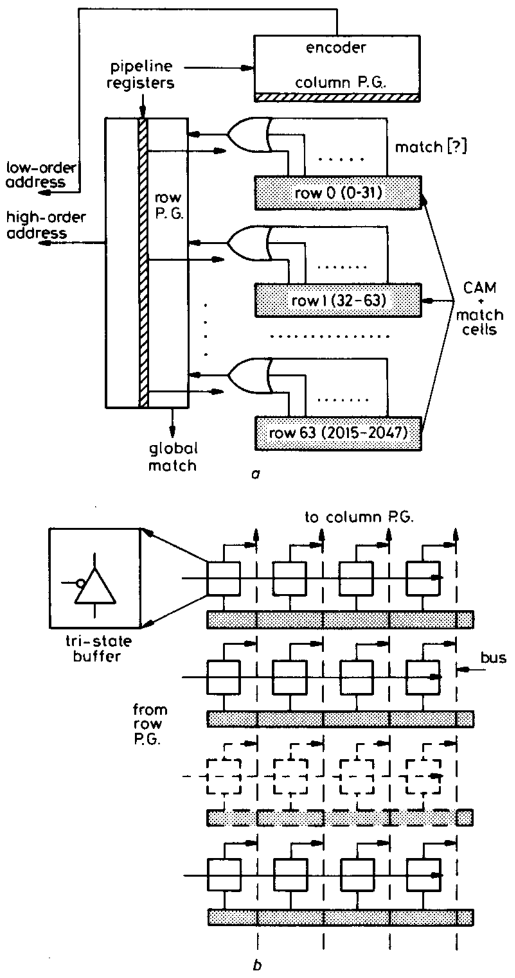


Fig. 8 *Priority generator for the row and column address*
a The CAM is reconfigured to 64 by 32 and a pipeline section can be included to improve speed
b Shows that a set of tristate buffers are added for row selection

added at each row to select the matched row for column-address generation. Then these selected match signals are pipelined at the input to the column priority generator. According to this arrangement only (64 + 32) pipeline registers are needed. If we distribute pipeline registers in another way, say inserted at the input of the row priority generator, then (64 × 32 + 64) pipeline registers are needed [7]. Although the latter arrangement does outperform the former in speed by 5%, its hardware overhead of pipeline registers is 22 times more than that of the former. By trading of area cost and speed, the former arrangement is used in our design.

The complete design is obtained by using a commercial 0.8 μm CMOS double metal process technology. The critical path is estimated to be about 17.9 ns using an HSPICE simulator. Therefore a clock speed of up to 50 MHz can be achieved.

3.5 Decoding process

Although the above mentioned architecture is derived for encoding purposes, it can also be used for decoding. When compressed data are to be decoded, it first checks the ID code and then performs decompression. When the received ID is '1', its followed data are source symbols, and can be sent out and stored in the CAM simultaneously. On the other hand, if the ID is '0', source symbols can be obtained from the CAM by decomposing the codeword into start address and match length. Note that symbols have to be read from the CAM in this case, therefore the sense amplifier is needed to improve access time.

4 Evaluation and discussions

To evaluate the architecture proposed in this paper, we give some comparison data, in terms of speed and compression capability, with those available hardware solutions for the LZ77 algorithm found in the literature. In Reference 4, a systolic array is proposed to obtain speed and throughput. However, some idle cycles can be allocated in processor elements during the coding process, leading to some hardware overhead. In addition, each input sample requires three cycles on average for compression. Also the achievable compression ratio for real-life applications is hard to achieve because of the limited number of processor elements. In Reference 6, a RISC architecture is proposed. However, throughput can only be up to a few hundred Kbytes per second, which cannot meet high-speed requirements. In References 3 and 5, the CAM approach is exploited. However, the compression ratio in Reference 3 is very low due to the limit of the CAM size. The architecture proposed in Reference 5 needs one extra cycle to load the input stream from the buffer into the CAM, and the achievable compression ratio is not declared.

In terms of speed, our proposed architecture can reach 50 MHz, which also implies that a data rate of up to 50 million samples per second can be handled. This is sufficiently high for current applications with large volumes of data. In other words, our solution can achieve the highest throughput of the mentioned solutions. This is the first feature of our design.

The achievable compression ratio lies in the range of 1.5–3.5 [7], which is competitive with the others. However, it should be mentioned here that our architecture can be reconfigured for different maximum match lengths by changing the pre-defined value. This is the second feature of our design.

In addition, our design can be combined with some standards (e.g. JPEG and MPEG) for image coding, or with pre-processing as in Reference 10, or even with tree-based codes [11] added to the output stage, to further improve compression ratio.

5 Conclusions

In this paper, we have presented an ASIC architecture for single chip implementation of the well known LZ77 algorithm. The high-throughput data compressor design is achieved by exploiting the content addressable memory. By trading off algorithm complexity and compression ratio, an optimum set for buffer size and match length has first been determined in the design process. Then, by means of design hierarchy, we obtain an efficient VLSI architecture, and achieve high speed by exploiting partitioning and pipelining techniques. Simulation results have shown that the clock speed can be up to 50 MHz, which is sufficient for high-speed storage and networking applications where the data transfer rate is more than 100 Mbits/s.

6 References

1 ZIV, J., and LEMPEL, A.: 'A universal algorithm for sequential data compression', *IEEE Trans. Inform. Theory*, 1977, **IT-23**, pp. 337–343

2 STORER, J.A.: 'Data compression: methods and theory' (Computer Science Press, Rockville, MD, 1988)

3 JONES, S.R.: '100 Mbit/s adaptive data compressor design using selectively shiftable content addressable memory', *IEE Proc. G*, 1992, **139**, (4), pp. 498–502

4 RANGANATHAN, N., and HENRIQUES, S.: 'High-speed VLSI designs for Lempel-Ziv-based data compression', *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, 1993, **40**, (2), pp. 96–106

5 WEI, W.Y., TARVER, R., KIM, J.S., and NG, K.: 'A single chip Lempel-Ziv data compressor'. Proc. ISCAS'93, Chicago, May 1993, pp. 1953–1955

6 CHANG, J., JIH, H.J., and LIU, J.W.: 'A lossless data compression processor'. Proc. of 4th VLSI/CAD Workshop, Nan-Tou, Taiwan, August 1993, pp. 134–137

7 YANG, R.-Y.: 'High-speed Lempel-Ziv data compressor designs using content addressable memory'. NCTU/DEE Master thesis, Hsinchu, Taiwan, May 1994

8 WESTE, N.H.E., and ESHRAGHIAN, K.: 'Principles of CMOS VLSI Design: a systems perspective' (Addison-Wesley, 1993, 2nd edn.), pp. 589–590

9 LEE, C.Y., JUAN, S.C., and YANG, W.W.: 'An area-efficient maximum/minimum detection circuit for digital and video signal processing'. Proc. ISCAS'93, Chicago, Illinois, May 3–6, 1993, pp. 223–226

10 VENBRUX, J., YEH, P.S., and LIU, M.N.: 'A VLSI chip set for high-speed lossless data compression', *IEEE Trans. Circuits Syst. Video Technol.*, 1992, **2**, (4), pp. 381–391

11 MUKHERJEE, A., RANGANATHAN, N., FLIEDER, J.W., and ACHARYA, T.: 'MARVLE: a VLSI chip for data compression using tree-based codes', *IEEE Trans. VLSI Syst.*, 1993, **1**, (2), pp. 203–214