

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

5,900

Open access books available

145,000

International authors and editors

180M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Design and Implementation of FPGA- based Systolic Array for LZ Data Compression

Mohamed A. Abd El Ghany, Magdy A. El-Moursy* and Aly E. Salama**

Electronics Engineering Dept., German University in Cairo, Cairo, Egypt

*Electronics Research Institute, Cairo, Egypt, Mentor Graphics Corporation, Cairo, Egypt**

*Electronics and communication Dept., Cairo University, Cairo, Egypt***

1. Introduction

Data compression is becoming an essential component of high speed data communications and storage. Lossless data compression is the process of encoding ("compressing") a body of data into a smaller body of data which can, at a later time, be uniquely decoded ("decompressed") back to the original data. In lossy compression, the decompressed data contains some approximation of the original data.

Hardware implementation of data compression algorithms is receiving increasing attention due to exponential expansion in network traffic and digital data storage usage. Many lossless data compression techniques have been proposed in the past and widely used, e.g., Huffman code (Huffman, 1952) ; (Gallager, 1978);(Park & Prasanna, 1993), arithmetic code (Bodden et al., 2004);(Said, 2004);(Said, 2003); (Howard & Vetter, 1992), run-length code (Golomb, 1966), and Lempel–Ziv (LZ) algorithms (Ziv & Lempel, 1977);(Ziv & Lempel, 1978);(Welch, 1984);(Salomon, 2004). Among those, LZ algorithms are the most popular when no prior knowledge or statistical characteristics of the data being compressed are available. The principle of the LZ algorithms is to find the longest match between the recently received string which is stored in the input buffer and the incoming string. Once this match is located, the incoming string is represented with a position tag and a length variable linking the new string to the old existing one. Since the repeated data is linked to an older one, more concise representation is achieved and compression is performed. The latency of the compression process is defined by the number of clock cycles needed to produce a codeword (matching results).

To fulfill real-time requirements, several hardware realizations of LZ and its variants have been presented in the literature. Different hardware architectures, including content addressable memory (CAM) (Lin & Wu, 2000);(Jones, 1992);(Lee & Yang, 1995), Systolic array (Ranganathan & Henriques, 1993);(Jung & Burleson, 1998);(Hwang & Wu, 2001), and embedded processor (Chang et al., 1994), have been proposed in the past. The microprocessor approach is not attractive for real- time applications, since it does not fully explore hardware parallelism (Hwang & Wu, 2001). CAM has been considered one of the fastest architectures to search for a given string in a long world, which is necessary process in LZ. A CAM- based LZ data compressor can process one input symbol per clock cycle, regardless of the buffer length and string length. A CAM- based LZ can achieve optimum speed for compression.

However, CAMs require highly complex hardware and dissipate high power. The CAM approach performs string matching through full parallel search, while the systolic-array approach exploits pipelining.

As compared to CAM-based designs, systolic-array-based designs are slower, but better in hardware cost and testability (Hwang & Wu, 2001); (Hwang & Wu, 1995); (Hwang & Wu, 1997). Preliminary design for systolic-array contains thousands of processing elements (PEs) (Ranganathan & Henriques, 1993). High speed designs were then reported later, requiring only tens of PEs (Jung & Burlison, 1998); (Hwang et al., 2001). A technique to enhance the efficiency of systolic-array approach which is used to implement Lempel-Ziv algorithm is described in this chapter. A parallel technique for LZ-based data compression is presented. The technique employs transforming a data-dependent algorithm to a data-independent algorithm. A control variable is introduced to indicate early completion which improves the latency. The proposed implementation is area and speed efficient. The effect of the input buffer length on the compression ratio is analyzed. An FPGA implementation for the proposed technique is carried out. The implemented design verifies that data can be compressed and decompressed on-the-fly which opens new areas of research in data compression.

The organization of this chapter is as follows: In Section 2, the LZ compression algorithm is explained. The results and comments about some software simulations are discussed. The dependency graph (DG) to investigate the data dependency of every computation step in the algorithm is shown. The most recent systolic array architecture is described and an area and speed efficient architecture is proposed in Section 3. In Section 4, the proposed systolic array structure is compared with the most recent structures (Hwang et al., 2001) in terms of area, and latency. An FPGA implementation for the proposed architecture showing the real time operations is demonstrated in Section 5. Finally, conclusions are provided in Section 6.

2. Lempel-ziv coding algorithm

The LZ algorithm was proposed by Ziv and Lempel in (Ranganathan & Henriques, 1993). The relationship between n and L_s for optimal compression performance is briefly examined. The data dependency of every computation step in the LZ compression algorithm is investigated.

2.1 The compression algorithm:

The LZ algorithm and its variants use a sliding window that moves along with the cursor. The window can be divided into two parts, the part before the cursor, called the dictionary, and the part starting at the cursor, called the look-ahead buffer. The lengths of these two parts are input parameters to compression algorithm. The basic algorithm is very simple, and loops executing the following steps:

1. Find the longest match of a string starting at the cursor and completely contained in the look-ahead buffer to a string starting in the dictionary.
2. Output a triple parameter (I_p, L_{max}, S) containing the position I_p of the occurrence in the window, the length L_{max} of the match and the next symbol S past the match.
3. Move the cursor $L_{max} + 1$ symbols forward.

Let us consider an example with window length of ($n=9$) and look-ahead buffer length ($L_s=3$) shown in Fig. 1.

Let the content of the window be denoted as X_i , $i = 0, 1, \dots, n-1$ and that of the look-ahead buffer be Y_j , $j = 0, 1, \dots, L_s-1$ (i.e., $Y_j = X_{i+n-L_s}$). According to LZ algorithm, the content of look-ahead buffer is compared with the dictionary content starting from X_0 to X_{n-L_s-1} to find the longest match length. If the best match in the window is found to start from position I_p and the match length is L_{max} . Then L_{max} symbols will be represented by a codeword (I_p, L_{max}). The codeword length is L_c :

$$L_c = 1 + \lceil \log_2(n-L_s) \rceil + \lceil \log_2 L_s \rceil \quad \text{bits} \quad (1)$$

L_c is fixed. Assume w bits are required to represent a symbol in the window, $l = \lceil \log_2 L_s \rceil$ bits are required to represent L_{max} , and $p = \lceil \log_2(n-L_s) \rceil$ bits are required to represent I_p . Then the compression ratio is $(l + p) / (L_{max} * w)$, where $0 \leq L_{max} \leq L_s$. Hence the compression ratio depends on the match situation.

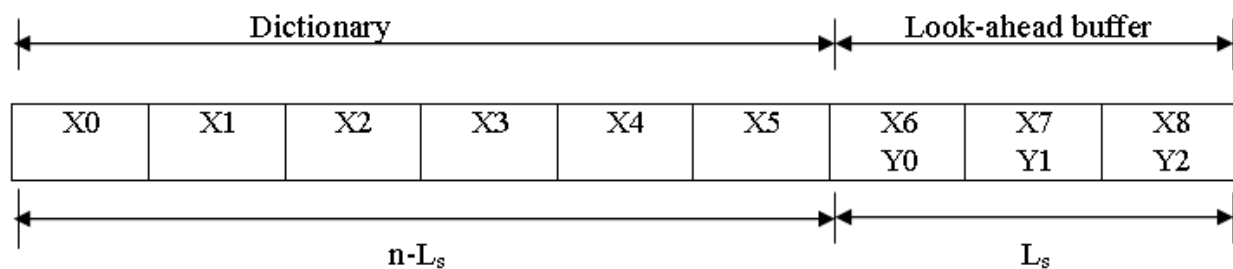


Fig. 1. window of the LZ compressor example

The codeword design and the choice of window length are crucial in achieving maximum compression. The LZ technique involves the conversion of variable length substrings into fixed length codewords that represent the pointer and the length of the match. Hence, selection of values of n and L_s can greatly influence the compression efficiency of the LZ algorithm.

2.2 Compression Algorithm Parameters Selection

Simulation for the performance of the LZ algorithm for different buffers lengths is performed using the Calgary corpus and the Silesia Corpus (Deorowicz, 2003) [16], as shown in Fig. 2 and Fig. 3, respectively. In these experiments, the codeword is up to 2 bytes long.

From Fig. 2 and Fig. 3, the compression ratio decreases when n exceeds 1024. The above improvement in the compression ratio can be obtained only when $L_s = 8, 16, \text{ or } 32$. Based on the results, the best L_s for a good compression ratio is 2^4 . Increasing L_s beyond that will require a much faster growing n (as well as hardware cost and computation time), with saturating or even decreasing compression ratio. The reason is that repeating patterns tend to be short, and that increasing L_s and n also increases the codeword length ($l + p$). To achieve a good performance for different data formats, L_s may range from 8 to 32, while n may be from 1k to 8 k. The simulation results verify the results proposed in (Arias et al., 2004).

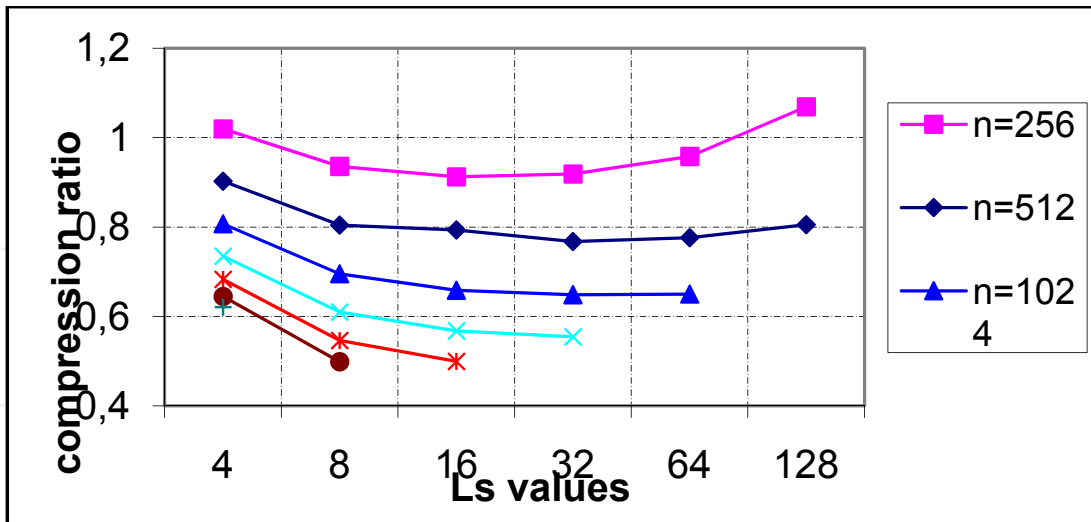


Fig. 2. The relationship between the compression ratio of Calgary corpus and L_s for different values of n

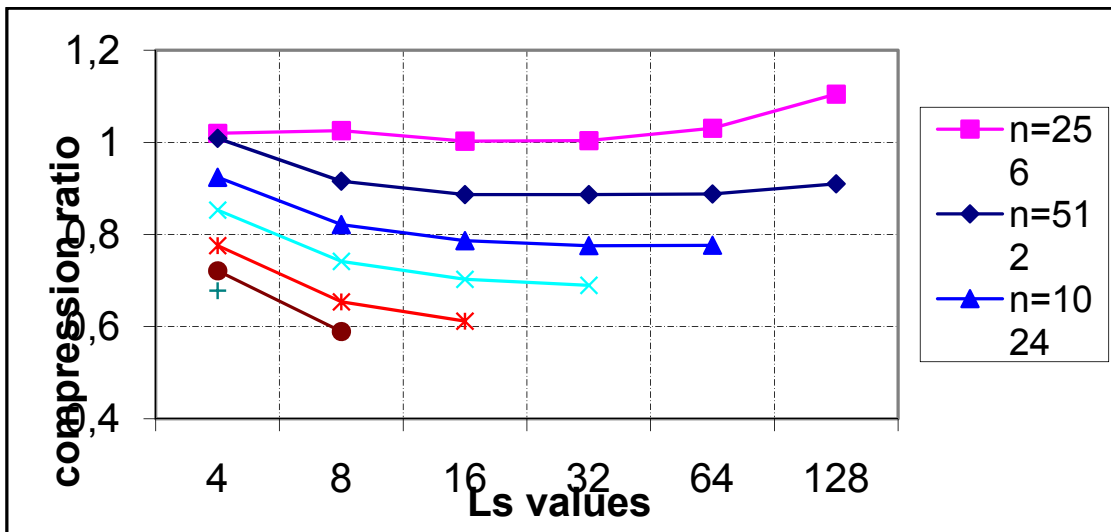


Fig. 3. The relationship between the compression ratio of Silesia corpus and L_s for different values of n .

2.3 Dependency graph:

A dependency graph (DG) is a graph that shows the dependence of the computations that occur in an algorithm. The DG of the LZ algorithm can be obtained as shown in Fig. 4. In the DG, L (match length) and E (match signal) are propagated from cell to cell. X (content of the window) and Y (content of the look-ahead buffer) are broadcast horizontally and diagonally to all cells, respectively. The DG shown in Fig. 4 is called a global DG, because it contains global signals. The global DG can be transformed into a localized DG, by propagating the input data Y and X from cell to cell instead of broadcasting them. Processor assignment can be done by projection of the DG onto the surface normal to the projection vector selected. After processor assignment, the events are scheduled using a schedule vector.

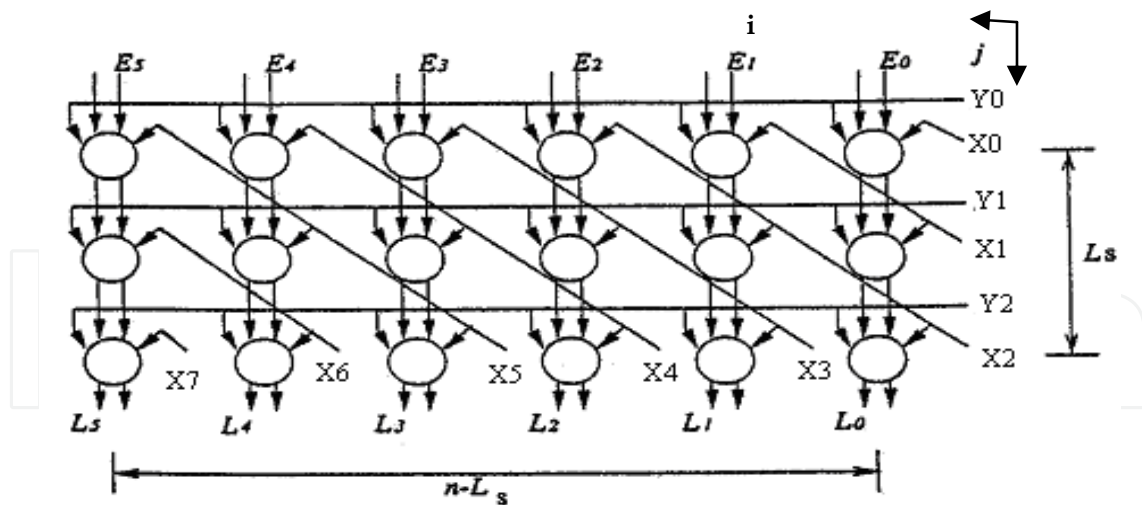


Fig. 4. The dependence graph of the LZ compression algorithm.

3. Systolic array architecture

The hardware architectures of LZ data compression demonstrate that systolic array compressors are better in hardware cost and testability. The main difference between systolic arrays and other architectures is that systolic arrays can operate at a higher clock rate (due to nearest-neighbor communication) and can easily be implemented and tested (due to regularity and homogeneity). In the following subsections, the most recent systolic array architectures are described. The high performance architecture is proposed.

3.1 Design-1

This architecture was first proposed in (Ziv & Lempel, 1977). The space-time diagram and its final array architecture are given in Fig. 5, where D represents a unit delay on the signal line between two processing elements. In Table 1, the six -sets of comparisons have to be done in sequence in order to find the maximum matching substring.

(1)	(2)	(3)	(4)	(5)	(6)
X0- Y0	X1- Y0	X2- Y0	X3- Y0	X4- Y0	X5- Y0
X1- Y1	X2- Y1	X3- Y1	X4- Y1	X5- Y1	X6- Y1
X2- Y2	X3- Y2	X4- Y2	X5- Y2	X6- Y2	X7- Y2

Table 1. The six- sets of required comparisons

Let us consider six processing elements (PE's) in parallel, each performing one vertical set of comparisons. Each processing element would require 3 time units ($L_s = 3$) to complete its set of comparisons. As shown in Fig. 5, the delay blocks in each PE delay the Y by two time steps and the X by one time step. A space-time diagram is used to illustrate the sequence of comparisons as performed by each PE. The data brought into PE0 are routed systolically through each processor from left to right. In the first time unit, X0 and Y0 are compared at PE0. In the second time unit, X1 and Y1 are compared, X0 flows to PE1, and Y0 is delayed by one cycle (time unit). In the third time unit, X2 and Y2 are compared at PE0. At this time, Y0 gets to PE1 along with X1, and PE1 performs its first comparison at the third cycle, PE0

completes all its required comparisons and stores an integer specifying the number of successful comparisons in a register called L_i . Another register called L_{max} , holds the maximum matching length obtained from the previous PE's. In the fourth time unit, PE0 compares the values of L_{max} (which for PE0 is 0) and L_i , and the greater of the two is sent to the L_{max} register of the next PE. The result of the $L_i - L_{max}$ comparison is sent to the next PE after a delay of one time unit for proper synchronization. Finally, the L_{max} value emerging out from the last PE (PE5 in this case) is the length of the longest matching substring. There is another register called PE's id, whose contents are passed along with the L_{max} value to the next PE. Its contents indicate the id of the processor element where the L_{max} value occurred which becomes the pointer to the match.

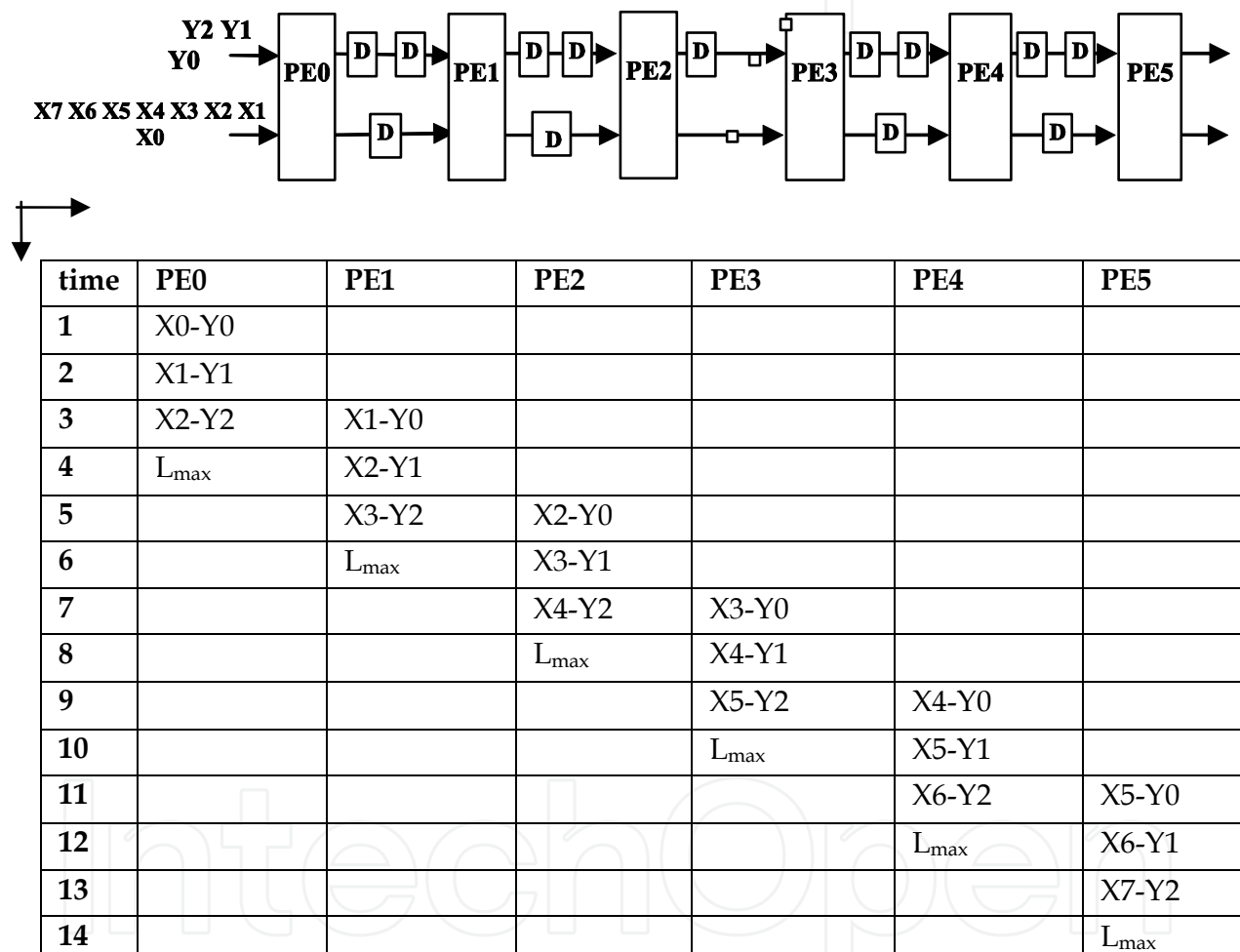


Fig. 5. Design-1 and its space-time diagram indicating the sequence of events in the 6 PE's.

The functional block of the PE is shown in Fig. 6, in which the control circuit is not included. Two comparators are needed in the PE: one is for equality check of Y_j and X_i and the other together with two multiplexers are for determining L_{max} and I_p . If Y_j and X_i are equal, a counter is incremented each time until an unsuccessful comparison occurs. Sequences X_i and Y_j can be generated by the buffer shown in Fig. 7, which is organized in two levels the upper level of the buffer holds the incoming symbols to be compressed. The contents of the upper level are copied into the lower level whenever the "load" line goes high. The lower level is used to provide data to the PE's in the correct sequence.

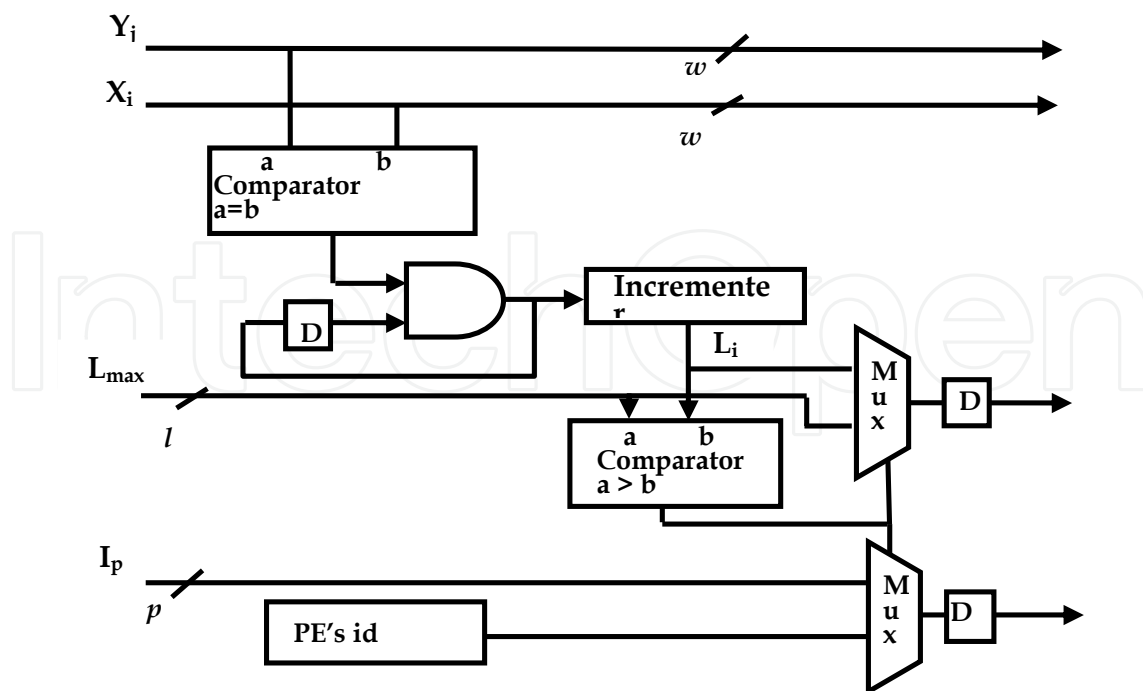


Fig. 6. The functional block of design-1 PE.

The operation of the buffer is as follows. When the longest match length is found, the same number of symbols are shifted into the in upper buffer from the source and then the symbols in the upper buffer are copied to the lower buffer in parallel to generate the next sequence to the processor array. In the Design- 1 array, The number of clock cycles needed to produce a codeword is $2(n-L_s)$, so the utilization rate of each PE is $L_s / [2(n-L_s)]$, which is low since the PE is idle from the moment when L_i is determined until the time the codeword is produced. The reason is that it seems impossible to compress subsequent input symbols before the present compression process is completed, because the number of input symbols needed to be shifted into the buffer is equal to the longest match length which is not available before the completion of the present compression process. Therefore, the design with more than L_s pipeline stages must have some idle PEs before the present codeword is produced.

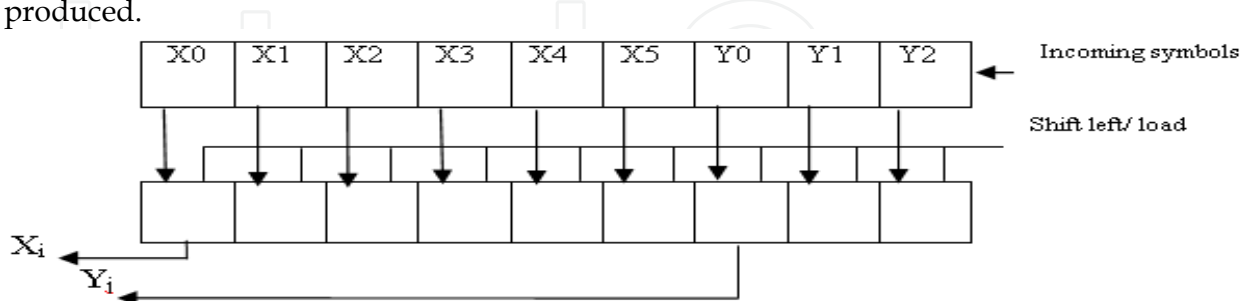


Fig. 7. the buffer.

3.2 Design-2:

The Design-2 was first proposed in (Hwang & Wu, 2001). The space-time diagram and its array architecture are given in Fig. 8. It consists of L_s process elements. The match element Y_j s stay in the PEs, and X_i and L_i both flow leftwards with delays of 1 and 2 clock cycles,

respectively. The first L_i from the leftmost PE will be obtained after $2 * L_s$ clock cycles. After that, one L_i will be obtained every clock cycle. The block diagram of the Design-2 PE is shown in Fig. 9.

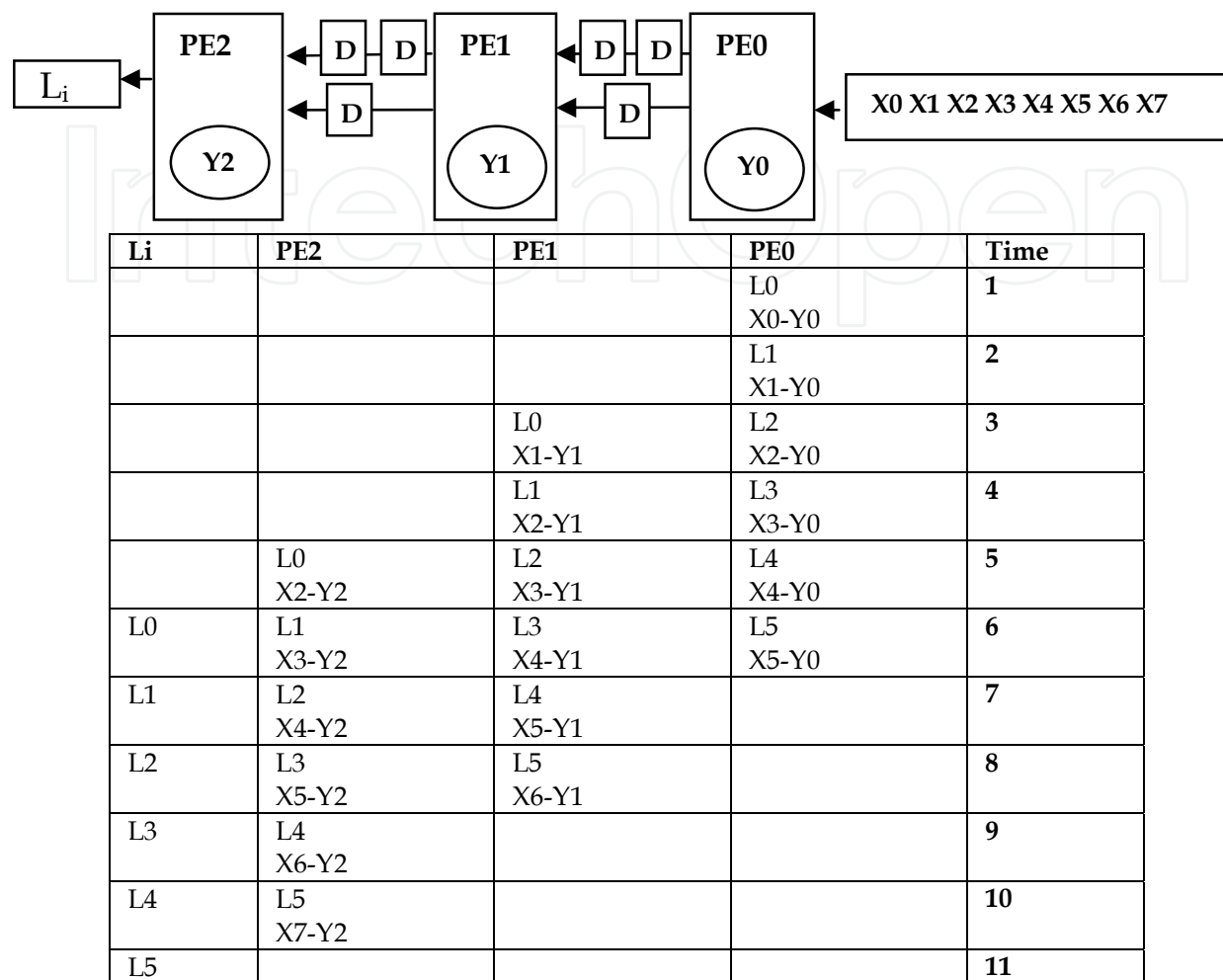


Fig. 8. Design- 2 space- time diagram and array.

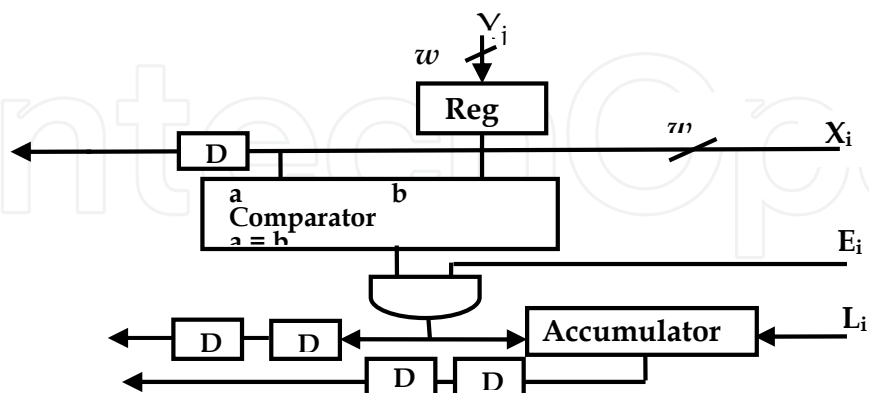


Fig. 9. The structure of Design- 2 PE.

3.3 Design-3:

Design-3 was proposed in (Ranganathan & Henriques, 1993). The space-time diagram and the resulting array are given in Fig. 10. The value of Y_i stays in the PE. The buffer element X_i moves systolically from right to left with delay of 2 clocks. L_i propagates from right to left with 1 clock cycle. The first L_i from the leftmost PE will be obtained after L_s clock cycles. After that, the subsequent ones will be obtained every clock cycle. The structure of Design-3 PE is shown in Fig. 11. MRB is needed to determine L_{max} and l_p .

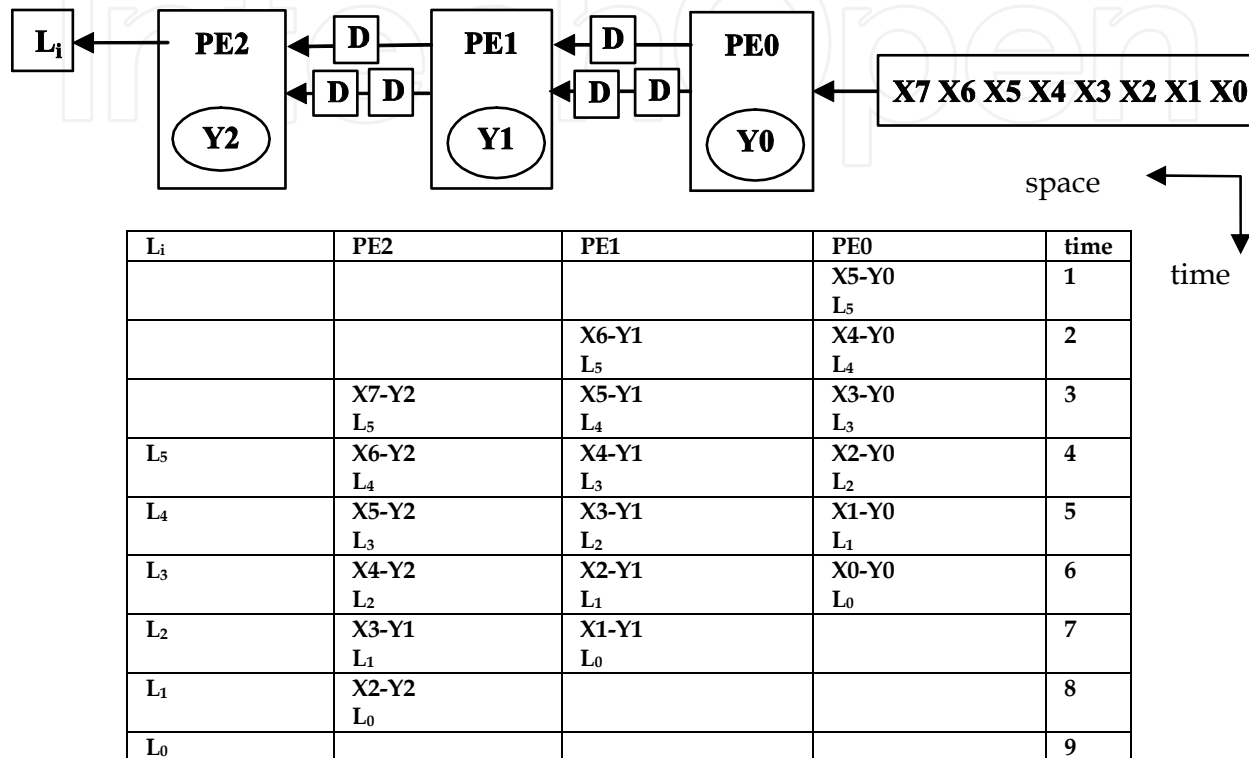


Fig. 10. The space-time diagram and the resulting array of Design-3.

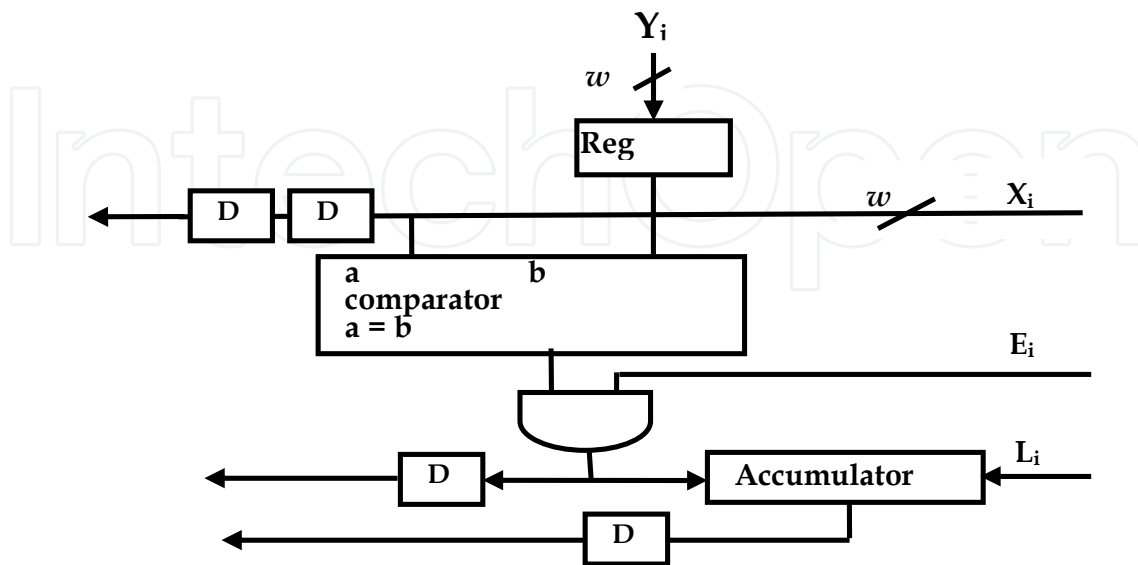


Fig. 11. The structure of Design-3 PE.

3.4 The Interleaved Design:

From the dependency graph shown in Fig. 4, the interleaved design is obtained by projecting all the nodes in a particular row to a single processor element. This design was first proposed in (Hwang & Wu, 2001). The space-time diagram and the resulting array of the interleaved design are given in Fig. 12. The Y_j s which need to be accessed in parallel do not change during the encoding step. The special buffer is needed to generate the interleaving symbols of X_i and $X_{i + [n - L_s/2]}$ as shown in Fig. 13. The first L_i will be obtained after L_s clock cycles from the leftmost PE and subsequent ones will be obtained every one clock cycle.

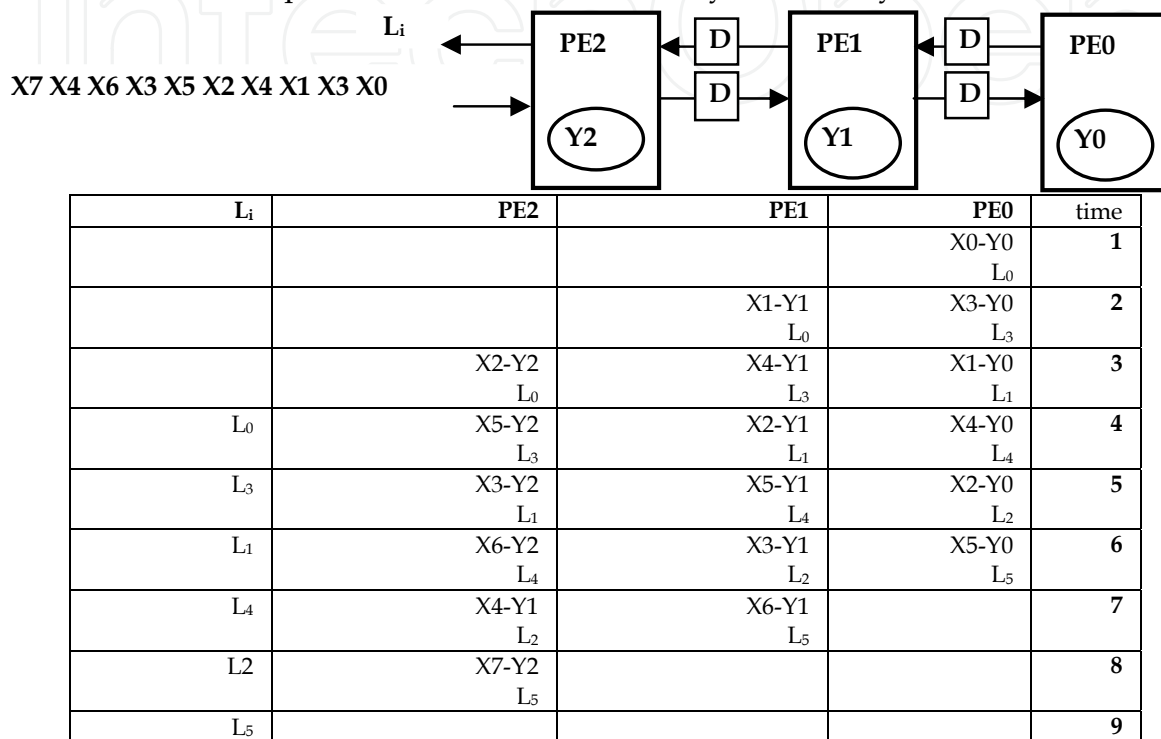


Fig. 12. The space-time diagram and the resulting array of interleaved design.

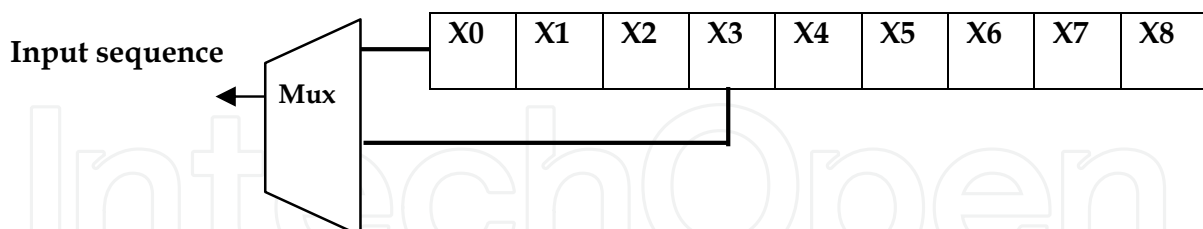


Fig. 13. buffer design for generating the interleaving symbols.

Before the encoding process, the Y_i s are preloaded which take L_s extra cycles. During the encoding process, the time to preload new source symbols depend on how many source symbols were compressed in the previous compression step, L_{max} . The block diagram of the processor element is shown in Fig. 14.

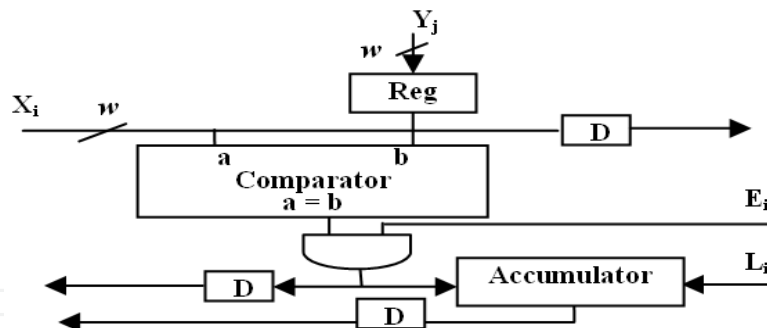


Fig. 14. The block diagram of interleaved design PE.

Match results block MRB is needed to determine L_{max} among the serially produced L_i s. MRB is shown in Fig. 15. The PEs do not need to store their ids to record the position of the L_i s. A special counter is needed to generate the sequence which interleaves the position of the first half of L_i s and the position for the second half as shown in Fig. 16. The compression time of the interleaved design is n clock cycles.

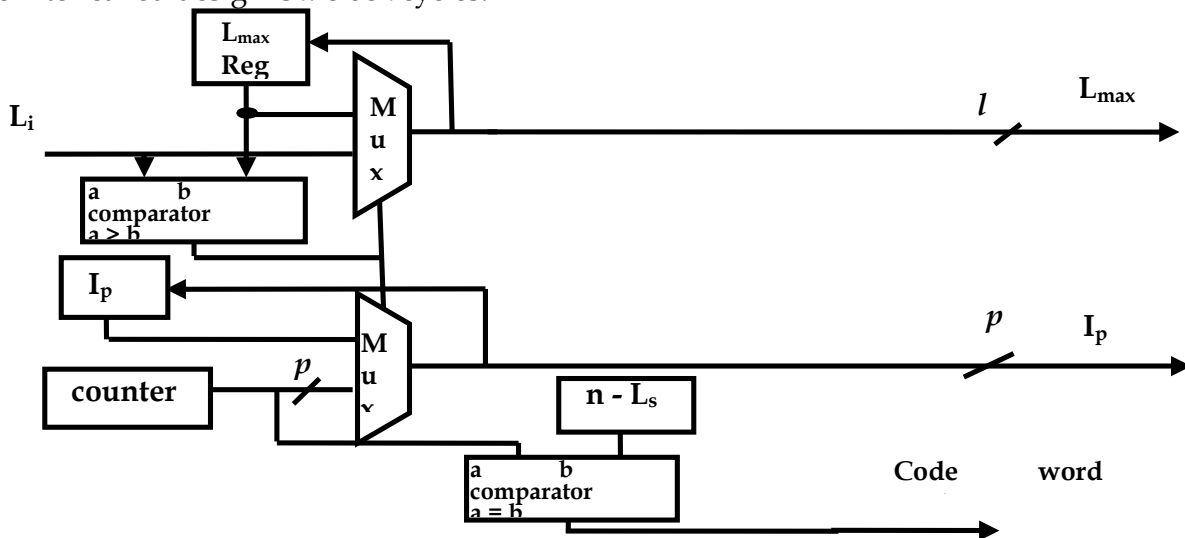


Fig. 15. the match results block

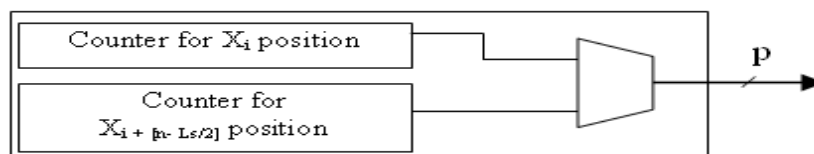


Fig. 16. Special position counter in MRB for interleaved Design.

3.5 Proposed Design (Design- P)

From the dependence graph shown in Fig. 4, all the nodes in a particular row are projected to a single processor element (Abd El Ghany, 2007). This produces an array of length L_s . The space-time diagram and the resulting array of Design-P are given in Fig. 17, where D represents a unit delay on the signal line between two processing elements.

As shown in Fig. 17, the architecture consists of L_s processing elements which are used for the comparison and L-encoder which is used to produce the matching length. Consequently, the look-ahead buffer symbols Y_{js} which do not change during the encoding step, stay in PEs.

The dictionary element X_i moves systolically from left to right with a delay of 1 clock cycle. The match signal E_i of the processing element moves to the L-encoder. The output L_i of the encoder is the matching length resulting from the comparisons at step $i-1$. The first L_i will be obtained after one clock cycle and the subsequent ones will be obtained every clock cycle. Before the encoding process, the Y_i s are preloaded to be processed and this takes L_s extra cycles. During the encoding process, the time to preload new source symbols depends upon how many source symbols were compressed in the previous compression step, L_{max} .

The functional block of the PE is shown in Fig. 18. Only one equality comparator is needed for comparing Y_j and incoming X_i . The comparator result E_i (match signal) propagates to L-encoder. The block diagram of L-encoder is shown in Fig. 19. According to E_i s (match signals), L-encoder computes the match length L_i corresponding to position i .

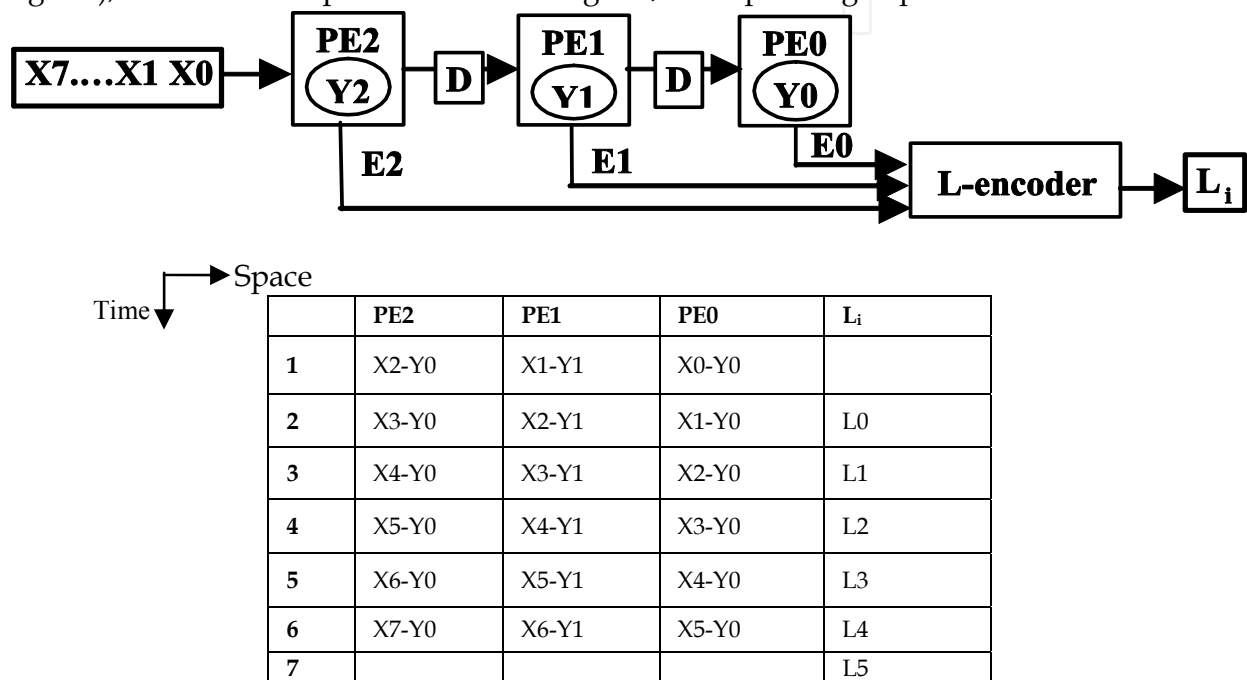


Fig. 17. Architecture of Design-P.

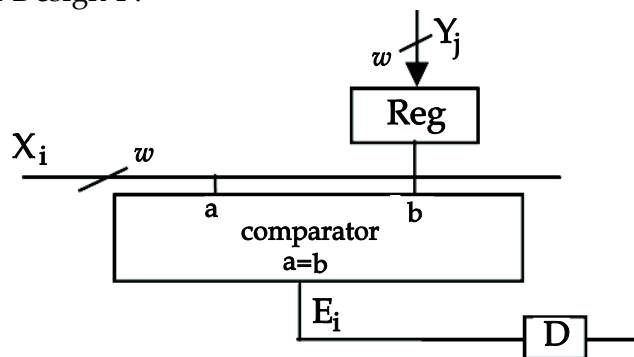


Fig. 18. The functional block of Design-P PE.

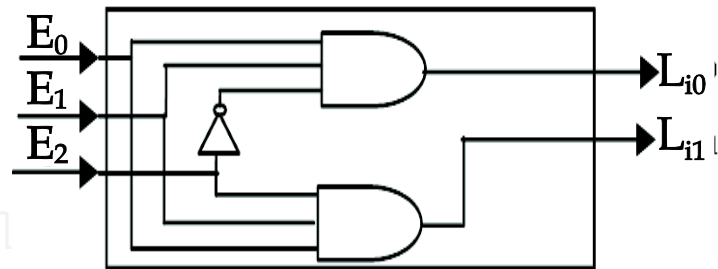


Fig. 19. L-encode

As shown in Fig. 17, it is clear that the maximum matching length is not produced by the L-encoder. So, a match results block (MRB) is needed, as shown in Fig. 20, to determine L_{max} among the serially produced L_i s. Also, the PEs need not store their ids to record the position of the L_i s (Ip). Since $p = \lceil \log_2(n-L_s) \rceil$ bits are required to represent Ip , only a p -bit counter is required to provide the position i associated with each L_i , since the time when L_i is produced corresponds to its position. MRB uses a comparator to compare the current input L_i and the present longest match length L_{max} stored in the register. If the current input L_i is larger than L_{max} , then L_i is loaded into the register and the content of the position counter is also loaded into another register which is used to store the present Ip . Another comparator is used to determine whether the whole window has been searched. It compares the content of the position counter with $n - L_s$, whose output is used as the codeword - ready signal. During the searching process, L_i might be equal to L_s when $i < n - L_s$, i.e., the content in the look-ahead buffer can be fully matched to a subset of the dictionary, and hence searching the whole window is not always necessary. An extra comparator is used to determine whether L_{max} is equal to L_s , and hence the string matching process is completed. Therefore, encoding a new set of data could start immediately. This will reduce the average compression time. The number of clock cycles needed to produce a codeword is $\lceil (n-L_s) + 1 \rceil$ clock cycles, so the utilization rate of each PE is $(n-L_s) / \lceil (n-L_s) + 1 \rceil$, which almost equals to one. This result is consistent since the PE is busy once L_i is determined until the time at which the codeword is produced.

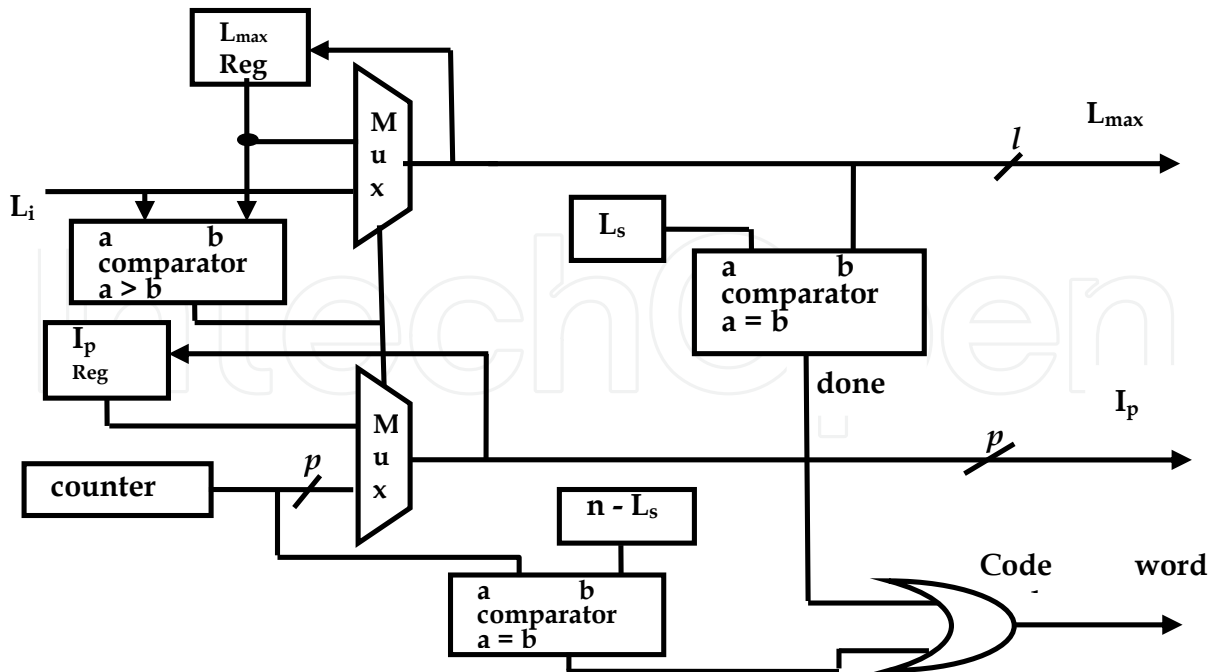


Fig. 20. The match results block (MRB) for design-P.

Parallel compression can be achieved by using an appropriate number of Design- P modules. For example, two modules of Design- P can be used, as shown in Fig. 21. The input sequence of the first module (X_i) is obtained from the first position of Buffer. The input sequence of the second module ($X_{i + [(n - L_s)/2]}$) is obtained starting at $((n - L_s)/2)$ position in the Dictionary. Note that the MRB now needs to determine L_{max} among L_I , L_{II} that are produced at the same time. So, the MRB could be modified. The speed is two times the Design -P array.

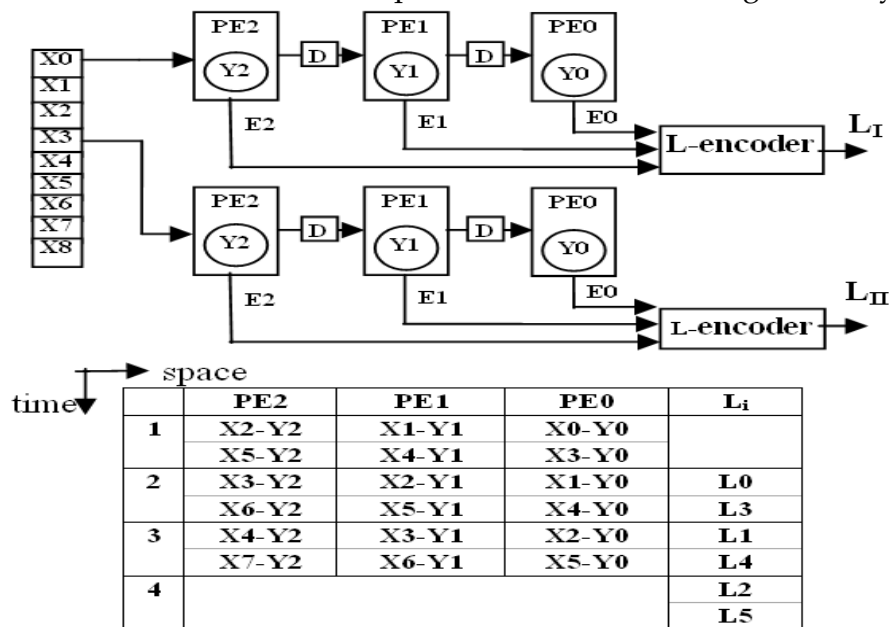


Fig. 21. Parallel compression.

4. The design comparison

The comparison of four designs is given in Table 2. The Design-1 contains $n - L_s$ processing elements, which are always active, resulting in a large area and high- power consumption. It is also slower than the others. Design-P has the maximum utilization rate, minimum latency time (high compression speed) and minimum area for LZ.

	Interleaved Design	Design-P	Design-1	Design-2	Design-3
Number of PEs	L_s	L_s	$n - L_s$	L_s	L_s
Utilization	$(n-L_s)/n$	$(n-L_s)/(n-L_s+1)$	$L_s/2(n-L_s)$	$(n-L_s)/(n+L_s-1)$	$(n-L_s)/n$
Latency	n	$n - L_s + 1$	$2(n - L_s) + 1$	$n + L_s - 1$	n
DFFs per PE	$2w+l+1$	$2w + 1$	$3w+l+2p+1$	$2w+2l+2$	$3w+l+1$
accumulator Per PE	1	---	1	1	1
Counter per MRB	2	1	---	2	2
Multiplexer per MRB	3	2	---	3	3
Multiplexer per buffer	1	---	---	---	---
Equality comparator per MRB	1	2	---	1	1

Table 2. The comparison between Design-P and Design-i.

5. FPGA Implementation

In this section, the proposed architecture for LZ is implemented on FPGA. As shown in Fig. 22, the architecture consists of 3 major components: systolic-array LZ component (SALZC), block RAM, and host controller. The length of window (n) is assumed to be 1K, and the length of look-ahead buffer (L_s) is assumed to be 16. SALZ component contains 16 PEs and implements the most cost effective array architecture (Design -P). Full - custom layout is straight forward since the array is very regular. Only a single cell (PE) was hand - laid out. The other 15 PEs are copies of it. Since the array is systolic, routing also is simplified. Block RAM that is used as the data buffer (Dictionary) is not included in SALZ component. Thereby, the dictionary length can be increased by directly replacing the block RAM with a larger one. The dictionary length is a parameter to cover a broad range of applications, from text compression to lossless image compression.

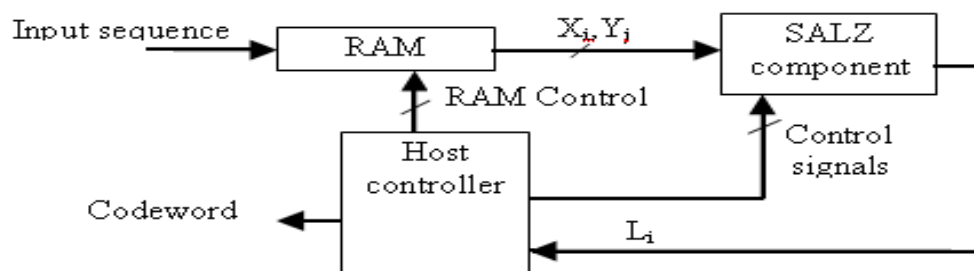


Fig. 22. LZ compression chip

The implementation of the proposed design (Design-P) and interleaved design (Design-i) are carried out using XILINX (Spartan II XC200) FPGA, for $n=1\text{ k}$, $L_s=16$, $w=8$. The implementation results are shown in Table 3. The number of components and their percentage as compared with the available components on the chip are calculated.

The compression rate of a compressor is defined as the number of input bits which can be compressed in one second. The compression rate (R_c) can be estimated as follows:

$$R_c = clk \times [(L_s W) / (n - L_s + 1)] \quad (2)$$

Where clk is the operating frequency. Note that only estimated R_c can be obtained, since it depends on the input data. It is not possible to predict exactly how many words will be compressed (L_s at most) and how many clock cycles will be required ($(n - L_s + 1)$ at most) for every compression step. In the proposed implementation, if the window length (n) is 1k, $L_s=16$, $w=8$, and $clk=105\text{ MHz}$, The R_c is 13 M bit per second.

	Number of Slices		Number of Slice of Flip Flops		Number of 4 input LUTs		Number of BRAMs		Maximum Frequency
Design-P	310	13%	408	8%	419	8%	2	14%	105 MHz
Design-i	471	20%	511	10%	650	13%	2	14%	79 MHz

Table 3. The implementation results of Design-P and Design-i

In order to use the parallel scheme to increase the compression rate, the host controller could be modified and an appropriate number of LZ compressor components could be connected in parallel, as shown in Fig. 23. Note that, the MRB now needs to determine L_{max} among L_I , L_{II} and L_{III} . e.g., ten components could be implemented in one chip of large size to achieve a compression rate about 130 M bits per second. Moreover, by modifying the host controller and including, e.g., dictionaries, the proposed design can be used for other string-matching based LZ algorithms, such as LZ78 and LZW. The Design-P is flexible.

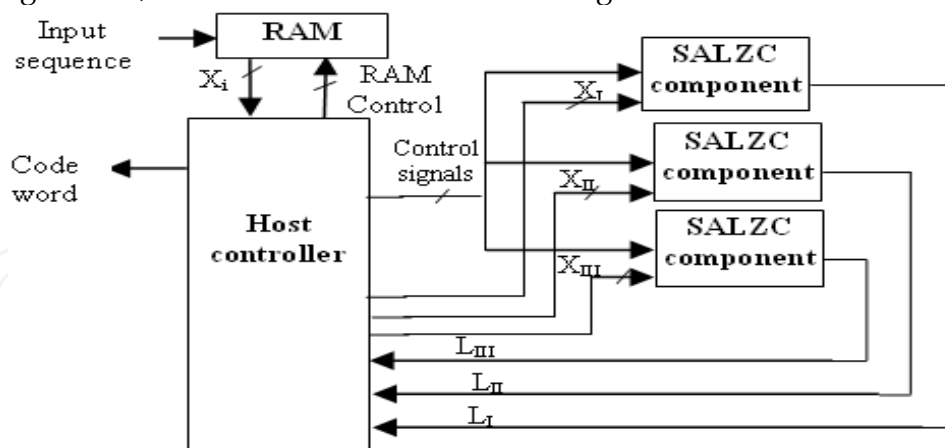


Fig. 23. Multiple SALZC system.

6. Conclusions

In this chapter, a parallel algorithm for LZ based data compression is described by transforming a data-dependent algorithm to a data-independent regular algorithm. To further improve the latency, a control variable to indicate early completion is used. The

proposed implementation is area and speed efficient. The compression rate is increased by more than 40% and the design area is decreased by more than 30%. The design can be integrated into real - time systems so that data can be compressed and decompressed on - the - fly.

7. References

- Abd El Ghany, M. A.; Salama, A. E. & Khalil, A. H. "Design and Implementation of FPGA-based Systolic Array for LZ Data Compression" in *Proc. IEEE International Symposium on Circuits and Systems ISCAS'07*, May 2007
- Arias, M.; Cumplido, R. & Feregrino C., (2004) "On the Design and Implementation of an FPGA-based Lossless Data Compressor.", *ReConFig'04*, Colima, Mexico, Sep., 2004
- Bodden, E.; Clasen, M.; Kneis, J. & Schluter, R., (2004) "Arithmetic coding revealed" pp. 8-57, May 2004
- Chang, J.; Jih, H. J & Liu, J. W., (1994) "A lossless data compression processor," in *Proc. 4th VLSI Design/CAD Workshop*, Aug. 1994, pp.134-137
- Deorowicz, S., (2003) "Universal lossless data compression algorithms" *Dissertation, Silesian University*, pp 92-119, 2003
- Gallager, R. (1978)"Variations on a theme by Huffman," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 668-674, 1978
- Golomb, S., (1966) "Run-length encoding," *IEEE Trans. Inform. Theory*, Vol. IT-12, pp. 399-401, July 1966
- Howard, P. G. & Vetter, J. S., (1992) "Practical Implementations of Arithmetic Coding" *Brown University, Department of Computer Science, Technical Report No. (CS-91-45)*, April 1992
- Huffman, D. A. (1952) "A method for the construction of minimum-redundancy codes" *Proc. IRE*, Vol. 40, pp. 1098-1101, September, 1952.
- Hwang, S. -A & Wu, C. -W., (2001) "Unified VLSI Systolic Array Design for LZ Data compression," *IEEE Trans. VLSI Systems*, vol. 9, pp. 489- 499, August 2001
- Jung, B. & Burleson, W. P., (1998) "Efficient VLSI for Lempel-Ziv compression in wireless data communication networks," *IEEE Trans. VLSI Syst.* Vol. 6, pp. 475-483, Sept. 1998
- Lin, K. -J. & Wu, C.-W., (2000) "A low-power CAM design for LZ data compressor," *IEEE Transactions on Computers*, Vol.49, No.10, pp. 1139-1145, Oct. 2000
- Ranganathan, N. & Henriques, S., (1993) "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
- Park, H. & Prasanna, V.K. (1993) "Area efficient VLSI architectures for Huffman coding", *IEEE Trans. Analog and Digital Signal Processing*, Vol.40, pp.568-575, Sep.1993
- Said, A., (2003), "Fast Arithmetic Coding," in *Lossless Compression Handbook*, 2003
- Said, A., (2004) " Introduction to Arithmetic coding - Theory and Practice" *Imaging Systems Laboratory, HP Laboratories Palo Alto*, April 2004
- Salomon, D., (2004) "Data Compression" *The complete reference*, Computer Science Department, California State University, Northridge, pp. 22-205, 2004
- Sandoval, M. M. & Uribe, C. F. (2005) "A hardware Architecture for Elliptic Curve Cryptography and Lossless Data Compression," *IEEE Computer Society*, May 2005

- Welch, T., (1984) "A technique for high-performance data compression," *IEEE computer*, Vol. 17, pp. 8-19, 1984
- Ziv, J. & Lempel, A., (1977) "A universal algorithm for sequential data compression," *IEEE Trans. Inform. Theory*, Vol. IT-23, pp. 337-343, 1977
- Ziv, J. & Lempel, A., (1978) "Compression of individual sequences via variable rate coding," *IEEE Trans. Inform. Theory*, Vol. IT-24, pp. 530-536, 1978

IntechOpen

IntechOpen



Data Storage

Edited by Florin Balasa

ISBN 978-953-307-063-6

Hard cover, 226 pages

Publisher InTech

Published online 01, April, 2010

Published in print edition April, 2010

The book presents several advances in different research areas related to data storage, from the design of a hierarchical memory subsystem in embedded signal processing systems for data-intensive applications, through data representation in flash memories, data recording and retrieval in conventional optical data storage systems and the more recent holographic systems, to applications in medicine requiring massive image databases.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Mohamed A. Abd El Ghany, Magdy A. El-Moursy and Aly E. Salama (2010). Design and Implementation of FPGA-based Systolic Array for LZ Data Compression, Data Storage, Florin Balasa (Ed.), ISBN: 978-953-307-063-6, InTech, Available from: <http://www.intechopen.com/books/data-storage/design-and-implementation-of-fpga-based-systolic-array-for-lz-data-compression>

INTECH
open science | open minds

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2010 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the [Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License](#), which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen