International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 5– No.4, March 2013 – www.ijais.org

Hardware Implementation of LZMA Data Compression Algorithm

E.Jebamalar Leavline
Department of ECE Bharathidasan Institute of Technology, Anna University Tiruchirappalli – 620 024 Tamilnadu, India

D.Asir Antony Gnana Singh
Department of CSE Bharathidasan Institute of Technology, Anna University Tiruchirappalli – 620 024 Tamilnadu, India

ABSTRACT
Data transmission, storage and processing are the integral parts of today's information systems. Transmission and storage of huge volume of data is a critical task in spite of the advancements in the integrated circuit technology and communication. In order to store and transmit such a data as it is, requires larger memory and increased bandwidth utilization. This in turn increases the hardware and transmission cost. Hence, before storage or transmission the size of data has to be reduced without affecting the information content of the data. Among the various encoding algorithms, the Lempel Ziv Marcov chain Algorithm (LZMA) algorithm which is used in 7zip was proved to be effective in unknown byte stream compression for reliable lossless data compression. However the encoding speed of software based coder is slow compared to the arrival time of real time data. Hence hardware implementation is needed since number of instructions processed per unit time depends directly on system clock. The aim of this work is to implement the LZMA algorithm on SPARTAN 3E FPGA to design hardware encoder/decoder with reduces circuit size and cost of storage.

General Terms
Data Compression, VLSI

Keywords
Data compression, encoding, decoding, unknown byte stream, LZMA algorithm, compression ratio, FPGA

1. INTRODUCTION
Reducing the size of the data so as to reduce the storage space and transmission time is essential in information and communication systems. The demand for on-line storage is growing day by day due to the drastic improvements in the information technology during the past decade. The cost of communication channels such as cellular telephone and satellite television transmission is another driving force for increase in the use of data compression in the field of communication. Thanks to the compression algorithms, these facilities are available at affordable cost. Due to the explosion of the world-wide web network, too much data have to be moved at a time with the available channel capacity. With compression, it is possible to reduce the amount of data transmitted, without sacrificing the amount of information conveyed [1].

Data-compression techniques fall under two broad categories namely lossy and lossless compression. In spite of achieving higher compression rates, lossy data compression schemes allow certain loss of accuracy of the data. Lossy compression is effective in applications such as image and voice coding where loss of accuracy is tolerable up to certain levels. Until recently, lossy compression has been primarily implemented using dedicated hardware. In the past few years, powerful lossy-compression programs have been moved to desktop CPUs, but even so the field is still dominated by hardware implementations [2]. On the other hand, the loss less compression schemes generate an exact duplicate of the input data stream after a compress/expand cycle. Storing database records, spreadsheets, or word processing files uses loss less compression the loss of even a single bit could be calamitous. Lossless data compression is generally implemented using one of two different types namely statistical modeling or dictionary-based compression.

Statistical modeling reads in and encodes a single symbol at a time using the probability of that character's appearance. Dictionary based modeling uses a single code to replace strings of symbols. In dictionary-based modeling, the coding problem is significantly reduced, leaving the model supremely important [2, 3]. Also, Adaptive models have been proposed [4-8] in which, data does not have to be scanned once before coding in order to generate statistics. Instead, the statistics are continually modified as new characters are read in and coded. One problem with adaptive models is that they start knowing essentially nothing about the data. Most adaptive algorithms tend to adjust quickly to the data stream and will begin turning in respectable compression ratios after only a few thousand bytes. Yet, the adaptive algorithms are able to adapt themselves to local conditions [2].

A dictionary-based compression scheme reads in input data and looks for groups of symbols that appear in a dictionary. A pointer or index into the dictionary can be output instead of the code for the symbol if a string match is found. The compression ratio depends on the amount of match occurs. But dictionary based methods introduce overhead because the dictionary needs to be transmitted along with the text [2].

Huffman coding was the choice for data compression for many decades [8]. However in recent years the Lempel Ziv algorithm has taken over the Huffman algorithm almost completely, which belongs to the class of universal source coding algorithms. This algorithm is a variable to fixed length coding scheme. This means any sequence of source output is uniquely parsed into phrases of varying length and these phrases are encoded using code words of equal length [9].

Lempel Ziv Marcov chain Algorithm is a variant of this Lempel Ziv Algorithm. LZMA is found to be highly efficient for unknown data stream compared with Huffman algorithm.

Depending on the processor speed the compressing and decompressing speed differs. LZMA is best suited for compressing data streams generated at 10-20 Mbps in Real-time environment.

In this paper, we propose the hardware architecture for the LZMA algorithm. The proposed architecture was implemented in Spartan 3E FPGA family under Xilinx environment. The rest of this paper is organizes as follows. In section 2, the related background work is presented. In section 3, the LZMA is explained. The proposed hardware architecture and experimental setup is detailed in section 4. The synthesis results are presented in section 5 and the paper is concluded in section 6.

2. BACKGROUNDS

Most of the general-compression schemes in the data compression literature use statistical modeling. But the in two algorithms proposed in [10, 11] by Jacob Ziv and Abraham Lempel drew attention towards dictionary-based methods to achieve better compression ratios [2].

2.1 LZ78

The first simple compression algorithm described by Ziv and Lempel is commonly referred to as LZ77 [10]. The dictionary consists of all the strings in a window into the previously read input stream. When new groups of symbols are being read in, the algorithm searches for matches with strings found in the previous data already read in. Then the matches are encoded as pointers and sent as the output. LZ77 encoding is simple and faster. The variants of LZ77 are used in popular programs such as PKZIP and LHarc [2]. LZ77 is effective only when the input data is highly redundant or repetitive.

2.2 LZ78

LZ78 [11] builds its dictionary with of all of the previously seen symbols in the input text rather than having a limited-size window into the preceding text. But instead of having access to all the symbol strings in the preceding text, a dictionary of strings is built a single character at a time. This incremental procedure works very well at isolating frequently used strings and adding them to the table. Unlike LZ77 methods, strings in LZ78 can be extremely long, which allows for high-compression ratios [2].

2.3 LZMA

The Lempel–Ziv–Markov chain algorithm (LZMA) is an algorithm for lossless data compression and it was first used in the 7z format of the 7-Zip archiver [12]. The first idea of LZMA implementation was found in [13]. There LZMA is proposed as a subordinate block in image compression. Further, FPGA implementation issues of LZ77 are discussed in [15, 16]. Being inspired by [14-17], FPGA implementation of LZMA data compression algorithm has been developed using SPARTAN 3E FPGA.

3. LZMA DATA COMPRESSION ALGORITHM

The dictionary based LZ77 algorithm codes byte sequences from former contents instead of the original data. In general only one coding scheme exists; all data will be coded in the same form:

- Address to already coded contents
- Sequence length
- First deviating symbol

If no identical byte sequence is available from former contents, the address is 0, the sequence length is 0 and the new symbol will be coded. LZ77 [10] also uses a dynamic dictionary to compress unknown data with the use of sliding window algorithm. LZMA uses a Delta Filter and Range Encoder in addition to the original LZ77 algorithm as shown in Figure 1.

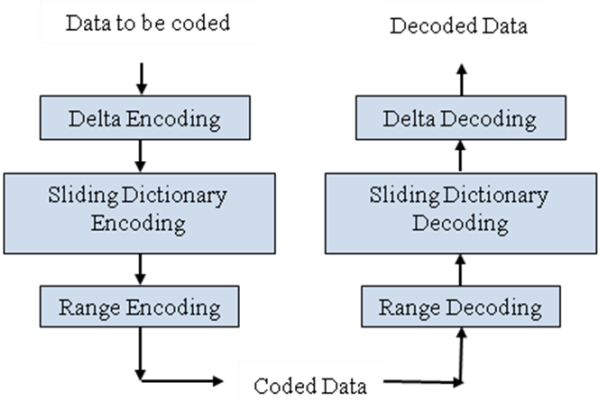

Figure 1. LZMA Coding Scheme

3.1 Delta Encoding and Decoding

The Delta Filter shapes the input data stream for effective compression by the sliding window. It stores or transmits data in the form of differences between sequential data rather than complete files. The output of the first byte delta encoding is the data stream itself. The subsequent bytes are stored as the difference between the current and its previous byte. For a continuously varying real time data, delta encoding makes the sliding dictionary more efficient [18, 19].

Example:

Sample input sequence	: 2,3,4,6,7,9,8,7,5,3,4
Output sequence encoded	: 2, 1, 1, 2, 1, 2,-1,-1,-2,-2, 1
Number of symbols in input	: 8
Number of symbols in output	: 4

3.2 Sliding Dictionary Algorithm

There are two types of dictionaries namely static dictionary and adaptive dictionary. In static dictionaries the entries are predefined and constant according to the application of the text whereas in adaptive dictionaries, the entries are taken from the text itself and created on-the-fly. A search buffer is used as dictionary as in Figure 2, and the sizes of these buffers depend on the parameters of the implementation. Patterns in text are assumed to occur within range of the search buffer. The offset and length are encoded separately, and a bit-mask is also encoded. Use of suitable data structure for the buffers will reduce the search time for longest matches. Sliding Dictionary encoding is more difficult than decoding as it needs to find the longest match [17]. The basic steps in sliding dictionary algorithm are shown in Figure 3.

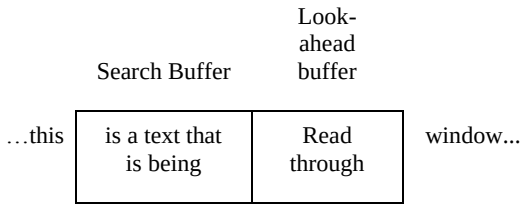

Figure 2. Sliding Dictionary Concept

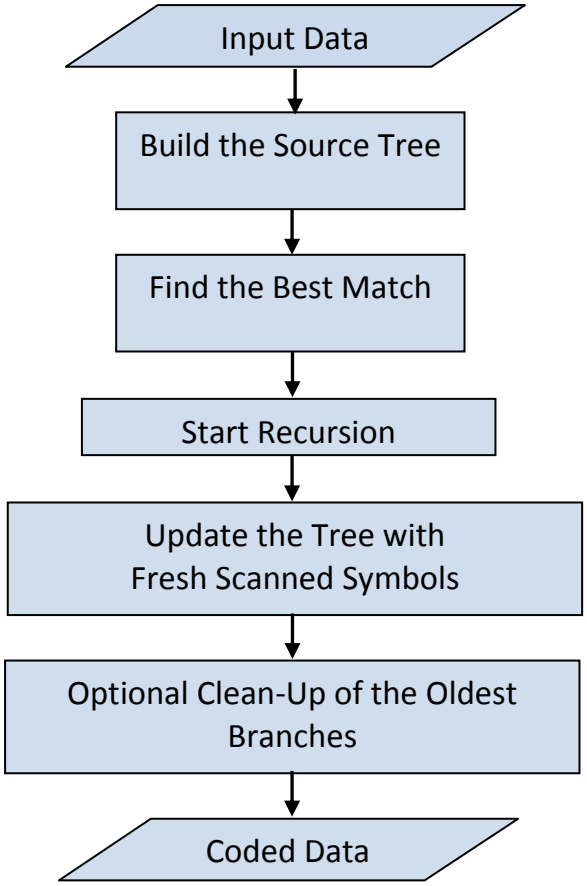

Figure 3: Sliding Dictionary Algorithm

3.3 Range Coder

Range encoder encodes all the symbols of the message into a single number to achieve greater compression ratios. It efficiently deals with probabilities that are not exact powers of two. The range encoder uses the following steps.

- Given a large-enough range of integers, and probability estimation for the symbols.
- Divide the initial range into sub-ranges whose sizes are proportional to the probability of the symbol they represent.
- Encode each symbol of the message by reducing the current range down to just that sub-range which corresponds to the next symbol to be encoded.

The decoder must have the same probability estimation the encoder used, which can either be sent in advance, derived from already transferred data [20].

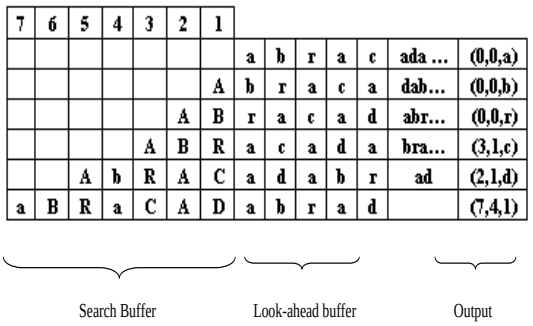

Figure 4: Sample Sliding Dictionary Coding. 12 characters compressed into 6 tuples. (String to be encoded: abracadabrad, Output tuple: (offset, length, symbol)

Input		7	6	5	4	3	2	1
(0,0,a)								a
(0,0,b)							a	b
(0,0,r)						a	b	r
(3,1,c)				a	B	r	a	c
(2,1,d)		A	B	r	A	c	a	d
(7,4,d)	abrac	A	D	a	B	r	a	d

Figure 5 Sample sliding Dictionary Decoding

4. FPGA IMPLEMENTATION OF LZMA DATA COMPRESSION

The proposed architecture for hardware implementation of LZMA is shown in Figure 6.

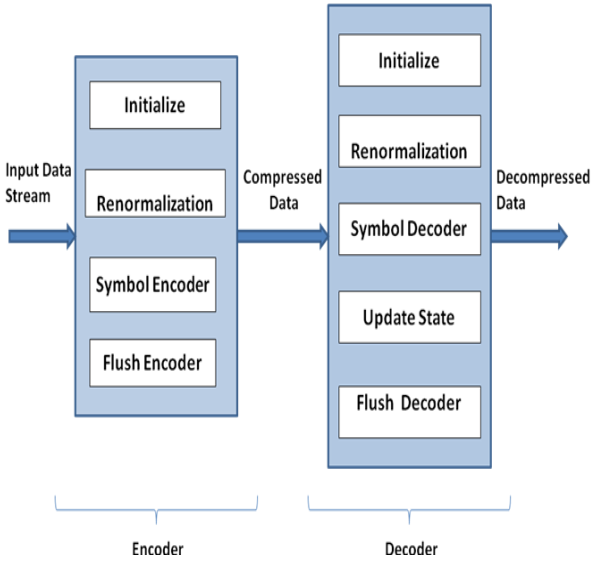

Figure 6. Proposed architecture for hardware implementation of LZMA

The encoder is made of four routines: init, renormalize, encode a symbol and flush encoder. Init must be called before starting to encode symbols, both encode and renormalize code the symbols, and flushing is done when you have encoded all the symbols. The decoding has five routines: init, renormalize, decode symbol, update state and flush. Update

state must be called after knowing the symbol's range, so you can extract it from the number. These routines have been developed using VHDL [21, 22].

5. EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Experimental Setup

The experimental set up for the proposed hardware architecture for LZMA is shown in Figure 7. The proposed architecture is tested on Spartan 3E FPGA with Xilinx ISE 11.1i. The real data input is given from PC through serial communication to the FPGA development board. The FPGA is programmed using USB programmer. The Compressed and decompressed data are displayed with LEDs.

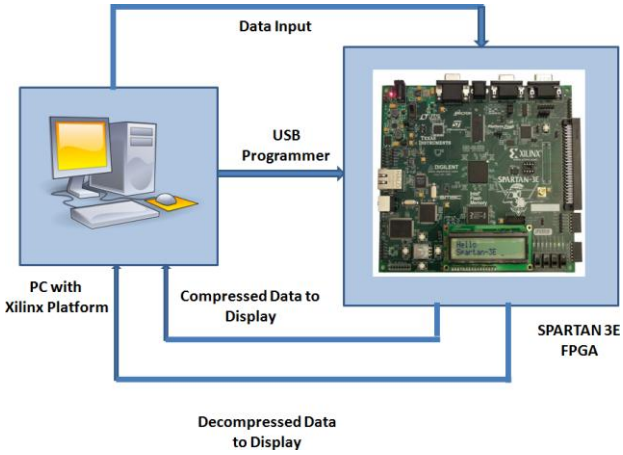

Figure 7. Experimental set up for the proposed hardware architecture for LZMA

5.2 Results and Discussion

The compression ratio for various input data sizes are shown in Table 1. The HDL synthesis Summary and the device utilization details for both encoder and decoder are given in Table 2. The LZMA is consistent over a range of input data sizes and the average compression ratio achieved is 57.135 which is comparable with the compression ratio achieved in 7Zip. The compression ratio (CR) is calculated as follows.

CR = (Output Data Size/Input Data Size) X 100%

CR Achieved in 7ZIP = (138 bytes/260 bytes) *100 = 53.1 %

CR Achieved Practically = (37 bytes/64 bytes) *100= 57.8 %

Table 1: Compression Ratio achieved for various data sizes with LZMA

S.No	Size of Input Data (Byte)	Size of Compressed Data (Byte)	Compression Ratio (in %)
1	32	18	56.275
2	64	37	57.813
3	128	73	57.031
4	256	146	57.422
Average Compression Ratio			57.135

6. CONCLUSIONS

The simple, portable and efficient LZMA compression algorithm is implemented using HDL that provides an excellent platform for Real-time compression applications. The proposed hardware architecture is tested with Spartan 3E FPGA device for various input data sizes and the compression ratio was calculated. Encoding and decoding processes are fully functional at 50 MHz and Compression Ratio is comparable with that of real compression ratio. Further this architecture can be extended to application specific integrated circuits so as to design a specific hardware chip for the LZMA compression algorithm.

7. REFERENCES

[1] Alistair Moffat, Andrew Turpin klwer. Compression and Coding Algorithms. Academic Publishers Massachusetts (2002).

[2] Mark Nelson, Jean-loup Gailly, “The Data compression Book”,2nd Edition, M&T Books, New York, NY (1995).

[3] S. Shanmugasundaram and R.Lourdusamy. A Comparative Study Of Text Compression Algorithms. International Journal of Wisdom Based Computing. 1, 3(2011)

[4] Gennady Pekhimenko Vivek Seshadri Onur Mutlu, “Base-Delta-Immediate Compression: A Practical Data Compression Mechanism for On-Chip Caches” SAFARI Technical Report No. 2012-001 (June 19, 2012)

[5] Fout, Nathaniel, Ma and Kwan-Liu, An Adaptive Prediction-Based Approach to Lossless Compression of Floating-Point Volume Data. IEEE Transactions on Visualization and Computer Graphics. 18 , 12 (2012) 2295- 2304.

[6] M.Smith, I.Posner and Paul Newman. Adaptive compression for 3D laser data” The International Journal of Robotics Research. 30,7, (2011) 914–935.

[7] Sacaleanu, D.I ,Stoian, R, and Ofrim, D.M, An adaptive Huffman algorithm for data compression in wireless sensor networks. 10th International Symposium on Signals, Circuits and Systems (ISSCS), 2011 Page(s): 1-4

[8] Satpreet Singh and Harmandeep Singh. Improved Adaptive Huffman Compression Algorithm. International Journal of Computers & Technology.1 ,1(2011) 16-22

[9] John G. Proakis, Masoud Salehi. Fundamentals of Communication Systems. Pearson Education (2006).

[10] J.Ziv and A.Lampel. A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory. 23,3(1997) 337–343.

[11] J.Ziv and A.Lampel. Compression of Individual Sequences via Variable-Rate Coding. IEEE Transactions on Information Theory. 24, 5(1978) 530–536.

[12] Igor Pavlov, “7z format”, http://www.7-zip.org/7z.html

[13] Ranganathan, N and Henriques, S. High-speed VLSI designs for Lempel-Ziv-based data compression. Circuits and Systems II: Analog and Digital Signal Processing, 40,2,(1993) 96–106.

[14] Zongjie Tu and Shiyong Zhang. A Novel Implementation of JPEG 2000 Lossless Coding Based on LZMA.

Proceedings of the Sixth IEEE International Conference Computer and Information Technology, (2006).

[15] Mohamed A. Abd El Ghany, Magdy A. El-Moursy and Aly E. Salama. Design and Implementation of FPGA-based Systolic Array for LZ Data Compression. Proceedings of IEEE International Symposium on Circuits and Systems, (2007).

[16] S Rigler, W Bishop and A Kennings. FPGA-Based Lossless Data Compression using Huffman and LZ77 Algorithms. Proceedings of Canadian Conference on Electrical and Computer Engineering, (2007).

[17] Ashwini M. Deshpande, Mangesh S. Deshpande and Devendra N. Kayatanavar. FPGA Implementation of AES encryption and decryption. Proceedings of International Conference on Control, Automation, Communication And Energy Conservation (2009).

[18] David Salomon. Data Compression: The Complete Reference. Second Edition, Springer New York, Inc. (2000).

[19] Simon Haykin. Communication Systems. 4th Edition, John Wiley and Sons,(2001).

[20] Arturo Campos. Range encoder.URL: http://www.arturocampos.com/ac_range.html

[21] Wayne Wolf. FPGA-Based System Design, Pearson Education, (2004).

[22] Jayaram Bhasker.. A VHDL Primer. Third Edition, Prentice Hall P T R, (1999).

Table 2 :HDL Synthesis and Device Utilization Summary for LZMA Encoder and Decoder

Macro Statistics	Encoder		Decoder	
	Details	**Quantity**	**Details**	**Quantity**
# ROMs	16x8-bit ROM	2	16x14-bit ROM	1
# Adders/ Subtractors	32-bit adder - 7 32-bit Subtractor - 17	24	32-bit Subtractor	2
# Counters	32-bit down counter - 2 32-bit up counter- 2	4	32-bit down counter-1 32-bit up counter -2	3
# Registers	1-bit register - 127 14-bit register - 16 32-bit register - 4 7-bit register - 1	148	1-bit register -5 14-bit register - 1 3-bit register -2 32-bit register - 2 8-bit register - 17	27
# Comparators	16-bit comparator equal - 6 24-bit comparator equal - 5 32-bit comparator equal - 4 32-bit comparator great equal - 3 32-bit comparator greater - 2 32-bit comparator less - 2 32-bit comparator less equal - 3 40-bit comparator equal - 3 48-bit comparator equal - 2 56-bit comparator equal - 1 8-bit comparator equal - 7	38	32-bit comparator great equal - 2	2
# Multiplexers	120-bit 4-to-1 multiplexer -1 14-bit 16-to-1 multiplexer - 1	2	32-bit 4-to-1 multiplexer - 5 32-bit 8-to-1 multiplexer - 2 8-bit 16-to-1 multiplexer - 1 8-bit 4-to-1 multiplexer -32 8-bit 8-to-1 multiplexer -48	88
Timing Summary Speed Grade: -4	Minimum period Minimum input arrival time before clock Maximum output required time after clock Maximum combinational path delay Maximum Frequency	17.342ns 4.335ns 4.283ns No path found 57.66MHz	Minimum period Minimum input arrival time before clock Maximum output required time after clock Maximum combinational path delay Maximum Frequency	18.542ns 4.335ns 4.283ns No path found 53.93MHz

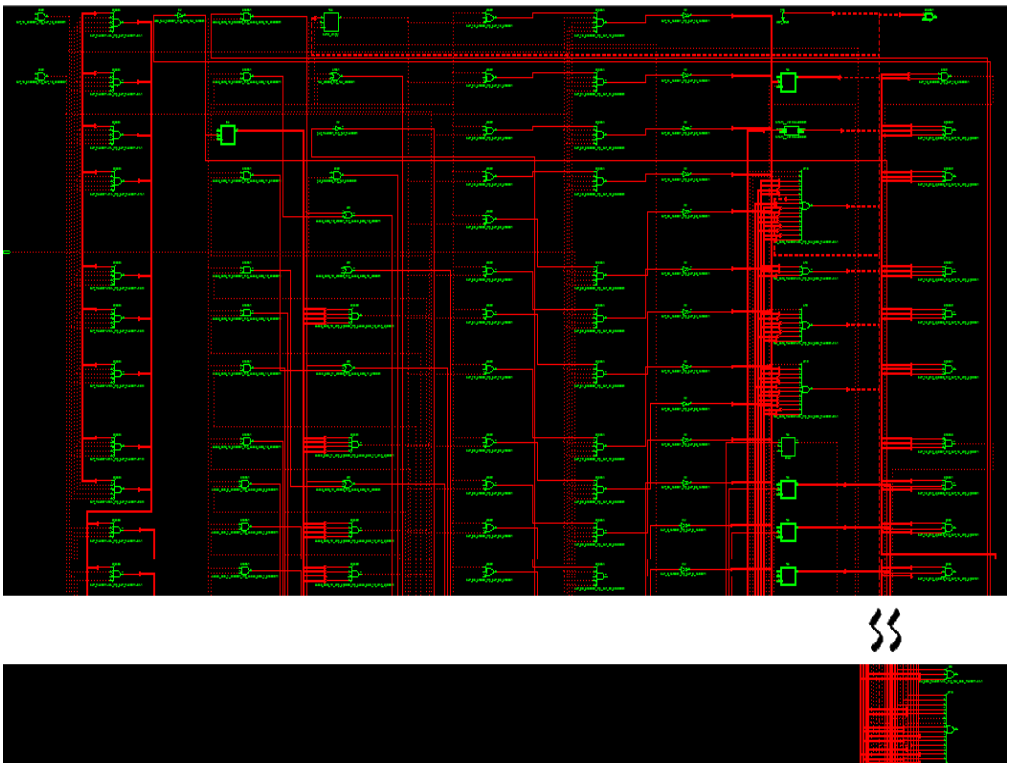

Figure 8: RTL Schematic View of LZMA Encoder

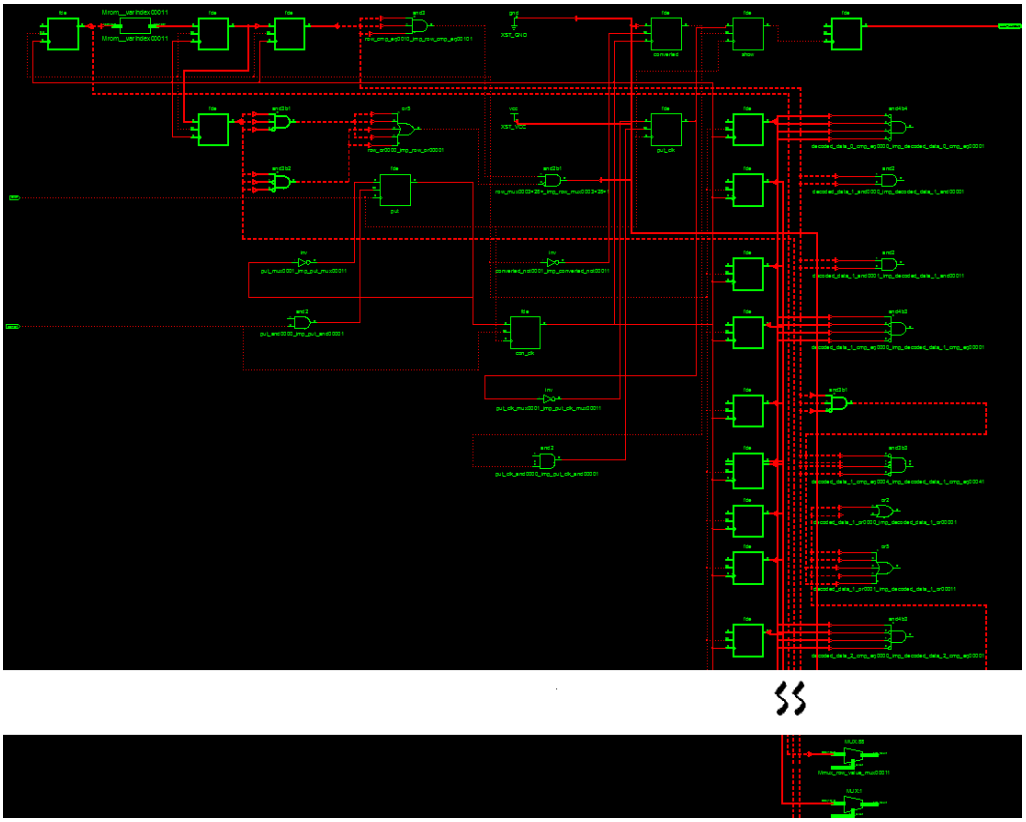

Figure 9. RTL Schematic View of LZMA Decoder