

Prototyping of Boolean Function Classification Schemes for Lossless Data Compression using FPGA

M. I. Reaz, F. Mohd-Yasin, M. S. Sulaiman, K. T. Tho, K. H. Yeow

Faculty of Engineering
Multimedia University
63100 Cyberjaya, Selangor
Malaysia
mamun.reaz@mmu.edu.my

Abstract – In this paper, we present the realization of Boolean function classification schemes on Altera FLEX10K FPGA device for lossless data compression. The compression algorithm is performed by incorporating Boolean function classification into Huffman coding. This allows for more efficient compression because the data has been categorized and simplified before the encoding is done. The design is followed by the timing analysis and circuit synthesis for the validation, functionality and performance of the designated circuit which supports the practicality, advantages and effectiveness of the proposed hardware realization for the applications. The average compression ratio is 25% to 37.5% from numerous testing with various text inputs with a maximum clock frequency of 27.9 MHz.

I. INTRODUCTION

The term Data Compression refers to the process of reducing the amount of the required data representing a given quantity of information. Data compression is increasingly more and more important in the development of computer and data communications technology. Various data compression technologies have been developed since the past few decades, using different algorithms for different applications. Some of the data compression techniques are Null Suppression, Run-Length Encoding, Huffman coding, Arithmetic coding, Lempel-Ziv-Welch coding, Discrete Cosine Transform, Joint Photographic Expert Group and Boolean Compression algorithm [1].

Boolean function classification technique has been traditionally designed for digital circuit applications. The main feature of this technique is due to the fact that the functions belonging to some classes may be implemented more efficiently than the general sum of product implementation. Boolean function classification plays an important role in the field like technology mapping for digital circuit design, function mapping for minimization and the development of universal logic modules [2].

The FPGA offers a potential alternative to speed up the hardware realization [3, 4]. From the perspective of computer-aided design, FPGA comes with the merits of lower cost, higher density, and shorter design cycle [5]. It comprises a wide variety of building blocks. Each block consists of programmable look-up table and storage registers, where interconnections among these blocks programmed through the hardware description language [6, 7]. This programmability and simplicity of FPGA made it favourable for prototyping digital system.

In this work we proposed the framework of FPGA-based hardware prototyping of Boolean compression algorithm by incorporating Boolean function classification into

Huffman encoding [8]. By performing the Boolean function classification, the binary data is grouped into their classes and through Huffman encoding, the compression is done in a more efficient way because the data has been categorized and simplified before the encoding is done. The result has higher compression ratio. We had studied the existing Boolean classification schemes that are suitable for using in data compression. We also studied the new and alternative classification schemes that can be implemented in the algorithm. After finalizing the algorithm, the VHDL is selected as the hardware description language to realize the scheme as VHDL design environment permits extensive simulation for verification of the algorithm [9]. In the computation of method, the problem is first divided into small pieces, each can be seen as a submodule in VHDL. Following the software verification of each submodule, the synthesis is then activated that performs the translations of hardware description language code into an equivalent netlist of digital cells. The synthesis helps integrate the design work and provides a higher feasibility to explore a far wider range of architectural alternative. In this study, to validate the effectiveness of the method, various text inputs has been used. The method provides a systematic approach for hardware realization, facilitating the rapid prototyping of Boolean Function Classification Schemes for Lossless Data Compression.

II. ALGORITHM DEVELOPMENT

In Boolean function classification algorithm, 16-bit of data bits are extracted from data input. The first 16-bit Boolean block is then used to generate fractal. The fractal is then used to match with the fractals for all other bits in the data source. If there are identical matches between the first fractal with the fractals in the data source, the counter for the fractal is incremented. After all the data bits are matched with the first fractal, the first unmatched 16-bit data will be used as the next fractal, and to be matched with the remaining bits of data. Again, when there are matches between the second fractal with the data source, the counter for the second fractal will be incremented. The same algorithm continues until there is no more unmatched data source. Figure 1 shows the flow chart for the algorithm as explained.

When there is no more unmatched data source with the fractal, the algorithm continues with the Huffman encoding to compress the classified data source. From the counters for each of the fractals, the frequency of occurrence for each fractal can be determined. Based on Huffman algorithm [10], the data bits with higher number of

occurrence are to be encoded with shorter codes, whereas the data bits with lower number of occurrence are to be encoded with longer codes. The same concept applies in this compression. For the fractal with higher number of occurrence, the fractal is encoded with a short code and saved with a header to enable the data to be retrieved or decompressed. This is to be explained in the decompression section. Each of the fractals is encoded using Huffman encoding and this completes the Boolean compression. Figure 2 shows the flow chart for Huffman encoding algorithm.

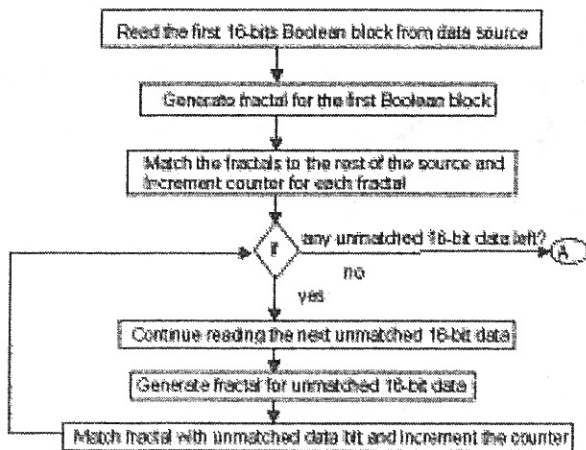


Figure 1: Boolean function classification algorithm

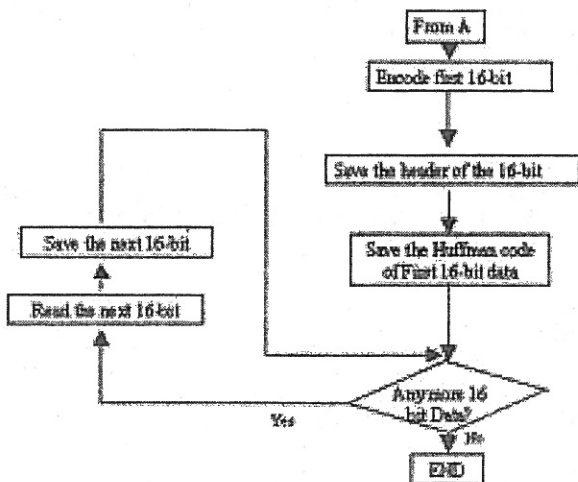


Figure 2: Huffman encoding algorithm

The decompression algorithm involves re-building the Huffman tree from a stored frequency table in the header of the compressed file, and converting or decompressing the bit streams of variable encode length into characters. Beginning at the root node based on the header stored in the compressed data, and depending on the value of the bit, the right or left branch of the Huffman tree is taken, and then return to read another bit for the next branch. When the node selected is a leaf, which means that it has no right and left child nodes, its character value is written to the decompressed file and go back to the root node for the next bit. This algorithm is continued till all the compressed bits of variable encode length are decompressed. Figure 3 shows the decompression algorithm.

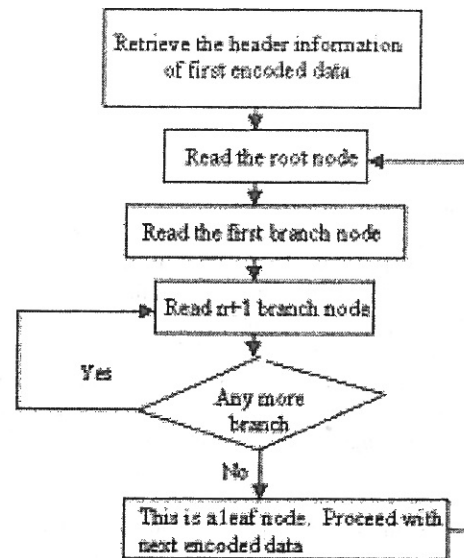


Figure 3: Decompression algorithm

III. VHDL IMPLEMENTATION

The design started with two blocks, compressor and decompressor. The input of the compressor is integer stream, where 8-bit binary data is read in as input for the compression. The data input is classified and compressed into another form. The classified data is shown in its respective class, which is represented by the 3-bit class output pin. The output data is a variable-length Huffman code, which is the compressed data. This compressed data is available from output pins. Each 8-bit data from the input will be compressed according to the look-up table built. The input data into the decompressor chip is a bit stream with length of 1-bit to 6-bit, and the class of 3-bit. The output of the decompression is an 8-bit data.

VHDL implementation had been performed following the algorithm discussed in section 2. The implementation is started by building a statistical lookup table for all the possible text inputs, ranging from a to z for small case, A to Z for uppercase and some special characters like semicolon, each with specific class. An ASCII-to-binary program written in javascript is used to convert the text input into binary bits. The ASCII table for each of these characters is also referred. This ASCII table is used as a reference to specify the bit patterns for each of the input character. The bit patterns for each of the input character are important in the formation of the lookup table consisting of all possible text inputs. Since the compressor can recognize 71 characters, these characters are classified into their respective classes. After determining the size of the class, the next step is to determine the characteristic of the class. The equations to determine the class using definition for direct symmetric Boolean function is

$$f(z) = f(x, y) = f(y, x) \quad (1)$$

where x is initialized to 001, and $y = m-1$, where m is the length of the encoded bits. x is initialized to 001 to represent the class for encoded data with length of 1 bit.

Thus,

$$f(z) = f((001)_2, (m-1)_{10}) = f((m-1)_{10}, (001)_2) \quad (2)$$

for the function to be a direct symmetric function. $f(z)$ represents function for classification of the Boolean function derived using definition for direct symmetric Boolean function.

To perform the compression, the input data will first match with all the predefined inputs in the lookup table. When the input data is matched successfully, the length of the encoded output will be shown, and the output will be displayed. The output is of variable length. Thus, careful declaration of the vector size is needed to ensure correct compilation and simulation. The possible length of the encoded output ranges from 1 bit to 6 bits, which is lesser than the uncompressed form of data for each character, which is 8 bits.

For the decompression of the encoded data, the class of the encoded data and the compressed data are used as the inputs to run the decompression program. Again, the compressed data may be of different length, which varies from 1-bit to 6-bit. The inserted class and compressed data will then be matched with all the predefined data in the lookup table. When the inserted class and compressed data matches with the predefined data in the lookup table, then the output, which is the original data before compression can be obtained.

IV. SIMULATION

The system was coded in IEEE-compliant VHDL and compiled and simulated using the MAX+PLUS2. This provides an opportunity to detect and correct errors early in the design process [11, 12]. Both compression and decompression modules was designed and tested in isolation before being incorporated into the higher levels of the design.

Both compression and decompression modules were first simulated individually to verify their functionalities. Each module was fed a fix inputs and the correct outputs were observed. After the successful individual simulations were performed, the modules were integrated together. This enables detailed simulation at the top level.

The results are generated using waveform editor. The clock signal and outputs are shown in the timing diagram. One example of the simulation was shown in section 4.1 and 4.2 using the 72-bit input binary data.

A. Compression Simulation

In Figure 4, the generated data inputs are 00110101, 00110110, 00110111, 01100011, 00111001, 01101011, 01101100, 01110000, and 01110111. The encoded outputs are 0000, 0001, 0010, 1001, 0100, 1000, 1011, 0111, and 1110. The outputs are exactly the same as the output in the lookup table. This yields that the compression is performed correctly. In this simulation, the compression ratio is 50%. The best compression ratio for this algorithm is 87.5%, which is the case when all the inputs are having encoded output of 1 bit. However, this rarely occurs since text inputs usually consist of various different characters, which have their respective class and output bits as defined

in the lookup table. The average compression ratio is 25% to 37.5% from numerous testing with various text inputs. This is verified from the lookup table as well, since class 101 and class 110 have most input texts, and their encoded bits range from 5 bits to 6 bits.

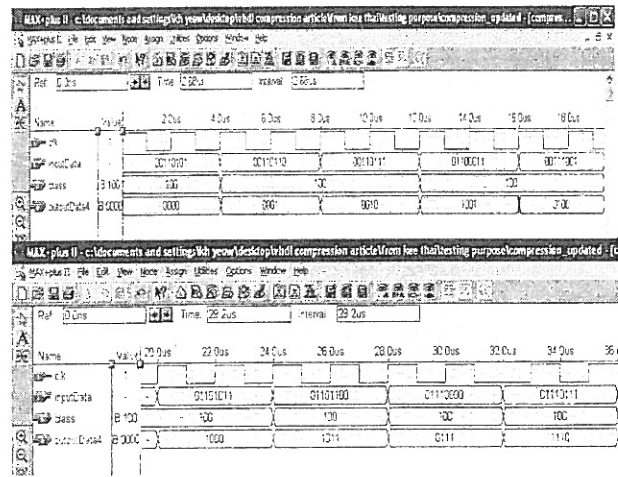


Figure 4: Simulation results for compression

B. Decompression Simulation

In Figure 5, the input is the compressed data and class. The compressed data are 0000, 0001, 0010, 1001, 0100, 1000, 1011, 0111 and 1110. The outputs are 00110101, 00110110, 00110111, 01100011, 00111001, 01101011, 01101100, 01110000, and 01110111. The outputs are exactly the original inputs which verify the correct functionalities of the algorithm.

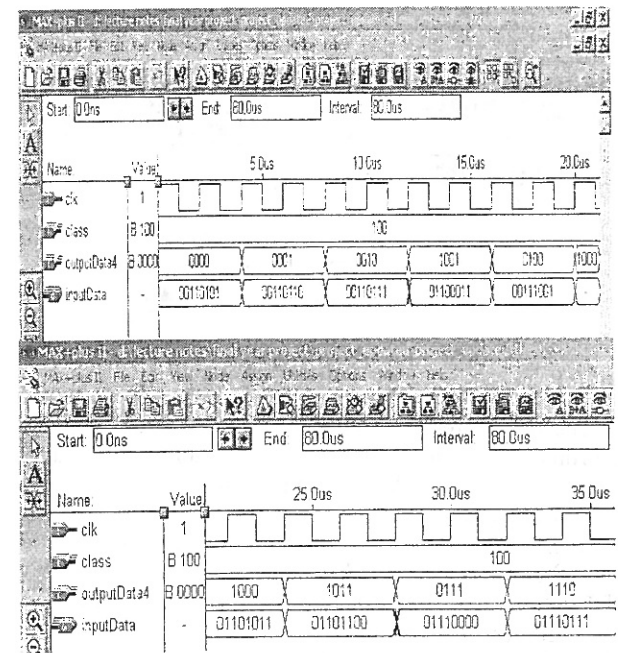


Figure 5: Simulation for decompression

V. SYNTHESIS

In regard to the designated hardware realization, The VHDL code is synthesized by considering Altera FLEX10K: EPF10K10LC84-3 FPGA chip on LC84 package. The physical hardware layout is generated using the synthesis tool Synplify version 7.0. The FLEX 10K family provides the density, speed, and features to integrate entire systems, including multiple 32-bit buses into a single chip. A comparatively low critical path frequency was achieved which was 27.9 MHz. The design took a minimum resource i.e. 366 logic cells, which is 63.54% of the device EPF10K10LC84. Table 1 shows a details report of the usage of resources. Partial view of sheet 3 (out of 15) of the top level RTL view and the sheet 3 (out of 15) of top-level technology view are shown in Figure 6 and 7 respectively.

TABLE 1
THE USAGE OF RESOURCES

Logic Resources (EPF10K10LC84-3)	Logic resources: 366 LCs of 576 (63.54%)
	Number of Nets: 444
	Number of Inputs: 1843
	I/O cells: 66
	Cells in logic mode: 323
	Cells in cascade mode: 43

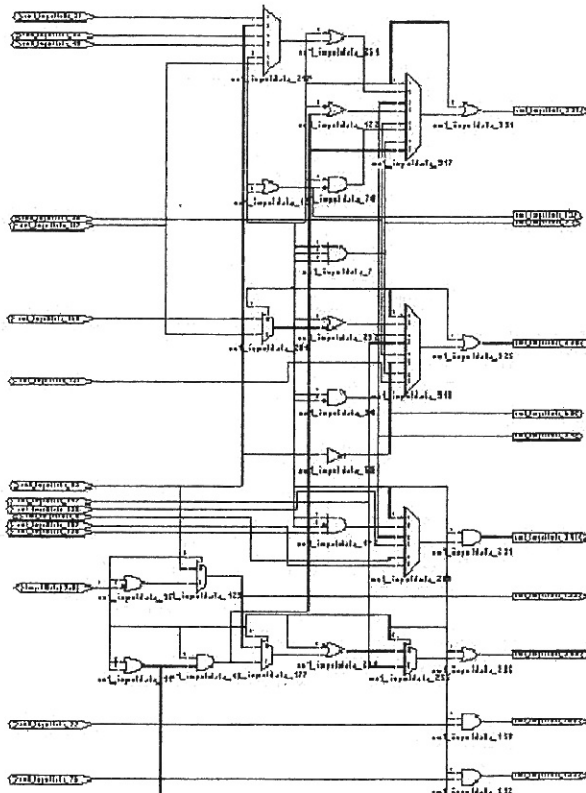


Figure 6: Top level RTL view - partial sheet 3 of 15

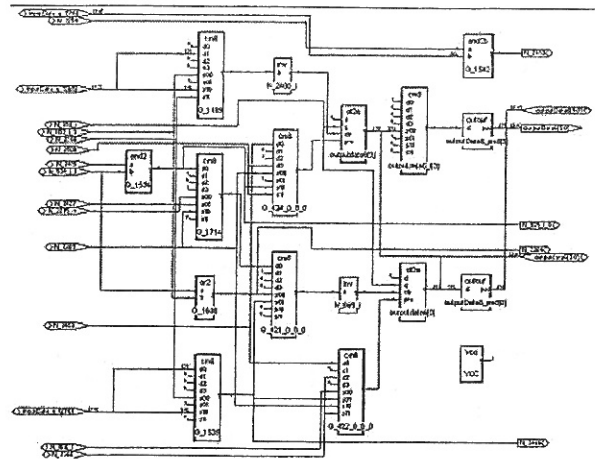


Figure 7: Top level technology view - sheet 3 of 15

VI. CONCLUSION

The objective of this project was to hardware prototyping of Boolean compression algorithm using VHDL. The Boolean function classification schemes are incorporated into Huffman coding for a better compression algorithm. The modules were successfully compiled, simulated and synthesized with achieved maximum frequency of 27.9 MHz and a minimum resource usage of 63.5% of the total cells. The hardware implementation demonstrated complete, correct functionality and met all the initial system requirements. Currently we are conducting further research to reduce the hardware complexity and increase the critical frequency in terms of synthesis by improving the VHDL coding.

VII. REFERENCES

- [1] Visweswariah, K., Kulkarni, S.R. and Verdu, S., "Universal Coding for Nonstationary Sources", IEEE Transaction on Information Theory, Vol. 46, No. 4, July 2000, pp 1633-1637
- [2] Chip-Hong Chang, Bogdan J. Falkowski, "Operations on Boolean Functions and Variables in Spectral Domain of Arithmetic Transform", IEEE International Symposium on Circuits and Systems (ISCAS '96 - Connecting the World), Georgia, 1996, Vol. 4, pp. 400-403.
- [3] Alexandre Schmid, Yusuf Leblebici, and Daniel Mlynek, "Hardware Realization of A Hamming Neural Network with On-Chip Learning," IEEE International Symposium on Circuits and Systems, Monterrey CA, 1998, vol. III, pp. 191-194.
- [4] B. K. Fawcett, "Tools to Speed FPGA Development," IEEE Spectrum, November 1994, vol. 31, pp. 88-94.
- [5] C. E. Cummings, "Verilog Simulation Xilinx Designs," Proc. Int. Verilog HDL Conf., Santa Clara, CA, 1994, pp. 93-100.
- [6] A. Rushton, VHDL for Logic Synthesis, Wiley, New York, 1998.
- [7] B. K. Fawcett, "Tools to Speed FPGA Development," IEEE Spectrum, November 1994, vol. 31, pp. 88-94.

- [8] Chien-Chung Tsai, Malgorzata Marek-Sadowska, "Boolean Function Classification via Fixed Polarity Reed-Muller Forms", IEEE Transactions on Computers. 1997, Vol. 46, No. 2, pp. 173-186
- [9] Mamun Bin Ibne Reaz, Sayed Zahidul Islam, Mohd. Alauddin Mohd. Ali, Mohd. Shahiman Sulaiman, "FPGA Realization of Backpropagation for Stock Market Prediction", Proceedings of the 9th International Conference on Neural Information Processing, Singapore, 18-22nd November, 2002, Vol. 2, pp 960-964.
- [10] Huffman, D.A., "A method for the Construction of Minimum Redundancy Codes," Proceedings of the Institute of Radio Engineers, New York, 1952, pp. 1098-1101
- [11] R.D.M. Hunter, "Introduction to VHDL", Chapman & Hall, Summit Design Inc., USA, 1996, 482 pages.
- [12] Peter J. Ashenden, "The Designer's Guide to VHDL", Morgan Kaufmann Publishers Inc., San Francisco, California, 1996, 688 pages.