AN FPGA IMPLEMENTATION OF A MICROPROGRAMMABLE CONTROLLER TO PERFORM LOSSLESS DATA COMPRESSION BASED ON THE HUFFMAN ALGORITHM

Tiago Maritan Ugulino de Araújo, Eduardo Ribas Pinto, José Antônio Gomes de Lima and Leonardo Vidal Batista

Departamento de Informática – CCEN – UFPB
Cidade Universitária, 58051-900

tiagomaritan@lavid.ufpb.br, ducaribas@yahoo.com.br, jose@di.ufpb.br and leonardo@di.ufpb.br

ABSTRACT

This work describes an FPGA implementation of a Microprogrammable Controller to perform lossless data compression based on the Huffman Algorithm. The Huffman Algorithm is a lossless statistical coding algorithm used in many modern compression systems, such as the JPEG and MPEG-2 standards. The Microprogrammable Controller architecture was implemented in blocks, using the hardware description language VHDL and the Huffman microprogram was implemented using the microinstrutions of this architecture. The description and functionalities of the main blocks that compose its architecture are presented as well as the simulation and synthesis methodology. By changing the microcode, the controller can also implement other compression algorithms.

1. INTRODUCTION

Data compression is a key component in many applications dealing with large amounts of data that must be efficiently stored or transmitted. With the advances in microelectronics, new integrated circuit technologies (such as FPGA - Field Programmable Gate Array), and new CAD (Computer Aided Design) tools, complex data compression algorithms can be implemented in high-level languages and mapped to circuits in an efficient way.

Most hardware implementations of data compression methods are not flexible enough to allow quick adjustments in the basic algorithm. On the other hand, the use of microprogrammable controllers provides a very flexible way to implement several data compression algorithms, such as Golomb, Golomb-Rice, Huffman and Arithmetic Coding [1].

Huffman algorithm [2] is the optimum integer-length coding algorithm, generally achieving data rates close to the message entropy [3]. Huffman coding is present in many modern compression methods, such as the JPEG and MPEG-2 standards.

The Huffman algorithm creates a different binary tree for each information source to be coded. This binary tree associates a code for each symbol of the source alphabet. This binary tree is not easy to implement in hardware, so most hardware implementations of the Huffman algorithm [4],[5] use static codes stored in a table. This solution codes any message in the same manner, independently of the characteristics of each specific information source, which is reflected in loss of compression.

This paper presents an FPGA implementation of a microprogrammable controller to perform lossless compression based in the Huffman algorithm. The Huffman binary trees are generated dynamically, i.e., a different binary tree for each information source.

The rest of this paper is organized as follows. In Section 2 the Huffman Algorithm is described. Section 3 presents the architecture of the MCC (Microprogrammable Controller for Data Compression), details of its implementation. Section 4 presents the simulation and synthesis methodology. Section 5 presents synthesis results in terms of the FPGA devices used. Finally, conclusions are presented in Section 6.

2. THE HUFFMAN ALGORITHM

The Huffman algorithm is a lossless statistical coding algorithm that depends on the source symbols probabilities. With lossless compression, the original information can be perfectly reconstructed from the coded message.

Huffman algorithm associates a weighted tree for each symbol of the information source. Initially, each weighted tree has only one node with weight equal to the associated symbol probability. In each iteration of the algorithm, the two trees with the smallest weights become the right and left subtrees of a new tree whose weight is the sum of the weights of the two subtrees. The routine stops when only one tree remains.

The code for each symbol is obtained by traversing the final tree from the root to the corresponding symbol node (a leaf node). A bit '1' corresponds to a left branch and a bit '0' corresponds to a right branch or vice versa.

3. MCC ARCHITECTURE

The MCC (Microprogrammable Controller for Data Compression) architecture is composed by the following blocks: Microprocessor Module (MM); Microprogrammable Control Unit (MCU); Sequencing Logic (SL); Stack (SK); Tree Memory (TM); Memory Data Register (MDR); Memory Adress Register (MAR). Figure 1 shows the MCC architecture implemented in FPGA.

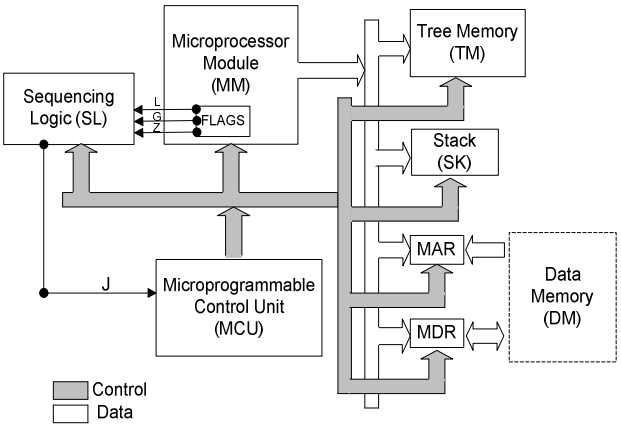

Figure 1. MCC architecture

The Data Memory (DM), shown in Figure 1 is an external memory block that stores the data to be processed (coded/decoded) by the MCC. This memory is divided in two blocks: the Header Block and the Raw/Coded Data Block, as shown in Figure 2.

The Header block contains the symbols probability. During encoding, the Raw/Coded Block contains the raw data that will be coded and, for decoding, it contains the coded data that will be decoded.

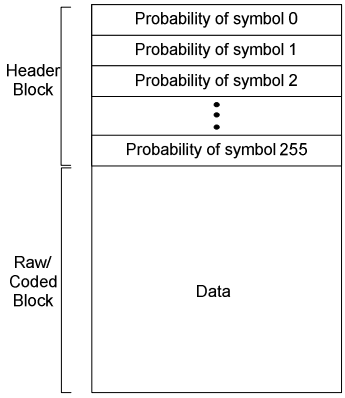

Figure 2. Data Memory

3.1. Microprocessor Module (MM)

Figure 3 shows the behavioral VHDL descriptions used in the MM implementation [6].

The MM is responsible to temporarily store variables and to execute logic and arithmetic operations. Its contains sixteen identical 16-bits scratchpad registers, called RA, RB, RC, RD, RE, RF, RG, RH, RI, RJ, RK, RL, RM, RN, RO and RQ. The sixteen registers can be loaded by the C bus. The registers RA up to RH can output its contents onto the internal A bus and the registers RI up to RQ can output its contents onto B bus.

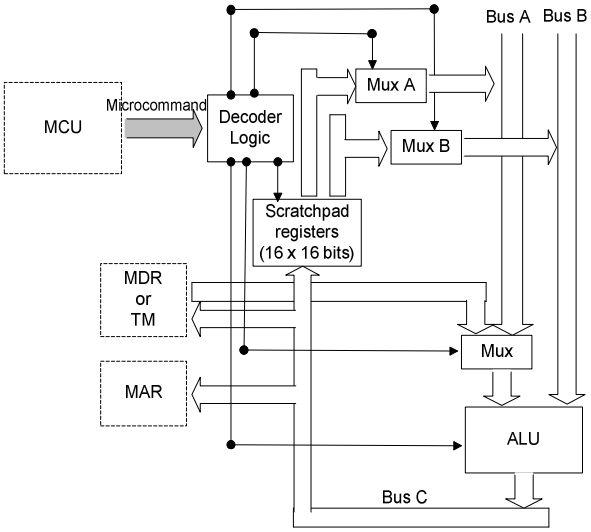

Figure 3. Microprocessor module

The ALU can perform 16 operations specified by the control lines. It generates the flags that will be used to control the flow of execution of the microprogram. The behavioral VHDL description of the ALU conducts to a "Carry Look Ahead" adder implementation increasing the arithmetic operations of the module.

3.2. MAR and MDR

MAR and MDR control the data flow between the MCC and DM (Figure 1). The MAR stores the address of the DM that is read or written across the MDR. The MM can write to MAR, read and write in the MDR to exchange data with DM.

3.3. TM and SK

Figure 3 shows the interface between the MM module and the TM and SK modules. The TM stores the Huffman tree. Moreover, the TM stores a structure array that contains symbols with non-null probabilities and the respective address. This array is used to locate the node of the symbols in the Huffman tree.

The SK is an efficient data structure that performs the storage of the variables. The SK is a Last in- First out (LIFO) structure that is used to traverse the Huffman tree.

3.4. MCU

The MCU controls the blocks of the MCC, the ALU operations and the data path using horizontal microinstructions, as shown in Figure 4. The microinstructions are divided in 11 fields. Each field

controls only one block allowing concurrent operations of the blocks. Each field has a set of possible micro-commands that form a set of instructions for that field. Each instruction has its equivalent mnemonic and users can write programs in Assembly.

The Huffman algorithm is implemented by using the available microinstructions of each field. The resulting microprogram is stored in the MCU ROM. Both coded and decoded microprogram are sequentially stored in the same memory. Other compression algorithms can be implemented by changing the content of the MCU ROM only.

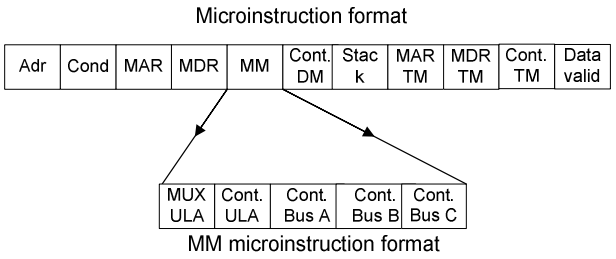

Figure 4. MCC microinstruction format

3.5. SL

SL determines which next microinstruction will be run. The output of this logic controls the MCC which routes either "current address + 1" or ADR, i.e. the address of the next microinstruction. The output flags of the MM, and the COND field of the microinstruction indicate the choice to be made.

4. SYNTHESIS METHOD

The MCC architecture was implemented in blocks, as shown in Figure 1, using the hardware description language VHDL [7]. The blocks were divided in sub-blocks. The blocks and sub-blocks behavioral descriptions were made and their structural descriptions were constructed and validated using the Altera Quartus II 2.2 software [9].

A semi-adaptative Huffman algorithm [1] was implemented using the MCC microinstructions, i.e. as a microprogram. The implementations of both coding and decoding, microprograms are done in four stages:

1. Read the Header block in DM and put the symbols and its probability in TM;
2. Sort the symbols in TM by crescent probability, using the BubbleSort algorithm;
3. Create the Huffman tree and the array of symbols and addresses of their nodes in this tree. That array is used only in the coding stage. The tree and the array should be put in TM.
4. Read the Raw/Coded Block of DM and code/decode the symbols using the Huffman Tree in TM.

To support the development of the microprogram a tool called MccME (MCC Microprogram Editor) was implemented. The MccME was developed in Object Pascal.

Each block of the MCC architecture has an instruction mnemonic set and an entire MCC microprogram can be written in Assembly. The MccME generates a "mif" file (Memory Initialization File) after compiling the program as shown in Figure 6. A "mif" file is an ASCII text file that specifies the initial content of the ROM block of the MCU, used as an input file for memory initialization in the Altera Compiler and Simulation tools [8].

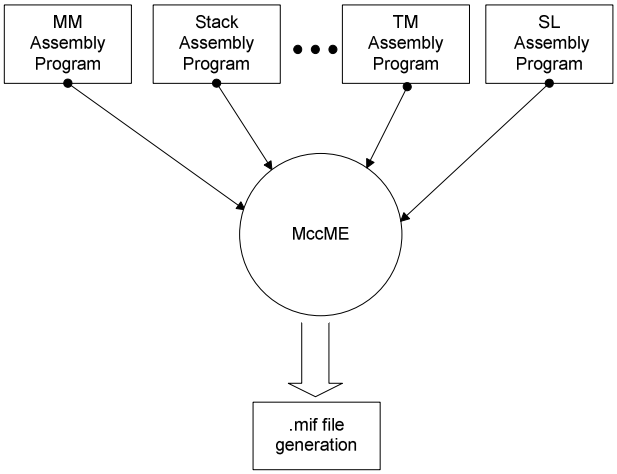

Figure 6. Microprogram development

5. RESULTS AND DISCUSSIONS

The descriptions of the MCC were synthesized using Quartus II 2.2 software from Altera [9]. Results in terms of number of logic cells, number of memory bits utilized, numbers of pins and frequency of operation for four FPGA devices are shown in Figure 7.

The results presented in Figure 7, show that MCC architecture uses less than 10% of the devices logic cells and from 50% up to 85% of the device memory bits.

To validate the proposed architecture, we coded five electrocardiogram (ECG) records from MIT-BIH Arrhythmia Database [10] on MCC and on Winzip 9.0 program. The ECG records are one-dimensional signals with 11 bits per sample. To optimize the compression, the prediction error was computed on the ECGs signals [11].

ECG records with error prediction were used here because we intend to implement MCC plus a predictor in *Holter* portable *ECG* devices for long-term cardiac monitoring

A *Holter* device records the ECG signal continuously for 24 or 48 hours. For high quality records the storage requirements may be prohibitive without data compression.

The metric used to quantity confrontation between MCC and Winzip program was the compression ratio (CR). The CR is defined as the ratio between the number of bits in the original signal and the number of bits used to represent the compressed signal.

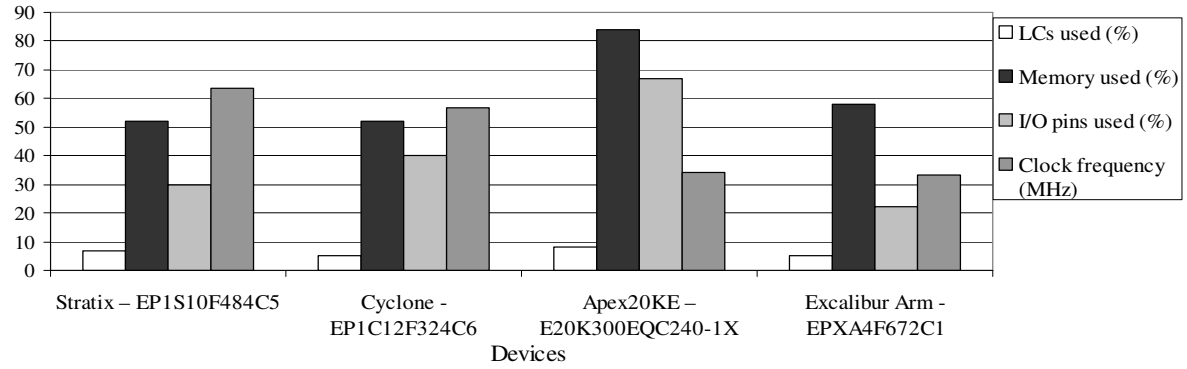

Figura 7. Results of MCC synthesis

The average CR obtained to MCC architecture was 2.62:1, while to WinZip program was 1.73:1. This result shows the efficiency of the MCC Huffman microprogram.

6. CONCLUSIONS AND FUTURE WORKS

An FPGA implementation of a microprogrammable controller that performs the Huffman Algorithms has been described. The results show that it is viable to incorporate this implementation in a single FPGA device. Furthermore, the use of FPGA devices brings many possibilities of reconfiguration.

In the future, other data compression algorithms will be implemented, such as the arithmetic coding algorithm. Besides, we intend to adjust the flexible architecture proposed in this paper for the Open Core Protocol (OCP).

7. REFERENCES

[1] T. Bell, J. G. Clearly and I. H. Witten, "Text Compression". Prentice Hall Reference Series, 1990.

[2] D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes", *Proceedings of the Institute of Radio Engineers*, n. 40, pp. 1098-1101, 1952.

[3] C. E. Shannon, C. E. "A Mathematical Theory of Communication", Bell Syst. Tech. J., pp. 379-423, 1948.

[4] S. H. Sun and S.J. Lee. "A JPEG Chip for Image Compression and Decompression", *Journal of VLSI Signal Processing*, Kluwer Academic Publishers, pp. 43–60, August 2003.

[5] C.J. Lian, Y.W. Huang, H.C. Fang, Y.C. Chang and L.G. Chen, "JPEG, MPEG-4, and H.264 Codec IP Development", *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, IEEE Computer Society, March 2005.

[6] J. A. G. Lima, E. U. K. Melcher and H. S. Silva, "An FPGA Implementation of the ATM Layer", *XIII Symposium on Integrated Circuit Design and System Design(SBCCI 2000)*, Amazonas-Brazil, pp. 185-190, September 2000,.

[7] J. H. Moreno, M. Ercegovac and T. Lang, "Introdução aos Sistemas Digitais". 1a ed. Bookman, 2000.

[8] *Altera Corporation*. "Altera Data Book", 1995.

[9] *Altera Corporation.* "Using Quartus II Verilog HDL & VHDL Integrated Synthesis", 2002.

[10] G. B. Moody, R.G. Mark, *MIT-BIH Arrhythmia Database Directory*. Second Edition, BMEC TR010, Massachusetts Institute of Technology, Biomedical Engineering Center, August 1988.

[11] M.M. Meira, J.A.G.Lima and L.V.Batista. **"**An FPGA Implementation of a Lossless Electrocardiogram Compressor based on Prediction and Golomb-Rice Coding for Telemedicine Applications**",** *XXI Simpósio Brasileiro de Telecomunicações (SBT'04)*, Pará-Brasil, September 2004.