FPGA-Based Modelling Unit for High Speed Lossless Arithmetic Coding

Riad Stefo[1], José Luis Núñez[2], Claudia Feregrino[2], Sudipta Mahapatra[3], and Simon Jones[2]

[1] Electronic and Electrical Engineering Department, Institut für Grundlagen der Elektrotechnik und Elektronik (IEE) TU Dresden, 01062 Dresden, Germany
stefo@iee.et.tu-dresden.de

[2] Electronic Systems Design Group, Loughborough University
Loughborough, Leics., LE11 3TU, U.K.
{J.L.Nunez-Yanez, C.Feregrino-Uribe, S.R.Jones}@lboro.ac.uk

[3] Dept. of Computer Science Engineering & Applications, Regional Engineering College, Rourkela - 769 008, Orissa, India
Sudipta@rec.ori.nic.in

Abstract. This paper presents a hardware implementation of an adaptive modelling unit for parallel binary arithmetic coding. The presented model combines the advantages of binary arithmetic coding where the coding process is simplified, with the benefits of multi-alphabet arithmetic coding where any type of data can be compressed. The modelling unit adopts a simple method to store and modify the information, making it able to process 8 bits per clock cycle and to increase substantially the arithmetic coding speed. This model has been implemented in an A500K130 ProASIC FPGA and offers a throughput of 256 Mbits/s.

1 Introduction

Data compression allows representing data in a format that requires less space than is usually needed. It is particularly useful in communications because it enables devices to transmit the same amount of data in fewer bits. Data compression can be lossy or lossless [1]. Unlike lossy compression where the decompressed data may be different from the original, lossless compression requires decompressed data to be an exact copy of the original.

Arithmetic coding is one of the best algorithms that can be used in lossless data compression. It replaces a stream of symbols with a coding range of real numbers between 0 and 1. The low end of this coding range is then used as output code for this stream of symbols. At the beginning of the compression process, the coding range has 0 and 1 as its low and high ends, and a cumulative probability interval (Q_x , Q_{x+1}) is allocated to every possible input symbol. So, the symbol x has the probability $P_x = Q_{x+1} - Q_x$. Every time a symbol is encoded, the coding range is narrowed to a portion allocated to the symbol according to its cumulative probability interval. The task of arithmetic coder is

G. Brebner and R. Woods (Eds.): FPL 2001, LNCS 2147, pp. 643–647, 2001.

to calculate the new low and high ends of the coding range using the equations (1) and (2), where $old\_range = old\_high - old\_low$.

$$\text{new\_high} = \text{old\_low} + \text{old\_range} \times Q_{x+1} \quad (1)$$

$$\text{new\_low} = \text{old\_low} + \text{old\_range} \times Q_x \quad (2)$$

Instead of the new high end of the coding range the new coding range can be obtained using equation (3) to execute the arithmetic coding task.

$$\text{new\_range} = \text{old\_range} \times P_x \quad (3)$$

Arithmetic coding can be implemented for either multialphabet or binary alphabet. Because of the complexity of multialphabet arithmetic coding few implementations have been presented. One of them [2] presents a software implementation of arithmetic coding at byte level based on the equations (1) and (2). The complexity of computations in this implementation makes it impractical for real time applications. Other software implementation [3] presents a multialphabet arithmetic coding using a simpler algorithmic structure than the presented in [2], and offers similar compression ratio.

Due to the simplicity of binary arithmetic coding several binary arithmetic coding implementations have been presented. One of the best known, Marks [4], presents the Qx-coder which uses a 7^{th} order binary model and processes a single bit per clock cycle. Kuang *et al.* [5] presents another implementation of arithmetic coder that uses a 10^{th} order binary model and needs 8.5 clock cycles to process a bit.

In Section 2 we review a parallel implementation of arithmetic coding. Section 3 discusses the hardware architecture of the modelling unit. Section 4 reports the design's implementation and Section 5 concludes this paper.

2 Review of Parallel Arithmetic Coding

In [6] we proposed a parallel implementation of arithmetic coder that follows the scheme showed in [7]. The model is 0^{th} order and processes symbols of 8 bits using a binary tree of $n = log_2 N$ levels to store the frequency information of the symbols, where N is the size of the alphabet. Part of such tree is shown in Fig. 1. The complete tree has 8 levels for an alphabet of 256 symbols.

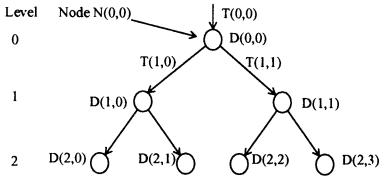

Fig. 1. Part of the binary tree

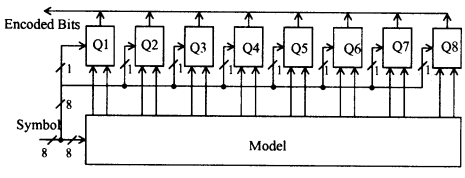

Fig. 2. Structure of the compressor

Each node N(i,j)stores frequency information of the symbol in a single variable D(i,j). The value of this variable splits the codespace of each node in two different halves. This codespace has zero as its left limit and the data received from the parent node T(i,j) as its right limit which is the total count of the symbols. The root node stores the size of the alphabet as the data from its parent node, which is increased after each symbol is encoded. It is assumed during the initialization operation that each symbol of the alphabet has occurred once.

Next we show the modelling algorithm, where b_i refers to the i^{th} bit of the symbol to be encoded (b_0 is the MSB).

1 Initialize the nodes of the tree with the symbols frequency information $D(i,j)$ and set $T(0,0)$ to 256
$D(i,j) = 2^{n-i-1},\ 0 \le i \le n-1,\ 0 \le j \le 2^i - 1,\ n = 8$
$T(0,0) = 2^n$
2 Send $T(0,0)$ to the root of the tree
3 Send the next symbol to be encoded to the first level and set $j = 0$
4 FOR i = 0 to n-1, execute the following operations:
a) Receive the value $T(i,j)$ from the parent node
b) IF $i < n-1$
IF $b_i = 0$
set $T(i+1,2j) = D(i,j)$ and send it to the left child $N(i+1,2j)$
ELSE
set $T(i+1,2j+1) = [T(i,j) - D(i,j)]$ and send it to the right child $N(i+1,2j+1)$
c) Send the values $D(i,j)$ and $T(i,j)$ as the modelling information to the corresponding coder
d) Update $D(i,j) = D(i,j) + (1 - b_i)$
IF $b_i = 0$
set $j = 2j$
ELSE
set $j = 2j + 1$
5 Update $T(0,0) = T(0,0) + 1$
6 IF there are more symbols to encode go to step 2
ELSE
EXIT

Fig. 2 shows the compressor model. The coders work in parallel such that each coder encodes one bit of the symbol. Each coder Q encodes either the left half or the right half of the code space according to the bit received from the symbol using the equations (2) and (3). The code bits generated by each coder Q are sent to the decompressor, which uses them to retrieve the original data. For details about the structure of the decompressor the reader is refered to [6]. A software implementation for the compressor and the decompressor has been done using C++. The average compression ratio for Canterbury Corpus files

[8] is 0.64, which is still a good compression ratio for a universal lossless data compressor.

3 Model Architecture

The fact that only one node in each level of the tree sends the modelling information to the corresponding coder makes the realisation of each tree level with one node possible. In this case each tree level contains one node and the corresponding memory to store the frequency information of the symbols for all the nodes in this level. The designed model works basically in two phases. The first phase is for initialization where the memory locations in the tree levels are initialized with the frequency information of the symbols. The second phase is for encoding where the model analyses the symbols in the input data and sends the modelling information to the coders. The model works in blockwise fashion, this means that the initialization phase is executed for each new block of data. Fig. 3 shows the architecture of the model. The total counter generates the data for the next phrase. The initialize counter generates the addresses to the memory locations during the initialization phase. Each tree level receives data from the previous level and the corresponding bit of the symbol from the symbol register. It analyses these data according to the modelling algorithm and sends data to the next level and modelling information to the corresponding coder. The signals mid point1...8 , range top1...8 in Fig. 3 form the modelling information that each tree level sends to its corresponding binary coder. Fig. 4 shows the architecture of a tree level. The intermediate logic updates the frequency information of the symbol and sends it to the RAM. It also calculates the data to be send to the next tree level. The index logic define which child node in the next level will receive the data sent from the current level. The comparator is used to avoid collisions between the read and write addresses of the RAM.

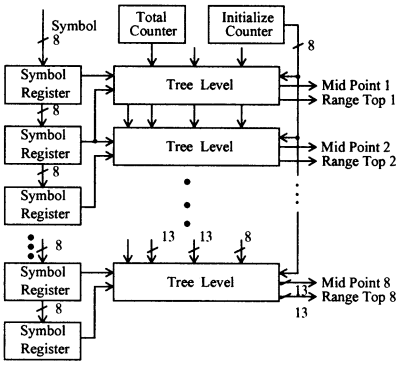

Fig. 3. Model architecture

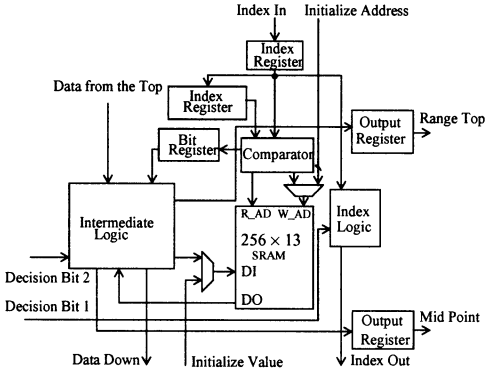

Fig. 4. Tree level architecture

Table 1. Comparing our implementation with other available implementations

Implementation	Speed	Throughput	Symbol processed	Technology
Marks[4]	75 MHz	64 Mbits/s	0.8 bit/clock cycle	CMOS 5S 0.35 μm IBM
Kuang [5]	25 MHz	3 Mbits/s	0.12 bit/clock cycle	0.8 μm single poly double metal SPDM
Presented implementation	32 MHz	256 Mbits/s	8.0 bits/clock cycle	ProASIC A500K FPGA

4 Implementation

The model has been implemented in a non-volatile reprogrammable ProASIC A500K130 FPGA. The design only uses 32.6% of the device logic and 80% of the embedded RAM available. The device can be clocked at 32 MHz and processes a data block up to 256 Kbytes. Table 1 compares the results of our implementation with other available implementations. The results from Table 1 show clearly that the presented implementation outperforms other implementations.

5 Conclusions

A hardware implementation of a 0^{th} order adaptive statistical modelling unit that is able to support parallel binary arithmetic coding is presented in this paper. The described model combines the advantages of binary arithmetic coding, with the benefits of multi-alphabet arithmetic coding. Parallel binary arithmetic coding increases the arithmetic coding speed substantially offering a throughput of 256 Mbits/s.

References

1. M. Nelson: The Data Compression Book, Prentice Hall (1991)
2. I.H.Witten et al: Arithmetic Coding for Data Compression , Communications of the ACM, Vol. 30, No.6, (1987), pp. 520-540
3. J. Jiang.: Novel Design of Arithmetic Coding for Data Compression, IEE Proceedings Computers and Digital Techniques, Vol. 142, No. 6, (1995),pp. 419-424
4. K. M. Marks: A JBIG-ABIC Compression Engine for Digital Document Processing, IBM Journal of Research and Development, Vol. 42, No.6, (1998), pp. 753-758
5. S. Kuang, J. Jou, Y. Chen: The Design of an Adaptive On-Line Binary Arithmetic Coding Chip, IEEE TCAS-I, Vol. 45, No. 7, (1998), pp. 693-706
6. S. Mahapatra, J. L.Núñez, C. Feregrino-Uribe and S. Jones: Parallel Implementation of a Multialphabet Arithmetic Coding Algorithm, IEE Colloquium on Data Compression: Methods and Implementations, IEE Savoy Place, London, (1999)
7. A. Moffat: Linear Time Adaptive Arithmetic Coding, IEEE Transaction on Information Theory, Vol. 36, No. 2, (1990), pp. 401-406
8. R. Arnold, T. Bell: A Corpus for the Evaluation of Lossless Compression Algorithms, Data Compression Conference,(1997), pp. 201-210