Efficient Implementation of Arithmetic Compression for EEG

D. O'Shea, R. McSweeney, C. Spagnol, E. Popovici

Department of Electrical and Electronic Engineering, University College Cork

Email :dejoshea@gmail.com, richardmcs@ue.ucc.ie, c.spagnol@ue.ucc.ie, e.popovici@ucc.ie

Abstract— **This paper presents some hardware and software implementations of Arithmetic Coding for lossless compression of electroencephalogram (EEG) data. Arithmetic Coding achieves near optimum levels of compression set by the entropy of the source. Both the hardware and software implementations of the Arithmetic Coding algorithm are implemented for EEG data. We achieve a compression ratio of 0.5 on average. This compression ratio was achieved through the use of reversible filtering, whereby the source information is filtered to reduce its entropy and after decoding an inverse filter returns the original information without loss. A differentiator was used at the input of the encoder and an integrator at the output of the decoder. This allowed for an increase in compression by 18.1%. The codec was implemented on Xilinx Spartan FPGA and power analysis was carried out.**

Keywords **– EEG, lossless compression, arithmetic coding, FPGA, low power.**

I INTRODUCTION

For many low power wireless applications, data compression can significantly reduce the power consumption of the system. In many such systems the RF transceiver carries the largest power requirement and any reduction in the number of bits to transmit will have a large power saving associated with it. Data compression reduces the number of bits to transmit while still relaying the same information [1]. This can lead to an overall power saving if the power required to carry out the compression/decompression is less than the power required to transmit the extra bits of information, which is generally the case. Another advantage of compression is that it reduces the amount of storage, thus contributing to both power and cost reduction [3].

This paper focuses on the use of the arithmetic coding algorithm as the data compression scheme for lossless compression of electroencephalography (EEG) data [5]**.** The main benefits of using data compression here is to reduce the size of the data for transmission and also for reducing the storage requirement. Being able to reduce the overall power consumption of the system through data compression is also important.

EEG can be used as a medical diagnosis technique and for such applications it is imperative that no information is lost which could affect the result of the examination. The EEG can be also used in several other applications including Human Computer Interface (or Brain Computer Interface), gaming control, fatigue detection, consciousness level monitoring, etc. It is a recording of electrical potentials along the scalp that originate from electrical activity within the brain. Sensors are placed on the scalp and the waveforms captured contain variations in both amplitude and frequency. These variations depend on the mental activity of the patient, and change for example, while the patient is conscious or sleeping. Multiple channels are recorded from the sensors, and patterns within these channels are used to diagnose neurological diseases such as seizure.

The compression of EEG is desirable for a number of reasons. Although the sampling frequencies are low, multiple channels (order of 10 to 100) are recorded simultaneously and large amount of data are recorded. The ability to compress this data and transmit it efficiently is desirable. It can allow alternative ways of processing the EEG recordings, so as being able to carry it out in the home or a hand held device rather than in a hospital setting.

By being able to efficiently transfer EEG data, it would be possible to produce wearable EEG devices [2], so that the patient could use the device at home. Two types of wearable EEG could make use of data compression, such as portable storage or wireless EEG devices. The portable storage can be used to

record the information on a storage element within the device for a later transfer/analysis, or a wireless device, that would relay the EEG data over a wireless communication network and be connected to the internet to pass the information to the hospital or private practice for diagnosis [4].

Data compression can bring power saving to the wireless device by reducing the power needed to transmit redundant data. The power cost of the compression is generally less than that needed to transmit the data, and this reduces the power consumption and also the time needed to transmit the data. This could lead to a reduced battery requirement for the device, which would lower the size and weight.

The compression performance can easily be indicated by the compression ratio (*CR*):

$$CR = 1 - (C / O) \quad (1)$$

where, *C* denotes the file size after compression while *O* denotes the original file size.

II ARITHMETIC CODING

Arithmetic Coding is a form of variable length entropy coding [9,10,11]. It relies on a probability model to achieve compression, and the accuracy of this model will impact on the overall performance of the compression scheme. With Arithmetic Coding, there is an intrinsic separation between the model and the encoder unit itself. The model can generally be changed without any need to adapt the encoder, allowing the same encoder to be used in a variety of applications or source variations.

In practice this model can be implemented as a look-up table, and generally cumulative frequencies are used (rather than the actual probabilities) as it simplifies the implementation. The model can either be static or adaptive. The use of adaptive models over static models allows greater coding efficiency when the statistical property of the source changes compared to static model. This benefit comes at a cost in the form of updating the model every time a symbol is encoded or decoded, and hence it leads to increased complexity.

a) Arithmetic Encoder

Arithmetic coding works by assigning an initial range, and as input symbols are encoded this range reduces, the less probable a symbol is of occurring the further the reduction in the range that will take place. Compression is achieved as the more probable a symbol is of occurring the wider is the range, and this leads to less precision in the range being needed. It uses integer instead of floating point operation.

Figure 1 shows the flow diagram of the encoder. The encoding process starts with the initialization: this is setting the initial coding range which is accomplished by setting the initial values of upper and lower bounds of the range. Once initialization is complete, a symbol is read into the encoder. If the symbol is the end of transmission symbol, the encoder is flushed of the remaining output bits and the encoding process ends.

For the case when the symbol is not the end of transmission symbol, the encoding process continues. Based on the outputs of the model for the current symbol, and also the previous values of the current range and the lower bound of the range, the new values for the current range and lower and upper bound are calculated. To determine if the bits can be the encoder output, the upper and lower bounds of the range are checked for convergence. If the Most Significant Bit(MSB) of the upper and lower bounds are equal, the value of the MSB can be the output. If the underflow counter is greater than 0, the inverse of the MSB is the output and the counter decremented until the counter reaches zero. As the MSBs of the limits of the range have converged they will not change again and as a result the limits are shifted to the left to free up the limited precision of the encoder. Once this is completed the limits of the range are once again checked for convergence.

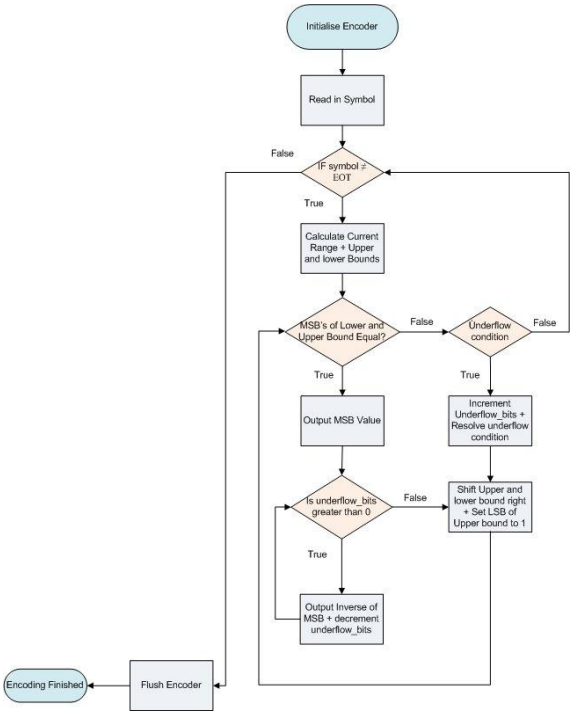

Figure 1: Flow diagram of the arithmetic encoder

When checking the limits for convergence they are also checked for the underflow condition, which occurs when the second MSB of the lower bound is 1 and 0 for the upper bound. When this occurs it can lead to lock up of the encoder and the process would fail to continue. To avoid this problem, the underflow condition is removed, by discarding the second MSB of both bounds. The discarded bits are

still important for generating the codeword and an underflow counter is used to keep track of the number of times this discarding of bits has occurred since the last MSB match. After the underflow condition has been removed the bounds are shifted to the left and again the process returns to checking the bounds for convergence.
When the MSBs of the bounds no longer match and there is no underflow condition present, the encoding process has finished and the encoder returns to reading in the next symbol and the process begins again. A block diagram of the encoder is presented in Figure 2.

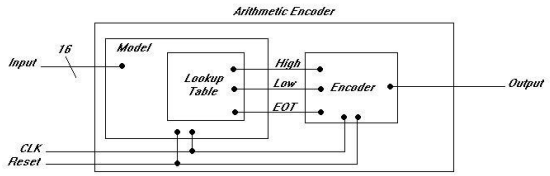

Figure 2: Arithmetic Encoder Architecture

b) Arithmetic Decoder
Decoding is accomplished by initially having the range set the same as the initial encoder range. The codeword is checked to see what symbols proportion of the range it is found in. The next step it removes the symbol from the codeword. This is done by using the following formulas.

$$Range = Upper_{bound} - Lower_{bound} \quad (2)$$

$$Codeword = (Codeword - Lower_{bound}) / Range \quad (3)$$

Figure 3 shows the flow diagram for the decoder. Decoding begins as with the encoder with the initialization of the upper and lower bounds. Also during the initialization the first of the codeword bits are read in and the number of bits equals the same as the number in the bounds of the range. The initial range of both the encoder and the decoder must match for correct decoding.
After initialization the current range is calculated from the upper and lower bounds. A temporary variable is also calculated, which is used to determine the symbol. To accomplish this, the model is searched to find the symbol that relates to the value of temp and the following equality shows how this is done.
This search can be carried out in a number of ways, such as the use of a linear or a binary search. However in most cases the linear search takes too long to complete and the time to complete varies depending on where the symbol lies in the model. To overcome these drawbacks a binary search can be implemented and this leads to shorter and predictable search times as the search will required N cycles were N is the number of bits in the model's address.

Once the symbol is found, it is checked to see if it is the end of transmission symbol. If found the decoding process stops as all symbols have been decoded. Otherwise the new bounds for the range are calculated based on this symbol and the bounds are checked for convergence as is carried out in the encoder. The only differences to this stage is that no output bits are generated and when the bounds of the range are shifted, so is the codeword, and the LSB of the codeword is set to the next code bit.
Once the convergence check has completed the decoder returns to calculating the current range and the new value for temp and the process begin again.

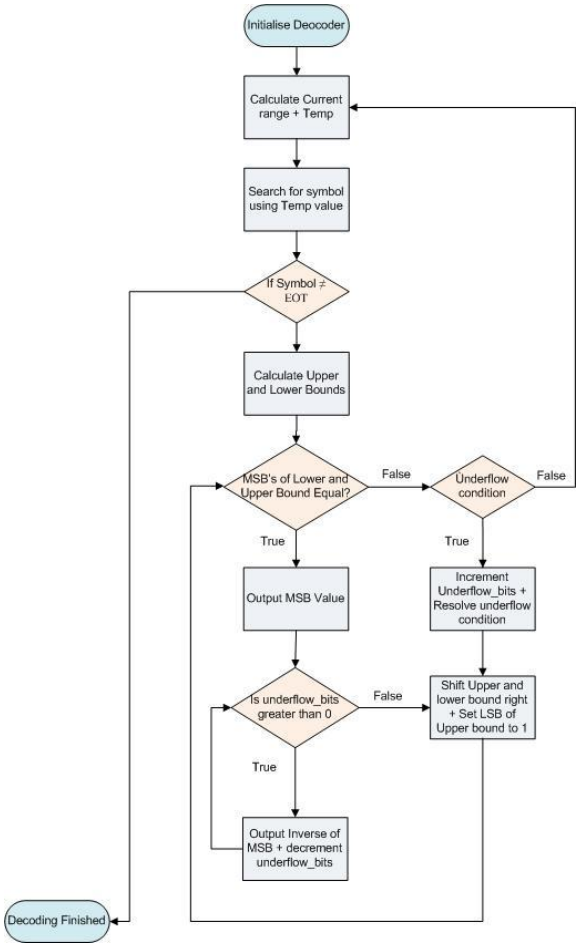

Figure 3: Flow diagram of the arithmetic decoder

III SOFTWARE IMPLEMENTATION

The EEG samples were based on 16 bit analogue to digital converter outputs sampled at 256 Hz from a database provided by the University of Freiburg [6]. The data segments considered contains 6 channels of EEG with a run-length of 921600 samples. A program was written to examine the EEG sample and return occurrences of each of the 65536 symbols.
A model that reflects the statistical properties was then created. The arithmetic codec is implemented using the C-code language [7][8][9]. Several

versions of the codec are investigated, and their compression performances are discussed.

a) The 8-bit codec

In order to reduce the size of the encoder and also the model's memory, an 8 bit version of the Arithmetic Coding algorithm is investigated. The 16 bit input is split into two 8 bit symbols, which lowers the amount of memory needed for the model's look-up table from 65536 down to 256. The size of the registers in the design decreases over the 16bit case and this leads to reduced complexity and area/memory. The reduced size of the model and computational constraints allows the use of 16-bit microcontroller for this version of the algorithm.
This approach offers many advantages such as reduced memory requirement, but it also suffers from a number of drawbacks. In order to make use of this version, the input has to be split into two symbols, which leaves two symbols to encode instead of one. Therefore both the encoder and the decoder need to perform double the amount of calculations over the 16-bit implementation, which affects the system throughput.

b) The 16-bit codec

This implementation encodes the entire input symbol at once, and therefore the size of the look-up table in this version increases to 65536 for the 16bit input. The precision of the internal encoder and decoder registers are increased to 22bits and this limits the use of the implementation to 32-bit microcontrollers. However this implementation achieves better compression performance than the 8 bit input version due to the initially lower entropy of the source.
To further increase the compression performance, the derivative of the EEG is used. This is achieved by passing the EEG data through a differentiator, which reduces the entropy and allows for higher compression ratios to be achieved. The derivative signal has an offset applied so that the output is always positive. At the decoding stage, the original signal is retrieved by using an integrator. From this section on, the combination of this system is simply referred to as the reversible filter.

b) Compression Performance

The 8 bit input version gives the lowest compression performance but does offer reduced complexity and memory requirements due to its reduced model size. One major drawback from this implementation is the creation of two symbols for each symbol read in, and therefore increasing the amount of calculations to be carried out.
The 16 bit input version gives an increase in compression performance over the 8 bit input version at the expense of an increased lookup table size, increasing from 256 to 65536 values. The addition of a differentiator at the input of the encoder and output of the decoder gives a large increase in the performance of the compression scheme for little additional computational complexity. The derivative gives on average an 18.1% increase in compression ratio from the architecture without the use of the differentiator.

Table 1: Compression Performance

Implementation	***Average CR***	***Entropy H***
8-bit	22.60%	6.179 bits
16bit	31.50%	10.53 bits
16-bit + derivative	49.60%	7.74 bits

Due to the performances and the benefits versus the drawback of each of the different versions, the 16 bit input with the reversible filter is chosen as the version to be implemented in hardware.

IV HARDWARE IMPLEMENTATION

The hardware implementation is based upon the 16 bit input with reversible filter. This offers good compression performance without much extra complexity. Both the encoder and the decoder are structured into blocks to form the overall coding scheme.

b) Encoder Architecture

The encoder consists of a filter, model, encoding and a controller block (Figure 6). The output is a parallel 16bit bus that is generated when the encoder reaches a multiple of 16 bit outputs. When the bus is ready for reading, the read pin on the output goes high. Once it is read, an acknowledgement is sent to the encoder. When the "End of transmission symbol" is read, the encoder enters the flush state and the output is padded with zeroes to reach 16 bits if necessary.

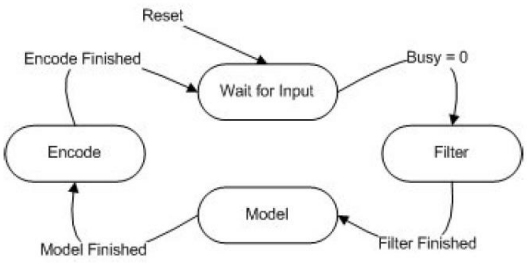

Figure 6: Control block of the encoder

The controller is responsible for enabling the other modules within the design. The busy pin input is connected to this module and when its goes low, its enables the filtering block. When the filtering has

been completed, the signal filter pin goes high, indicating the process is finished and that the model module can now be enabled. Once again when the model block has finished, it signals to the controller to start the encoding processing making use of the output of the model block. Once the encoding block has finished, it again signals its completion and the controller reverts to its initial state of waiting for the next symbol to arrive. The filter block is enabled by the controller, two signals connect the two, one signals to the filter block to begin and the second signals to the controller that filtering has been carried out.

c) Decoder Architecture

The decoder contains a FSM, which controls the operation of the decoding module. It also controllers the operation of the model/filter block. This is in contrast to the encoder where there is a separate controller module. The use of only a single module is sufficient as the tasks are simpler and the decoder and model/filter block work together throughout the decoding stage, whereas the block works sequentially in the encoder.
The search in the decoder requires interaction with the Model/filter module, in order to return the values of high and low for use in the search. Once the symbol had been found, the filtering process can be started in the model and the decoded symbol output from the system.

d) Synthesis and Power

The Verilog HDL designs of the 16-bit codec with the reversible filter are implemented on the Xilinx Spartan 3 C3S200 FPGA. Synthesis and power estimation are carried out using Xilinx Xpower software package.

Table 2: Synthesis results of the codec on the Xilinx Spartan 3 FPGA

Device	***Slice Flip-Flops***	***4 input LUT***
Encoder	344	3021
Decoder	1358	3173

It is seen from Table 2 that the decoder takes almost 4 times as much slice flip-flops than the encoder. The LUT number however remains similar.

Table 3: Power Consumption figures

Platform	***Quiescent Power***	***Dynamic Power***
Xilinx Encoder	37 mW	4 mW
Xilinx Decoder	56 mW	7 mW

The power analysis in Table 3 shows that the quiescent power consumption is the largest contributor to the overall power consumption of both the encoder and the decoder for the Xilinx implementations. This power consumption is inherent in the FPGA and changes to the switching activity or the design will have little effect overall. This large power consumption can take away from some of the benefits of using data compression when implemented into a low power system.
This power consumption can be further improved if one would use flash based FPGAs such as Actel Igloo or Xilinx CoolRunner. This would allow for the quiescent power to be minimised (in the regions of μW), and by their design, further reduce the dynamic power.
Such a system composed of low power codec with very low static power (< 1mW) and low dynamic power (<5 mW) can be employed for wireless transmission scenarios for saving power. Considering a low power transceiver (such as the Nordic NRF2401), the current consumption in communicating at a power level of -10 dBm is typically 9 mA. With a supply voltage of 2V, the power consumption is 18 mW. Since the codec achieves an average CR of almost 50 %, this means that the transceiver can operate half as long as it normally would without compression. Therefore the average system power consumption for the codec and transceiver combination is 12 mW.

IX CONCLUSION

Hardware and software implementations of arithmetic codec for lossless compression of EEG data were presented.
For the first analysis, an 8 bit input implementation of the algorithm was investigated. The 16 bit input was split into two 8bit symbols so that the size of the model could be reduced from 65536 down to 256 entries. However by splitting the information, there was an increase in the entropy and as a result the maximum compression ratio was reduced to 0.22.
When using a 16 bit input version, the compression ratio achieved on average of 0.31. This increase in the compression ratio is at the cost of the increased size of the LUT and the model.
The complexity of the encoder is much smaller than that of the decoder. This makes feasible the implementation of the encoder on resource constrained sensor node while decoding can be done at a base station which has more resources.
The power consumption of the design was large on the Xilinx FPGAs, with much of the total power consumption resulting from the FPGA's quiescent power consumption. However, it is envisioned that the use of a low power FPGA/hardware would allow the power consumption to be sufficiently low to warrant the use of the codec in a body area network

scenario. Further power reductions are expected if one would target low power ASIC technologies.

AKNOWLEDGEMENTS

We would like to acknowledge the support of SFI through grant number 07/SRC/I1169.

REFERENCES

[1] Shannon C. E. "A Mathematical Theory of Communication" The Bell System Technical Journal, Vol. 27, July, October, 1948, pp. 379-423 and 623-656

[2] Yates, D.C.; Rodriguez-Villegas, E.;, "A Key Power Trade-off in Wireless EEG Headset Design," Neural Engineering, 2007. CNE '07. 3rd International IEEE/EMBS Conference on , , pp.453-456

[3] Khalid Sayood, Introduction to data compression, 3rd ed. 2005, Morgan Kaufmann.

[4] Marnane, W.P., Faul, S., Bleakley, C., Conway, R., Jones, E., Popovici, E., de la Guia Solaz, M., Morgan, F. and Patel, K., "Energy Efficient On-Sensor Processing In Body Sensor Networks,", 32nd Annual International IEEE EMBS Conference, Buenos Aires Sheraton Hotel, Buenos Aires, Argentina, August 31-September 4, 2010, pp. 2025-2029.

[5] Neeraj Magotra, Giridhar Mandyam, Mingui Sun, Wes McCoy "Lossless Compression of Electroencephalographic (EEG) Data" Circuits and Systems, 1996. ISCAS '96. vol.2, no., pp.313-315

[6] Albert-Ludwigs-Universitat Freiburg, Available at: https://epilepsy.unifreiburg.de/freiburg-seizure-prediction-project/eeg-database.

[7] Cliff Wootton, A Practical Guide to Video and Audio Compression, Elsevier, 2005.

[8] Wayner, Peter C., Data Compression for Real Programmers, Morgan Kaufmann, 2000.

[9] Paul G. Howard and Jeffrey Scott Vitter "Practical Implementations of Arithmetic Coding" Brown University, Department of Computer Science, Technical Report No. 92-18, April 1992.

[10] Ian H. Witten, Radford M. Neal, and John G. Cleary "Arithmetic Coding for Data Compress", Communications of the ACM, June 1987, Volume 30, Number 6, pp.520-540.

[11] Eric Bodden, Malte Clasen and Joachim Kneis "Arithmetic Coding revealed - A guided tour from theory to praxis" Sable Technical Report No. 2007-5, McGill University School of Computer Science Sable Research Group.