Hardware Architecture for Simultaneous Arithmetic Coding and Encryption

Amit Pande
Department of Computer Science
University of California
Davis, CA, USA
Email: amit@cs.ucdavis.edu

Joseph Zambreno
Electrical and Computer Engineering
Iowa State University
Ames, IA, USA
Email: zambreno@iastate.edu

Prasant Mohapatra
Department of Computer Science
University of California
Davis, CA, USA
Email: prasant@cs.ucdavis.edu

***Abstract*—Arithmetic coding is increasingly being used in upcoming image and video compression standards such as JPEG2000, and MPEG-4/H.264 AVC and SVC standards. It provides an efficient way of lossless compression and recently, it has been used for joint compression and encryption of video data. In this paper, we present an interpretation of arithmetic coding using chaotic maps. This interpretation greatly reduces the hardware complexity of decoder to use a single multiplier by using an alternative algorithm and enables encryption of video data at negligible computational cost. The encoding still requires two multiplications. Next, we present a hardware implementation using 64 bit fixed point arithmetic on Virtex-6 FPGA (with and without using DSP slices). The encoder resources are slightly higher than a traditional AC encoder, but there are savings in decoder performance. The architectures achieve clock frequency of 400-500 MHz on Virtex-6 xc6vlx75 device. We also advocate multiple symbol AC encoder design to increase throughput/slice of the device, obtaining a value of 4.**

I. Introduction

The state-of-the-art video coding standards such as SVC (technically Annex G of MPEG-4/H.264 AVC) [1] is been widely adopted in current video application systems due to its outstanding coding performance, and scalable properties which allow deployment in fluctuating channel conditions and to serve heterogeneous clients. There are three entropy coding tools adopted in H.264/SVC. One is Context-based Adaptive Binary Arithmetic Coding (CABAC), based on arithmetic coder. The other are Context-based Adaptive Variable Length Coding (CAVLC) and Exp-Golomb coding (to code syntax elements). CABAC can achieve averaged bit-rate savings of 9% to 14% at the cost of higher computational complexity in comparison to CAVLC. However, the increased computational complexity and strong data dependencies significantly restrict the throughput of CABAC decoder. This restriction becomes a challenge in hardware design of CABAC coder making CAVLC more suitable for decoding in low-power embedded systems.

Arithmetic coding is a data compression technique that encodes data by creating a code string which represents a fractional value on the interval [0, 1). When a string is compressed using arithmetic coder, frequently-used characters are stored with fewer bits and not-so-frequently occurring characters are stored with more bits, resulting in fewer bits used in total [2]

This paper discusses arithmetic coding from a slightly different perspective. Recent work has established how arithmetic coding can be viewed as an iteration on piece-wise linear chaotic maps [3], [4]. Further, many researchers have studied the use of arithmetic coding for joint encryption and compression [5], [6], [7]. For example- In [8], a chaos-based adaptive arithmetic coding technique was proposed. The arithmetic coder's statistical model is made varying in nature according to a pseudo-random bitstream generated by coupled chaotic systems. Many other techniques based on varying the statistical model of entropy coders have been proposed in literature, however these techniques suffer from losses in compression efficiency that result from changes in entropy model statistics and are weak against known attacks [9]. Recently, Grangetto et al. [5] presented a Randomized Arithmetic Coding (RAC) scheme which achieves encryption by inserting some randomization in the arithmetic coding procedure at no expense in terms of coding efficiency. RAC needs a key of length 1-bit per encoded symbol. Kim et al. [6] presented a generalization of this procedure, called as Secure Arithmetic Coding (SAC). The SAC coder builds over a Key-Splitting Arithmetic Coding where a key is used to split the intervals of an arithmetic coder, adding input and output permutation to increase the coder's security.

In this paper, we extend this discussion to hardware community - to study the hardware optimizations in design of such schemes. Particularly, we study the implementation of arithmetic coding using piece-wise chaotic maps [3], [4]. As we shall study, this implementation has lower decoder requirements than the commercial implementations. Apart from these, chaotic maps have also been used in cryptography and for pseudo random number generation [10].

The reduced decoding efficiency of arithmetic coding allows it to trend towards the low computational complexity of Huffman coders, allowing BAC to enter embedded systems market. The aspects of context-modeling and adaptation and renormalization, as done in CABAC coder are beyond the scope of this work, where we focus on architectural optimizations on encoder and decoder processes.

Why another design?

An inquisitive question which comes to mind at this point is the need for hardware implementation of chaotic maps. When arithmetic coding is already been done using traditional ways, why do we need yet another architecture?

The motivation to develop a hardware architecture for chaotic maps iterations is summarized below:

1) Arithmetic coding done using chaotic maps is asymmetric in nature, (explained in later sections) making the decoder architecture simpler than existing framework for AC. The reduced decoder complexity is highly desired to reduce the power and computational requirements of video decoding in low power mobile devices. Current mobile video profiles use Huffman coding instead of Arithmetic coding to reduce the computational complexity, which leads to average compression inefficiency of 15%, particularly poor performance in coding events with symbol probabilities greater than 0.5, due to the fundamental lower limit of 1 bit/symbol on Huffman coding [11].
2) Recently, arithmetic coding based encryption schemes have been proposed in research literature for joint compression and encryption purposes [7], [12]. It would be interesting to integrate both coding and encryption using chaotic maps at a computational complexity lower than existing implementations. This motivates the need of coding and encryption architecture using chaotic maps.
3) Chaotic maps can be used to Pseudo-Random Number Generation (PRNG) [10] and stream ciphers [13], apart from arithmetic coding. These have been found to be light weight and simple.

Contributions

The main contributions of this paper are as follows:

1) We introduce arithmetic coder architecture using chaotic maps which has potential advantages in reducing decoder complexity and allows combined encryption.
2) We present two architectures for FPGA implementation of the proposed scheme: one using explicit multipliers from DSP48E1 slices on Virtex-6 FPGA, while other using reconfigurable multipliers and mapping to hardware 6-LUTs.
3) We advocate the multiple-symbol encoding which makes sense for throughput/ area.

Scope of the work

In the regular coding mode, prior to the actual arithmetic coding process the given binary data enters the context modeling stage, where a probability model is selected such that the corresponding choice may depend on previously encoded syntax elements. Then, after the assignment of a context model, the bin value along with its associated model is passed to the regular coding engine, where the final stage of arithmetic encoding together with a subsequent model updating takes place (see Figure 1). We shall restrict the focus of further discussions on the final arithmetic encoding (and decoding) stages of CABAC coder.

II. Literature Review

Adaptive minimum-redundancy (Huffman) coding is expensive in both time and memory space, and is handsomely outperformed by adaptive AC besides the advantage of AC in compression effectiveness [14]. Fenwicks structure requires just n words of memory to manage an n-symbol alphabet, whereas the various implementations of dynamic Huffman coding [15], [16] consume more than 10 times as much memory [17].

Hardware architectures have been proposed in research literature for arithmetic coding using CACM model [18] or related works [14], [19], [20]. CABAC or Context-Adaptive Binary Arithmetic Coder is used in H.264 AVC and SVC. The critical path of coder is the multiplier, which is removed in CABAC and recent implementations [21], [22], [23] by using a look-up approximation (leading to some compression inefficiency).

There has been little work [24], [25], however, in implementation of chaotic maps on hardware. However, the recent trend toward joint compression and encryption using chaotic maps and arithmetic coding for low power embedded systems would be greatly complimented by an efficient hardware architecture, as presented in this paper.

Binary Arithmetic Coding (BAC)

Binary arithmetic coding is based on the principle of recursive interval subdivision. We start with an initial interval [0,1) and keep dividing it into subintervals based on the probability of incoming symbols. A good detailed overview of BAC is presented in [14]

III. How do we interpret AC using Chaotic Maps?

A description of equivalence between binary arithmetic coding and chaotic maps is given in earlier works [4], [7]. In this section, we gave a brief overview of N-alphabet arithmetic coding to familiarize the reader with coding using piece-wise linear chaotic maps.

Scenario: We have a string $S = x_1, x_2, ...x_M$ consisting of M symbols (N unique symbols) to be encoded. The probability of occurrence of a symbol $s_i,\ i \in 1, 2, ...n$ is given by p_i such that $p_i = N_i/N$ and N_i is the number of times the symbol s_i appears in the given string S.

Description: Consider a piece-wise linear map (ρ) with the following properties:

- It is defined on the interval $[0, 1)$ to $[0, 1)$ i.e.

$$\rho :\ [0, 1) \longrightarrow [0, 1)$$

- The map can be decomposed into N piece-wise linear parts ϱ_k i.e.

$$\rho = \bigcup_{k=1}^{N} \varrho_k$$

- Each part ϱ_k maps the region on x axis $[beg_k, end_k)$ to the interval $[0, 1)$ i.e.

$$\varrho_k :\ [beg_k, end_k) \longrightarrow [0, 1]$$

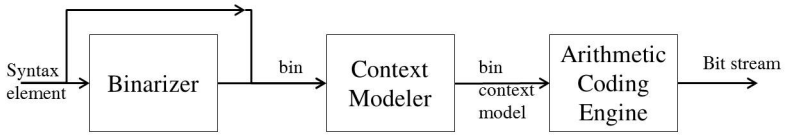

Fig. 1. Block diagram of CABAC coder

The last two propositions lead to:

$$\bigcup_{k=1}^{N}[beg_k, end_k) = [0,1)$$

- The map ϱ_k is one-one and onto i.e.:

$$\forall x \in [beg_k, end_k)$$
$$\exists y \in [0,1) : y = \varrho_k(x), \text{ and}$$
$$\forall y \in [0,1)$$
$$\exists x \in [beg_k, end_k) : \varrho_k(x) = y$$

- ρ is a many-one mapping from $[0,1)$ to $[0,1)$. This implies that the decomposed linear maps (ϱ_k) don't intersect each other i.e.

$$\forall (k \neq j) : [beg_k, end_k) \bigcap [beg_j, end_j) = 0$$

- Each linear map ϱ_k is associated uniquely with one symbol s_i. The mapping $\varrho_k \longrightarrow s_i$ is defined arbitrarily but one-one relationship must hold.
- The valid-input width of each map (ϱ_k), given by ($end_k - beg_k$) is proportional to a probability of occurrence of symbol s_i.

$$end_k - beg_k \propto p_i$$
$$\Rightarrow end_k - beg_k = C \times p_i$$

We recall that $\sum_{k=1}^{N}(end_k - beg_k)$ is same as the input width of $\bigcup_{k=1}^{N} \varrho_k = \rho$, which is 1. Also, $\sum_{i=1}^{N} p_i = 1$. Thus, we get the value of constant C to be 1.

$$\Rightarrow end_k - beg_k = p_i$$

Figure 2 shows a sample map fulfilling these properties. Figure 2(a) shows the full map with different parts $\varrho_1, \varrho_2, ...\varrho_N$ present while Figure 2(b) zooms into individual linear part ϱ_k. The maps are placed adjacent to each other so that each input point is mapped into an output point in the range $[0,1)$.

Encoding/ Decoding

The decoding process is quite simple. The encoded value is considered as an initial value IV. This value is iterated over the piece-wise linear map ρ, M times to get M iterated values IV_i. Each value is mapped to piece-wise linear part ϱ_i and thus to corresponding s_i.

The encoding process is done by reversing the input string to $x_M, x_{M-1}, ...x_1$. Each input character is mapped to unique symbols s_i and then to piece-wise linear maps ϱ_i. Thus, we get a sequence of piece-wise linear maps corresponding to input string $\varrho_{x_M}, \varrho_{x_{M-1}}...\varrho_{x_1}$. We start with the initial interval [0,1) and back-iterate this interval over chaotic maps using the string $\varrho_{x_M}, \varrho_{x_{M-1}}...\varrho_{x_1}$ to get a final interval. The output codeword is chosen as the shortest binary number from final interval.

Compression Efficiency and Equivalence

Arithmetic coding has been shown to be achieve Shannon's limit on compression efficiency asymptotically. The same result holds true for coding using piecewise linear maps because of the following observations:

The width of final interval is given by $\lceil \prod_{i=1}^{N} f_i^{n_i} \rceil$, where f_i is the probability of occurance of symbol s_i, and n_i is the frequency of occurance of symbol s_i. This value asymptotically approaches Shannon's value for maximum entropy []. It can be observed that while CAC scales the codeword or initial value to map them to the intervals corresponding to different symbols, the standard arithmetic coder keeps the codeword constant and instead scales the map in every iteration to find the symbol. It is immaterial - whether one scales the map to suit the codeword or scales the codeword to suit the map - the relative ratios remain the same, hence output of both procedures is the same.

Use of Chaotic Maps in Encryption

[12], [7] present two different scenarios of using chaotic maps for arithmetic encryption. The first case uses N-ary arithmetic coding and has high cryptographic strength and implementation cost, while the second case uses binary arithmetic coding to encrypt data with low computational resources. In both the cases, the choice of multiple piece-wise linear maps to encode the input symbol is used for key generation. This property is used for encryption, for without knowledge of the correct map, an adversary cannot decode the input stream correctly.

A. Applications

The CAC can be used as a joint compression-cum-encryption technique for data encryption. It is particularly beneficial for data-intensive tasks such as multimedia encryption and compression and can be integrated into the standard video compression algorithms such as JPEG2000, JPEG, MPEG etc.

CAC can be used for full or selective encryption of multimedia data. For full encryption, the entire volume of multimedia data is passed through BCAC (Binary CAC) encoder while in case of selective encryption only the important parts of data are passed through BCAC encoder. If we reveal the first K bits of the key publicly, then a part of the bitstream can

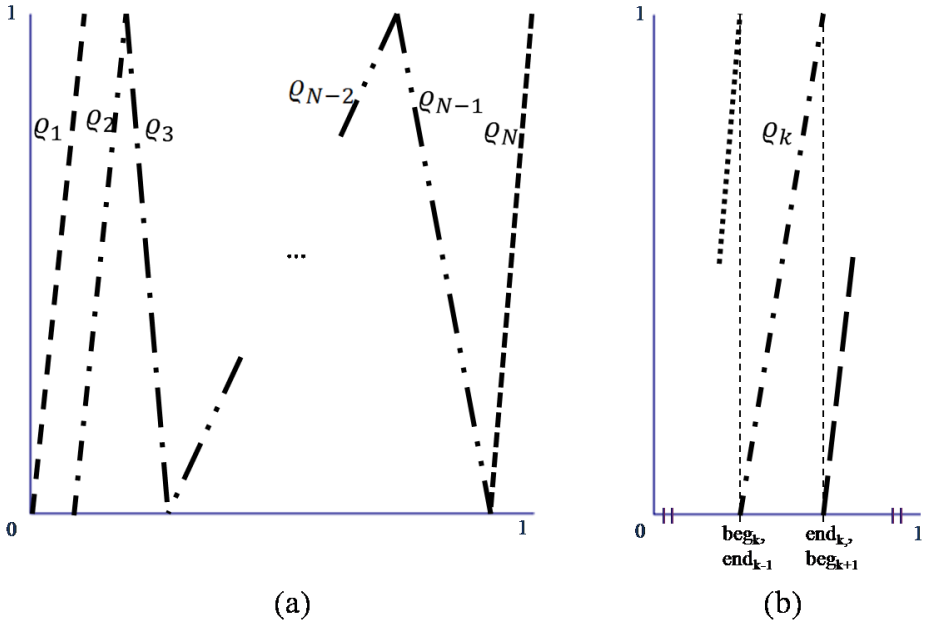

Fig. 2. A sample piece-wise linear map for arithmetic coding like compression (a) The entire map is shown (ρ) (b) A single linear part of the map (ϱ_k) is zoomed. It can have a positive or negative slope depending on choice

be decoded correctly while decoding the entire bitstream will require knowledge of the entire key. Thus, BCAC can be used to provide conditional access to the multimedia content.

Implementation Efficiency

For a normal binary arithmetic coder, at each iteration the starting interval $[I_s, I_e)$ is updated at one end. On encoding a '0' the final interval becomes $[I_s + p(I_e - I_s), I_e)$ while on encoding a '1' the final interval becomes $[I_s, I_s + p(I_e - I_s)$. Thus, every iteration requires one multiplication and two addition operations. The decoding procedure for a binary arithmetic coder involves updating the interval $[I_s, I_e)$ at one end depending on whether the last decoded symbol was a '0' or a '1'. Thus, every iteration again requires one multiplication and two addition operations.

For chaotic arithmetic encoder, both end of interval are updated at every iteration using a linear transformation $x = my+c$ thus requiring two multiplications and two additions for encoding. The decoding is simple as it involves iteration on the chaotic map according to the linear transformation $y = nx+c$ involving a multiplication and an addition operation. There are some additional table lookups (an 8-input LUT required for BCAC to choose the exact chaotic map) involved in chaotic coding to choose the right chaotic map at every iteration which can be efficiently implemented in software or hardware. Thus, CAC encode requires more computations than BAC encode while CAC decode requires less computations than BAC decode. [11] present a multiplier-free binary arithmetic coder, called as modulo coder (M-coder) which can be shown to have negligible performance degradation by some interval approximations. Our BCAC coder is similar to a BAC coder except the variable slope and intercept of the lines of chaotic map which is decided by choice of map. These values can be mapped to look-up tables and the remaining operation can be optimized similar to BAC. However, we skip this detail in this paper for the sake of brevity.

IV. Hardware architecture

In this section, we discuss the hardware architecture for arithmetic coding using chaotic maps, and N-ary chaotic arithmetic encryption.

The chaotic encoder operation inverse inverse mapping of interval [0,1) on the chaotic map according to input symbol. For binary arithmetic coder, we have a fixed map to be iterated in each cycle.

Figure 3(a) shows the basic architecture for coding using chaotic maps. The control unit receives the input bit stream, which is passed on to the chaotic map Iterator (CMI). The control unit passes the bitstream, one symbol per cycle (unless in the case of multiple symbol encoding, which will be discussed later). For encoding, the initial interval passed to CMI is [0,1), which is transmitted as the beginning (B_n) and end (E_n) interval values. Both the intervals are then iterated over CMI (using two instances of CMI), and then the output is sorted so that $B_n < E_n$. If the difference ($D_n = E_n - D_n$) is lower than a threshold, we need to renormalize the encoder. The renormalization procedure for arithmetic coding has been discussed in [14]. A similar extension of renormalization procedure may be possible for chaotic maps. But, for the evaluation designs considered in this work, we have considered 64 bit encoder without any renormalization procedure.

In case of decoding, Control Unit (CU) transmits the coded symbol into CMI, which is then iterated over Piece-wise linear map and reported back to CU. The CU makes a comparison

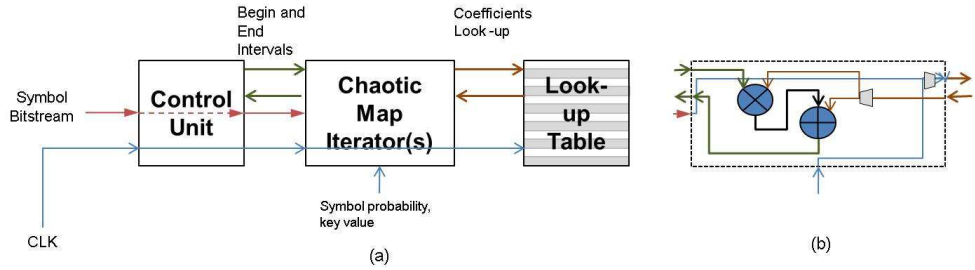

Fig. 3. Generalized Hardware Architecture for Chaotic Maps. (a) Generalized architecture and (b) Circuit details for Chaotic map Iterator

with chaotic map indicated by the key and outputs a single bit output.

CMI has a multiplier and an adder to perform chaotic iteration. The internal details of this operation are given in Figure 3(b). The multiplication and addition coefficients are obtained from a look-up table/ RAM collating the input symbol, key value and probability value as the input address. The Look-ed up value or a word is demultiplexed to obtain the multiplication and addition coefficients. This option can work fine for at most binary case, and for the case where p value is limited to fixed precision, say 8 bits. Such fixed precision approximations have been introduced in CABAC [11], however it leads to approximation of results. Alternatively, we can use a multiplexer which can implement look-up using physical circuits to compute the return values. The second approach has been implemented in this work, as it allows more flexibility in design and accuracy in computation.

For implementation, the input and output intervals to the Chaotic Map Iterator are represented in 64 fixed point (0 bits integer and 64 bits fraction, shortly I.F 0.64) arithmetic. The symbol probability has been quantized to 8 bits (I.F 0.8).

Binary Arithmetic Coder (BAC) architecture

To implement BAC in proposed architecture, we target a design with processes 1 symbol (1 bit in this case) per cycle. The CMI has 1 bit symbol input, 8 bit symbol probability and no bits for choice of chaotic map (there is only one map in this case). The 9 bit lookup can be implemented using a 512 words RAM or Look-up Table. One word is 16 bits - 8 bits each for multiplication and addition coefficients. Alternatively, this can be implemented using a multiplexer and hardware adder/ subtracter to obtain the coefficients. The later approach was used for BAC implementation. The design was synthesized in Xilinx Virtex-6 XC6VLX75t FPGA using Xilinx ISE Design Suite 12.0 environment. The same target FPGA, which is one of the low end Virtex-6 family member is used in all synthesis/ translate/ map/ place and routes.

The two 64x8 bit multiplications are mapped in hardware into 10 DSP48E slices. A slice usage of 302 was obtained and the design achieved a clock frequency of 510 MHz, with one symbol per clock cycle. The optimized implementation of multiplication, using carry-chains of FPGA fabric was synthesized to remove the use of DSP slices. This implementation requires 1585 slices and achieves a clock frequency of 500 MHz. The throughput of this implementation is 1 bit per cycle with a 500 MHz clock, i.e. 500 Mbps.

Binary Chaotic Arithmetic Coder and Encryption (BCAC) architecture

The architecture for BCAC differs from binary arithmetic coder in the sense that, the choice of chaotic map is made based on a key value, and is not precomputed. For this implementation, the CMI has 1 bit symbol input, 8 bit symbol probability and 3 bits for choice of chaotic map (for binary case $N = 2$, hence number of different chaotic maps is $N2^N = 8$. The 12 bit lookup can be implemented using a 512 words RAM or Look-up Table, with 16 bits word. Alternatively, we used 8-to-1 multiplexer to obtain the coefficients corresponding to a key, each cefficient being generated based on value in Table 1 in [7]. The implementation on target FPGA gave a clock frequency of 500 MHz, utilizing 321 slices and 10 DSP48E1 slices (which have optimized multiplier and accumulator operation implemented in VLSI). Mapping these multiplication to FPGA logic increased the slice usage to 1474, without any change in achievable clock frequency.

The BCAC decoder hardware utilization was 173 slice LUT with 5 DSP slices (806 slice LUTs with LUT multiplier) with a clock frequency of 510 MHz (500 MHz). The 64x8 bit multiplier is implemented by ISE into 5 DSP slices. However, the same multiplier can be optimized and implemented without hardware multipliers using other multiplier such as square root multiplier, reconfigurable constant multipliers etc. The hardware requirements are basically dependent on size of Look-up logic which increases exponentially with increase of N. The throughput of this implementation is 1 bit per cycle with a 510 MHz clock, i.e. 510 Mbps. To consider the area effectiveness of this design, we consider throughput per slice, with the second implementation where we implement multiplication in LUTs rather than using DSP48E1 slices present in device. The throughput/ slice for this design is obtained as 322 Kb/slice.

Cost of encryption

Comparing the BAC and BCAC architectures, we obtain a zero latency, same throughput and little hardware overhead (20

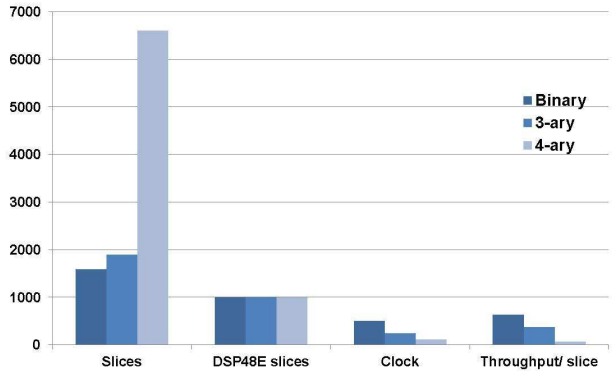

Fig. 4. N-ary arithmetic coding and encryption architectures: Comparative performance. The # of slices, # of DSP slices (x100), clock frequency (MHz) and throughput per slice (x1000) are reported in the figure. It can be observed that increasing the size of dictionary significantly reduces the throughput. The figure is drawn by scaling the throughput/slice legend to consider the fact that a 4 symbol dictionary will require half the words as a 2 symbol dictionary.

slice LUTs) in implementing this encryption scheme against AES or other schemes which have significant overhead. For instance,Chang et al. [26] reports AES implementation using 156 slices, 2 Block RAMs to obtain a lower clock of 306 MHz.

To increase the throughput per slice for a bitstream, we intuitively consider the dimension of increasing the number of symbols in dictionary used in arithmetic coding. For example - considering 3 or 4 symbols in the dictionary.

N-ary Chaotic Arithmetic Coder and Encryption (NCAC) coding

N-ary arithmetic encryption using the entire possible key space quickly turns out-of-bounds for a FPGA device. Moving from 2 to 3 piece-wise linear maps, we have a tremendous increase in key-size. We implemented tri-nary CAC coder in FPGA device to obtain a device usage of 492 slices and 10 DSP48E slices (1800 slices without DSP slices), but the achievable clock frequency dropped to 127 MHz. The tri-nary decoder hardware utilization was 419 slice LUT with 5 DSP slices (1052 slice LUTs with LUT multiplier) with a clock frequency of 442 MHz (369 MHz). The hardware requirements are basically dependent on size of Look-up logic which increases exponentially with increase of N ($N|!2^N$), making it infeasible to scale-up the throughput/slice.

A simple way to restrict this bandwidth explosion is to used the algorithm for encryption proposed in [12]. They restrict the keyspace and instead use only a small fragment of keys from the entire range, for encryption. However, the approach presented in [12] has other computationally-inefficient parts.

The results are shown in Figure 4. The number of slice LUTs is reported directly, number of DSP slices is scaled directy and clock frequency is measured in MHz. The throughput comparison is tricky because using a 4-symbol dictionary (4-ary coding) will lead to reduced bitstream (around 50% reduction) than the bitstream generated by 2-symbol dictionary. Thus, to compare these values on a graph, we multiply each throughput with N value (2 for binary) to indicate relative throughput. It can be observed that increasing the size of dictionary significantly reduces the throughput, even after such considerations due to exponential increase in hardware usage for key implementation.

Although our experiment to scale to multiple-symbol dictionary failed, the reason is not the same as for traditional designs for arithmetic coding [11]. Rather, the key explosion is the main reason for such limitations. We next consider increasing the system throughput by encoding multiple binary symbols in a single pass. This approach is different than the previous approach in the sense that multiple probability values are not involved.

Multiple symbol per cycle arithmetic coding

Let us consider the case of arithmetic coding where we want to encode two symbols in a single iteration of chaotic map. In this case, the chaotic map will spit into multiple (four instead of two) piece-wise maps. Arithmetic coding with encryption is still going to suffer with band-width expansion, but we observe that the bandwidth expansion is much less (or order of 2^N) instead of $N2^N$. Consider, for example the case where we want to encode two symbols together (‘01’ instead of ‘0’ and ‘1’ in two separate iterations) using BAC. In this case, the resultant chaotic iterator will have 4 (instead of 2) piece-wise linear maps and their precision of implementation will be increased (16 instead of 8 bits). This analysis can be extended to three, four or more symbols.

In this case, the increase is caused by increase in fixed point precision of coefficients (and hence multipliers and adders), and increase in number of piece-wise maps. However, against the case of MCAC where there was a bandwidth explosion due to increase in key size, we observe a considerable different result of implementation on Virtex-6 device. These results are reported in Figure 5. The results are interesting to note, because contrasting with the traditional notion of one-symbol per cycle, we show that we can scale upto 4 symbols per cycle and achieve a higher throughput per slice. As we go from 2 to 4 case, we observe a increase in throughput which is then checked by the exponential increase in hardware resources caused by multiple symbols use. This value of 4 cannot be a device constraint (restrictions due to finite area or size of device) because the pure LUT mapping based implementation requires only 5480 slices out of 43000 slices present in target xc6vls75 device. The highest throughput achievable is 431 Kbits per slice for 4 symbols case.

For the sake of brevity, we have restricted our discussion in last sections to NCAC and multiple symbol BAC encoder, but the same trend follows for the decoder also.

V. Conclusion

In this paper, we presented architecture for simultaneous coding and encryption using chaotic maps. After presenting the hardware requirements and computations involved in chaotic maps, we mapped these designs into a Virtex-6 FPGA to obtain

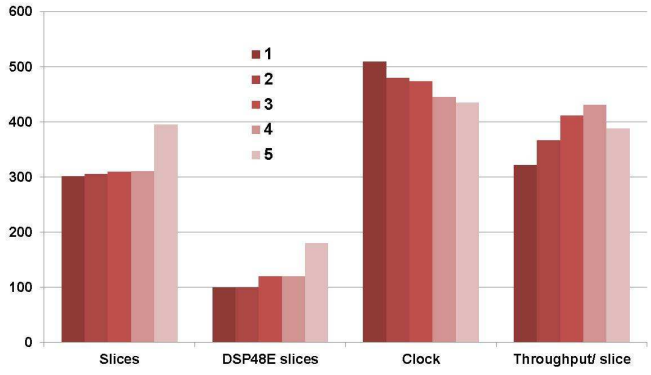

Fig. 5. Multiple symbols per cycle (BAC): Comparative performance. The # of slices, # of DSP slices (x10), clock frequency (MHz) and throughput per slice (x1000) are reported in the figure. It can be observed that 4 symbols per cycle achieve highest throughput before LUT explosion due to increased precision and maps.

a performance analysis on real hardware. We investigated the key-explosion problem which avoided the implementation of simultaneous coding and encryption using larger dictionaries. However, we found that the hardware resource explosion is not much in case of multiple character coding using BAC (indicating 5 symbols be encoded simultaneously). This work is one of the earliest hardware implementation of chaotic maps, first reported implementation of chaotic maps for simultaneous coding and encryption. It achieves encryption at insignificant hardware cost, against use of encryption ciphers such as AES which require separate modules for encryption operation.

We are looking for, and encourage other readers also for future work in two directions:

1) Looking for ways to solve key-explosion problem using circuit level techniques.
2) Incorporating re-normalization and context to this encoder, so that it can be added to CABAC or other encoders.

ACKNOWLEDGEMENT

This research is supported by the National Science Foundation under Grant #1019343 to the Computing Research Association for the CIFellows Project.

REFERENCES

[1] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H. 264/AVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103–1120, 2007.

[2] G. Langdon and J. Rissanen, "Compression of black-white images with arithmetic coding," *IEEE Trans. Communications*, vol. 29, no. 6, pp. 858–867, Jun 1981.

[3] M. Luca, A. Serbanescu, S. Azou, and G. Burel, "A new compression method using a chaotic symbolic approach," in *Proc. IEEE Commun. Conf.* Citeseer, 2004, pp. 3–5.

[4] N. Nagaraj, P. Vaidya, and K. Bhat, "Arithmetic coding as a non-linear dynamical system," *Communications in Nonlinear Science and Numerical Simulation*, vol. 14, no. 4, pp. 1013–1020, 2009.

[5] M. Grangetto, E. Magli, and G. Olmo, "Multimedia selective encryption by means of randomized arithmetic coding," *IEEE Trans. Multimedia*, vol. 8, no. 5, pp. 905–917, Oct. 2006.

[6] H. Kim, J. Wen, and J. Villasenor, "Secure arithmetic coding," *IEEE Trans. Signal Processing*, vol. 55, no. 5, pp. 2263–2272, May 2007.

[7] A. Pande, J. Zambreno, and P. Mohapatra, "Joint video compression and encryption using arithmetic coding and chaos," in *IEEE International Conference on Internet Multimedia Systems Architecture and Application*, 2010.

[8] R. Bose and S. Pathak, "A novel compression and encryption scheme using variable model arithmetic coding and coupled chaotic system," *IEEE Trans. Circuits and Systems I*, vol. 53, no. 4, pp. 848–857, April 2006.

[9] G. Jakimoski and K. Subbalakshmi, "Cryptanalysis of some multimedia encryption schemes," *IEEE Trans. Multimedia*, vol. 10, no. 3, pp. 330–338, April 2008.

[10] T. Stojanovski and L. Kocarev, "Chaos-based random number generators-part I: analysis [cryptography]," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 48, no. 3, pp. 281–288, 2002.

[11] D. Marpe, H. Schwarz, G. Blttermann, G. Heising, and T. Wieg, "Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, pp. 620–636, 2003.

[12] K.-W. Wong, Q. Lin, and J. Chen, "Simultaneous arithmetic coding and encryption using chaotic maps," *IEEE Trans. Circuits and Systems*, vol. 57, pp. 146–150, February 2010. [Online]. Available: http://dx.doi.org/10.1109/TCSII.2010.2040315

[13] S. Lian, J. Sun, J. Wang, and Z. Wang, "A chaotic stream cipher and the usage in video protection," *Chaos, Solitons & Fractals*, vol. 34, no. 3, pp. 851–859, 2007.

[14] A. Moffat, R. Neal, and I. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems (TOIS)*, vol. 16, no. 3, pp. 256–294, 1998.

[15] G. Cormack and R. MORSPOOL, "Algorithms for adaptive Huffman codes," *Information Processing Letters*, vol. 18, no. 3, pp. 159–165, 1984.

[16] J. Vitter, "Design and analysis of dynamic Huffman codes," *Journal of the ACM (JACM)*, vol. 34, no. 4, pp. 825–845, 1987.

[17] A. Moffat, N. Sharman, I. Witten, and T. Bell, "An empirical evaluation of coding methods for multi-symbol alphabets," *Information Processing & Management*, vol. 30, no. 6, pp. 791–804, 1994.

[18] I. Witten, R. Neal, and J. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[19] P. Howard and J. Vitter, "Analysis of arithmetic coding for data compression," *Information Processing & Management*, vol. 28, no. 6, pp. 749–763, 1992.

[20] G. Langdon, "An introduction to arithmetic coding," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, 1984.

[21] R. Osorio and J. Bruguera, "Arithmetic coding architecture for H. 264/AVC CABAC compression system," 2004.

[22] T. Chuang, Y. Chen, Y. Chen, S. Chien, and L. Chen, "Architecture Design of Fine Grain Quality Scalable Encoder with CABAC for H. 264/AVC Scalable Extension," *Journal of Signal Processing Systems*, vol. 60, no. 3, pp. 363–375, 2010.

[23] C. Lo, S. Tsai, and M. Shieh, "Reconfigurable architecture for entropy decoding and inverse transform in H. 264," *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 3, pp. 1670–1676, 2010.

[24] T. Addabbo, M. Alioto, A. Fort, S. Rocchi, and V. Vignoli, "Low-hardware complexity prbgs based on a piecewise-linear chaotic map," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 53, no. 5, pp. 329 – 333, May 2006.

[25] A. Pande and J. Zambreno, "Design and hardware implementation of a chaotic encryption scheme for real-time embedded systems," in *Signal Processing and Communications (SPCOM), 2010 International Conference on*. IEEE, 2010, pp. 1–5.

[26] C.-J. Chang, C.-W. Huang, K.-H. Chang, Y.-C. Chen, and C.-C. Hsieh, "High throughput 32-bit aes implementation in fpga," in *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, 30 2008.