Partial Encryption of Compressed Images Employing FPGA

M. B. I. Reaz, F. Mohd-Yasin, S. L. Tan, H. Y. Tan
Faculty of Engineering
Multimedia University
63100 Cyberjaya, Selangor, Malaysia
mamun.reaz@mmu.edu.my

M. I. Ibrahimy
Dept. of Electrical and Computer Engineering
International Islamic University Malaysia
53100 Jalan Gombak, Kuala Lumpur, Malaysia

Abstract— **In this paper, we present the realization of of partial encryption of compressed images on Altera FLEX10K FPGA device that allows for efficient hardware implementation. The compression algorithm decomposes images into several different parts. A secure encryption algorithm is then used to encrypt only the crucial parts, which are considerably smaller than the original image. This will result in significant reduction in processing time and computational requirement for encryption and decryption. The breadth-first traversal linear lossless quadtree decomposition method is used for the partial compression and RSA is used for the encryption. Functional simulations are carried out to verify the functionality of the individual modules and the system on four different images. Comparisons, verification and analysis made validate the advantage of this approach. The design has utilized 2928 units of LC and a system frequency of 13.42MHz.**

I. Introduction

The rapid growth of image and video communication nowadays is powered by ever-faster systems demanding greater speed and security. Real-time secure image and video communication is challenging due to the processing time and computational requirement for encryption and decryption. In order to cope with these concerns, innovative image compression and encryption techniques are required.

Although a vast number of compression and encryption algorithms exist, they have been traditionally developed independently of each other. A partial encryption scheme for images that takes advantage of the image compression algorithm has been proposed by Cheng and Li in [1], [2] and [3]. The scheme makes use of a compression algorithm that decomposes an image into several different parts. A secure encryption algorithm is then used to encrypt only the crucial parts, which are considerably smaller than the original image. This will result in significant reduction in processing time and computational requirement for encryption and decryption.

Other researchers have also proposed partial encryption, or combined compression and encryption methods. Dang and Chau [4] has proposed the joint image compression and encryption scheme using Discrete Wavelet Transform (DWT) and Data Encryption Standard (DES). Jakobsson *et al.* [5] developed a “Scramble All, Encrypt Small” technique that encrypts only a small block of an arbitrarily long message. However, the former is less efficient than a partial encryption scheme and utilizes an encryption algorithm (DES) that is no longer secure. The latter requires an ideal hash function that is hard to realize and may not be suitable for images as it was designed for data encryption.

Traditionally, image compression and encryption algorithms have been restricted to the software realm and developed separately. Although the advantages of software are ease of update, flexibility and portability, hardware implementation is faster and more physically secure, especially when secret key storage security are concerned.

This work aims to investigate the hardware feasibility and performance of a novel partial encryption scheme for compressed images using FPGA, which combines compression and a secure encryption algorithm that encrypts only crucial parts of the compressed image. The algorithms chosen for implementation are the lossless quadtree compression and the RSA algorithm. The hardware implementation was done using Altera FLEX10KE device.

II. Design Methodology

The partial encryption scheme depends on a compression algorithm that decomposes the input image into a number of different logical parts. The output consists of parts that provide significant amount of information about the original image, referred to as the important parts. The remaining parts have little meaning without the important parts, hence known as the unimportant parts. In this partial encryption approach, only the important part needs to be encrypted by a secure encryption algorithm. When the important part is considerably smaller than the total output of the compression, the encryption and decryption time can be reduced significantly.

A. Quadtree Compression

The quadtree decomposition method converts an image into a quadtree structure with intensity values attached to the leaf nodes of the tree. The quadtree structure reveals the outline of objects in the original image [1]. Since the quadtree indicates the location and size of each homogeneous block in the image while the intensity values do not reveal much information, partial encryption is possible by encrypting only the quadtree structure. Here, the quadtree structure is the important part whereas the intensity values form the unimportant part.

In the case of lossless compression on a b-bit image, the total size of the leaf values is $b(3k+1)$ bits, where k is the number of internal nodes, which is equivalent to multiplying the size of each leaf value with the number of leaf nodes in the quadtree. An approximate upper bound on the relative quadtree size, which is the ratio of the size of the quadtree and the total size of the compressed image, is given in equation 1:

$$\frac{4k+1}{4k+1+b(3k+1)} = \frac{4+\frac{1}{k}}{3b+4+\frac{b+1}{k}} \approx \frac{4}{3b+4} \tag{1}$$

where,

Size of quadtree = number of nodes = $4k+1$

Size of compressed image = size of quadtree + size of leaf values = $4k+1+b(3k+1)$

For 8-bit images, $b=8$, the size of the quadtree relative to the lossless quadtree compression output is at most 14.3%. The approximation is valid for large value of *k*, which is typically at least 1000 for 256×256 images and greater for larger images. For lossy compression, this calculation is not applicable because variable number of bits is used to represent leaf values. Results collected from experiments performed by Cheng H. [2] on test images show that for typical images, the relative quadtree size is between 13% and 27%. Therefore, only 13 to 27% of the output of lossy quadtree algorithm is encrypted for typical images.

The lossless quadtree compression algorithm with Leaf ordering II has been used in this research, as it is computationally simpler and secure.

B. *Linear Lossless Quadtree*

Representing quadtrees in a tree structure requires the use of pointers. However, the amount of space required for pointers from a node to its children is not trivial. Samet [6] suggested that each node in a quadtree is stored as a record containing six fields. The first five fields contain pointers to the node's parent and its four children labeled as NW, NE, SW and SE; whereas the sixth field describes the intensity value (color) of the image block that the node represents. The pointers would occupy nearly 90% of the memory space required to store the quadtree [7]. As a result, several pointerless quadtree representations have been proposed by researchers such as Lin [7], and Gargantini [8].

This research is based on the breadth-first traversal of linear quadtree proposed by Chan [9]. It consists of two lists, i.e. a tree list and a color list. The tree list stores the quadtree structure, where '0' denotes a leaf node and '1' denotes an internal node. The color list simply stores the pixel values of the image in a sequence defined by the tree structure.

C. *RSA Encryption*

Since the encrypted part of the proposed partial encryption scheme is preferably small, public-key algorithms has been applied directly to it.

In RSA a plaintext block *M* is encrypted to a ciphertext block *C* by:

$$C = M^e \bmod n \tag{2}$$

and the plaintext block is recovered by:

$$M = C^d \bmod n \tag{3}$$

RSA encryption and decryption are mutual inverses and commutative, due to symmetry in modular arithmetic. Also, (2) and (3) show that both encryption and decryption are based on the same operation, which is modular exponentiation. Therefore, hardware implementation of RSA allows the encryption and decryption to share the same architecture, which helps reduce the hardware size.

III. VHDL Modeling

The VHDL model for the proposed work consists of four sub-modules. The overall implementation is known as the PARTIAL_ENCRYPT chip and it consists of the functional sub-modules Compression, Encryption, Decryption and Decompression.

A. *Linear Quadtree Compression/Decompression*

Linear quadtree compression and decompression are implemented in two separate blocks, QT_ENCODER and QT_DECODER respectively. The combination of these two functional blocks is named QT_CODEC. The linear quadtree codec connects both QT_ENCODER and QT_DECODER in parallel and to the memory block RAM256X8. There are four input control signals, *i.e.* CLK, RESET, GO and E_D. The architectures of QT_CODEC, QT_ENCODER and QT_DECODER are implemented using Moore state machines with asynchronous reset. The reset signal (RESET) is used to set the state machine to its initial idle state, while a high GO signal switches it from idle state to the next state. A low E_D signal activates the QT_ENCODER while a high E_D activates the QT_DECODER. The READY signal is high when the compression operation is completed.

For compression, the input image is scanned in an order, where each quadrant is scanned in the NW, NE, SW and SE directions. The input image is stored in the RAM from addresses 00 to 3F (hex) in raster scan order, *i.e.* from left to right and from top to bottom. For a pixel in an 8×8 image indexed by row *I* and column *J* where *I, J* = 0, 1, 2, , 7, its corresponding address in the RAM is expressed by:

$$Address = (8\times I)+J \tag{4}$$

For 8×8 input images, the sequence of RAM addresses in the appropriate scan order is:

$$Address = 32K_5 + 4K_4 + 16K_3 + 2K_2 + 8K_1 + K_0 \tag{5}$$

where K_5, K_4, K_3, K_2, K_1 and K_0 are the individual bit (0 or 1) values of a 6-bit counter that counts from 0 to $(111111)_2$.

The output of the compression is a tree list that describes the quadtree structure ('0' for leave node and '1' for internal node), and a color list that contains the intensity values of the quadtree. On the other hand, the linear quadtree decompression performed by the QT_DECODER block is just the reverse process of the compression.

In linear quadtree compression, if a 2×2 block of an image is homogeneous, it is reduced to one block containing the pixel value; otherwise it is reduced to an 'I' block. The intensity values in an 'I' block are stored in a list. This continues recursively until the 8×8 image is reduced to only one block. The tree list and color list are stored at RAM addresses beginning with 40(hex) and 80(hex) respectively.

B. *RSA Encryption Implementation*

RSA Module consists of 3 sub-modules. They are RSA_LOAD, RSA_CORE and RSA_OUTPUT. RSA_CORE performs encryption and decryption, RSA_LOAD serially captures incoming message to be encrypted or decrypted and RSA_OUTPUT serially outputs the decrypted/encrypted message. A RAM with 2048 bits in size was design to provide the storage element for the RSA encryption modules.

Arithmetic Logic Unit accepts 32 bits data as input, and produce 32 bits output. The input data is stored temporary in a larger register (34-bits). Arithmetic operations are performed on the temporary register. The working result is then moved to the output port when the operation is done. The design uses four large registers (34 bits) to hold the working results, and 2 small registers (5 bits) to hold the loop variables (i, j). The extra 2 bits in the 4 registers are used in order to prevent overflowing during addition operations.

After consideration on the trade-off between security and speed, the size of parameters and signals of the RSA_CORE module for the VHDL model are chosen as follow:

- *M* is the 32-bit plaintext for encryption, or the 32-bit ciphertext for decryption.
- *E* and *N_C* are the 32-bit public key (*e*, *n*) used for encryption, or the 32-bit private key (*d*, *n*) used for decryption.
- *CLK* is the clock input signal.
- *RST* sets the state machine implemented in RSA_CORE architecture to the initial idle state.
- *GO* switches the state machine from idle state to the next state.
- *C* is the 32-bit ciphertext produced by encryption, or the 32-bit plaintext recovered by decryption.
- *DONE* is high when the encryption or decryption operation is completed, otherwise it is always low.

C. Top Level Design

The overall design incorporates the RSA_CORE module into the linear quadtree codec. The top-level entity is named as PARTIAL_ENCRYPT where a low E_D signal activates the QT_ENCODER block to perform linear quadtree compression on the input image stored in the RAM256X8 block. When compression is completed, the RSA_CORE is activated to encrypt the tree list stored at RAM addresses 80 to 82 (hex). The encrypted tree list is then stored at addresses 88 to 8B. On the other hand, a high E_D signal starts the decryption operation of RSA_CORE on the encrypted tree list to recover the tree list. Then decompression is performed by the QT_DECODER to reconstruct the original image.

Four test images are used as inputs to verify the correctness of the design using functional simulation. All of the test images are grayscale with dimensions 8×8. For clarity, each image is arranged in a 8×8 table, in which the cells correspond to the pixel intensity values (grayscale level or color). The size of each pixel is 8 bits and its value is expressed in 2 hexadecimal digits.

IV. Simulation, Synthesis and Discussion

A. Theoretical Results for Test Image 1 and its Quadtree

The output of linear lossless quadtree compression is a tree list that contains the quadtree nodes and a color list that contains the pixel values of the image. In the tree list, binary '0' denotes a leaf node and '1' denotes an internal node. The root node and bottommost leaf nodes are omitted in the tree list in order to achieve better compression ratio, as the decompression algorithm does not need them. Since decompression is simply the reverse process of compression, its results can be deduced from those of the compression.

The results of linear lossless quadtree compression are:

Tree list = 100111000001_2 = 9C1

Color list = 00 FF 00 FF 00 00 FF 00 00 FF FF 00 FF 00 FF FF FF 00 00

Size of image = 64×8 bites = 512 bits

Size of tree list = 12 bits

Size of color list = 152 bits

Compression ratio = Size of image / (Size of tree list + Size of color list)

$$= \frac{512}{12+152} = 3.12{:}1$$

B. Functional Simulation for Linear Quadtree Compression / Decompression

Functional simulation is performed to test the logic function of the hardware design and it is presented to verify the correctness of the algorithms implemented by the quadtree codec and the partial encryption module.

Functional simulation of the linear quadtree codec (QT_CODEC) is performed on the four test images with 20ns simulation clock period (50 MHz). The time interval between high GO signal and high READY signal is divided by the simulation clock period to calculate the processing time for compression or decompression. From the results it is observed that the processing time is longer with smaller compression ratio and decompression is faster than compression. Table I shows the functional simulation results on four test images.

TABLE I: COMPARISON OF FUNCTIONAL SIMULATION

Image	**Compression Ratio**	**Compression (clock cycles)**	**Decompression (clock cycles)**
Test Image 1	3.12:1	291	291
Test Image 2	1.91:1	350	332
Test Image 3	56.89:1	219	200
Test Image 4	0.96:1	489	350

C. Partial Encryption Simulation

Functional simulation of the partial encryption module (PARTIAL_ENCRYPT) is performed on four-test image with 40ns simulation clock period (25 MHz). In this paper the simulation for test image 1 is shown in Fig. 1 and 2 using following key pairs:

- Public key for encryption, E = $0000041B_{16}$, N_C = $FCB17208_{16}$.
- Private key for decryption, E = $013A0C23_{16}$, N_C = $FCB17208_{16}$.

The time interval between high GO signal and high READY signal is divided by the simulation clock period to calculate the processing time for combined compression and partial encryption or partial decryption and decompression, and the results are compared in Table II. It is concluded that the partial decryption and decompression is much slower because the decryption time of the RSA_CORE module is twice longer than the encryption time.

TABLE II: COMPARISON OF THE COMBINED PROCESSING TIME FOR TEST IMAGE 1 (FUNCTIONAL SIMULATION).

Image	**Compression Ratio**	**Compression and Partial Encryption (clock cycles)**	**Partial Decryption and Decompression (clock cycles)**
Test Image 1	3.12:1	1918	4173

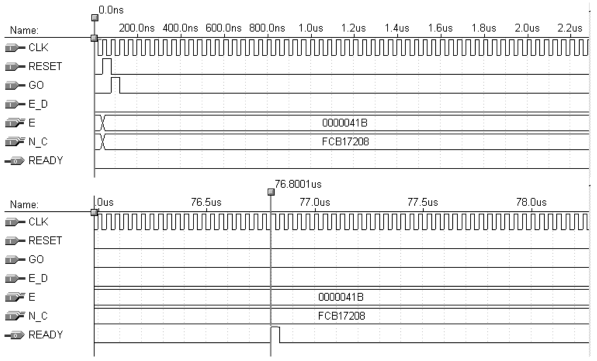

Fig. 1. Simulation of Test Image 1 compression and partial encryption.

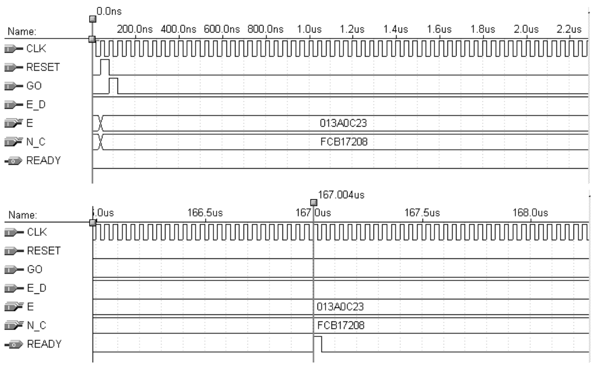

Fig. 2. Simulation of Test Image 1 partial decryption and decompression.

D. *Synthesis*

In regard to the designated hardware realization, The VHDL code is synthesized by considering Altera FLEX10K: EPF10K10LC84 FPGA chip on LC84 package. The FLEX 10K family provides the density, speed, and features to integrate entire systems, including multiple 32-bit buses into a single chip. The RTL view for the output layer is shown in Fig. 3.

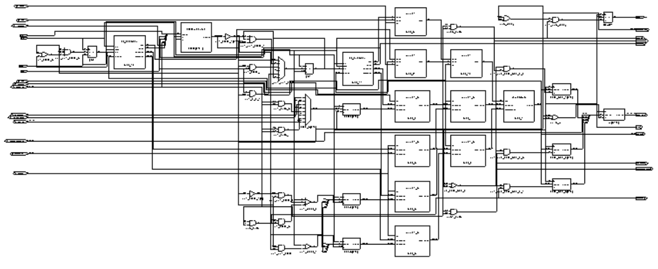

Fig. 3. RTL view of the PARTIAL_ENCRYPT (1 of 3 sheets)

Throughout the synthesis results, there are a few points worth to be discussed. Firstly, from the synthesis results, the RSA Core module utilized around 20% of the chosen FLEX 10KE device. Nevertheless, the clock frequency report showed the critical frequency is only 34.7MHz. This has given the limitation of the frequency of the RSA Module, even though the serial to parallel and parallel to serial converters could achieve 133.9MHz and 89.6MHz respectively. For the RSA Core module, though the 34.7MHz is acceptable, it is not fast enough compare to today's FPGA technology. However, the critical frequency can possibly be increased further by optimizing the circuit through place & route the internal probes. The synthesis of the whole RSA encryption, which included the RAM implementation, has taken up 554 units of logic cell (LC). This is about 35% utilization of the chosen device. Lastly the top level design, which is the PARTIAL ENCRYPT entity, was synthesized. A total of 2928 units of LC were used and it is about 58% utilization of the device (Altera EPF10K100EQC208-1). The frequency achieved was 13.42MHz.

V. Conclusion

In this research project, the FPGA prototyping of a partial encryption of compressed images algorithm that allows for efficient hardware implementation had been implemented. The lossless quadtree compression and RSA encryption algorithms are chosen for implementation due to their computational simplicity in hardware.

It is found from the simulation results that in linear quadtree approach the compression process is faster than the decompression process. Moreover, the RSA simulations show that the encryption process is faster than the decryption process for all four images tested.

References

[1] Cheng, H., and Li, X. "Partial Encryption of Compressed Images and Videos." IEEE Transactions on Signal Processing, Vol. 48, No. 8, pp. 2439-2451, August 2000.

[2] Cheng, H. "Partial Encryption for Image and Video Communication." M.S. Thesis, University of Alberta, Edmonton, Canada, 1998.

[3] Cheng, H., and Li, X. "On the Application of Image Decomposition to Image Compression and Encryption." Communications and Multimedia Security II: Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security, pp. 116-127, Essen, Germany, September 1996.

[4] Dang, P.P., and Chau, P.M. "Image Encryption for Secure Internet Multimedia Applications." IEEE Transactions on Consumer Electronics, Vol. 46, No. 3, pp. 395-403, August 2000.

[5] Jakobsson, M., Stern, J.P., and Yung, M. "Scramble All, EncryptSmall."http://www.julienstern.org/files/scramble/scramble.html, visited in December 2002.

[6] Samet, H. "Data Structures for Quadtree Approximation and Compression." Communications of the ACM, Vol. 28, No. 9, pp. 973-993, September 1985.

[7] Lin, T.W. "Image Compression Using Fixed Length Quadtree Coding." Proceedings of the 3rd International Conference on Signal Processing (ICSP), pp. 970-973, Beijing, China, October 1996.

[8] Gargantini, I. "An Effective Way to Represent Quadtrees." Communications of the ACM, Vol. 25, No. 12, pp. 905-910, December 1982.

[9] Chan, Y.K., and Chang, C.C. "Concealing a Secret Image Using the Breadth First Traversal Linear Quadtree Structure." Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications (codas), pp. 194-199, Beijing, China, April 2001.

[10] Rivest, R., Shamir, A., and Adleman, L. "A Method for Obtaining Digital Signatures and Public Key Cryptosystems." Communications of the ACM, Vol. 21, No. 2, pp. 120-126, February 1978.