ANALYSIS AND ARCHITECTURE FOR MEMORY EFFICIENT JBIG2 ARITHMETIC ENCODER

Chun-Chia Chen, Yu-Wei Chang, Hung-Chi Fang, and Liang-Gee Chen
DSP/IC Design Lab, Graduate Institute of Electronics Engineering and
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
{chunchia, wayne, honchi, lgchen}@video.ee.ntu.edu.tw

***Abstract*— JBIG2 is the latest international standard for bi-level image compression. It partitions a bi-level image into three types of region—text, halftone, and generic region. This paper addresses the problem of large memory requirements for the contexts. The arithmetic encoders of text region in JBIG2 require a large memory for the contexts. In this work, we proposed several algorithms to reduce the memory requirements. A large portion of total memory requirements are reduced by the proposed algorithms. Moreover, the proposed algorithms are implemented as a memory efficient hardware architecture, the unified arithmetic encoder. The experimental results show that the proposed algorithms are efficient. The proposed algorithms can reduce the memory requirements for the contexts by 98.7%. Moreover, the drop of coding gain due to memory reduction for the contexts is only 4.83% in average.**

I. INTRODUCTION

JBIG2 is the latest international standard for lossless or lossy bi-level image compression [1]. In JBIG2, a bi-level image is first segmented into three types of region—text, halftone and generic region [2]. Then, they are coded separately by suitable algorithms [3]. The text region is coded into a symbol dictionary and a text region data structure, and then coded separately [6]. The halftone region is decomposed into a pattern dictionary and a gray image, and then coded separately [5]. The generic region is coded by a context-based arithmetic encoder. In this paper, we focus on text coding.

In JBIG2 text coding, the text coding algorithms are well discussed and described in [6] [7] [8]. However, there is no discussion on the hardware issues of the Arithmetic Encoders (AEs) for text coding. Thus, we focus on the hardware issues of the AEs. The huge ConteXts (CXs) memory of AEs of JBIG2 is a serious problem for hardware implementation. Therefore, analysis and techniques overcoming this problem are proposed in this work.

In JBIG2, the computational complexity of AEs occupies a considerable portion of total complexity. Table I shows the average time profiling results of coding CCITT standard bi-level images. The AE computation time occupies 29.6% of total computation time of JBIG2. Moreover, the AE involves many bit-level operations, which are not suitable for processors. Therefore, hardware acceleration of the AE is an efficient way to implement the JBIG2 system.

The AEs require a large memory for the CXs. If the memory is put on chip, it will occupy most of the chip area. This problem makes low cost AE accelerator of JBIG2 unrealistic. Thus, we proposed algorithms to reduce the CXs memory.

This paper is arranged as follows. Section II introduces the AEs of JBIG2 and the problem of huge CXs memory. To solve this problem, algorithms in Sec. III is proposed. A contexts memory efficient hardware architecture to support AEs in JBIG2 is proposed in Sec. IV. Section V lists the experimental results. Finally, Sec. VI is the conclusion.

II. PRELIMINARY

JBIG2 text coding has several stages. To code a text image, a dictionary is built in a sequentially-symbol-matching manner. During dictionary building, the text image is decomposed into a dictionary and side information of text symbols. Finally, the dictionary and the side information of text symbols are coded by several AEs. Here, we describe the coding flow step by step.

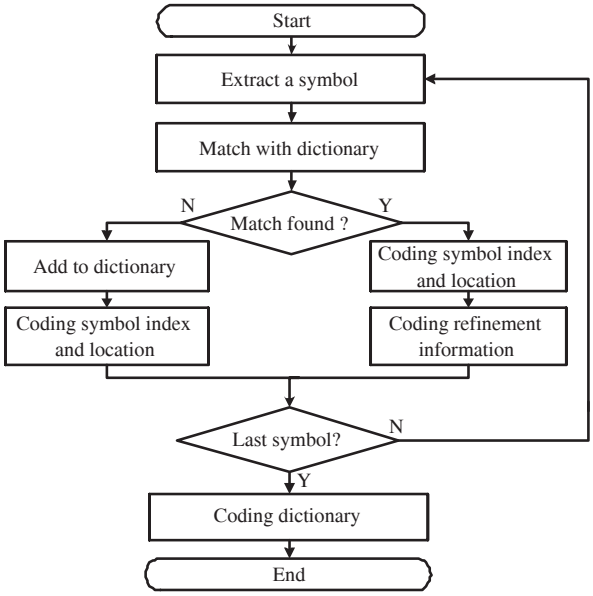

Fig. 1. The Flowchart of Text Coding.

TABLE I
AVERAGE TIME PROFILING OF CODING FLOW IN JBIG2.

Arithmetic Encoder	Symbol Matching	Side Info.	Other
29.6%	36.5%	12.0%	21.9%

A text image consists of symbols. In JBIG2, a text image is decomposed into a symbol dictionary and side information of symbols. The decomposition is a sequential symbol matching process, as shown in Fig. 1. After extracting from the image, a text symbol is matched with the dictionary symbols. If there is no match, the text symbol is added into the dictionary as a new dictionary symbol. Otherwise, the text symbol is matched. Often, the matched dictionary symbol is not perfectly the same as the text symbol. For lossless coding , the difference between the text symbol and the dictionary symbol must be coded as a refinement. Thus, the side information of every text symbols after the decomposition includes text symbol location, symbol index in dictionary, and refinement information.

After the decomposition, the dictionary and side information of

0-7803-9197-7/05/$20.00 © 2005 IEEE.

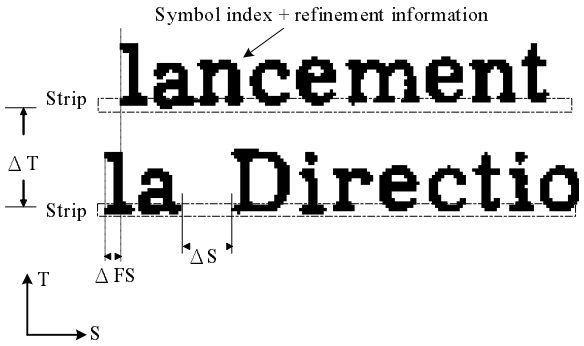

Fig. 2. The Structure of Side Information.

text symbols are coded into bitstreams by entropy coding. For understanding the entropy coding, the structure of side information of symbols is illustrated in Fig. 2. Figure 2 shows a sample text region of a bi-level image. The horizontal axis is denoted as S coordinate, the vertical coordinate is denoted as T coordinate. Instead of directly coding S and T value, JBIG2 uses a strip-based differential coding scheme. JBIG2 group symbols into strips. A text image consists of many strips; each strip consists of many symbols with similar T coordinates. There are many kinds of side information of text symbols, we just list some important types in Fig. 2. The integer ΔFS means the difference of FS values of two neighboring strips, while the First S (FS) means the S coordinate of the first symbol of a strip. The integer ΔT means the difference of T coordinates of two neighboring strips. The integer ΔS is equal to the distance between the right-side edge of the current symbol and the left-side edge of the next symbol in the same strip. Integers ΔFS, ΔT and ΔS are fed to AEs to encode.

The last step of text coding is the entropy coding. The entropy coding of JBIG2 is achieved by the AEs. JBIG2 has several AEs, each AE is responsible for a certain type of information. All AEs in JBIG2 can be classified into two types, Integer Arithmetic Encoder (IAE) and Context-based AE (CAE). The input to an IAE is an integer. The input to a CAE is a symbol bitmap or refinement bitmap. Here, we just focus on entropy coding on side information of text symbols. The side information includes symbol location, symbol index, and refinement information. The symbol locations and symbol indexes is encoded by IAEs. The refinement information is encoded by CAE.

In JBIG2, every AE has a set of CXs. Every CX needs several bits of memory. Thus, the number of CXs decides the memory requirements of this AE. Each AE has two stages, Context Formation (CF) and MQ Coder. The CF generates CX and Decisions (D). The Ds are coded by the MQ coder with the associated CXs.

III. Algorithm

In this section, memory saving algorithms for the CXs memory are proposed. They are elementary saving algorithms, dummy insertion algorithm, and image level refinement algorithm.

In JBIG2, some AEs have options and parameters that will affect the number of CXs. By choosing them properly, the number of CXs can be reduced. The AEs in JBIG2 can be classified into three categories. First category contains IAEs which input integer is theoretical small, thus these IAEs use few CXs. Thus, they are not the discussed here. The second category contains CAEs, which number of CXs can be set by parameters and options. For this category, we just select the parameter or option to minimize the number of CXs while maintaining the coding efficiency. The AEs of the third category are IAEs that can be discarded in text coding while preserve formal syntax of bitstreams. For these IAEs, we discard them and their CXs can be saved.

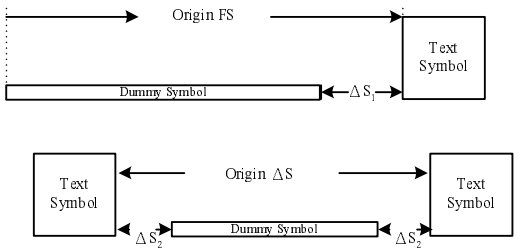

Fig. 3. Example of DIA on FS and ΔS.

After the elementary saving step, the critical part of number of CXs is three IAEs and the refinement AE. These three IAEs contain IAE for coding ΔS (IADS), IAE for coding ΔT (IADT), and IAE for coding ΔFS (IAFS). Hereafter, we call them critical IAEs. In the follows, we propose two algorithms to reduce the number of CXs.

A. Dummy Insertion Algorithm

In this section, we focus on saving the CXs of the three critical IAEs. Because of the CF rule of IAE, the number of used CXs is related to the dynamic range of input integers of IAE. The larger the dynamic range of an IAE, the more the number of used CXs. For CXs reduction, it's clear to confine the dynamic range of the input integers of IAEs. Thus, the number of used CXs is reduced. The unused CXs are discarded, and memory of these CXs are reduced.

However, the dynamic range of input integers of IAEs depends on the text image. Thus, the dynamic ranges are not bounded. In order to support lossless coding, it's difficult to confine the dynamic ranges while not changing the original image. Thus, we proposed the Dummy Insertion Algorithm (DIA) to achieve this goal.

The DIA decomposes a large input integer into several smaller input integers by inserting dummy symbols or dummy strips. Thus, the input integers of critical IAEs can be confined to a predefined value.

In the DIA, we add several blank symbols into the symbol dictionary. These blank symbols have various widths. The insertion of a dummy symbol is just to index the corresponding blank symbol from dictionary. Figure 3 shows how the DIA confines ΔS and ΔFS. If ΔS is too large, the DIA inserts a dummy symbol with suitable width into ΔS, thus the large ΔS is cut by the dummy symbol. The same idea applies to confine ΔFS. As in Fig. 3, we insert a dummy symbol with suitable width to the leftmost of every strip. The original large FS is replaced by the dummy symbol, thus a large FS can be decomposed into a zero ΔFS and a small ΔS. Originally, the coded ΔFS is unbound. By the DIA, the dynamic ranges of critical IAEs are confined to a value. Therefore, the number of CXs is reduced.

Figure 4 shows how the DIA confines ΔT. When the ΔT is too large, the DIA inserts several dummy strips between the current strip and the next strip. Every dummy strip is blank, consisting of only one dummy symbol. Inserting dummy strips decomposes the original ΔT into several smaller ΔTs.

By the DIA, the used CXs of critical IAEs are reduced. Moreover, the DIA doesn't introduce artifacts because all the dummy symbols and dummy strips are blank. The image is still lossless. However, the DIA also satisfy the syntax format of the JBIG2 standard.

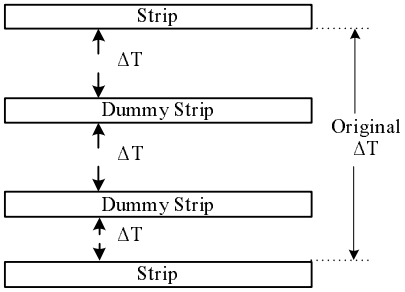

Fig. 4. Example of DIA on ΔT.

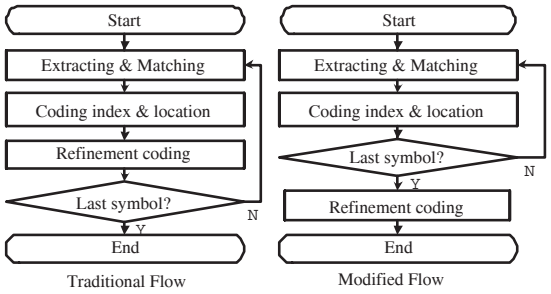

Fig. 5. Modified Flow of Image Level Refinement.

B. Image Level Refinement

To further reduce the total number of CXs, we want to share the CXs of the refinement coding AE with IAEs. By CXs sharing, the total CXs can be reduced. But, the CXs of the refinement coding AE and IAEs cannot be shared in the traditional coding flow. As shown in Fig. 5, in the traditional flow, the refinement coding is performed symbol by symbol. After coding the location and symbol index of a symbol by the IAEs, the refinement coding is performed only for this symbol. During refinement coding, the CXs memory for the IAEs must be remained for the coding of symbol index and location for the next symbol. Thus, the CXs of the refinement coding AE and the IAEs cannot be shared.

For CXs sharing, we propose the image level refinement. The idea is to modify the coding flow, as shown in Fig. 5. In the modified coding flow, the refinement coding is performed after all symbol locations and symbol indexes are coded. After all IAEs coding, the CXs for symbol locations and symbol indexes become useless. Because they are not used now, data in CXs of IAEs can be reset and reusable thereafter. For lossless coding, the refinement information of text symbols must be coded. In the image level refinement, the refinement information of every symbols are grouped into a refinement image. The refinement coding AE encodes the refinement information by the refinement image. Because the CXs of IAEs are reusable now, the refinement coding AE uses the reusable CXs to encode the refinement image. Therefore the CXs between IAEs and the refinement coding AE can be shared.

The image level refinement satisfy the syntax format of the JBIG2 standard. In JBIG2, there is a region refinement data type. The image level refinement information is coded into the region refinement data type.

IV. Architecture

In this section, an AE accelerator, Unified Arithmetic Encoder (UAE) is proposed. The UAE can support all IAE and CAE types in text coding. It's a computation accelerator for entropy coding in JBIG2. The UAE is a context efficient engine because it adopts CXs sharing method and the DIA.

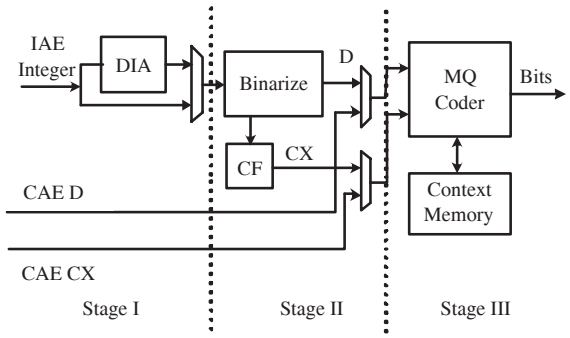

Fig. 6. Unified Arithmetic Encoder Architecture.

As shown in Fig. 6, our UAE has three stages. Input integers of critical IAEs are fed into stage I. Input integers of other IAEs pass through another path to stage II. Contexts and decisions of the CAEs are fed into stage III.

Stage I is the DIA stage. The input integers of the DIA module are decomposed into several smaller integers. Then, the smaller integers are fed to stage II one by one.

Stage II is the binarization and context formation stage, implementing the binarization and CF rules of IAEs of JBIG2. It also has context sharing method. It shares the context memory among side information coding, dictionary coding and refinement coding. This stage generates CXs and decisions for Stage III. As in Fig. 6. The binarization module binaries an input integer into several decisions. The binarization module also directs the CF module to generate corresponding CXs.

Stage III is the MQ Coder Stage. It has two components, a MQ Coder and a CXs memory. The CXs memory is a SRAM to store all CXs. The MQ coder receives the address of CXs from stage II and accesses the CXs from the CXs memory. Finally, the output bytes are generated by the MQ coder.

V. Experimental Results

In our experiments, the JBIG2 encoder is built ourselves named Jerichi JBIG2 Encoder. The coding gain of our Jerichi JBIG2 Encoder is nearly equal to the reference encoder called PowerJB2. The test bi-level images are the text regions of CCITT standard images.

In our experiments, we focus on CXs reduction performance, the effect of CXs on compression ratio and hardware gate count analysis. Table II shows the number of CXs after our reduction algorithms. In Table II, "ES" means all our elementary saving methods. The elementary saving methods reduces a large portion of original CXs. After elementary saving methods, the bottleneck of reduction of CXs is in critical IAEs, those are IAE for ΔS (IADS), IAE for ΔT (IADT) and IAE for ΔFS (IAFS). In Table II, "ES+DIA" means that both elementary saving methods and the DIA are applied. It shows that the DIA totally reduces 71.4% of original CXs of critical IAEs. Note that "ALL" in Table II means all our methods are applied, those are elementary saving methods, the DIA and image level refinement. After all methods are applied, the total CXs are reduced to 1024CXs. Totally, our methods reduce 98.7% of original CXs.

The penalty of saving CXs is the drop of compression ratio. The experimental results of the drop of coding gain are listed in Table III. Table III shows the coded file sizes of selected test text images after

TABLE II
NUMBER OF CXS AFTER REDUCTION ALGORITHMS.

	Origin	ES	ES+DIA	ALL
IADS	512	512	293	293
IADT	512	310	85	85
IAFS	512	512	4	4
CAE	73728	1536	1536	1024
Others	5721	1209	984	984
Total	80895	4079	2902	1024

TABLE III
FILE SIZE AFTER CXS REDUCTION IN KBYTES.

Image	Origin	ES	ES+DIA	ALL
f01	10.98	10.92	11.06	11.36
f03	14.07	14.23	14.31	14.93
f04	35.12	35.84	36.16	36.88
f05	16.67	16.87	17.09	17.54

our algorithms. The file size of every test image in our experiments is 505.3 KB. Table III shows averagely low coding gain drops of our methods. Although our CXs saving techniques can reduce a large portion of CXs, the coding gain drop is low. The total coding gain drop is 3.9% of image "f01", 5.7% of image "f03", 4.8% of image "f04", and 4.9% of image "f05". The average coding gain drop of test text images is 4.83%.

Now, we discuss the gate count analysis of our UAE. After synthesizing our design, stage I has 1496 gates, stage II has 735 gates and stage III has 3094 gates. The gate count of stage I is approximately the gate count of the DIA module. The gate count of the DIA module is small compared to total gates, thus the hardware cost of the DIA is low. In addition to the low overhead, the most important function of the DIA module is to reduce the CXs memory. Without the DIA module, the CXs memory in stage III will increase dramatically.

VI. CONCLUSION

In JBIG2, a large context memory is required for the arithmetic encoders. To solve the problem, we proposed the dummy insertion algorithm and the image level refinement algorithm to effectively reduce the contexts memory size. We also proposed a hardware accelerator to support all AEs of JBIG2 text coding with small contexts memory. The experimental data show that the proposed algorithms can save the contexts by 98.7%. Besides, the coding gain drop is only 4.83% in average.

REFERENCES

[1] JBIG2 Final Draft Standard, ISO/IEC JTC 1/SC 29/WG 1, July, 1999

[2] D. Tompkins and F. Kossentini, "A Fast Segmentation Algorithm for Bi-level Image Compression Using JBIG2", Intl. Conf. on Image Processing, Oct. 1999.

[3] Paul G. Howard, Faouzi Kossentini, Bo Martins, Soren Forchhammer, and William J. Rucklidge, "The Emerging JBIG2 Standard", IEEE Trans. Circuits Syst. Video Technol., vol. 8, no. 7, Nov. 1998

[4] Paul G. Howard, "Lossless and Lossy Compression of Text Images by Soft Pattern Matching", IEEE Data Compression Conf., March 1996

[5] S. Forchhammer and Kim S. Jensen, "Data Compression of Scanned Halftone Images", IEEE Trans. on Communications, vol. 42, no. 2/3/4, Feb./March/April 1994.

[6] Yan Ye and Pamela Cosman, "Dictionary Design for Text Image Compression with JBIG2", IEEE Trans. on Image Processing, vol. 10, no. 6, June 2001.

[7] Yan Ye and Pamela Cosman, "Fast and Memory Efficient Text Image Compression With JBIG2", IEEE Trans. on Image Processing, vol. 12, no. 8, August 2003.

[8] Yan Ye and Pamela Cosman,"Feature monitored shape unifying for lossy SPM-JBIG2" Int. Symp. Signal Processing and Its Applications, August 2001.