A JBIG–ABIC compression engine for digital document processing

by K. M. Marks

This paper describes the implementation of a combined JBIG and ABIC compression–decompression engine, which has been integrated into a digital document-processing microcontroller used in imaging applications.

Introduction

With print resolution for digital documents rising to 1200 dpi, the requirements for controller memory and controller throughput are increasing steadily for both print and digital copy applications. Storage requirements are dictated by features such as collation or duplexing, which make use of full image storage.

As shown in **Table 1** for the case of a copier, generation of subsequent pages takes place while previous pages are printed in order to keep the output engine continuously busy. Pages P_n are split into bands P_n B_x for that purpose. This requires quasi-simultaneous compression of the rendered or scanned and screened images to storage and repetitive decompression of the stored images for print. This can be accomplished either by providing separate compression and decompression engines or by supplying a single engine with sufficient throughput to allow time-sharing between compression and decompression tasks.

To meet the real-time constraints of laser engines (once a page is started, it is necessary to keep up with the engine), predictability or low data dependency of the throughput is very important. At 1200 dpi, a single letter-size page contains approximately 128 million pixel elements, or 16 MB of bitonal data. A high average compression ratio as well as good worst-case compression behavior is therefore essential. Adaptive arithmetic compression as used in ABIC [1, 2] and JBIG [3, 4] provides by far the best average and worst-case compression ratios on bitonal data.

JBIG and its predecessor ABIC are adaptive arithmetic compression methods that convert an image stream into a binary fraction. This is accomplished in two steps—an image preprocessing (model unit) and a subsequent adaptive arithmetic coding process. **Figure 1** shows the basic architecture of a JBIG compressor.

The image preprocessing extracts individual image pixels and converts them into a context word correlated with the pixel to be coded. In other words, each individual pixel of the complete image has as additional information an associated context describing the neighboring pixels

0018-8646/98/$5.00 © 1998 IBM

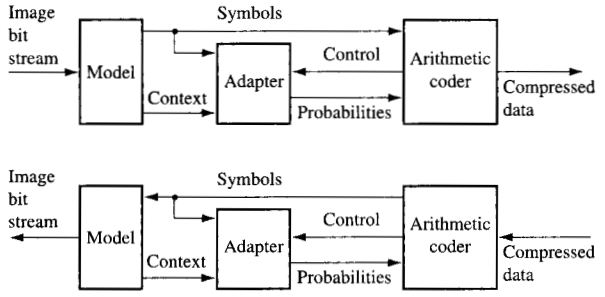

Figure 7

JBIG compressor architecture for compression and decompression.

Table 1 Copier pipeline with JBIG shared for compression and decompression.

Time	*Scan*	*Scale and screen*	*JBIG compress**	*JBIG decompress**	*Print*
T_0	$P_n B_0$	$P_{n-1} B_7$	$P_{n-1} B_6$	$P_{n-m} B_3$	$P_{n-m} B_2$
T_1	$P_n B_1$	$P_n B_0$	$P_{n-1} B_7$	$P_{n-m} B_4$	$P_{n-m} B_3$
T_2	$P_n B_2$	$P_n B_1$	$P_n B_0$	$P_{n-m} B_5$	$P_{n-m} B_4$
T_3	$P_n B_3$	$P_n B_2$	$P_n B_1$	$P_{n-m} B_6$	$P_{n-m} B_5$
T_4	$P_n B_4$	$P_n B_3$	$P_n B_2$	$P_{n-m} B_7$	$P_{n-m} B_6$
T_5	$P_n B_5$	$P_n B_4$	$P_n B_3$	$P_{n-m+1} B_0$	$P_{n-m} B_7$
T_6	$P_n B_6$	$P_n B_5$	$P_n B_4$	$P_{n-m+1} B_1$	$P_{n-m-1} B_0$
T_7	$P_n B_7$	$P_n B_6$	$P_n B_5$	$P_{n-m+1} B_2$	$P_{n-m+1} B_1$
T_8	$P_{n+1} B_0$	$P_n B_7$	$P_n B_6$	$P_{n-m+1} B_3$	$P_{n-m+1} B_2$

*These tasks share the time intervals T_n.

as well as their spatial relationship. At the edges of the data matrix or image, special rules apply for context bits outside the actual image or data matrix. JBIG uses a context built from ten neighboring pixels of the current pixel being examined. These are taken either from the previous line and the current line or from the two previous lines and the current line. Also, one context bit from the previous line can be replaced by a more distant pixel in the current line to pick up horizontal frequencies to improve the compression ratio. These options are programmable parameters. How each image pixel is associated with its context and how image pixels make up a context are shown below in the model unit discussion.

An adaptive arithmetic coder consists of an adapter and the so-called arithmetic coder. The adapter contains a storage table memory, which, for example, in the case of JBIG, has 1024 (2^{10}) entries. The context bits are used as an index into this table memory. The information elements stored in each entry are the expected value for the current pixel given the surrounding pixel values defined by the context, and a probability index for this expected value. Both the probability indices and the expected value are adapted dynamically to optimize compression. The probability indices are converted into a binary fraction that can be viewed as an interval width. The higher the probability of an expected value, the lower is the information content of an event confirming the expected value, and, accordingly, the chosen size for the corresponding interval. For each image pixel being coded, the adapter passes the interval size Qe and a flag, specifying to the arithmetic coder whether or not the expectation for this image pixel has been met.

The coder contains an accumulator C to generate the compressed code word as well as an accumulator A to track the actual interval width. Initial conditions are C = 0 and A = 1.0000. For each pixel being coded, the interval size is adapted; if the pixel meets the expected value, the value in A is replaced by A − Qe; otherwise the value in A is replaced by Qe. The value in C is replaced by C + Qe or retains its value accordingly. Whenever A drops below 0.75, both A and C are shifted left until A is again greater than or equal to 0.75. This allows the coder to generate a binary fraction of infinite precision with an accumulator of finite precision. The compression of the coding is based on the fact that for pixels with a high probability for the expected value, Qe is very small and therefore can be subtracted many times from A before an underflow occurs and the simultaneous renormalization of C with A generates code bits.

Decoding works similarly by loading C with the accumulated code word. Then the unit tries to subtract Qe from C. If Qe can be subtracted from C without causing C to underflow, it must have been added during encoding; i.e., the pixel being coded equaled the expected value. Otherwise, C retains its value, and the pixel being coded did not equal the expected value. A is processed the same way as during encoding.

Most significantly, JBIG allows specific accommodations for halftone images that other compression methods such as T.4 or T.6 fax compression do not offer. JBIG provides more capability for tuning the template of the model unit to particular images and has a higher-resolution probability estimation, both of which can help to improve the compression ratio. However, ABIC's bit stuffing is more efficient than JBIG's byte stuffing and is not prone to the considerable worst-case uncertainty in throughput caused by carry resolution during compression [5, 6].

With no compression hardware existing at the time that could support the requirements of the multifunction peripheral market, Xionics decided in 1995 to develop

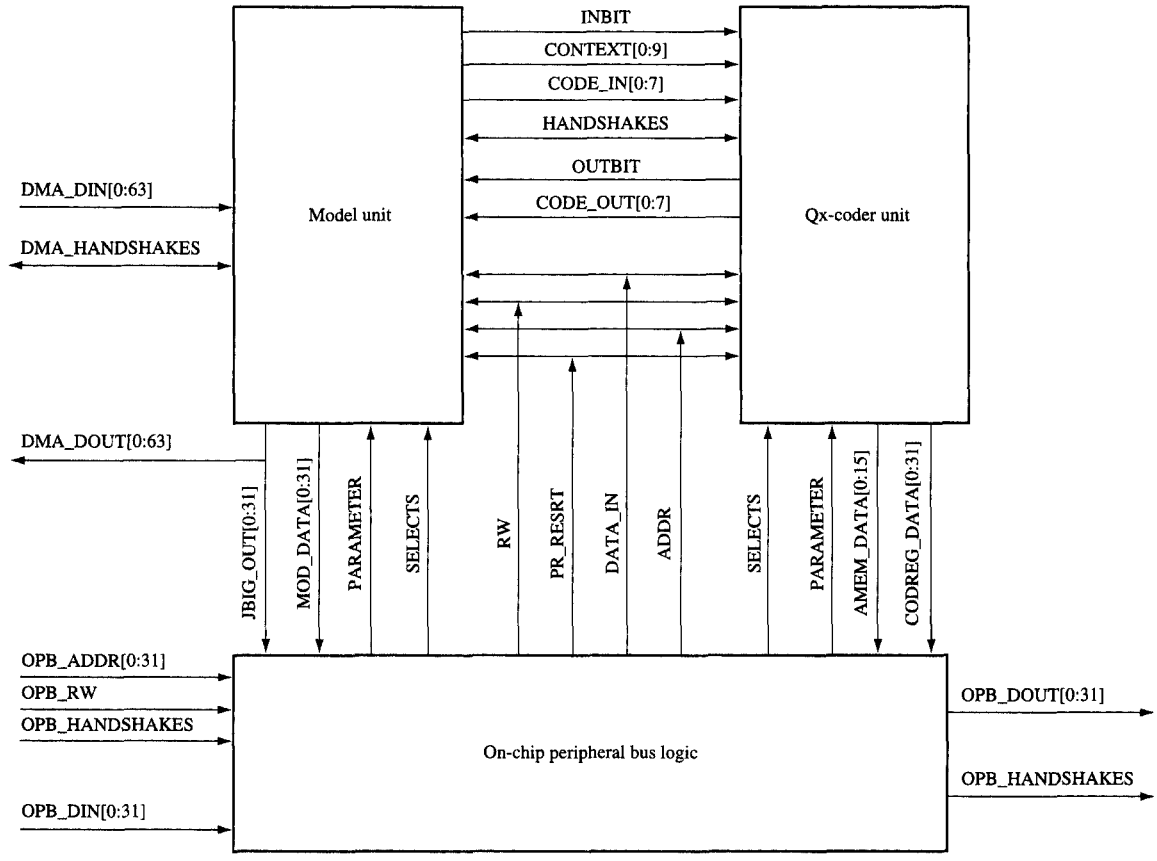

Figure 2

Partitioning of ABIC–JBIG macro.

such a core as part of their integrated MFP silicon solution, XipChip, and began a collaboration with IBM Microelectronics to develop a bitonal compression solution. This paper describes the implementation of this combined JBIG and ABIC compression and decompression engine for digital document processing equipment and the design tradeoffs made to meet throughput requirements. XipChip is shipped today as one of Xionics' multifunction peripheral solutions.

Combined JBIG and ABIC compression and decompression engine

- *Overall engine architecture*

The combined compression engine preserves a partitioning similar to that of the original ABIC hardware [1]. As shown in **Figure 2**, it consists of three major modules: the model unit, the Qx-coder [5], and the bus interface logic.

Model unit

The model unit provides three pieces of functionality—generation of the context for the Qx-coder; generation of the input pixel for compression and assembly of the image from the decompressed pixels; and management of the input and output buffers.

The model unit allows selection from among three different templates for context generation: the seven-bit ABIC template [**Figure 3(a)**]; the two-line JBIG template [**Figure 3(b)**]; or the three-line JBIG template [**Figure 3(c)**]. The number following the X identifies the bit number within the context. The ABIC template wraps around from the previous line and to the next line at line boundaries, while the JBIG templates pad the context with 0-bits at the beginning and end of each line. In addition, the JBIG templates implement the adaptive template bit identified by an A instead of an X. It is shown in its default position. It can be programmed for a horizontal

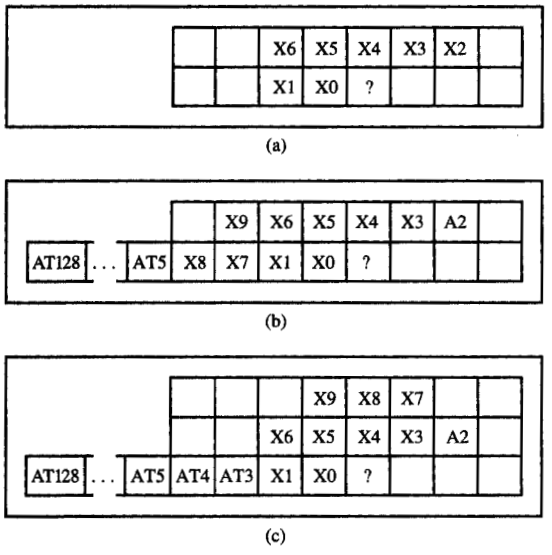

Figure 3

(a) ABIC template; (b) JBIG two-line template; (c) JBIG three-line template.

offset of up to 128 to allow picking up of halftone frequencies [see the JBIG template extensions at the left in Figures 3(a) and 3(b)]. Furthermore, in JBIG mode the model unit can be programmed to check for typical prediction.

Rather than implementing complete line buffers for the template reference lines, the model unit implements three 256-bit buffers. These buffers hold the currently active parts of the lines and move like a window through the image. The buffers are implemented with separate controls for master and slave flip-flops. This allows the unit to pipeline the read accesses and prefetch the next 256 bits for each line into the master latches while it is working on the previously loaded 256 bits stored in the slave latches. This avoids pipeline stalls due to data fetching. In addition, the moving windows do not impose any restriction on the image width, as do fixed buffers used in other implementations. Image width and height are restricted only by the size of the counters and a small overhead for the image boundaries. Image width and height are limited to 65530 pixels.

Qx-coder

The Qx-coder comprises two modules, the adapter and the coder. They are pipelined for optimal performance and hide the adapter's memory-access latency whenever possible. This is an implementation of the Qx-coder described in [5].

Adapter The adapter (**Figure 4**) implements the adapter memory and the hardware tables to convert the stored Q-index values to the Q-values used by the coder and to generate the Q-index transitions for adaptation. The adapter memory has separate read and write ports to allow updating the memory while fetching new values for subsequent coding steps. The memory is 16 bits wide so as to be able to fetch a pair of indices and MPS values with each access. This is important for decompression speed, where the last bit generated will not be available in time to become part of the adapter memory address. By reading both the value for a decoded 1-bit and the value for a decoded 0-bit, and by having dual hardware tables for the simultaneous conversion of the Q-indices to Q-values, the correct Q-value can be selected with a fast multiplexer as soon as the bit becomes available. In addition, it detects whether the same memory location will be accessed on subsequent accesses. For memory updates, the updated values are cached in a register until the actual memory update has occurred, so that they can be accessed without update latency in case they are needed for the next coding step. The adapter memory has 512 entries to accommodate the larger contexts of the JBIG template. For ABIC compression, only the lower 64 entries of the memory are used. The hardware tables for Q-value generation and Q-index transitions implement both the Q-coder tables for ABIC and the QM-coder tables for JBIG. One can decide which tables are to be used when setting up the unit and program accordingly.

Coder The implementation of the coder combines aspects of the original ABIC hardware design [1] with the changes required for JBIG. The native coding convention for this coder is the original ABIC coding. JBIG data streams are handled by inverting the code on its way out for compression and on the way in for decompression, as described in the companion paper [5] by Slattery and Mitchell in this issue.

Besides the handling of byte stuffing and the insertion and stripping of escape sequences, the major difference compared to the original ABIC design is the implementation of the renormalization shifts. The renormalization logic is part of the critical timing path in the coder and therefore required a thorough analysis. The use of a barrel shifter accommodating all possible shifts would have increased the minimum cycle time of the coder significantly. On the other hand, adding additional clock cycles for all required shifts would have reduced throughput in a similar way. A statistical analysis of the shifts occurring during coding showed that the vast majority of shifts occurring are by one bit position only. Therefore it was decided to insert in-line one-bit shifter logic into the data path to facilitate shifts by one bit without requiring an additional cycle. The advantage of

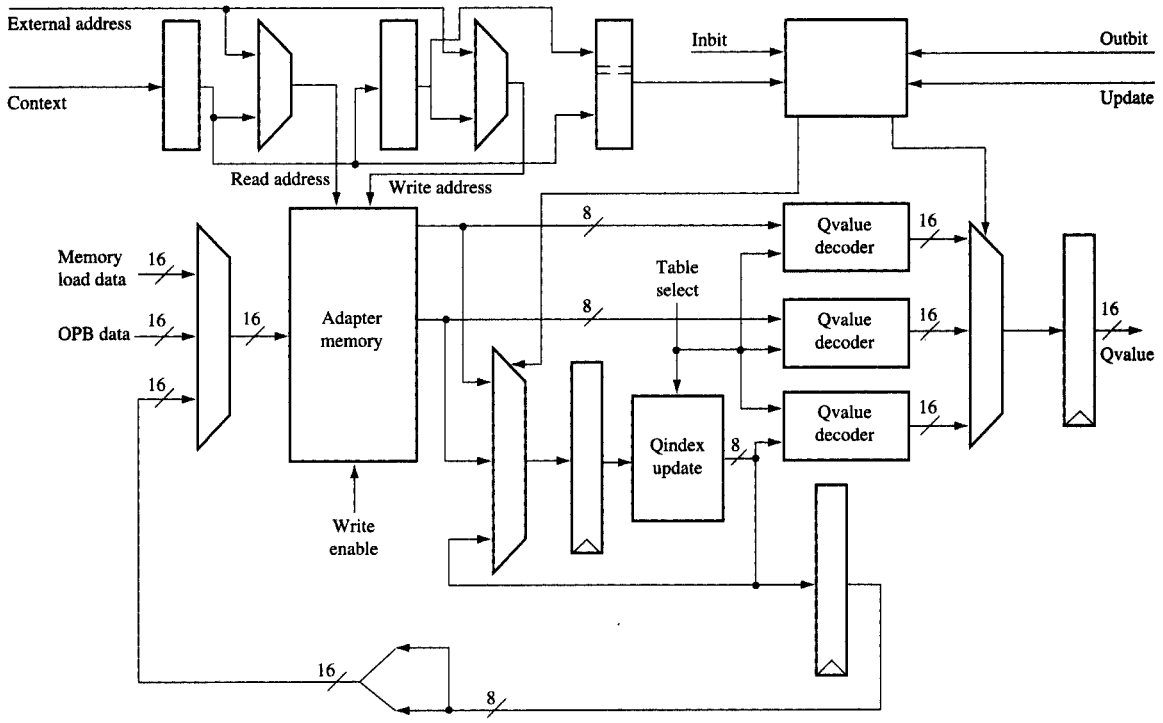

Figure 4

Adapter block diagram.

this approach is that the depth of the shift logic is small enough so that it does not significantly increase the cycle time of the coder. Additional cycles are required only for the less frequent multibit shifts. Additional details can be found in the companion paper [6] by Kampf in this issue.

The coder also supports the generation of trailing zeros on the input code stream during decode as long as the end of the document is not reached. This is required to automatically handle truncated encodings, where trailing zeros have been removed as allowed by the JBIG standard.

Physical implementation and throughput analysis

The unit has been synthesized to 80 MHz in CMOS 5S and CMOS 5SE and integrated into Xionics XipChip imaging microcontrollers. Tests with real-life images in digital copier applications have been conducted, using an image width of 4992 pixels and a band height of 64 to 80 lines. These tests have consistently shown a throughput of more than 64 million pixels per second at 75-MHz operation of the unit. This allows up to 15 ppm throughput at 1200 dpi and up to 60 ppm throughput at 600 dpi sharing the unit for compression and decompression. Assuming a worst-case compression ratio of 2:1, the external memory bandwidth required by the macro is 20 MB/s for the two-line template and 28 MB/s for the three-line template.

Summary

We have described the implementation of a combined JBIG and ABIC compression/decompression engine for digital imaging applications. It provides both the flexibility to handle both JBIG and ABIC data bases and image width to 65530 pixels and the high sustainable throughput required for state-of-the-art digital copier and collated print and copy applications.

References

1. R. B. Arps, T. K. Truong, D. J. Lu, R. C. Pasco, and T. D. Friedman, "A Multi-Purpose VLSI Chip for Adaptive Data Compression of Bilevel Images," *IBM J. Res. Develop.* **32,** 775–795 (1988).
2. J. L. Mitchell and W. B. Pennebaker, "Optimal Hardware and Software Arithmetic Coding Procedures for the Q-Coder," *IBM J. Res. Develop.* **32,** 727–736 (1988).
3. *ITU-T Rec. T.82/ISO/IEC 11544:1993 Information Technology*—Coded Representation of Picture and Audio

Information—Progressive Bi-Level Image Compression (JBIG standard).
4. W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard,* Van Nostrand Reinhold, New York, 1993 (ISBN 0-442-01272-1).
5. M. J. Slattery and J. L. Mitchell, "The Qx-Coder," *IBM J. Res. Develop.* **42,** 767–784 (1998, this issue).
6. F. A. Kampf, "Performance as a Function of Compression," *IBM J. Res. Develop.* **42,** 759–766 (1998, this issue).

Received April 1, 1998; accepted for publication September 24, 1998

Karl M. Marks *Xionics Document Technologies, Inc., 70 Blanchard Rd., Burlington, Massachusetts 01803.* Mr. Marks studied electrical engineering from 1980 to 1985 at the Ruhr-University, Bochum, Germany, and Purdue University; he received his Diploma in electrical engineering from the Ruhr-University in 1985. From 1986 to 1988 he worked as an assistant at the Institute for Electronic Devices at the University of Dortmund, Germany. In 1988 he founded MacroTek, which engaged in the design of support chips for Motorola's 88110 and IBM's PowerPC 60x family. During 1995 and 1996 he worked for the Heinz-Nixdorf Institute at the University of Paderborn, Germany, on the XipChip project with Xionics Document Technologies. In 1996 he joined Xionics as Senior Director of ASIC technology. He has received patents for error detection and correction and L2 cache technology.