The Qx-coder

by M. J. Slattery
J. L. Mitchell

The IBM Adaptive Bilevel Image Compression (ABIC) algorithm depends upon the hardware-optimized Q-coder. The Joint Bi-level Image Experts Group (JBIG) settled upon a software-optimized QM-coder. This paper explores the incompatibilities of the hardware- and software-optimized binary arithmetic coding conventions and reports on the solution that allowed a merged Qx-coder in hardware. A unique hardware solution is presented for the termination of the JBIG data stream (CLEARBITS). The probability estimation is presented in a common format. Detailed flowcharts are included in the Appendix. An ASIC core is available that supports both the ABIC and JBIG bilevel data compression standards using this merged Qx-coder.

1. Introduction

Binary arithmetic coding is a data compression method that generates compressed data as a finite-precision fraction which identifies an interval on the number line. Each picture element (pel) is encoded or decoded on the basis of a probability estimate which determines where the binary arithmetic coder splits the interval into two subintervals. The value of the pel determines which subinterval becomes the new interval. When the size of the new interval drops below a minimum value, Amin, renormalization shifts the precision until it is greater than or equal to the minimum size. With each shift, a bit is produced for the compressed data stream.

Some conventions have to be established about the order of the symbols. For example, the lower interval can be assigned to a 0 and the upper interval assigned to the 1. Alternatively, the lower interval can be assigned to the more probable symbol (MPS) or the less probable symbol (LPS); or the largest subinterval can always be assigned to the more probable symbol. Hardware conventions choose the upper interval as the MPS and the lower interval as the LPS. Software conventions select the lower interval as the MPS and the upper interval as the LPS.

Adaptive binary arithmetic coding uses information about the surrounding pels, the context, to modify the probability estimate. In fact, separate probability estimates are maintained for all possible contexts. A model generates context-pel pairs for the binary arithmetic coder based upon a template. The model must provide the same context for encoding and decoding. The probability estimate of a given context may be changed only when a renormalization is required.

IBM Adaptive Bilevel Image Compression (ABIC) employs the Q-coder for its adaptive binary arithmetic coder [1, 2]. The Q-coder was designed with parallel hardware conventions. Its model template consists of seven fixed, nearby pels. For each context, the expected symbol (0 or 1) and an indication of the estimated probability of the unlikely symbol are stored. After initialization of the 128 contexts, the context-decision/pel pairs are input into the binary arithmetic encoder to produce the compressed data stream. The international Joint Bi-level Image Experts Group created the international standard commonly known as JBIG, which employs a software-convention-based arithmetic coder, the QM-coder [3]. The JBIG sequential model template consists of ten nearby pels, one of which can move, creating 1024 probability-estimation contexts to track. Further details regarding ABIC and JBIG models and architectures are available in companion papers [4, 5].

Figure 1(a) illustrates the Q-coder hardware-optimized symbol-ordering conventions. The C register (C) holds the least significant bits of the compressed data and points to the base of the less-probable-symbol (LPS) interval. The LPS probability estimate Qe is the size of the LPS interval. The split between the LPS and the more-probable-symbol (MPS) intervals is at C + Qe and belongs to the MPS interval. The complete interval size is held in the A register (A). The sum, C + A, points to the

0018-8646/98/$5.00 © 1998 IBM

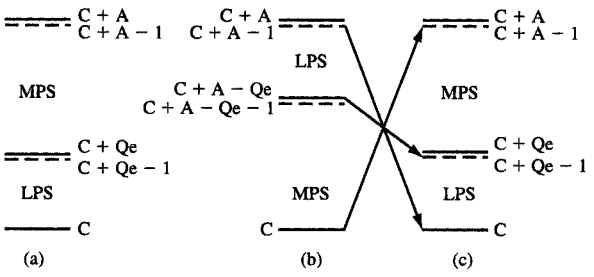

Figure 1

Binary arithmetic coding symbol order: (a) Q-coder hardware conventions with MPS/LPS; (b) QM-coder software conventions with LPS/MPS; (c) QM-coder converted to hardware conventions with MPS/LPS.

top of the interval and is not part of the MPS segment. The dashed line at `C + A − 1` shows the largest value that `C` can reach.

The software-optimized version of the Q-coder in Reference [6] kept the symbol-order conventions shown in Figure 1(a). Extra cycles to identically match the hardware output always occurred outside the inner loops. The hardware-optimized Q-coder-compressed data stream was constructed while the code stream was pointing to the top of the MPS interval. During the termination procedure, the data stream was moved to the bottom of the LPS segment and thus matched the hardware-generated compressed data. The hardware bit-stuffing conventions were carefully duplicated in the software algorithm.

When the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC) and the International Telecommunications Union—Terminal Sector (ITU-T), formerly known as the Consultative Committee of the International Telephone and Telegraph (CCITT), working jointly as JBIG with the Joint Photographic Experts Group (JPEG) [7, 8], arrived at a common QM-coder, it was defined in terms of a different optimal software convention. **Figure 1(b)** shows the JBIG/JPEG QM-coder conventions. For the QM-coder, the MPS interval occupied the lower portion of the number line. The point between the interval was assigned to the upper LPS interval. The code stream pointed to the base of the lower MPS interval.

The QM-coder, derived from ABIC, was designed with serial software conventions that make sharing of hardware difficult. However, the basic concept of the conversion of software-optimized arithmetic coding into hardware-optimized structures is known [9, 10]. **Figure 1(c)** shows a hardware-optimized QM-coder, in which the MPS over LPS (MPS/LPS) convention is used. The last valid value of the interval in Figure 1(b), `C + A − 1`, is mapped to the base of the LPS interval in Figure 1(c). The base of the LPS interval, `C + A − Qe`, maps to `C + Qe − 1`, which is the top of the LPS interval. Finally, the base of the MPS interval becomes the last valid value included in the MPS interval, `C + A − 1`.

This switch in conventions makes it possible to merge the Q-coder and the QM-coder in logic, adopting hardware conventions. The rest of this paper describes in detail how the two arithmetic coders were merged into the Qx-coder. Section 2 discusses more of their differences that must be resolved in the definition of the Qx-coder. Section 3 defines the Qx-coder and illustrates the integration of the Q-coder and QM-coder. Section 4 focuses upon a unique hardware solution to the encoder's termination technique for the JBIG-compressed data stream known as CLEARBITS. Section 5 discusses the various probability estimators and casts them into a common format. Section 6 summarizes the differences between the Q-coder and QM-coder, and the solutions provided by the Qx-coder. The Appendix contains detailed flowcharts implementing the Qx-coder encoder and decoder, and their descriptions.

2. Differences between the Q-coder and the QM-coder

Resolving the fundamental differences between hardware and software conventions is an essential first step in creating the merged Qx-coder. In addition, several other issues must be resolved: register precision, probability-estimation table, carry-over resolution, termination procedures, and byte stuffing.

The Q-coder `Qe` values have 12 bits of precision. The A register needs a 13th bit. This most significant bit is key to the renormalization-driven probability-estimation process. Whenever `A` drops below 0x1000, the index `I` of the MPS probability `Qe` is changed for the context `CX` just encoded/decoded.

The QM-coder `Qe` values have 15 bits of precision. The 16th bit in the A register drives the probability-estimation process. Whenever `A` drops below 0x8000, the index `I` is changed for the current context `CX`. During initialization a 17th bit can be needed for the A register.

To ensure that carries out of the C register could not affect more than the most recently generated compressed byte in the encoder, the Q-coder uses bit stuffing. After every 0xFF byte (aligned on byte boundaries), an extra bit is stuffed into the most significant bit of the next byte. The resolution of any carries that land in the stuffed-bit position is done in the decoder. The newly encoded compressed byte is extracted from the C register in such a way that at most one carry can propagate into it [9].

Table 1 Encoder register assignments.

	Coder	*msb*	*lsb*
C register	Q	fffffffff bbbbbbbb ssssxxxx	xxxxxxxx
	Q in Qx	0000cbbb bbbbbsss xxxxxxxx	xxxxx000
	QM	0000cbbb bbbbbsss xxxxxxxx	xxxxxxxx
	QM in Qx	0000cbbb bbbbbsss xxxxxxxx	xxxxxxxx
A register	Q	0001aaaa	aaaaaaaa
	Q in Qx	1aaaaaaa	aaaaa000
	QM	00000000 0000000a aaaaaaaa	aaaaaaaa
	QM in Qx	aaaaaaaa	aaaaaaaa
Amin	Q	00010000	00000000
	Q in Qx	10000000	00000000
	QM	10000000	00000000
	QM in Qx	10000000	00000000

The QM-encoder is designed to wait to output compressed bytes until any carries have been resolved in the encoder. A stack counter (SC) records the number of buffered 0xFF bytes. This counter typically contains small counts in the range of 1 to 3. However, it could potentially hold the entire compressed data stream and create a severe latency problem [11]. Since carries are resolved in the encoder, bit stuffing is not needed. However, byte stuffing of 0x00 bytes after every compressed 0xFF byte is used to prevent the accidental generation of marker codes.

During the termination of Q-coding, the bits remaining in the C register are shifted into the data stream. This has the nice property that the C register returns to 0x0000 when decoding has completed; otherwise, a nonzero value indicates that errors have occurred.

Since trailing 0x00 bytes may be discarded before the marker codes ending a JBIG stripe or image, the termination procedure for the QM-coder is not a simple flushing of the C register. Instead, the procedure CLEARBITS finds a point inside the interval with the most trailing 0 bits.

3. Qx-coder defined

The Qx-coder takes advantage of the fact that the Q-coder and QM-coder are both finite-precision arithmetic coders and use renormalization-driven probability estimation. The Q-coder hardware-optimized symbol order was chosen. This meant that the QM-coder had to be converted to hardware conventions without modifying the compressed-data bit streams. This was accomplished by inverting the symbol order, as shown in Figure 1(c) and then inverting the compressed data as the bytes are output.

The Qx-coder had to choose a register alignment. The QM and Q entries in **Table 1** and **Table 2** show the encoder and decoder register assignments as given in References [3] and [6], respectively. Since the precision of the Qe values is 12 bits for the Q-coder and 15 bits for the QM-coder, it was natural to shift the Q-coder values left three bits in the register to line them up with the more significant bits of the QM-coder values. This creates a common Amin value for triggering renormalizations. Table 1 shows that this aligns the output byte, bbbbbbbb, nicely.

Table 2 Decoder register assignments.

	Coder	*msb*	*lsb*
Chigh register	Q	000xxxxx	xxxxxxxx
	Q in Qx	xxxxxxxx	xxxxxbbb
	QM	xxxxxxxx	xxxxxxxx
	QM in Qx	xxxxxxxx	xxxxxxxx
Clow register	Q	nnnnnnnn	ffffffff
	Q in Qx	bbbbb000	00000000
	QM	bbbbbbbb	00000000
	QM in Qx	bbbbbbbb	00000000
A register	Q	0001aaaa	aaaaaaaa
	Q in Qx	1aaaaaaa	aaaaa000
	QM	a aaaaaaaa	aaaaaaaa
	QM in Qx	aaaaaaaa	aaaaaaaa

The extra spacer bit, s, in the Q-coder was a concern until it was realized that the spacer-bit definition changed between the documentation of the Q-coder and the QM-coder. The Q-coder documentation had as many xs as the precision of the Qe values. The QM-coder documentation has one more x than the precision of its Qe values, so by the more recent definition, both have just three spacer bits. The flag bits f in the most significant byte of the Q-coder C register are not needed because a counter CT keeps track of when to output bytes. The carry bit c is shown to the left of the output byte position. If the previous byte was a 0xFF, the carry bit will be the most significant bit of the output byte at the stuffed-bit position.

```
T = (A - 1 + C) and 0xFFFF0000
if T < C
  C = T + 0x8000
else
  C = T
endif
```

Figure 2

QM-coder procedure CLEARBITS.

```
T = C OR 0x0000FFFF
if T > (C + A - 1)
  C + T - 0x8000
else
  C = T
endif
```

Figure 3

QM-coder procedure CLEARBITSx for hardware conventions.

```
Chigh = Chigh
if (0x10000 > (Clow + Alow))
  Clow = 0x7FFF
else
  Clow = 0xFFFF
endif
```

Figure 4

A simplified CLEARBITSx.

The A register has to be kept in alignment with the C register and therefore is also shifted left three bits for the Q-coder in the Qx-coder. It still fits in a 16-bit register. The extra 17th bit of precision in the QM-coder was

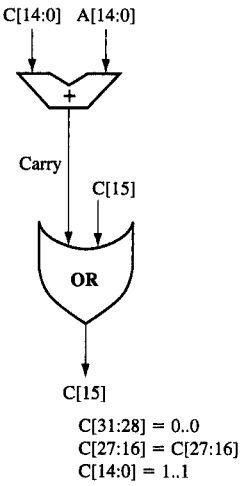

Figure 5

QM-coder procedure CLEARBITSx in hardware.

dropped. The JBIG specification explicitly states that the 17th bit can be avoided if initialization to 0x0000 produces the same output after the first subtraction as initialization to 0x10000 in the low-order 16 bits.

The decoder uses these same definitions of the A register and the `Amin` value. The entire renormalization-driven probability-estimation process is identical between encoder and decoder. Table 2 shows the decoder register assignments. The Q-coder registers are again shifted left three bits to align them with the QM-coder. The flag bits `f` in the least significant byte of the Q-coder `Clow` register are not needed because a counter `CT` identifies when to input another compressed data byte. The Q-coder input byte bits `n` are relabeled as `b` bits and are also shifted left three bits. The three `bbb` bits in the `Chigh` register do not disturb the decoding process because all tests against `Chigh` are "greater than or equal to" comparisons.

The QM-coder always has marker codes that follow the arithmetic-coded compressed data. Trailing 0x00 bytes may have been removed, so any missing data is filled in as 0x00 bytes. The Q-coder expects sufficient bits to completely decode the final decision in the compressed data stream. Since there are no marker codes, the end of compressed data will not always be known. To prevent waiting for nonexistent data upon a request for more data three shifts before the termination of compression, the Q-coder data is shifted up in the C register.

4. Hardware-optimized CLEARBITS

Since the Qx-coder selected the Q-coder symbol-ordering conventions, the flushing of the C register to terminate

Table 3 Qx-coder probability-estimation table for Q-coder.

Index	*Qe*	*NMPS*	*NLPS*	*SWITCH*	*Qe (binary)*			
0	5608	1	0	1	0101	0110	0000	1000
1	5408	2	0	0	0101	0100	0000	1000
2	5008	3	1	0	0101	0000	0000	1000
3	4808	4	2	0	0100	1000	0000	1000
4	3808	5	3	0	0011	1000	0000	1000
5	3408	6	4	0	0011	0100	0000	1000
6	3008	7	5	0	0011	0000	0000	1000
7	2808	8	5	0	0010	1000	0000	1000
8	2408	9	6	0	0010	0100	0000	1000
9	2208	10	7	0	0010	0010	0000	1000
10	1c08	11	8	0	0001	1100	0000	1000
11	1808	12	9	0	0001	1000	0000	1000
12	1608	13	10	0	0001	0110	0000	1000
13	1408	14	11	0	0001	0100	0000	1000
14	1208	15	12	0	0001	0010	0000	1000
15	0c08	16	13	0	0000	1100	0000	1000
16	0908	17	14	0	0000	1001	0000	1000
17	0708	18	15	0	0000	0111	0000	1000
18	0508	19	16	0	0000	0101	0000	1000
19	0388	20	17	0	0000	0011	1000	1000
20	02c8	21	18	0	0000	0010	1100	1000
21	0298	22	19	0	0000	0010	1001	1000
22	0138	23	20	0	0000	0001	0011	1000
23	00b8	24	21	0	0000	0000	1011	1000
24	0098	25	21	0	0000	0000	1001	1000
25	0058	26	23	0	0000	0000	0101	1000
26	0038	27	23	0	0000	0000	0011	1000
27	0028	28	25	0	0000	0000	0010	1000
28	0018	29	25	0	0000	0000	0001	1000
29	0008	29	27	0	0000	0000	0000	1000

Q-coding basically remained unchanged. However, the QM-coder software-optimized CLEARBITS procedure, designed to clear the most trailing bits, had to be converted to a hardware-optimized procedure CLEARBITSx designed to set the most trailing bits to 1. Then the output inversion process would clear those bits.

Figure 2 gives the QM-coder CLEARBITS procedure derived from Figure 29 in Reference [3]. A variable `T` (TEMP in the JBIG specification) is set to the top of the valid interval (`T = A - 1 + C`), and then its 16 least significant bits are cleared (`T = T AND 0xFFFF0000`). If `T` is no longer in the interval (`T < C`), 0x8000 is added back into `T` before setting `C` to `T`. This serial software approach is awkward in hardware.

Figures 1(b) and 1(c) show that the software-optimized upper point in the interval, (`C + A - 1`), is mapped directly to `C`. So a CLEARBITSx procedure with hardware conventions must set as many bits of `C` to 1 as possible, and, if the result is too large for the valid interval, decrement it by 0x8000. **Figure 3** defines this CLEARBITSx procedure. For the CLEARBITSx shown in Figure 3, let `Chigh` and `Clow` be the nonoverlapping most significant and least significant 16 bits of `C`, respectively. Since `A` uses only 16 bits, `A` equals `Alow`. Substituting these new definitions for `C` and `A` and eliminating the temporary variable `T` produces an alternate simplified CLEARBITSx, given in **Figure 4**.

Since `Alow` (after any required renormalizations) is always greater than or equal to 0x8000, the test is reduced to a bit test on `C[15]`. If the most significant bit of `Clow` is set (`C[15] = 1`), the test will always fail. If that bit is zero, the 15 low-order bits of `A` are added into `C` to allow a potential carry to set `C[15]`. The 15 low-order bits of `C` are set to 1s. The simplicity of this hardware is shown in **Figure 5**. A flowchart version of this figure is used for CLEARBITSx in Figure 16, shown later.

5. Probability estimation

The Q-coder and the QM-coder are both renormalization-driven probability estimators [3, 7, 8, 12]. Instead of collecting statistics on the occurrence of 0 and 1 decisions for each context `CX` and calculating probabilities, predetermined probabilities are referenced in a probability-estimation table. An index (`I`) into the probability-estimation table is saved at each context. From this index, the `Qe` value can be generated. The Qx-coder

Table 4 QM-coder probability-estimation table.

Index	Qe	NMPS	NLPS	SWITCH	Qe (binary)				Index	Qe	NMPS	NLPS	SWITCH	Qe (binary)			
0	5A1D	1	1	1	0101	1010	0001	1101	60	00F6	61	58	0	0000	0000	1111	0110
1	2568	2	14	0	0010	0101	0110	1000	61	00CB	62	59	0	0000	0000	1100	1011
2	1114	3	16	0	0001	0001	0001	0100	62	00AB	63	61	0	0000	0000	1010	1011
3	080B	4	18	0	0000	1000	0000	1011	63	008F	32	61	0	0000	0000	1000	1111
4	03D8	5	20	0	0000	0011	1101	1000									
5	01DA	6	23	0	0000	0001	1101	1010	64	5B12	65	65	1	0101	1011	0001	0010
6	00E5	7	25	0	0000	0000	1110	0101	65	4D04	66	80	0	0100	1101	0000	0100
7	006F	8	28	0	0000	0000	0110	1111	66	412C	67	81	0	0100	0001	0010	1100
8	0036	9	30	0	0000	0000	0011	0110	67	37D8	68	82	0	0011	0111	1101	1000
9	001A	10	33	0	0000	0000	0001	1010	68	2FE8	69	83	0	0010	1111	1110	1000
10	000D	11	35	0	0000	0000	0000	1101	69	293C	70	84	0	0010	1001	0011	1100
11	0006	12	9	0	0000	0000	0000	0110	70	2379	71	86	0	0010	0011	0111	1001
12	0003	13	10	0	0000	0000	0000	0011	71	1EDF	72	87	0	0001	1110	1101	1111
13	0001	13	12	0	0000	0000	0000	0001	72	1AA9	73	87	0	0001	1010	1010	1001
									73	174E	74	72	0	0001	0111	0100	1110
14	5A7F	15	15	1	0101	1010	0111	1111	74	1424	75	72	0	0001	0100	0010	0100
15	3F25	16	36	0	0011	1111	0010	0101	75	119C	76	74	0	0001	0001	1001	1100
16	2CF2	17	38	0	0010	1100	1111	0010	76	0F6B	77	74	0	0000	1111	0110	1011
17	207C	18	39	0	0010	0000	0111	1100	77	0D51	78	75	0	0000	1101	0101	0001
18	17B9	19	40	0	0001	0111	1011	1001	78	0BB6	79	77	0	0000	1011	1011	0110
19	1182	20	42	0	0001	0001	1000	0010	79	0A40	48	77	0	0000	1010	0100	0000
20	0CEF	21	43	0	0000	1100	1110	1111									
21	09A1	22	45	0	0000	1001	1010	0001	80	5832	81	80	1	0101	1000	0011	0010
22	072F	23	46	0	0000	0111	0010	1111	81	4D1C	82	88	0	0100	1101	0001	1100
23	055C	24	48	0	0000	0101	0101	1100	82	438E	83	89	0	0100	0011	1000	1110
24	0406	25	49	0	0000	0100	0000	0110	83	3BDD	84	90	0	0011	1011	1101	1101
25	0303	26	51	0	0000	0011	0000	0011	84	34EE	85	91	0	0011	0100	1110	1110
26	0240	27	52	0	0000	0010	0100	0000	85	2EAE	86	92	0	0010	1110	1010	1110
27	01B1	28	54	0	0000	0001	1011	0001	86	299A	87	93	0	0010	1001	1001	1010
28	0144	29	56	0	0000	0001	0100	0100	87	2516	71	86	0	0010	0101	0001	0110
29	00F5	30	57	0	0000	0000	1111	0101									
30	00B7	31	59	0	0000	0000	1011	0111	88	5570	89	88	1	0101	0101	0111	0000
31	008A	32	60	0	0000	0000	1000	1010	89	4CA9	90	95	0	0100	1100	1010	1001
32	0068	33	62	0	0000	0000	0110	1000	90	44D9	91	96	0	0100	0100	1101	1001
33	004E	34	63	0	0000	0000	0100	1110	91	3E22	92	97	0	0011	1110	0010	0010
34	003B	35	32	0	0000	0000	0011	1011	92	3824	93	99	0	0011	1000	0010	0100
35	002C	9	33	0	0000	0000	0010	1100	93	32B4	94	99	0	0011	0010	1011	0100
									94	2E17	86	93	0	0010	1110	0001	0111
36	5AE1	37	37	1	0101	1010	1110	0001									
37	484C	38	64	0	0100	1000	0100	1100	95	56A8	96	95	1	0101	0110	1010	1000
38	3A0D	39	65	0	0011	1010	0000	1101	96	4F46	97	101	0	0100	1111	0100	0110
39	2EF1	40	67	0	0010	1110	1111	0001	97	47E5	98	102	0	0100	0111	1110	0101
40	261F	41	68	0	0010	0110	0001	1111	98	41CF	99	103	0	0100	0001	1100	1111
41	1F33	42	69	0	0001	1111	0011	0011	99	3C3D	100	104	0	0011	1100	0011	1101
42	19A8	43	70	0	0001	1001	1010	1000	100	375E	93	99	0	0011	0111	0101	1110
43	1518	44	72	0	0001	0101	0001	1000									
44	1177	45	73	0	0001	0001	0111	0111	101	5231	102	105	0	0101	0010	0011	0001
45	0E74	46	74	0	0000	1110	0111	0100	102	4C0F	103	106	0	0100	1100	0000	1111
46	0BFB	47	75	0	0000	1011	1111	1011	103	4639	104	107	0	0100	0110	0011	1001
47	09F8	48	77	0	0000	1001	1111	1000	104	415E	99	103	0	0100	0001	0101	1110
48	0861	49	78	0	0000	1000	0110	0001									
49	0706	50	79	0	0000	0111	0000	0110	105	5627	106	105	1	0101	0110	0010	0111
50	05CD	51	48	0	0000	0101	1100	1101	106	50E7	107	108	0	0101	0000	1110	0111
51	04DE	52	50	0	0000	0100	1101	1110	107	4B85	103	109	0	0100	1011	1000	0101
52	040F	53	50	0	0000	0100	0000	1111									
53	0363	54	51	0	0000	0011	0110	0011	108	5597	109	110	0	0101	0101	1001	0111
54	02D4	55	52	0	0000	0010	1101	0100	109	504F	107	111	0	0101	0000	0100	1111
55	025C	56	53	0	0000	0010	0101	1100									
56	01F8	57	54	0	0000	0001	1111	1000	110	5A10	111	110	1	0101	1010	0001	0000
57	01A4	58	55	0	0000	0001	1010	0100	111	5522	109	112	0	0101	0101	0010	0010
58	0160	59	56	0	0000	0001	0110	0000									
59	0125	60	57	0	0000	0001	0010	0101	112	59EB	111	112	1	0101	1001	1110	1011

Table 5 Qx-coder probability-estimation table for JPEG-FA.

Index	*Qe*	*NMPS*	*NLPS*	*SWITCH*	*Qe (binary)*			
0	5601	1	1	1	0101	0110	0000	0001
1	3401	2	6	0	0011	0100	0000	0001
2	1801	3	9	0	0001	1000	0000	0001
3	0ac1	4	12	0	0000	1010	1100	0001
4	0521	5	29	0	0000	0101	0010	0001
5	0221	38	33	0	0000	0010	0010	0001
6	5601	7	6	1	0101	0110	0000	0001
7	5401	8	14	0	0101	0100	0000	0001
8	4801	9	14	0	0100	1000	0000	0001
9	3801	10	14	0	0011	1000	0000	0001
10	3001	11	17	0	0011	0000	0000	0001
11	2401	12	18	0	0010	0100	0000	0001
12	1c01	13	20	0	0001	1100	0000	0001
13	1601	29	21	0	0001	0110	0000	0001
14	5601	15	14	1	0101	0110	0000	0001
15	5401	16	14	0	0101	0100	0000	0001
16	5101	17	15	0	0101	0001	0000	0001
17	4801	18	16	0	0100	1000	0000	0001
18	3801	19	17	0	0011	1000	0000	0001
19	3401	20	18	0	0011	0100	0000	0001
20	3001	21	19	0	0011	0000	0000	0001
21	2801	22	19	0	0010	1000	0000	0001
22	2401	23	20	0	0010	0100	0000	0001
23	2201	24	21	0	0010	0010	0000	0001
24	1c01	25	22	0	0001	1100	0000	0001
25	1801	26	23	0	0001	1000	0000	0001
26	1601	27	24	0	0001	0110	0000	0001
27	1401	28	25	0	0001	0100	0000	0001
28	1201	29	26	0	0001	0010	0000	0001
29	1101	30	27	0	0001	0001	0000	0001
30	0ac1	31	28	0	0000	1010	1100	0001
31	09c1	32	29	0	0000	1001	1100	0001
32	08a1	33	30	0	0000	1000	1010	0001
33	0521	34	31	0	0000	0101	0010	0001
34	0441	35	32	0	0000	0100	0100	0001
35	02a1	36	33	0	0000	0010	1010	0001
36	0221	37	34	0	0000	0010	0010	0001
37	0141	38	35	0	0000	0001	0100	0001
38	0111	39	36	0	0000	0001	0001	0001
39	0085	40	37	0	0000	0000	1000	0101
40	0049	41	38	0	0000	0000	0100	1001
41	0025	42	39	0	0000	0000	0010	0101
42	0015	43	40	0	0000	0000	0001	0101
43	0009	44	41	0	0000	0000	0000	1001
44	0005	45	42	0	0000	0000	0000	0101
45	0001	45	43	0	0000	0000	0000	0001

has unique probability-estimation tables for the Q-coder and the QM-coder. Formats for these tables are presented in **Table 3** and **Table 4**. A third table, nicknamed the JPEG-FA table for its reuse of `Qe` values in two "fast attack" paths, is presented in **Table 5** [8].

An entry in the probability estimation table consists of a current index, `I`, a probability estimate, `Qe`, two possible next indices, `NMPS` and `NLPS`, and a `SWITCH` value. As each decision is processed, the index is changed any time a renormalization is required. For renormalizations triggered by the coding of an MPS, the `NMPS` gives the new index. Renormalizations are always required after the coding of an LPS. The `NLPS` gives the new index for this case. In addition, on an LPS renormalization, the sense of the MPS bit stored at each context may have to be switched, as indicated by a 1 in the `SWITCH` column.

Table 6 Differences between the Q-coder and QM-coder, and the Qx-coder solution.

Q-coder	*QM-coder*	*Qx-coder*
Hardware convention	Software convention	Convert software convention to hardware convention
LPS at bottom	MPS at bottom	LPS at bottom
12-bit Qe values	15-bit Qe values	Left-justify 12-bit Qe values
13-bit A register	16- or 17-bit A register	16-bit A register
30 Qe values (5 bits/context)	113 Qe values (7 bits/context)	Both at 7 bits/context
Describes four spacer bits	Describes three spacer bits	Same number when properly defined
LPS/MPS boundary with MPS	LPS/MPS boundary with LPS	LPS/MPS boundary with MPS
Renormalization on A < 0x1000	Renormalization on A < 0x8000	Shift Q-coder A register left three bits so renormalization on A < 0x8000 for both
Decoder resolves carry	Encoder resolves carry	Both
	ENCODER	
25-bit C register	28-bit C register	28-bit C register (25-bit shifted left three bits)
Flush C register	Clear 15 or 16 bits	Both (see CLEARBITSx procedure)
Initialize: A = 0x1000 CT = 12	Initialize: A = 0 CT = 11	Both
Bit stuffing	Byte stuffing	Both
Compressed data	JBIG standard compressed data	Both—Invert JBIG compressed data on output
None	Conditional exchange	Conditional exchange for QM-coder only
	DECODER	
Bit unstuffing	Byte unstuffing	Both
Compressed data	JBIG standard compressed data	Both—Invert JBIG compressed data on output
C register equals 0 when done	Supply 0x00s when out of data	Both
Initialize: A = 0x1000	Initialize: A = 0	Both

In Table 4, many of the Qe values are used at most once per context because they are part of the initial learning. Only 46 entries are part of the nontransient QM estimation state machine. The gaps in Table 4 show where the Qe values break a monotonically decreasing pattern. At these breaks, the NMPS index is not an increment of 1 away from the current index.

The JPEG-FA table has a much simpler initial learning structure. It is a variant of the Q-coder table extended to 15-bit precision. The JBIG committee has selected this table for the MQ-coder used in the new JBIG-2 lossy and lossless compression algorithm [13]. The MQ-coder follows the hardware conventions of the Q-coder with bit stuffing, but always assigns the MPS to the largest subinterval (conditional exchange), as is done in the QM-coder. Since probability estimates have fifteen bits of precision, the three trailing zeros of the Q-coder in the Qx-coder are replaced with real data.

6. Summary

The hardware-optimized Q-coder and the software-optimized QM-coder are merged into a common hardware-optimized Qx-coder. The Q-coder 13-bit A register located in the Qx-coder corresponds to the high-order 13 bits of the 16-bit QM-coder A register. Similarly, the Q-coder C register in the Qx-coder is shifted left three

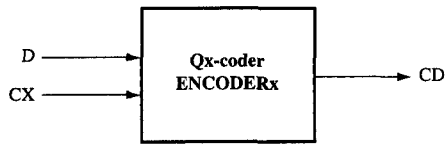

Figure 6

Qx-coder ENCODERx inputs and output.

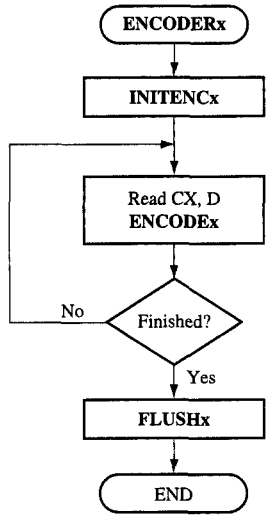

Figure 7

Encoder for the Qx-coder.

bits in the QM-coder C register. This allows the output byte to be co-located in the hardware. The QM-coder hardware-optimized compressed data is inverted at the output to create the identical QM-coder software-optimized compressed data. The CLEARBITSx procedure is optimized for hardware and can be executed in one cycle compared to the CLEARBITS and Clear_final_bits procedures given in the JBIG and JPEG standards respectively, which take several cycles. Both Q-coder bit stuffing and QM-coder byte stuffing are implemented.

Table 6 summarizes the differences between the Q-coder and the QM-coder and the solution chosen for the Qx-coder. A complete set of flowcharts for the Qx-coder are presented and discussed in the Appendix. Test sequences for the Q-coder are found in [6]; test sequences for the QM-coder are found in Table 26 of [3] and Annex K of [7] and [8]. More details about the rest of the ABIC/JBIG ASIC core which implements the Qx-coder architecture are found in companion papers [4, 5] and on the IBM Blue Logic Internet site [14].

Table 7 Comparing Qx-coder with Q-coder and QM-coder notations.

Qx-coder	*JBIG QM-coder*	*ABIC Q-coder*	*JPEG QM-coder*
A	A	A	A
AND	&	AND	AND
B	BUFFER	B	B
C	C	C	C
CD	SCD	—	—
Chigh	CHIGH	Cx	Cx
COUNT	—	CT	—
CT	CT	bits in C	CT
CX	CX	S	S
D	PIX	YN	D
I(CX)	ST(CX)	Qe_index	Index(S), I
MPS(CX)	MPS(CX)	MPS(S)	MPS(S)
NLPS[I(CX)]	NLPS[ST(CX)]	Qe_index−Decr_LPS	Next_Index_LPS(I)
NMPS[I(CX)]	NMPS[ST(CX)]	Qe_index+Incr_MPS	Next_Index_MPS(I)
Qe[I(CX)]	LSZ[ST(CX)]	Qe(S)	Qe(S)
SC	SC	—	SC
SWITCH[I(CX)]	SWTCH[ST(CX)]	MPSexch_flag	Switch_MPS(I)
T	TEMP	—	T
XOR	—	—	—
1−MPS(CX)	1−MPS(CX)	LPS(S)	1−MPS(S)
≪	≪	SLL	SLL
≫	≫	SRL	SRL

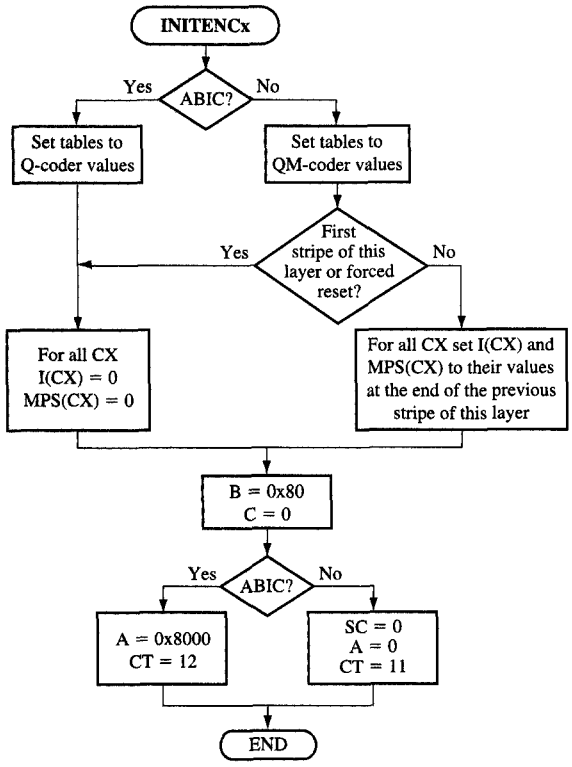

Figure 8

Initialization of the encoder.

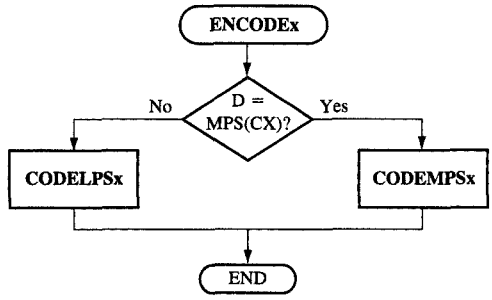

Figure 9

Deciding to encode an MPS or an LPS.

Appendix: Description of the Qx-coder flowcharts

Figures 6 to 25 show flowcharts for the merged Qx-coder. A bold-faced name in a flowchart block indicates that a more detailed description of a procedure with that name can be found in the flowcharts. All tests in the flowcharts are logical, unsigned comparisons.

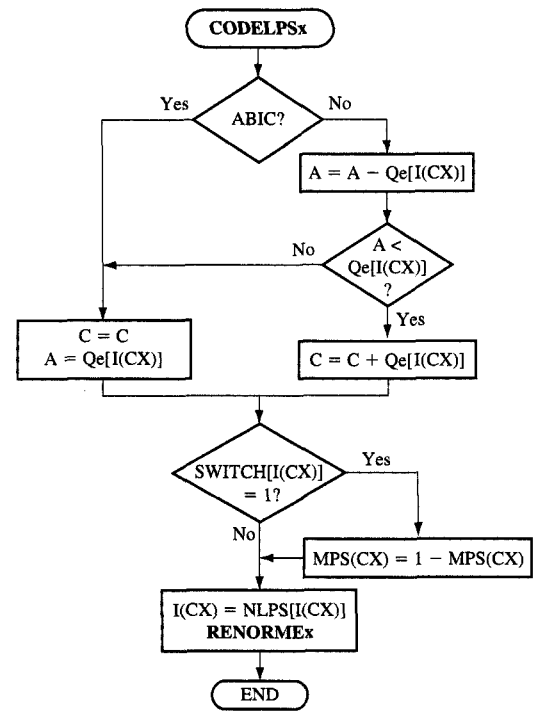

Figure 10

LPS encode and probability-estimate update.

The definitions used in the flowcharts are as follows:

A A 16-bit register containing the current interval.

ABIC A status flag selecting the Q-coder from the ABIC algorithm or the QM-coder from the JBIG algorithm. It also selects which probability-estimation table to use. If the ABIC status flag is 0, `Qe`, `NMPS`, `NLPS`, and `SWITCH` are taken from the QM-coder values in Table 3. If the ABIC status flag is 1, these values are taken from Table 3.

AND The AND logical operator.

B The latest compressed byte output in the encoder or compressed byte input in the decoder.

C A register containing the least significant bits of the encoder's compressed data and the most

	significant bits of the decoder's compressed data.
CD	The compressed data, including stuffed bytes/bits.
Chigh	The 16 high-order bits of C.
COUNT	The number of extra bits flushed for ABIC to guarantee that the decoder has enough bits to completely decode all decisions.
CT	The counter that identifies byte boundaries and, therefore, when to output or input bytes of compressed data.
CX	The context generated by the model. The model unit provides the same context to the encoder and decoder for a given decision.
D	The binary decision to be coded (0/1). In the encoder it is determined by the model unit; in the decoder, it is output from the Qx-decoder.
"Finished"	A status flag. For ABIC it is set when all of the lines have been encoded or decoded. For JBIG it is set when all of the lines in a stripe have been encoded or decoded.
"First stripe of this layer or forced reset"	A status flag that selects either initializing the statistics or preserving them.
I(CX)	The index of the current probability estimate for context CX.
MPS(CX)	The sense of the more probable symbol (0/1) for the given context CX
NLPS[I(CX)]	The next index for context CX after coding an LPS.
NMPS[I(CX)]	The next index for context CX after coding an MPS that triggered a renormalization.
Qe[I(CX)]	The current estimate of the less probable symbol probability.
SC	The stack counter used only during QM-coding.
SWITCH[I(CX)]	The switch flag which indicates that the sense of MPS(CX) must be switched after coding an LPS.
T	A temporary variable used for intermediate results.
XOR	The EXCLUSIVE-OR logical operator.
$\ll$	The binary shift left logical operator.
$\gg$	The binary shift right logical operator.

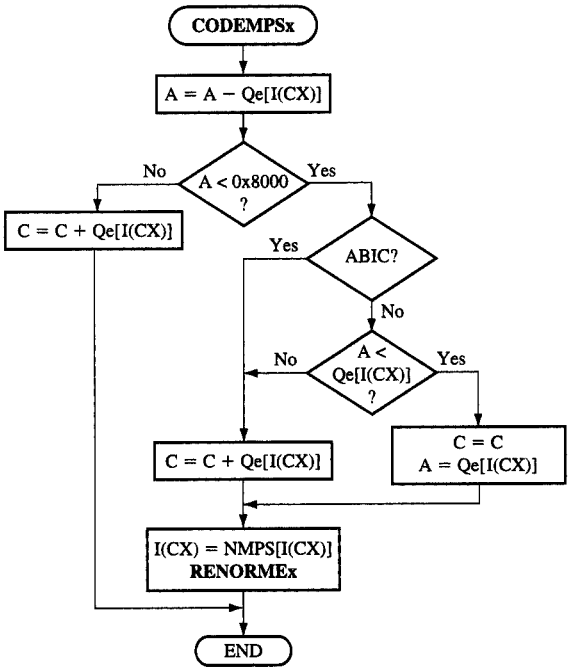

Figure 11

MPS encode and probability-estimate update.

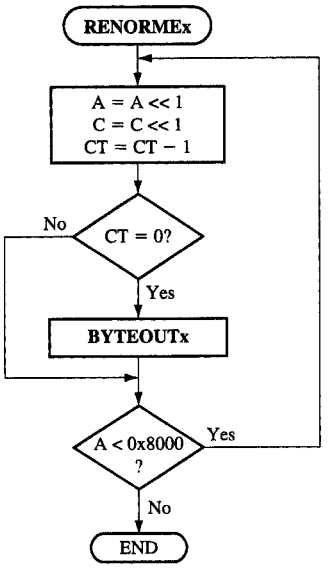

Figure 12

Encoder renormalization.

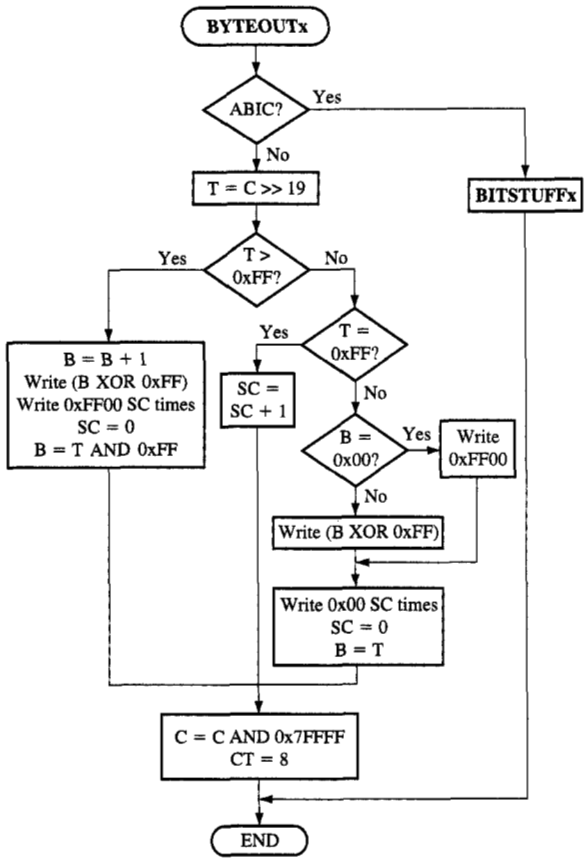

Figure 13

Shifting a byte from the C register to CD with byte stuffing.

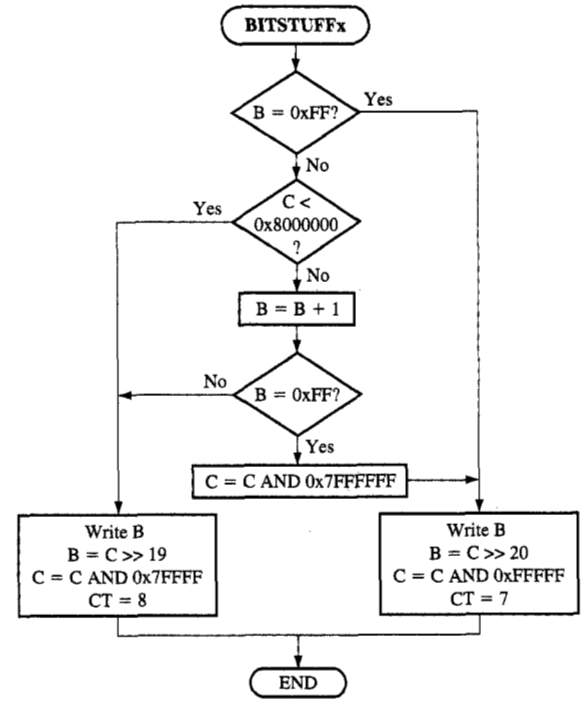

Figure 14

Shifting a byte from the C register to CD with bit stuffing.

Figure 6 is a block diagram showing the inputs and output to the Qx-encoder, ENCODERx. Figures 7 to 17 illustrate in greater detail the functional block in Figure 6. Figure 18 is a block diagram showing the inputs and output to the Qx-decoder, DECODERx. Figures 19 to 25 show more details of this decoder.

Table 7 compares the Qx-coder flowchart symbols with the notation used for the QM-coder as documented in the JBIG standard [3], with the notation used for the Q-coder [6], and with the notation used for the QM-coder as documented in the JPEG standard [7, 8].

Figure 6 shows a simple block diagram of the Qx-coder binary adaptive arithmetic encoder. The decision (D) and context (CX) pairs are processed together in the procedure ENCODERx to produce compressed data (CD) output. Both D and CX are provided by the model unit (not shown). CX selects the probability estimate to use during the coding of D. For a compression ratio of 10:1, about 80 decisions are coded before a byte of compressed data is output.

ENCODERx (**Figure 7**) initializes the Qx-coder through the INITENCx procedure. CX and D pairs are read and passed on to ENCODEx until all pairs have been read. Bytes of compressed data are output when no longer modifiable. When all of the CX and D pairs have been read (Finished?), FLUSHx concatenates the contents of the C register to the compressed data stream for ABIC and clears as many bits as possible for JBIG.

INITENCx (**Figure 8**) initializes the probability-estimation tables to either the ABIC values (Table 3) or the JBIG values (Table 4). For all ABIC images, the index I(CX) for the less-probable-symbol probability estimate for all contexts is set to 0, and the more-probable-symbol MPS(CX) for all contexts is also set to 0. This is also true for the first stripe of all JBIG images and any images that use a forced reset between stripes. If a JBIG image does not reset the contexts between stripes, the context indices and MPS remain as they were at the end of the previous stripe at the same level.

INITENCx initializes B to 0x80 and C to 0. For ABIC images, A is initialized to 0x8000, and the byte boundary

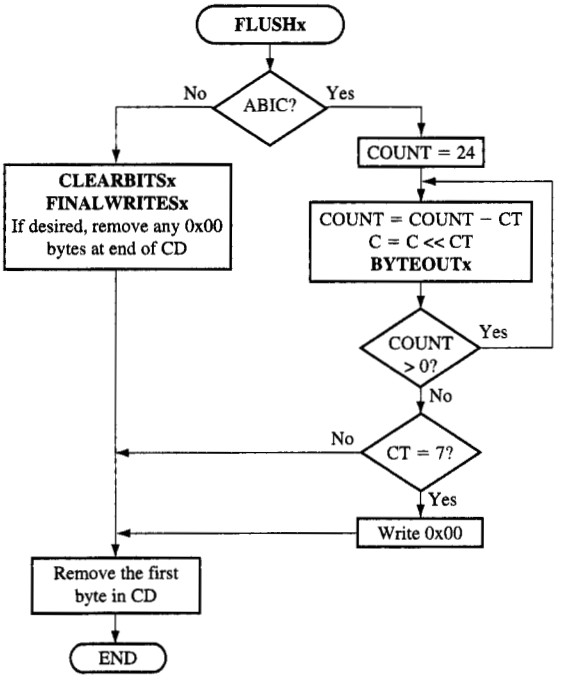

Figure 15

Flushing the final bits to CD.

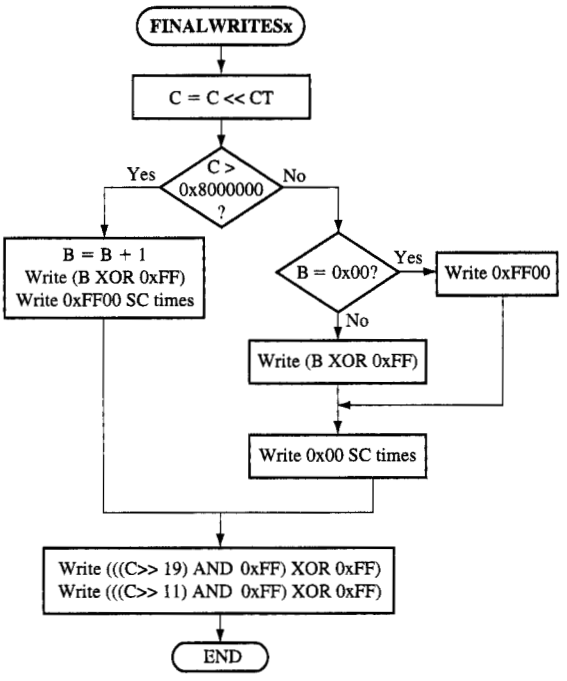

Figure 17

Writing out the final bytes with byte stuffing.

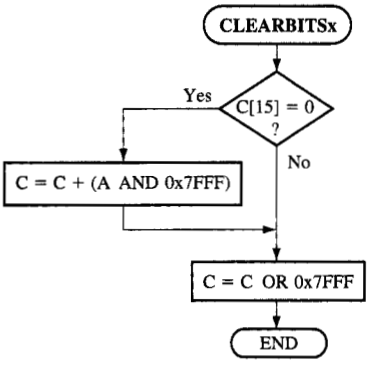

Figure 16

Clearing the final bits in the C register.

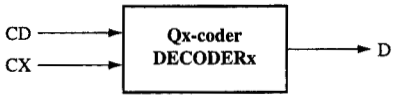

Figure 18

Qx-coder DECODERx inputs and output.

counter (CT) is initialized to 12. For JBIG images, A is initialized to 0x0000, and CT is initialized to 11. The stack counter is also used for JBIG images, and it is initialized to 0. If A were more than 16 bits for JBIG, it would be initialized to 0x10000. A note in the JBIG specification clarifies that 16 bits is sufficient if the effect of the initial subtraction from 0x0000 is the same as the initial subtraction from 0x10000.

ENCODEx (**Figure 9**) uses the context CX and decision D pair. It compares D with the more probable symbol for CX, MPS(CX). If they are equal, the decision is coded as a more probable symbol in CODEMPSx. Otherwise, the decision is coded as a less probable symbol in CODELPSx.

CODELPSx (**Figure 10**) codes a less probable symbol. If the algorithm is ABIC, C remains the same, and A is set to

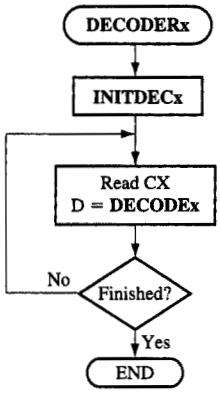

Figure 19

Decoder for the Qx-coder.

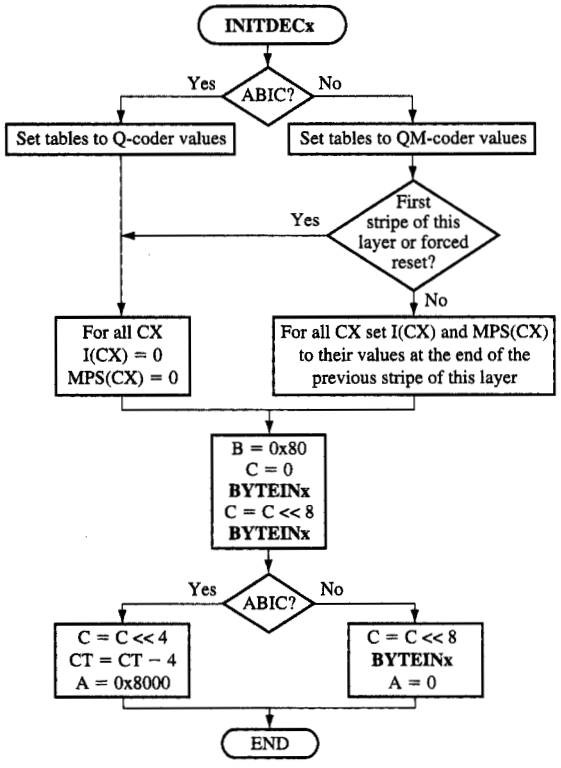

Figure 20

Initialization of the decoder.

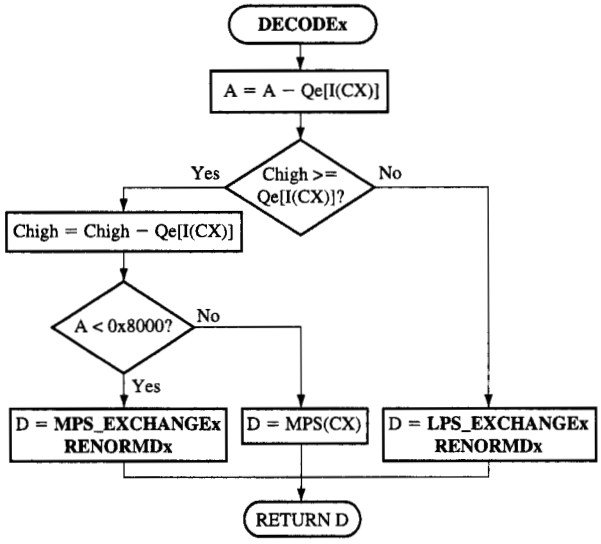

Figure 21

Decoding an MPS or an LPS.

`Qe[I(CX)]`. If the algorithm is JBIG, `A` is decremented by `Qe[I(CX)]` and compared to `Qe[I(CX)]` in the conditional exchange test. If `A` is less than `Qe[I(CX)]`, `C` is incremented by `Qe[I(CX)]`; otherwise, `C` remains the same and `A` is set to `Qe[I(CX)]`. Next, if `SWITCH(CX)` is set, the more probable symbol for the current context is switched to the opposite symbol. The index into the probability-estimation table for context `CX`, `I(CX)`, is updated with `NLPS[I(CX)]`, and RENORMEx is called.

CODEMPSx (**Figure 11**) codes a more probable symbol. `A` is reduced by the LPS probability, `Qe[I(CX)]`. If `A` is not below 0x8000, `C` is incremented by `Qe[I(CX)]` and CODEMPSx is complete. If `A` drops below 0x8000 and the algorithm is ABIC, `C` is increased by `Qe[I(CX)]`. If `A` drops below 0x8000 and the algorithm is JBIG, `A` is tested to see whether a conditional exchange is required (`A` < `Qe[I(CX)]`). If `A` is less than `Qe[I(CX)]`, `C` remains the same, and `A` is set to `Qe[I(CX)]`; otherwise, `C` is incremented by `Qe[I(CX)]`. `I(CX)` is then updated with the value stored in `NMPS[I(CX)]`, and RENORMEx is called. RENORMEx (**Figure 12**) left-shifts `A` and `C` until `A` is greater than or equal to 0x8000. After each shift, `CT` is decremented and then tested. If `CT` equals 0, BYTEOUTx is called.

BYTEOUTx (**Figure 13**) first checks which algorithm is being used. If the algorithm is ABIC, BITSTUFFx is called, and BYTEOUTx is complete. If the algorithm is JBIG, the temporary variable `T` is assigned `C` right-shifted by 19. If `T` is greater than 0xFF, a carry has occurred, and the byte in the output buffer, `B`, is incremented, inverted, and written to the compressed data stream. Then, the two

bytes 0xFF00 are written into the compressed data stream SC times (including no times if SC is still zero). The propagation of the carry through the counted 0xFF bytes converted them to 0x00 bytes. However, the inversion process restores them to 0xFF, and byte stuffing causes each 0xFF byte to be followed immediately by a stuffed 0x00 byte. SC is set to 0, and B is assigned the least significant eight bits of T.

If the test to see whether T is greater than 0xFF failed, then, if T is equal to 0xFF, the stack counter SC is incremented. Otherwise, B is tested for equality with 0x00. If so, the pair of bytes 0xFF00 are written into the compressed data stream; otherwise B is inverted and written. The byte 0x00 is written SC times; SC is set to 0, and B is set to T. Finally, the byte placed in the output buffer and the carry bit are removed from C by masking off the leftmost significant nine bits. CT is set to 8.

BITSTUFFx (**Figure 14**) tests to see whether the byte in the output buffer, B, is equal to 0xFF. If it is, B is written to the compressed data stream and then set to the seven most significant bits of C. These bits are then cleared from C, and CT is set to 7, indicating that bit stuffing has occurred. If B was not 0xFF, C is tested for a carry. If there was a carry, B is incremented and tested again. If B is now equal to 0xFF, the carry bit is removed from C; B is written and assigned the seven most significant bits of C; the seven most significant bits are cleared from C; and CT is set to 7. If B is not equal to 0xFF or no carry was set in C, B is written and assigned the eight most significant bits of C; the eight most significant bits of C are cleared, and CT is set to 8.

FLUSHx (**Figure 15**) first tests which algorithm is being used. If the algorithm is ABIC, COUNT is set to 24. COUNT is then decremented by CT; C is left-shifted by CT; and BYTEOUTx is called. These three steps are repeated while COUNT is greater than 0. Once COUNT is less than or equal to 0, if CT equals 7, 0x00 is written to output the trailing stuffed bit. If the algorithm is JBIG, CLEARBITSx and FINALWRITESx are called. If desired, any trailing 0x00 bytes at the end of CD may also be removed. FLUSHx is completed for both algorithms when the first byte in CD is removed.

CLEARBITSx (**Figure 16**) tests bit 15 of C. If bit 15 is 0, the sum of C and A masked with 0x7FFF will not carry into bit 16 of C, and the addition can be performed. The addition determines whether bit 15 of C will be set to 0 or 1. In either case, the 15 least significant bits of C are then set to 0x7FFF.

FINALWRITESx (**Figure 17**) left-shifts C by CT. If C is greater than 0x8000000, a carry exists. The byte in the output buffer, B, is incremented, inverted, and written; and the pair of bytes 0xFF00 is written SC times. If a carry does not exist, B is tested for equality with 0x00. If so, the pair of bytes 0xFF00 are written into the compressed data

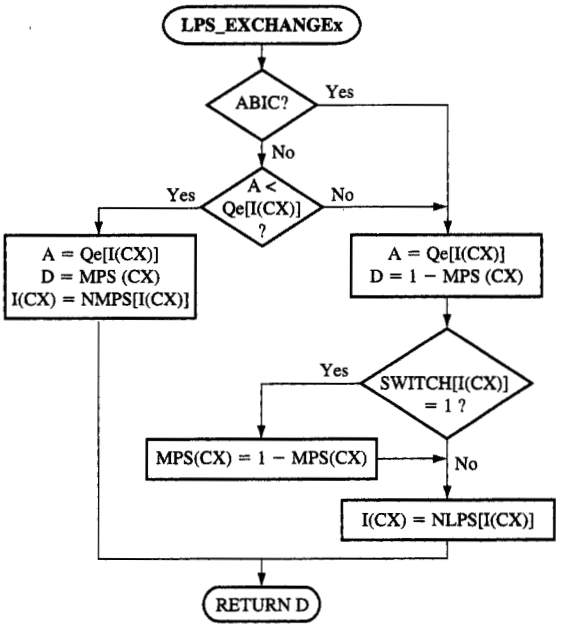

Figure 22

LPS decode and probability-estimate update.

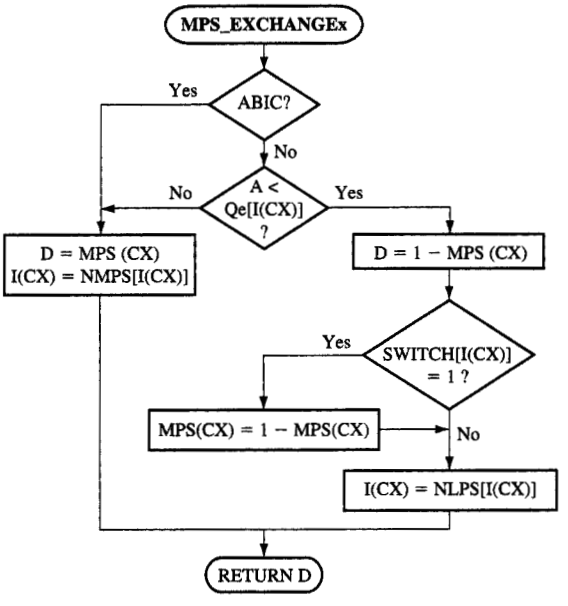

Figure 23

MPS decode and probability-estimate update.

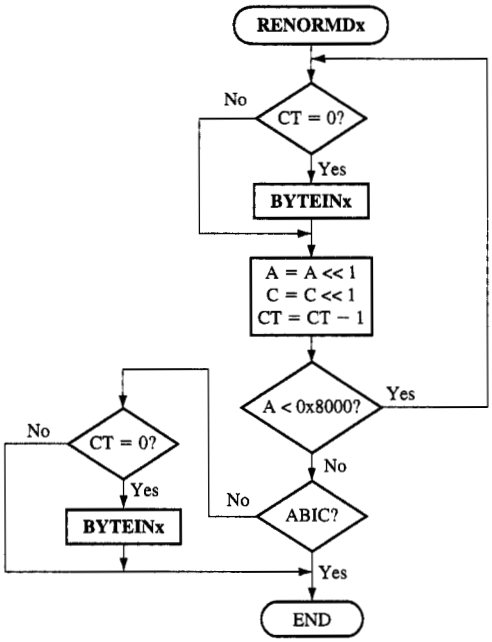

Figure 24

Decoder renormalization.

stream; otherwise B is inverted and written. The byte 0x00 is written SC times. Finally, the most significant 16 bits of the C register, not including the carry bit, are inverted and written.

Figure 18 shows a simple block diagram of a binary adaptive arithmetic decoder. The compressed data CD and a context CX from the decoder's model unit (not shown) are input to the arithmetic decoder, DECODERx. The decoder's output is the decision D. The encoder and decoder model units must supply exactly the same context CX for each given decision.

DECODERx (**Figure 19**) initializes the Qx-coder through INITDECx. Contexts, CX, and bytes of compressed data (as needed) are read and passed on to DECODEx until all contexts have been read. When all contexts have been read, the coded data has been decompressed.

INITDECx (**Figure 20**) initializes the probability-estimation tables to either the ABIC values (Table 3) or the JBIG values (Table 4). For all ABIC images, the index of the MPS probability estimate I(CX) and the more probable symbol MPS(CX) for all contexts are both set to 0. This is also true for the first stripe of all JBIG images and any images that use a forced reset between stripes. If a JBIG image does not reset the contexts between stripes, the context indices and more probable symbols remain as they were at the end of the previous stripe of this layer. B is initialized to 0x80. C is initialized to 0. BYTEINx is called to read in a byte of compressed data. C is left-shifted by 8. BYTEINx is called to read in another byte of coded data. For ABIC images, C is left-shifted by 4; CT is decremented by 4; and A is initialized to 0x8000. For JBIG images, C is left-shifted by 8; BYTEINx is called a third time; and A is initialized to 0. For implementations in which A has more than 16 bits, it is initialized to 0x10000.

DECODEx (**Figure 21**) decrements A by Qe[I(CX)]. If Chigh is less than Qe[I(CX)], LPS_EXCHANGEx and RENORMDx are called, and DECODEx completes. If Chigh is greater than or equal to Qe[I(CX)], Chigh is decremented by Qe[I(CX)]. If A is less than 0x8000, MPS_EXCHANGEx and RENORMDx are called; otherwise, the decision, D, is set to MPS(CX). D is returned to the calling DECODERx routine.

LPS_EXCHANGEx (**Figure 22**) tests which algorithm is used. If the algorithm is ABIC, A is set to Qe[I(CX)] and D is set to the LPS value, i.e., 1 − MPS(CX). If SWITCH[I(CX)] is set, MPS(CX) is inverted. The index, I(CX), is updated with NLPS[I(CX)]. The decoded decision D is returned to the calling routine.

If the algorithm is JBIG, A is tested against Qe[I(CX)]. If A is greater than or equal to Qe[I(CX)], the ABIC path is taken as above; otherwise, A is set to Qe[I(CX)], D is set to MPS(CX), and the I(CX) is updated with NMPS[I(CX)]. The decoded decision D is returned to the calling routine.

MPS_EXCHANGEx (**Figure 23**) tests which algorithm is used. If the algorithm is ABIC, D is set to MPS(CX). The index, I(CX), is updated with NMPS[I(CX)]. The decoded decision D is returned to the calling routine.

If the algorithm is JBIG, A is tested against Qe[I(CX)]. If A is greater than or equal to Qe[I(CX)], the ABIC path is taken as above; otherwise, D is set to the LPS value, 1 − MPS(CX). If SWITCH[I(CX)] is set, MPS(CX) is inverted. The index, I(CX), is updated with NLPS[I(CX)]. The decoded decision D is returned to the calling routine.

RENORMDx (**Figure 24**) left-shifts A and C and decrements CT until A is greater than or equal to 0x8000. Before each shift, if CT is 0, BYTEINx is called to read in another byte of compressed data. Once A is greater than or equal to 0x8000, if the algorithm is JBIG and CT equals 0, BYTEINx is called again.

BYTEINx (**Figure 25**) tests which algorithm is used. If the algorithm is ABIC, bit stuffing is used. For bit unstuffing, B is tested. If B is 0xFF, one byte of compressed data is read from CD, and B is set equal to it. C is incremented by B left-shifted 12 bits. The counter CT is set equal to 7, indicating that the byte includes a stuffed bit. If B is not equal to 0xFF, one byte of compressed data is read from CD, and B is set equal to it. C is incremented

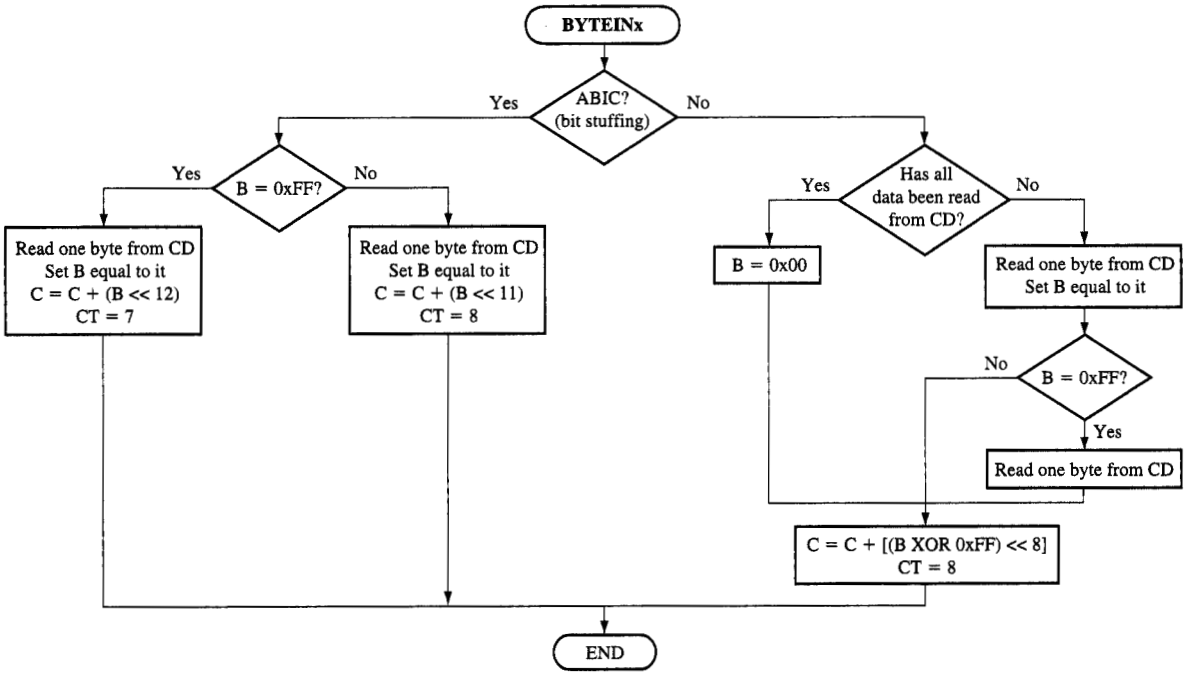

Figure 25

Inserting a new byte of CD into the C register.

by B left-shifted 11 bits. The counter CT is set to 8, indicating that eight bits are to be processed before another byte of compressed data is needed.

If the algorithm is not ABIC, a test is done to see whether all of the compressed data has been read from CD. If all the data has been read, B is set equal to 0x00. If not, a new byte is read from CD, and B is set equal to it. If B is equal to 0xFF, the next byte is read from CD. Once B is set, increment C by an inverted B left-shifted by eight bits. CT is set to 8.

Acknowledgments

The authors thank S. H. Burroughs for his persistence in asking whether merging a Q-coder and a QM-coder in VLSI was practical, the Xionics Corporation for demonstrating a market for the Qx-coder, and their colleagues, P. S. Colyer, F. A. Kampf, and K. M. Marks, for helpful discussions. In addition, they thank their managers, J. A. Oleszkiewicz, F. C. Mintzer, and T. R. Lattrell, for encouragement and support of this work; W. B. Pennebaker for his invaluable contributions to the pioneering work on hardware- and software-optimized arithmetic coding structures; R. B. Arps and G. G. Langdon, who played significant roles in laying the foundation of the hardware ABIC; and C. Constantinescu for rapid code updates to the software that was used to validate the design. The Q-coder work was done as a collaboration between the IBM Thomas J. Watson and Almaden Research Centers.

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. W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr., and R. B. Arps, "An Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic Coder," *IBM J. Res. Develop.* **32,** 717–726 (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. K. M. Marks, "A JBIG/ABIC Compression Engine for Digital Document Processing," *IBM J. Res. Develop.* **42,** 753–758 (1998, this issue).
5. S. H. Burroughs and T. R. Lattrell, "Data Compression Technology in ASIC Cores," *IBM J. Res. Develop.* **42,** 725–731 (1998, this issue).
6. J. L. Mitchell and W. B. Pennebaker, "Software Implementations of the Q-Coder," *IBM J. Res. Develop.* **32,** 753–774 (1988).
7. *ITU-T Rec. T81\|ISO/IEC 10918-1:1993 Information Technology*—Coded Representation of Picture and Audio

Information—Digital Compression and Coding of Continuous-Tone Still Images (JPEG standard).
8. W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard*, Van Nostrand Reinhold, New York, 1993.
9. 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).
10. J. L. Mitchell and W. B. Pennebaker, "Arithmetic Coding Data Compression/De-Compression by Selectively Employed, Diverse Arithmetic Coding Encoders and Decoders," U.S. Patent 4,891,643, January 2, 1990.
11. F. A. Kampf, "Performance as a Function of Compression," *IBM J. Res. Develop.* **42,** 759–766 (1998, this issue).
12. W. B. Pennebaker and J. L. Mitchell, "Probability Estimation for the Q-Coder," *IBM J. Res. Develop.* **32,** 737–752 (1988).
13. JBIG Committee, "Working Draft 14492 as of 20 April 1998," *ISO/IEC JTC 1/SC29/WG1 N839*, April 20, 1998.
14. *http://www.chips.ibm.com/products/asics/cores.html.*

Received February 3, 1998; accepted for publication May 21, 1998

Michael J. Slattery *IBM Microelectronics Division, Burlington facility, Essex Junction, Vermont 05452 (mslatter@us.ibm.com).* Mr. Slattery graduated from the University of Notre Dame with a B.S. degree in electrical and computer engineering. He joined the IBM Microelectronics Division in 1990 and has worked in VLSI failure analysis, polyimide films and via process engineering, encryption product design, adaptive and lossless data compression product design, arithmetic coding for VLSI implementations, and 2D graphical acceleration.

Joan L. Mitchell *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (joanm@us.ibm.com).* Dr. Mitchell graduated from Stanford University with a B.S. degree in physics in 1969. She received her M.S. and Ph.D. degrees in physics from the University of Illinois at Champaign–Urbana in 1971 and 1974, respectively, joining the Exploratory Printing Technologies group at the IBM Thomas J. Watson Research Center immediately after completing her Ph.D. She was a manager at the Research Center for nine years, worked for three years in IBM Marketing, and returned to the IBM Research Division in 1991 to work again in the Image Technologies group. In 1994, she left for a two-year leave of absence. During her leave, Dr. Mitchell co-authored a book on MPEG, consulted for IBM Burlington, and was a visiting professor at the University of Illinois for six months. Back at the IBM Thomas J. Watson Research Center, she is now a Research Staff Member in the Image Applications Department. Since 1976 Dr. Mitchell has worked in the field of image processing and data compression. She received IBM Outstanding Innovation Awards for two-dimensional data compression in 1978, for teleconferencing in 1982, for the image view facility in 1985, for resistive ribbon thermal transfer printing technology in 1985, for speed-optimized software implementations of image compression algorithms in 1991, and for the Q-coder in 1991. She is a member of APS, IEEE, Sigma Xi, and IS&T and is a co-inventor on 30 patents.