To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

USER'S MANUAL

Phase-out/Discontinued

μSAP70732-B02
μSAP703000-B02

JBIG MIDDLEWARE

TARGET DEVICE

μSAP70732-B02: V810 FAMILY™

μSAP703000-B02: V850 FAMILY™

Document No. U11057EJ2V0UM00 (2nd edition)
Date Published October 1996 P
Printed in Japan

T.82 and T.85 related patents:

Multiple patents exist for systems compliant with ITU-T Recommendations T.82 and T.85.
The customer is requested to handle the necessary rights related to these patents.
Note that NEC will assume no responsibility for any of these patents.

The information in this document is subject to change without notice.

NEC Corporation assumes no responsibility for any errors which may appear in this document.

NEC Corporation does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from use of a device described herein or any other liability arising from use of such device. No license, either express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Corporation or of others.

Phase-out/Discontinued

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Mountain View, California
Tel: 800-366-9782
Fax: 800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.1.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby Sweden
Tel: 8-63 80 820
Fax: 8-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-719-2377
Fax: 02-719-5951

NEC do Brasil S.A.
Sao Paulo-SP, Brasil
Tel: 011-889-1680
Fax: 011-889-1689

Major Revisions in This Version

Page	Contents
Throughout	Description about V850 family was added
p.16	Memory capacity in 1.3.3 (3) Memory size required was changed
p.17	Directory name in 1.3.4 Directory configuration was changed
p.19	Memory capacity in Table 2-1. Free Parameters was changed
p.20, 29	Description about VLENGTH was changed
p.21	Descriptions in 2.1.2 (2) and (3) were added and changed
p.23	2.1.4 Status Transition was added
p.26, 33, 38, 41	newlen_err_sts, invalid_code_sts, and length_err were added
p.30	Description on 2.2.1 (14) sdrst was added
p.31	Description on 2.2.1 (18) Tx, yAT was added
p.33, 34	Description on 2.2.1 (19) was added and changed
p.35	Description on 2.2.1 (21) and (22) sdrst were added
p.39	Description on Table 2-5 and 2-6 were added
p.42, 43	Description on Table 2-8 and 2-9 were added and changed
p.43	Description on 2.2.2 (2) (b) Image data was changed
p.48	Description on 3.2.1 (2) GHS version make file was changed
p.54, 55	Figure 4-1 and Figure 4-2 were changed
p.56	Memory capacity in Figure 4-3 V810 Family Memory Map Example was changed
p.59 to 61	APPENDIX (1) JBIG compression sample source was changed
p.62, 63	APPENDIX (2) JBIG expansion sample source was changed

The mark ★ shows major revised points.

PREFACE

Intended Readership : This manual is intended for users who are designing and developing V800 series™ application systems.

Purpose : This manual is intended to help users to understand the functions of the μSAP70732-B02 and 703000-B02.

Organization : This manual is broadly organized as follows:

- General Description
- Library Specifications
- Installation
- System Examples
- Appendix

Legend : Note : Explanation of items marked with Note in the text
Caution : Item to be especially noted
Remark : Supplementary information
Numeric notations : Binary ... xxxx or xxxxB
Decimal ... xxxx
Hexadecimal ... 0xxxxx or xxxxH
Suffix denoting the power of 2 (address space, memory capacity)
K (kilo) : $2^{10} = 1024$
M (mega) : $2^{20} = 1024^2$

Related Documents : Some related documents are preliminary versions, but please not that these documents are not marked “preliminary”.

V810 family related documents

Document Name	Data Sheet	User's Manual	
Product Name		Hardware	Architecture
V805™	ID-3292	IEU-1371	
V810™	ID-3293	IEU-1370	
V820™	ID-3301	Planned	
V821™	Planned	Planned	

V850 family related documents

Document Name	Data Sheet	User's Manual	
Product Name		Hardware	Architecture
V851™	Planned	U10935E	U10243E
V852™	Planned	U10038E	
V853™	Planned	U10913E	

V810 family development tool related documents

Document Name		Document No.
IE-70732-BX-A User's manual		On preparation
IE-70732-MC User's manual		On preparation
IE-70742-BX User's manual		Planned
IE-70741-BX User's manual		Planned
CA732 User's manual	Operation (UNIX^{TM} based)	U11013E
	Operation (Windows^{TM} based)	On preparation
	Assembly language	U11016E
	C language	U11010E
ID732 User's manual	Operation (UNIX based)	Planned
	Operation (Windows based)	Planned
	Installation (UNIX based)	Planned
	Installation (Windows based)	Planned
RX732 User's manual	Basic	U10346E
	Installation	U10347E
	Technical	U10490E
AZ732 User's manual	Operation	Planned

V850 family development tool related documents

Document Name		Document No.
IE-703000-MC-A User's manual - Hardware		On preparation
IE-703002-MC User's manual		Planned
IE-703003-MC User's manual		Planned
CA850 User's manual	Operation (Windows based)	On preparation
	Operation (UNIX based)	U11013E
	Assembly language	U10543E
	C language	U11010E
ID850 User's manual	Operation (Windows based)	Planned
	Installation (Windows based)	Planned
RX850 User's manual	Basic	U11037E
	Installation	U11038E
	Technical	U11117E
AZ850 User's manual	Operation	On preparation

Development tool related documents

For inquiries about the tools produced by Green Hills Software^{TM}, Inc. (GHS), please contact:

Green Hills Software, Inc.
510 Castillo Street, Santa Barbara,
California 93101

TEL: (805) 965-6044

Phase-out/Discontinued

[MEMO]

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

[MEMO]

CHAPTER 1 GENERAL DESCRIPTION

1.1 Middleware

Middleware consists of a group of software fine tuned to be able to exploit the performance of a processor. Middleware uses software to implement the processing that has been conventionally done by hardware. The concept of middleware becomes a reality thanks to the advent of high performance RISC processors and the establishment of an environment facilitating incorporation of the RISC into a system.

NEC offers the element technology for the V800 series to achieve a multimedia system. For example, NEC offers middleware such as voice CODEC and image data compression/expansion to assist development of the customer's system.

This product is the middleware to provide the FAX CODEC function.

1.2 JBIG

JBIG (Joint Bi-level Image Group) is a standard encoding system for 2-value still images (images that have only black and white colors) and has been recommended as ITU-T Recommendation T.82. The JBIG FAX version has newly been recommended as T.85 for use with FAX. Hereafter, the explanation of JBIG is common to T.82 and T.85, while that of the JBIG FAX version is only applicable to T.85.

- High compression performance
  Demonstrates high compression performance approximately 1.1 to 1.5 times that of the conventional MH/MR/MMR systems in character based documents. Compression/expansion is made available also for natural images.

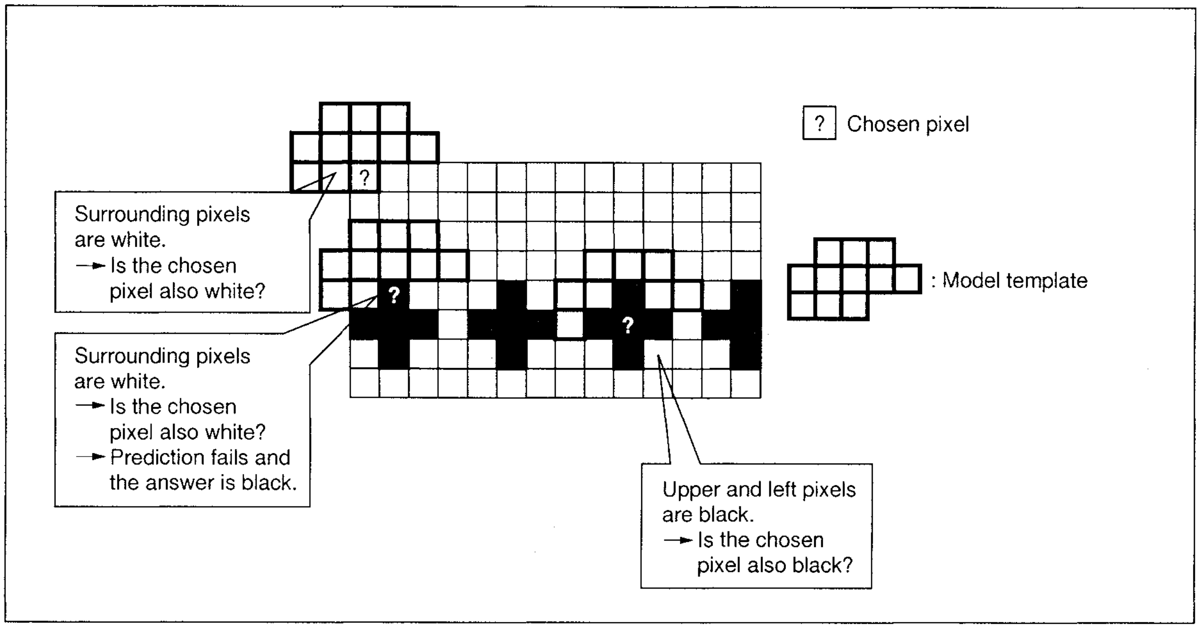
- Information integrity
  Preserves information integrity throughout compression/expansion (the JPEG system sometimes has problems with information integrity).

1.2.1 JBIG encoding

Encoding by JBIG is applied only when the black/white prediction of a chosen pixel according to the state of surrounding pixels (model template) fails (as shown in Figure 1-1). The μSAP70732-B02 learns from each prediction to increase the prediction hit rate in order to reduce the amount of encoded data.

In decoding, it also learns from each prediction, and thus only information for when prediction has failed suffices. Codes are represented in binary decimals between 0 and 1 by arithmetic encoding (refer to **1.2.9 AAE (Arithmetic Encoding)**).

Figure 1-1. JBIG Encoding

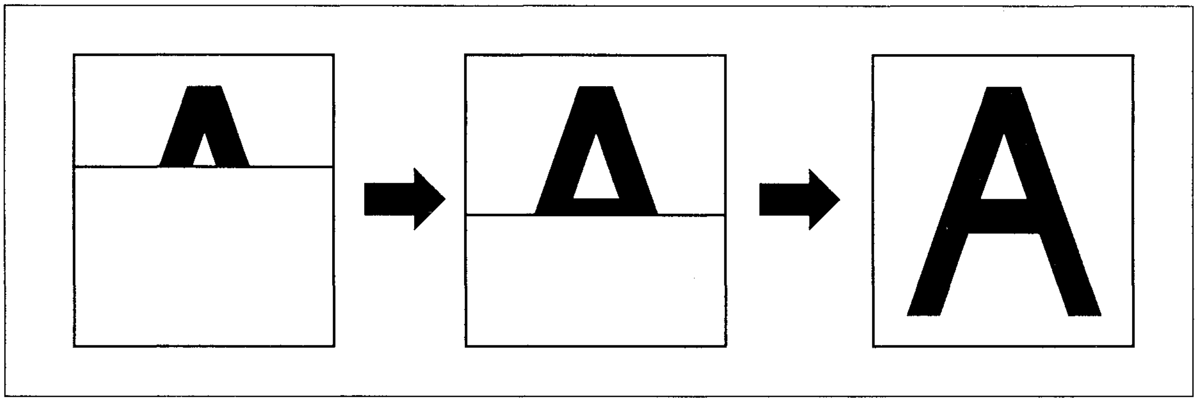

? Chosen pixel
: Model template
?
?
?
Surrounding pixels are white.
→ Is the chosen pixel also white?
Surrounding pixels are white.
→ Is the chosen pixel also white?
→ Prediction fails and the answer is black.
Upper and left pixels are black.
→ Is the chosen pixel also black?

1.2.2 JBIG transmission method

JBIG supports two transmission systems: sequential transmission (images are sequentially transmitted from top to bottom) and progressive transmission (a relatively coarse image is transmitted first and the precision of image quality is gradually improved by transmission of additional information). Progressive transmission is a layer-like transmission system.

The JBIG FAX version only supports sequential transmission. Figure 1-2 shows sequential transmission.

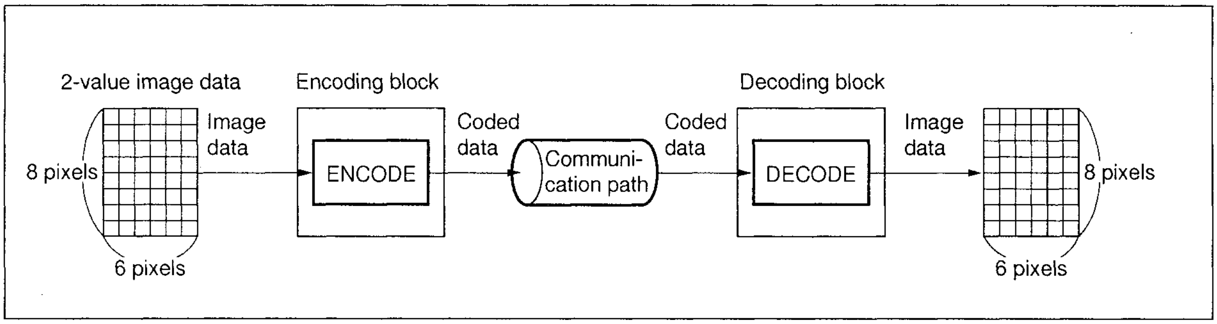
Figure 1-2. Sequential Transmission



1.2.3 Sequential transmission

An example of 2-value image data processing by sequential transmission is shown below.

Figure 1-3. Example of 2-Value Image Data Processing by Sequential Transmission



The JBIG FAX version free parameters necessary for transmission of 2-value image data layer 0 in Figure 1-3 are shown below.

- Number of difference layer[Note 1] (D) = 0
- Number of bit plane[Note 2] (P) = 1
- Number of pixels in horizontal direction in layer (Xd) = 6
- Number of pixels in vertical direction in layer (Yd) = 8
- Number of pixels in a stripe[Note 3] in layer (L0) = 4

Notes 1. Layer used for progressive transmission. In sequential transmission it is not required, so the number of difference layers is 0.

2. Since the JBIG FAX version uses black/white 2-value images, the number of bit planes is 1. 2 or more bit planes are used for color pixels.

3. Blocks obtained by splitting one page by every few lines.

1.2.4 Encoding

JBIG FAX encoding consists of the following 4 blocks.

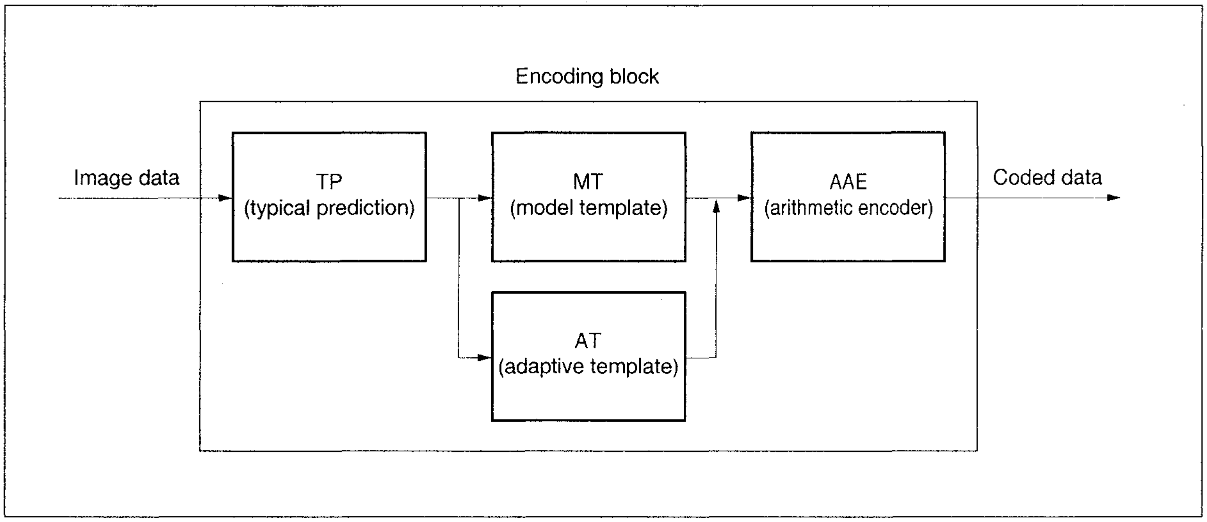
TP : Processing that reduces the number of pixels to be encoded. If the line to be encoded is the same as the preceding line, it is represented by one-bit information and not encoded (refer to **1.2.6 TP (Typical Prediction)**).

MT : Pattern of pixels surrounding the chosen pixel to be encoded (refer to **1.2.7 MT (Model Template)**).

AT : One of reference pixels of a model template (MT). It is a special pixel whose location can be changed. It is used for encoding of pixels that have correlation in a fixed period such as dither images (refer to **1.2.8 AT (Adaptive Template)**).

AAE : Arithmetic encoder. It predicts black/white from the contents of the model template (MT) for a given pixel, improves prediction by learning and outputs coded data (refer to **1.2.9 AAE (Arithmetic Encoding)**).

Figure 1-4. Encoding Block Diagram



1.2.5 Decoding

JBIG FAX decoding consists of the following 4 blocks.

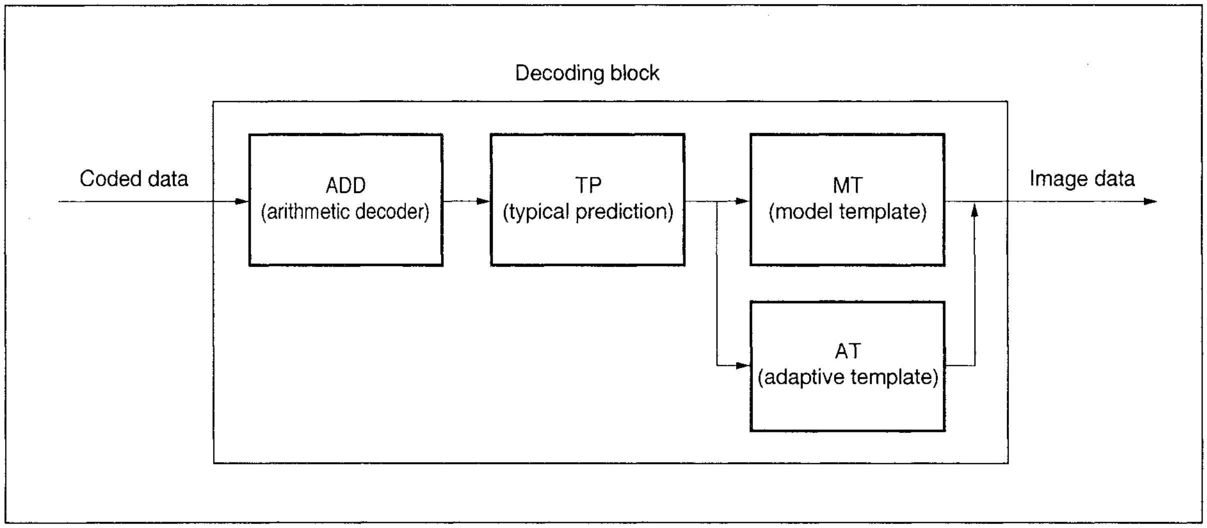
AAD : Arithmetic decoder.

TP : Processing that reduces the number of pixels to be decoded. If the line to be decoded is the same as the preceding line, it outputs the image data of the preceding line (refer to **1.2.6 TP (Typical Prediction)**).

MT : Pattern of pixels surrounding the pixel to be decoded (refer to **1.2.7 MT (Model Template)**).

AT : One of reference pixels of a model template (MT). It is a special pixel whose location can be changed. It is used for encoding of pixels that have correlation in a fixed period such as dither images (refer to **1.2.8 AT (Adaptive Template)**).

Figure 1-5. Decoding Block Diagram

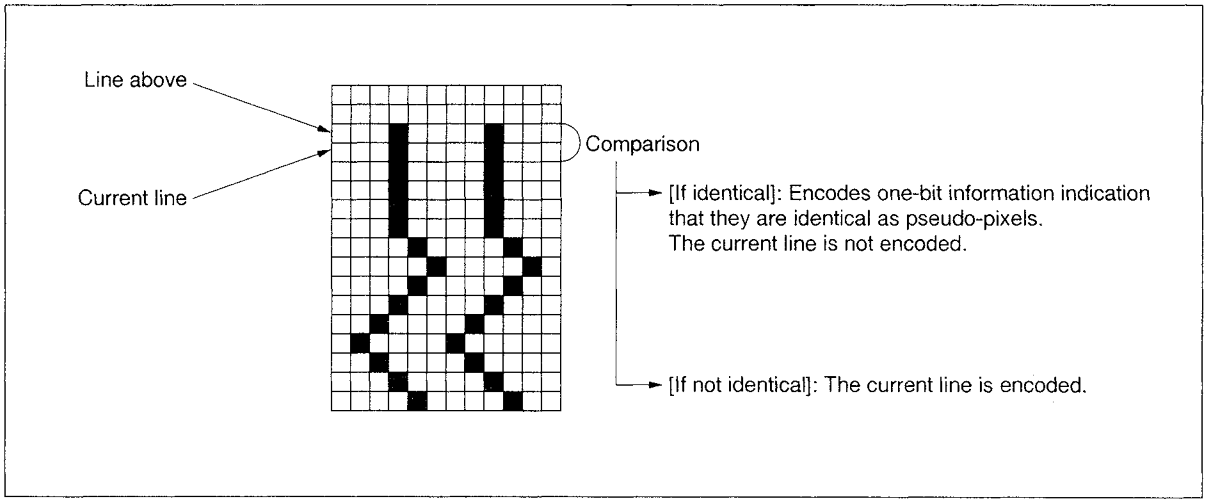


1.2.6 TP (Typical Prediction)

Processing that reduces the number of pixels to be encoded or decoded. In encoding, if a line to be encoded (current line) is the same as the previous line, it is represented by one-bit information and not encoded. In decoding, if a line to be decoded is the same as the previous line, the image data of the previous line is output.

Figure 1-6 shows current line pixel comparison. If the current line and the line above are compared and if they are identical, the one-bit information indicating that they are identical is encoded as pseudo-pixels and the current line is not encoded. This processing can reduce the number of pixels to be encoded and reduce coded data.

Figure 1-6. Comparison with Current Line

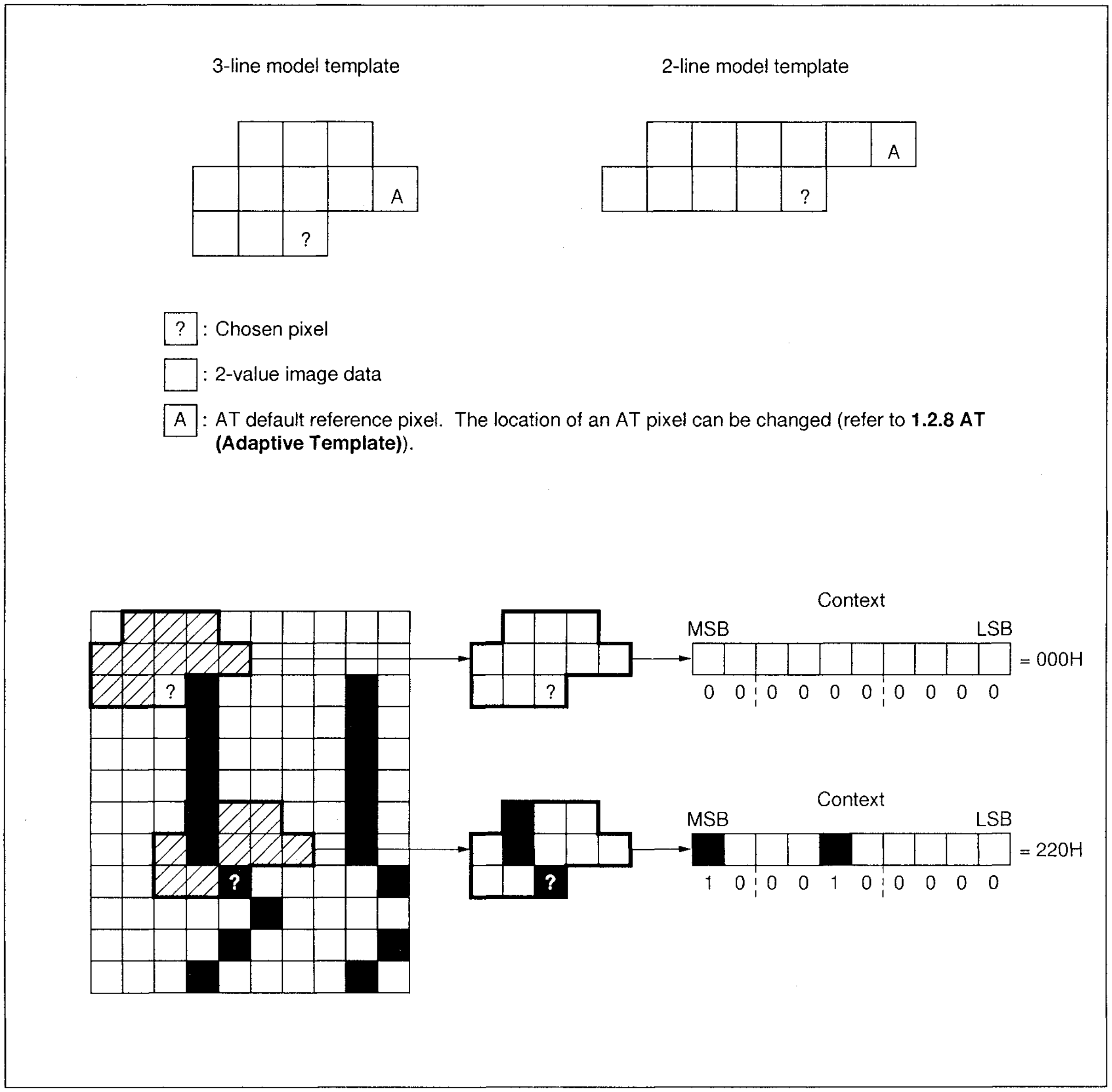

Line above
Current line
Comparison
[If identical]: Encodes one-bit information indication that they are identical as pseudo-pixels.
The current line is not encoded.
[If not identical]: The current line is encoded.

1.2.7 MT (Model Template)

The model template is a reference pixel pattern used as a prediction model for encoding. There are two kinds of model templates: 3-line model templates and 2-line model templates.

Figure 1-7 shows the procedure to convert a model template (2-dimensional) to a context (model template made to be 1-dimensional).

Figure 1-7. Model Template



? : Chosen pixel

□ : 2-value image data

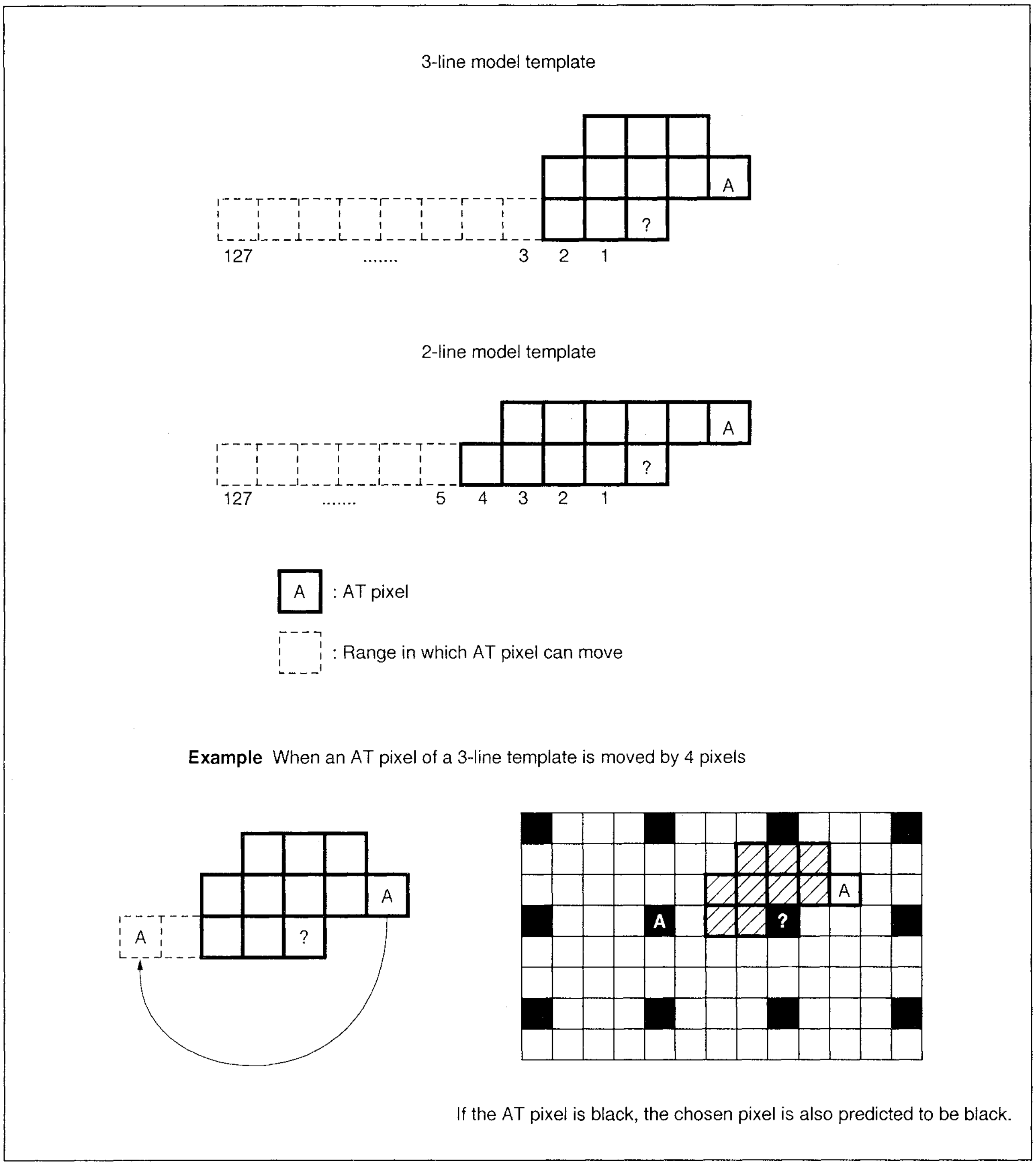
A : AT default reference pixel. The location of an AT pixel can be changed (refer to **1.2.8 AT (Adaptive Template)**).

1.2.8 AT (Adaptive Template)

AT is one of the model template reference pixels. Moving pixel “A” in Figure 1-8 is an effect on encoding of pixels that have correlation in a fixed period such as dither images.

An AT pixel can move by a maximum of 127 pixels. However, it should be set ensuring that it does not overlap pixels in the model template.

Figure 1-8. AT Pixels


3-line model template

2-line model template

Example When an AT pixel of a 3-line template is moved by 4 pixels

If the AT pixel is black, the chosen pixel is also predicted to be black.

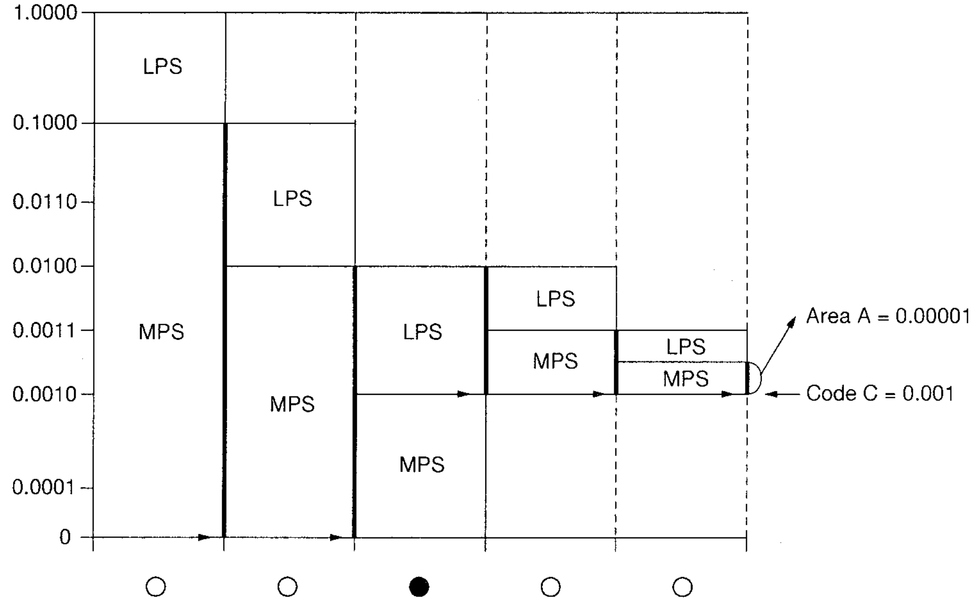
1.2.9 AAE (Arithmetic Encoding)

In JBIG, the color (black or white) of the chosen pixel is predicted according to the states of surrounding pixels (model template). The prediction results are represented in binary decimals between 0 and 1 using arithmetic encoding.

Figure 1-9 shows an example of encoding assuming that the probability of prediction proving accurate is 1/2 and that of prediction proving inaccurate is 1/2. Encoded symbols that appear frequently are called More Probable Symbols (MPS) and those that appear less frequently are called Less Probable Symbols (LPS). In Figure 1-9, white symbols are designated MPSs.

Figure 1-10 shows an example of encoding assuming that the probability of white prediction proving accurate is 1/2 and that of white prediction proving inaccurate is 1/2 (example of decoding the code C data).

Figure 1-9. Example of Arithmetic Encoding (when P_0 = 0.1, P_1 = 0.1)



Area A :	0.1	0.01	0.001	0.0001	0.00001
Code C :	0	0	0.001	0.001	0.001

A : Area that shows the probability of black/white permutation (initial value = 1.00)

C : Value that becomes data (initial value = 0)

P0 : Probability of MPS occurrence

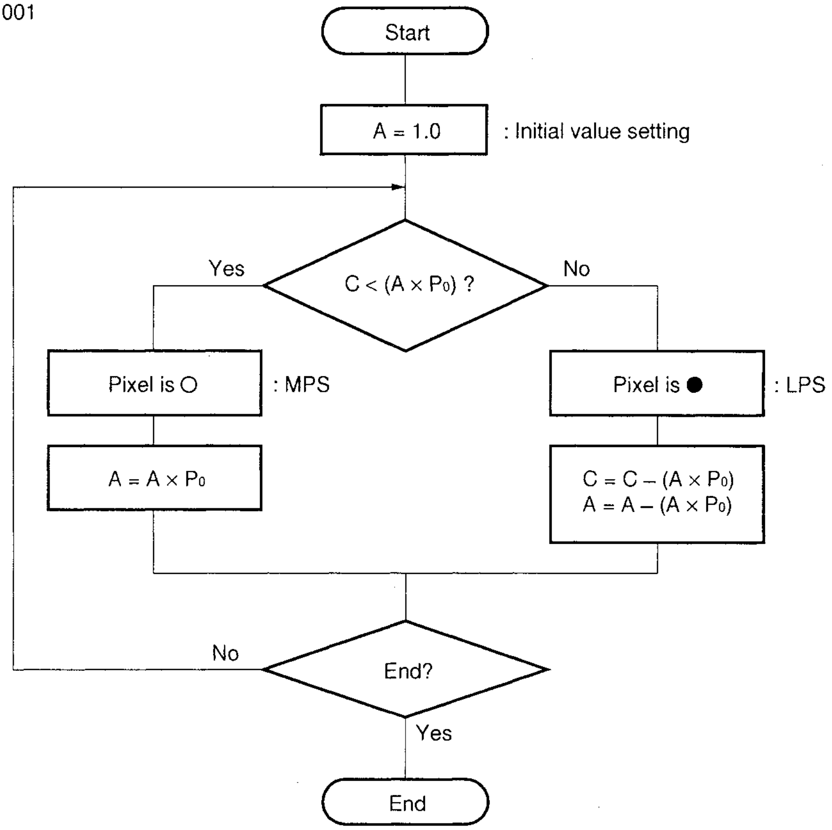
P1 : Probability of LPS occurrence

[Encoding expression] Area A : MPS : $A = A \times P_0$; LPS : $A = A \times P_1$

Code C : MPS : $C = C$; LPS : $C = C + (A \times P_0)$

Figure 1-10. Example of Decoding Flow

Area A = 0.00001
Code C = 0.001



A : Area that shows the probability of black/white permutation (initial value = 1.00)

C : Area that becomes data (initial value = 0)

P_0 : Probability of MPS occurrence

P_1 : Probability of LPS occurrence

1st loop : C = 0.001 < (1.0 × 0.1) = 0.1, therefore the 1st pixel is ○
A = 1.0 × 0.1 = 0.1

2nd loop : C = 0.001 < (0.1 × 0.1) = 0.01, therefore the 2nd pixel is ○
A = 0.1 × 0.1 = 0.01

3rd loop : C = 0.001 = (0.01 × 0.1) = 0.001, therefore the 3rd pixel is ●
C = 0.001 – (0.01 × 0.1) = 0
A = 0.01 – (0.01 × 0.1) = 0.01 – 0.001 = 0.001

4th loop : C = 0 < (0.001 × 0.1) = 0.0001, therefore the 4th pixel is ○
A = 0.001 × 0.1 = 0.0001

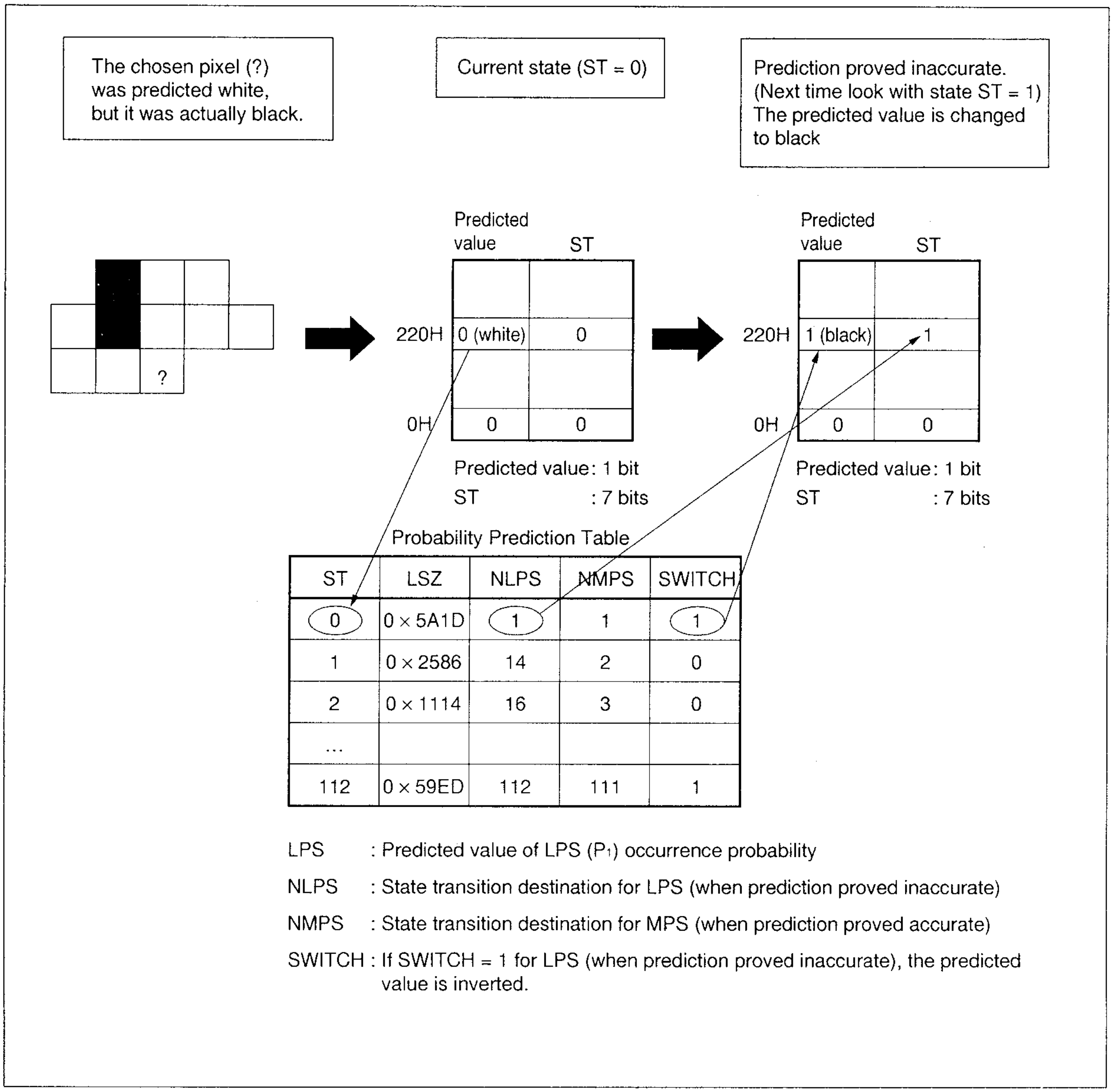
5th loop : C = 0 < (0.0001 × 0.1) = 0.000001, therefore the 5th pixel is ○

The context table and probably prediction table which are used for arithmetic encoding are explained below.

The context table stores the value and state of a pixel predicted from the value of the model template (10 pixels).

The probability prediction table shows the LPS area width (LSZ), the next state transition number indicated by the context table (NLPS, NMPS), and inversion of the predicted value (SWITCH). These values are obtained using statistical techniques.

Figure 1-11. Context Table and Probability Prediction Table



The chosen pixel (?) was predicted white, but it was actually black.

Current state (ST = 0)

Prediction proved inaccurate. (Next time look with state ST = 1) The predicted value is changed to black

	Predicted value	ST
220H	0 (white)	0
0H	0	0

Predicted value: 1 bit
ST : 7 bits

	Predicted value	ST
220H	1 (black)	1
0H	0	0

Predicted value: 1 bit
ST : 7 bits

Probability Prediction Table

ST	LSZ	NLPS	NMPS	SWITCH
0	0 × 5A1D	1	1	1
1	0 × 2586	14	2	0
2	0 × 1114	16	3	0
...				
112	0 × 59ED	112	111	1

LPS : Predicted value of LPS (P_1) occurrence probability

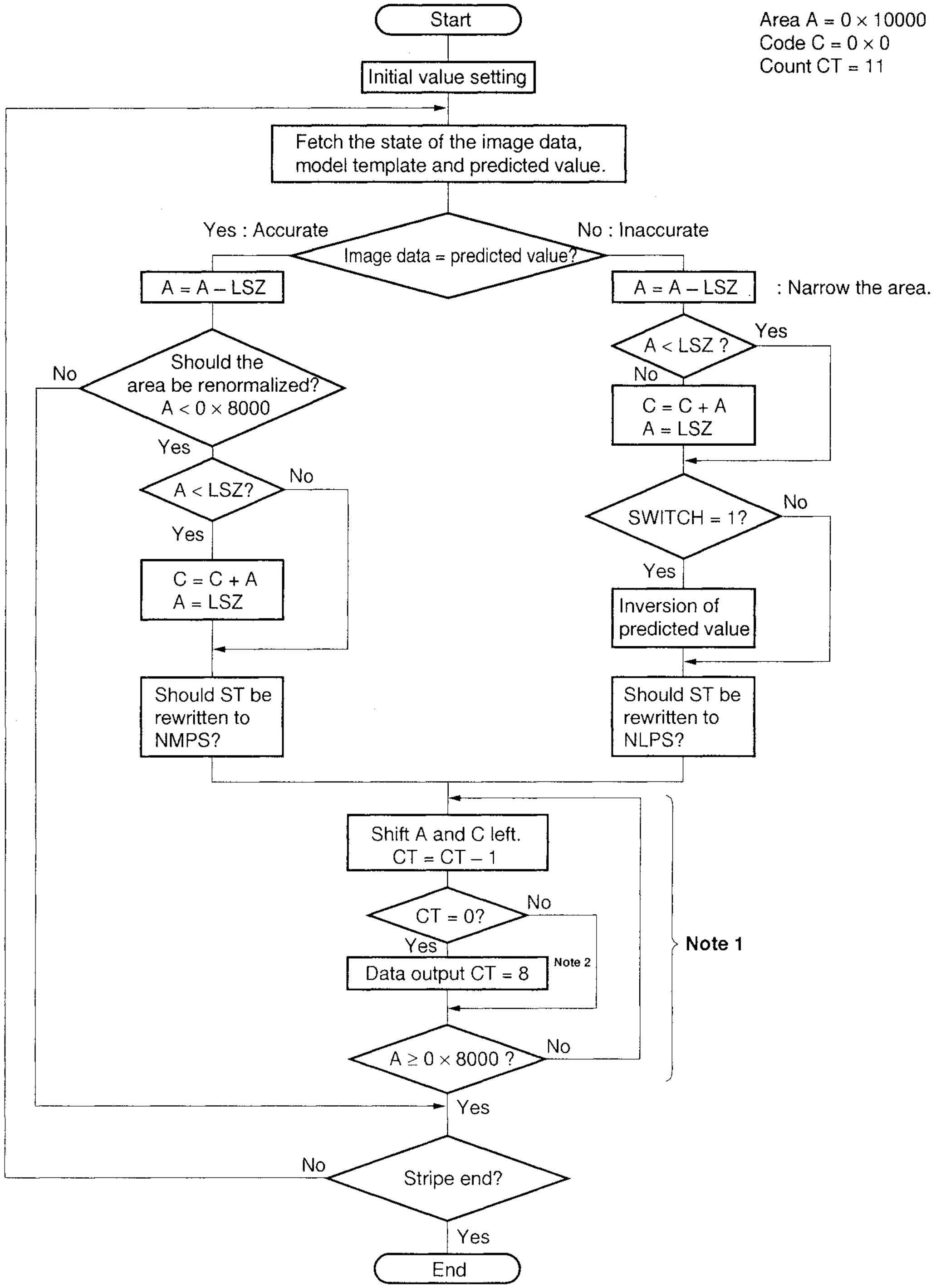
NLPS : State transition destination for LPS (when prediction proved inaccurate)

NMPS : State transition destination for MPS (when prediction proved accurate)

SWITCH : If SWITCH = 1 for LPS (when prediction proved inaccurate), the predicted value is inverted.

Figure 1-12 shows an outline flow of the arithmetic encoding. The values of area A which has been shown in decimals between 0 and 1 so far are now shown in hexadicimals between 0×8000 and 0×10000.

Figure 1-12. Outline Flow of Arithmetic Encoding



Notes 1. Renormalization
When area A becomes smaller than 0 × 8000, the values of A and C are shifted left (× 2) in order to make the value of A greater LSZ until A ≥ 0 × 8000 is reached.

2. Coded data output

The register configuration of area A and code C, and count CT are explained below.

Table 1-1. Register Configuration

	MSB LSB
Area A	00000000 0000000a aaaaaaaa aaaaaaaa
Code C	0000cbbb bbbbbsss xxxxxxxx xxxxxxxx

Remarks a : Area that indicates the probability of black/white permutation from 0×00000 to 0×10000.
b : Indicates the 8 bits of coded data.
x : Bit that receives data of area A
s : Transitory bit in process of shifting left by renormalization
c : Carry bit

Count CT is the counter to fetch JBIG coded data (8 bits) from code C. Values set in CT are 11 in the initial setting and 8 thereafter. Values are changed as shown in Table 1-1.

This is because the number of shifts to "b" where "x" is treated as coded data is 11, and the number of shifts from "s" to "b" after the coded data is output is 8.

1.2.10 Structure of JBIG coded data

The coded data (BIE) consists of a header (BIH) and data (BID).

BIE	BIH	DI			1 byte	Layer sent first
		D			1 byte	Number of difference layers
		P			1 byte	Number of bit planes
		–			1 byte	Reserved
		Xd			4 bytes	Number of pixels in horizontal direction in layer d
		Yd			4 bytes	Number of pixels in vertical direction in layer d
		L0			4 bytes	Number of stripe lines in layer d
		Mx			1 byte	Maximum horizontal offset allowed for AT pixel
		My			1 byte	Maximum vertical offset allowed for AT pixel
		Order	– (MSB)		1 bit	Reserved
			–		1 bit	Reserved
			–		1 bit	Reserved
			–		1 bit	Reserved
			HITOLO[Note 1]		1 bit	Transferred from higher position
			SEQ[Note 1]		1 bit	Sequential transfer
			ILEAVE[Note 1]		1 bit	Transfer order 1
			SMID[Note 1] (LSB)		1 bit	Transfer order 2
		Options	– (MSB)		1 bit	Reserved
			LRLTWO		1 bit	Template setting
			VLENGTH		1 bit	NEWLEN marker setting
			TPDON[Note 1]		1 bit	TP setting
			TPBON		1 bit	TP setting
			DPON[Note 1]		1 bit	DP setting
			DPPRIV[Note 1]		1 bit	DP setting
			DPLAST[Note 1] (LSB)		1 bit	DP setting
		DPTABLE[Note 1]			0/1728 bytes	DP setting
	BID	Floating marker segment[Note 3]	ATMOVE[Note 4]	ESC	1 byte	Escape (0×FF)
				ATMOVE	1 byte	AT move (0×06)
				yAT	4 bytes	AT move start line
				Tx	1 byte	Number of pixels per stripe involved in AT move in horizontal direction
				Ty[Note 1]	1 byte	Number of pixels per stripe involved in AT move in vertical direction
			NEWLEN[Note 5]	ESC	1 byte	Escape (0×FF)
				NEWLEN	1 byte	Number of new lines (0×05)
				Yd	4 bytes	Redefinition of number of page lines
			COMMENT[Note 6]	ESC	1 byte	Escape (0×FF)
				COMMENT	1 byte	Comment (0×07)
				Lc	4 bytes	Comment length
				Comment	Variable length	Comment with length specified by Lc
		SDE	PSCD		Variable length	Stripe data
			ESC		1 byte	Escape (0×FF)
			SDNORM/SDRST		1 byte	0×02 (stripe data end) / 0×03 (stripe data end and reset)
		Floating marker segment				
		SDE				
		⋮				
		Floating marker segment				
		SDE				

Notes 1. Not used in the JBIG FAX version.

2. When BID is terminated midway, add the following abort marker (2 bytes).
ESC (1 byte: 0×FF)
ABORT (1 byte: 0×04)

3. Gives information about move of the AT pixel, changes of the number of pixels in vertical direction (Yd) and comment addition.

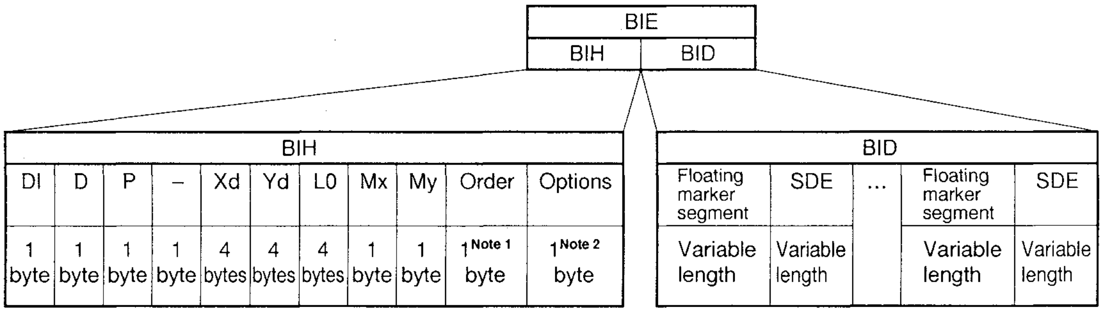
4. Gives information about move of the AT pixel.

5. Gives information about changes in the number of pixels in vertical direction (Yd).

6. Gives information about comments.

Caution SDNORM indicates the end of the stripe data. SDRST indicates the end of the stripe data and resets all the TP and DP functions in the same way as in the start of the image. Therefore, note that resetting by SDRST deteriorates compression efficiency.

Figure 1-13. Structure of JBIG Coded Data



BIE	
BIH	BID

BIH										
DI	D	P	–	Xd	Yd	L0	Mx	My	Order	Options
1 byte	1 byte	1 byte	1 byte	4 bytes	4 bytes	4 bytes	1 byte	1 byte	1[Note 1] byte	1[Note 2] byte

BID				
Floating marker segment	SDE	...	Floating marker segment	SDE
Variable length	Variable length		Variable length	Variable length

Notes 1. Order

Order							
–	–	–	–	HITOLO	SEQ	ILEAVE	SMID
1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit
MSB							LSB

2. Options

Options							
–	LRLTWO	VLENGTH	TPDON	TPBON	DPON	DPPRIV	DPLAST
1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit
MSB							LSB

1.3 Product Outline

1.3.1 Features

- Compression/expansion for specified lines
- Support of JBIG free parameters (compliant with ITU-T T.85 Recommendation)
- Can be called from NEC/GHS C compiler C language.
- Compatible with NEC/GHS real-time OS (re-entrant capability)
- JBIG coded data can be handled by both LSB_first and MSB_first.

Remark GHS: Green Hills Software, Inc.

1.3.2 Functions

(1) Compression system

Performs compression using a specified image data buffer and JBIG free parameters and outputs coded data to the compressed coded data buffer.

(2) Expansion system

Performs expansion from a specified receive buffer and JBIG free parameters and outputs expanded image data to the image data buffer.

1.3.3 Operating environment

(1) Applicable CPU

V810 family, V850 family ★

(2) Applicable linker

- NEC V810 family linker (Ver.2.00 or higher)
- NEC V850 family linker (Ver.1.00 or higher) ★
- GHS ELF version linker (Ver.1.8.7B or higher)

(3) Memory size required

(a) ROM (approximately 14K bytes)

- Program

	Compression system library	LSB_first version (approx. 6K bytes)
		MSB_first version (approx. 6K bytes)
★	Expansion system library	LSB_first version (approx. 8K bytes)
		MSB_first version (approx. 8K bytes)

- Probability prediction table (approx. 1K bytes)

(b) RAM

The memory size required varies depending on the image size, compressed coded data buffer size, etc.

- Image data buffer (the recommended memory size is that for 3 lines at least (number of lines corresponding to 1 stripe + reference line))
- Compressed coded data buffer
- MPS_St table (1K bytes)
- ★ I/O parameters + internal information area (for both compression/expansion systems: 232 bytes)

1.3.4 Directory configuration

The following are the package contents.

(1) V810 family

```
/ ─┬─ nectools ─┬─ <lib732> ──────────── libjbig.a     : Library                              ★
   │            │
   │            └─ <smpl732> ─ <jbig> ── enc_main.c    : Compression sample main source
   │                                     dec_main.c    : Expansion sample main source
   │                                     crt810.s      : Sample start-up routine
   │                                     jbig810.h     : Header file
   │                                     makefile      : Make file
   │                                     jbig_enc.lnk  : Link file
   │                                     jbig_dec.lnk  : Link file
   │
   └─ ghstools ─┬─ <lib810>  ─────────── libjbig.a     : Library                              ★
                │
                └─ <smpl810> ─ <jbig> ── enc_main.c    : Compression sample main source
                                         dec_main.c    : Expansion sample main source
                                         crt810.s      : Sample start-up routine
                                         jbig810.h     : Header file
                                         makefile      : Make file
```

(2) V850 family ★

```
/ ─┬─ nectools ─┬─ <lib850> ──────────── libjbig.a     : Library
   │            │
   │            └─ <smp850> ─ <jbig> ─── enc_main.c    : Compression sample main source
   │                                     dec_main.c    : Expansion sample main source
   │                                     crt850.s      : Sample start-up routine
   │                                     jbig850.h     : Header file
   │                                     makefile      : Make file
   │                                     jbig_enc.lnk  : Link file
   │                                     jbig_dec.lnk  : Link file
   │
   └─ ghstools ─┬─ <lib850>  ─────────── libjbig.a     : Library
                │
                └─ <smp850> ─ <jbig> ─── enc_main.c    : Compression sample main source
                                         dec_main.c    : Expansion sample main source
                                         crt850.s      : Sample start-up routine
                                         jbig850.h     : Header file
                                         makefile      : Make file
```

1.3.5 Target performance

★ **(1) V810 family**

Condition CPU : V810 (25 MHz), 32-bit bus, cache on
JBIG : TPBON=ON, LRLTWO=OFF, AT=default, Layer=Lowest
Data : ITU-T chart1 (1728 × 2376 dot)

Performance Compression time : Approximately 1.8 sec.
Expansion time : Approximately 1.8 sec.

★ **(2) V850 family (target value)**

Condition CPU : V850 family (33 MHz), 16-bit bus
JBIG : TPBON=ON, LRLTWO=OFF, AT=default, Layer=Lowest
Data : ITU-T chart1 (1728 × 2376 dot)

Performance Compression time : Approximately 1.0 sec.
Expansion time : Approximately 1.2 sec.

CHAPTER 2 LIBRARY SPECIFICATIONS

2.1 Processing Outline

2.1.1 Standard specifications

The following free parameters are defined in the standard recommendation (T.85) on the JBIG FAX version.

Table 2-1. Free Parameters (1/2)

Parameter	Size	T.85 Recommendation	Function
DI	1 byte	0 fixed	Layer sent first
D	1 byte	0 fixed	Number of difference layers
P	1 byte	1 fixed	Number of bit planes(1:2-value image)
–	1 byte	Don't care	Fill
Xd[Note]	4 bytes	1-0xFFFFFFFF	Number of pixels in horizontal direction in layer d
Yd[Note]	4 bytes	1-0xFFFFFFFF	Number of pixels in vertical direction in layer d
L0[Note]	4 bytes	1-Yd	Number of lines per stripe
Mx	1 byte	0-127	Maximum horizontal offset allowed for AT pixel
My	1 byte	0 fixed	Maximum vertical offset allowed for AT pixel

★

Note Parameter that can be set in this library.

Table 2-1. Free Parameters (2/2)

Parameter	Bit Name	Bit Location	T.85 Recommendation	Function
Order	–	7	–	–
	–	6	–	–
	–	5	–	–
	–	4	–	–
	HITOLO[Note 1]	3	0 fixed	From higher to lower
	SEQ[Note 1]	2	0 fixed	Sequential
	ILEAVE[Note 1]	1	0 fixed	Interleaved bit plane
	SMID[Note 1]	0	0 fixed	Central loop is stripe.
Options	–	7	–	–
	LRLTWO[Note 2]	6	0/1	Specification of template used 0 : Use of 3-line template 1 : Use of 2-line template
★	VLENGTH[Note 2]	5	0/1	Setting of NEWLEN marker 0 : Not set 1 : NEWLEN setting possible
	TPDON	4	0 fixed	Difference layer TP not used
	TPBON[Note 2]	3	0/1	Specification of minimum resolution layer TP 0 : Not used 1 : Used
	DPON	2	0 fixed	DP not used
	DPPRIV	1	0 fixed	DP table not used
	DPLAST	0	0 fixed	Last DP not used

Notes 1. Defines the stripe processing procedure.

2. Parameter that can be set in this library

2.1.2 Library processing

(1) MSB_first and LSB_first processing of coded data

Two libraries, MSB_first and LSB_first, are available for coded data.

	MSB_first Coded Data	LSB_first Coded Data
Compression	jbig_enc_m ( )	jbig_enc_l ( )
Expansion	jbig_dec_m ( )	jbig_dec_l ( )

(2) Processing unit and abort processing

Compression/expansion performs processing for specified lines. When processing for 1 stripe is completed, SDNORM/SDRST markers are added (for compression)/detected (for expansion) and a status indicating the completion of 1 stripe is used as the library return value. When processing for 1 page is completed, a status indicating the completion of 1 page is used as the library return value. ★

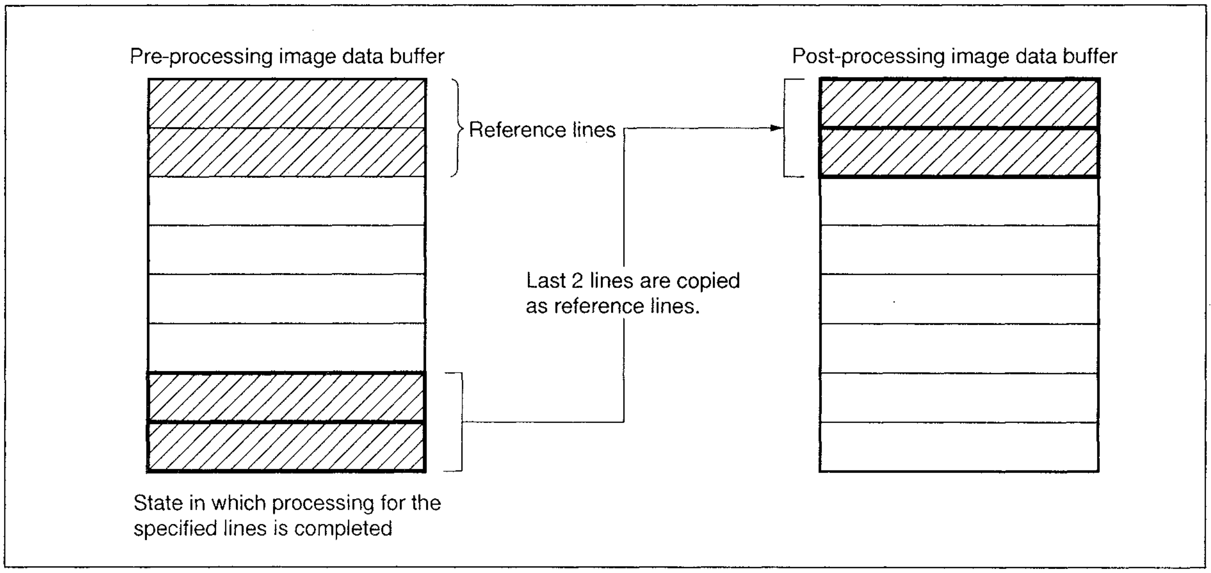
(3) BIH (JBIG header) setting

This library does not set BIN (JBIG header). It checks the VLENGTH bit and NEWLEN marker of BIH but does not check the other input/output parameters and BIH. For example, comparison between M_x of BIH (maximum horizontal offset allowed for AT pixel) and T_x of the ATMOVE marker (AT pixel offset in horizontal direction) is not checked. ★

(4) Reference line setting

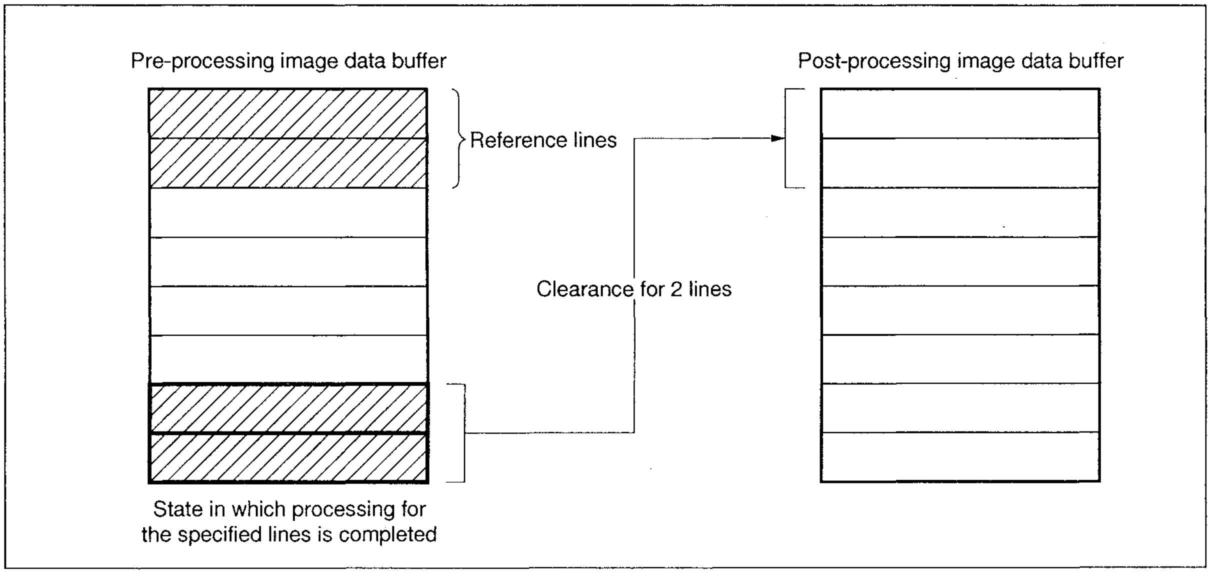
After compression/expansion is completed for specified lines, the last 2 lines (when a 3-line model template is used) or 1 line (when a 2-line model template is used) in the image data buffer are copied to a specified buffer as the reference line contents.

Figure 2-1. Copy as Reference Lines


Pre-processing image data buffer
Post-processing image data buffer
Reference lines
Last 2 lines are copied as reference lines.
State in which processing for the specified lines is completed

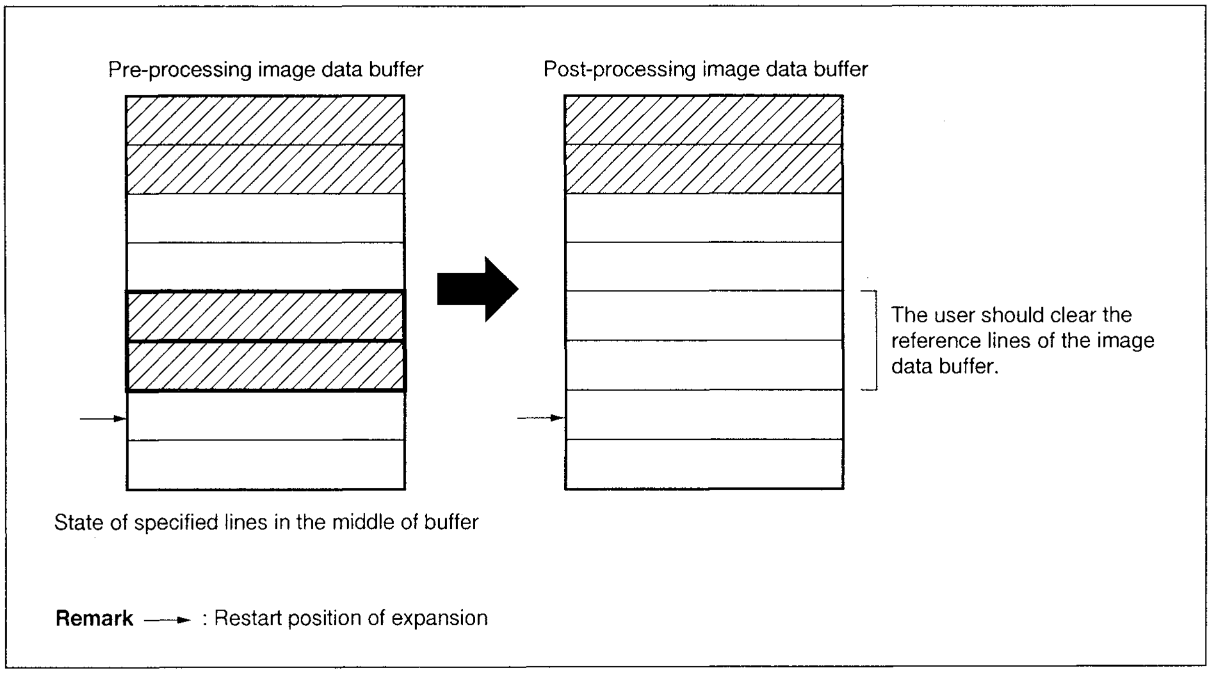
However, if SDRST is added at the stripe end, the specified buffer is cleared for the reference lines.

Figure 2-2. When Stripe End is SDRST


Pre-processing image data buffer
Post-processing image data buffer
Reference lines
Clearance for 2 lines
State in which processing for the specified lines is completed

Caution If SDRST is added to the middle of an expansion system image data buffer at the stripe end, the image data is not cleared, and therefore the user should clear the reference lines of the image data buffer.

Figure 2-3. When Processing is Restarted from the Middle of Image Data Buffer at Stripe End


Pre-processing image data buffer
Post-processing image data buffer
The user should clear the reference lines of the image data buffer.
State of specified lines in the middle of buffer
Remark ——► : Restart position of expansion

(5) Clearance of MPS and ST tables

If SDRST is set, it is processed in the library after the stripe end.

2.1.3 Handling image data and coded data

(1) Read/write system for image data and coded data

Image data is stored sequentially starting from the Least Significant Bit (LSB) of the byte scanned first with the scanner.

Coded data is read/written in two systems, LSB_first and MSB_first.

Data Type	Read/Write System
Image data	LSB_first
Coded data	LSB_first and MSB_first

(2) Marker handling

The following markers are automatically added (for compression) or automatically detected (for expansion).

SDNORM/SDRST
ATMOVE
NEWLEN
ABORT
COMMENT (for expansion only)
RESERVE (for expansion only)

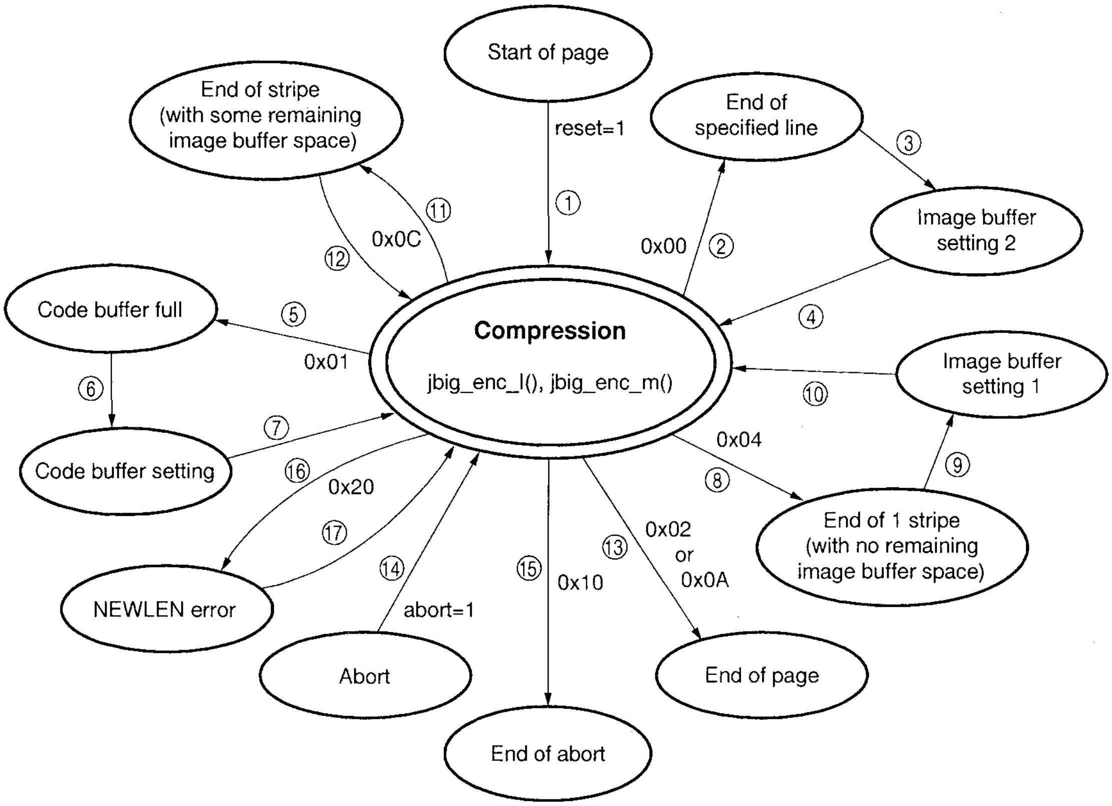
(3) Stuff byte handling

Stuff bytes are automatically added (for compression) or automatically discarded (for expansion).

2.1.4 Status transition ★

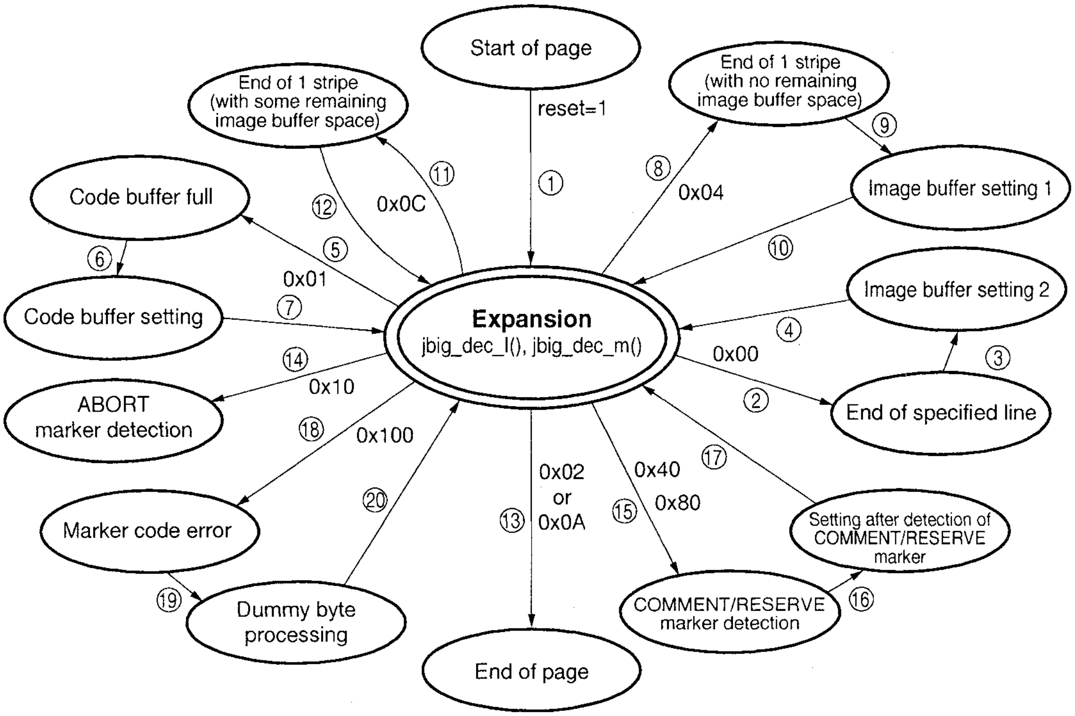
The followings are the block diagram of status transition in each processing.

Figure 2-4. Status Transition of Compression



Transition No.	Description	Return Value
①	Start of compression (reset=1)	–
②	End of specified line	0x00
③	Image buffer setting	–
④	Resumption of compression	–
⑤	Code buffer full	0x01
⑥	Code buffer setting	–
⑦	Resumption of compression	–
⑧	End of 1 stripe (with no remaining image buffer space)	0x04
⑨	Image buffer setting	–
⑩	Resumption of compression	–
⑪	End of 1 stripe (with some remaining image buffer space)	0x0C
⑫	Resumption of compression	–
⑬	End of page	0x02, 0x0A
⑭	Abort (abort=1)	–
⑮	End of abort	0x10
⑯	NEWLEN error	0x20
⑰	Resumption of compression after NEWLEN error	–
Others	Transition impossible	–

Figure 2-5. Status Transition of Expansion



Transition No.	Description	Return Value
①	Start of expansion (reset=1)	–
②	End of specified line	0x00
③	Image buffer setting	–
④	Resumption of expansion	–
⑤	Code buffer full	0x01
⑥	Code buffer setting	–
⑦	Resumption of expansion	–
⑧	End of 1 stripe (with no remaining image buffer space)	0x04
⑨	Image buffer setting	–
⑩	Resumption of expansion	–
⑪	End of 1 stripe (with some remaining image buffer space)	0x0C
⑫	Resumption of expansion	–
⑬	End of page	0x02, 0x0A
⑭	ABORT marker detection	0x10
⑮	COMMENT/RESERVE marker detection	0x40, 0x80
⑯	COMMENT/RESERVE marker processing	–
⑰	Resumption of expansion	–
⑱	Error marker code detection	0x100
⑲	Dummy byte processing (only when invalid_code_sts = 0x01)	–
⑳	Resumption of expansion	–
Others	Transition impossible	–

2.2 Function Specifications

The following is the specification used to call each library (written in C language).

2.2.1 Structure (parameter)

The parameters (J_PARA) used for all JBIG compression/expansion functions are shown below.

Table 2-2. Parameters (J_PARA)

Member Name	Type	Description
*pixel_buf	unsigned char	Image data buffer address
*next_pixel_buf	unsigned char	Next image data buffer address
pixel_buf_line	unsigned int	Number of image data lines
*code_buf	unsigned char	Compressed coded data buffer address
code_buf_size	unsigned int	Compressed coded data buffer remaining size
*Mps_St_tbl	unsigned char	Mps_St table start address
Xd	unsigned int	Number of pixels in horizontal direction
Yd	unsigned int	Number of pixels in vertical direction
line_cnt	unsigned int	Line counter
L0	unsigned int	Number of lines per stripe
Options	unsigned char	JBIG BIH Options Byte
reset	unsigned char	1: Reset execution
abort	unsigned char	1: ABORT
sdrst	unsigned char	After 1-stripe processing ends, 1: SDRST/0: SDNORM
★ newlen_err_sts	unsigned char	NEWLEN error status
★ invalid_code_sts	unsigned char	Error marker code status
★ dummy	unsigned char	Dummy byte
Tx	unsigned char	Number of pixels per stripe involved in AT move in horizontal direction
yAT	unsigned int	AT move start line, 0: stripe start
newlen	unsigned int	Redefinition of number of page lines, number of new lines
★ length_er	unsigned int	length value when NEWLEN marker has an error
Lc	unsigned int	Comment length
rstart_adr	unsigned int	Initialization/restart flag
reg_area [10]	unsigned int	Register save area
jbg_val [31]	unsigned int	JBIG variable save area (only for abort)

(1) *pixel_buf

Indicates image data buffer address.
pixel_buf specifies the address of a line to be compressed. Do not set the address of a reference line.
After specified lines have been processed, pixel_buf is assigned the address specified by next_pixel_buf.
Input a value aligned by a word for this member.

(2) *next_pixel_buf

Indicates the next image data buffer address.
next_pixel_buf specifies the current address of the image data buffer following the one that processed the specified lines.
After processing the specified lines, the last 2 lines (when 3-line model template is used) or 1 line (when 2-line model template is used) of the image data buffer are copied before next_pixel_buf. pixel_buf indicates the current line at this time.
Input a value aligned by a word for this member.

(3) pixel_buf_line

Indicates the number of image data lines set in the image data buffer.
Be sure to specify a value of one line at least. The number of lines necessary for the image buffer depends on the value of LRLTWO of Options.

LRLTW0=0: Requires 3 lines or more including 2 lines (reference lines) before the current line.
LRLTW0=1: Requires 2 lines or more including a line (reference lines) before the current line.

When the specified lines are processed, the set value is output.
The number of the remaining lines is output for the value of pixel_buf_line only in the following cases.

- When a stripe ends or page ends without completing the processing of the specified lines
- When aborted by compressed coded data buffer full

(4) *code_buf

Indicates the compressed coded data buffer address.
code_buf is incremented by +1 when one byte of the coded data is written (read) by the library.

(5) code_buf_size

Indicates the compressed coded data buffer remaining size.
code_buf_size is decremented by –1 when one byte of the coded data is written (read) by the library. If the compressed coded data buffer remaining size is 0 bytes (return value is 0×01), set it again.

(6) *Mps_St_tbl

Indicates the Mps_St table start address.
The Mps_St table requires 1K bytes.

(7) Xd

Indicates the number of pixels in horizontal direction.
It cannot be changed during page processing. The same keyword as that for the JBIG free parameters is used.

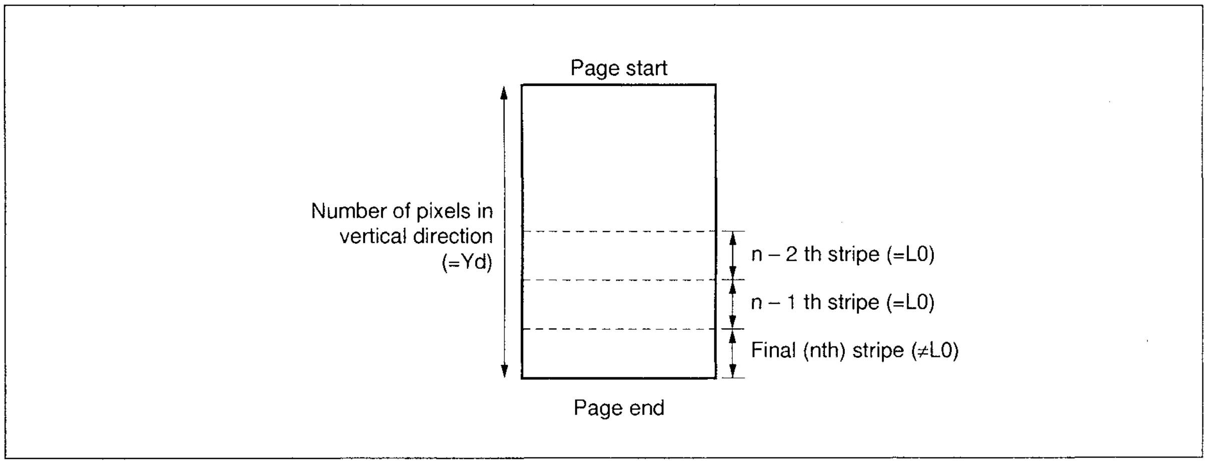
During compression : Specify a multiple of 8.
During expansion : No restrictions

(8) Yd

Indicates the number of pixels in vertical direction.

It cannot be changed during page processing. Its relation with the number of lines per stripe (L0) is shown below. Yd does not necessarily match multiples of L0. Even if the final stripe does not match L0, it is processed as one stripe. The same keyword as that for the JBIG free parameters is used.

Figure 2-6. Relation with Number of Lines per Stripe (L0)



(9) line_cnt

The line counter indicates the accumulated number of lines processed by the library.

Refer to it when the number of lines per page or stripe is unknown. reset=1 clears it.

(10) L0

Indicates the number of lines per stripe.

L0 cannot be changed during page processing. Even if the number of lines in the final stripe does not match L0, it is processed as one stripe (refer to **(8) Yd**). The same keyword as that for the JBIG free parameters is used.

(11) Options

This is Options Byte of JBIG BIH.

Options cannot be changed during page processing. The same keyword as that for the JBIG free parameters is used. Options are explained below.

Table 2-3. Options

Bit Location	Bit Name	Description
7	–	0 fixed
6	LRLTWO	Specification of model template 0: Use of 3-line model template 1: Use of 2-line model template
5	VLENGTH	Setting NEWLEN marker 0: Not set 1: NEWLEN setting possible
4	–	0 fixed
3	TPBON	Specification of minimum resolution layer TP 0: Not used 1: Used
2-0	–	0 fixed

★

Caution **If VLENGTH=0 and the NEWLEN marker is appended (during compression) or detected (during expansion) as a result of checking the relation between the VLENGTH and NEWLEN markers that are set by bit 5 of Options in 2.1.1 Standard specifications, the NEWLEN error status (0×01) is returned. During compression, the return value (0×20) of a function is returned to indicate a NEWLEN marker error. During expansion, however, processing is continued with the NEWLEN marker ignored.** ★

(12) reset

Initializes this library.

If this library is executed by a reset (reset=1), the Mps_St table, registers in the QM coder, reference lines (1 line/2 lines depending on Options), restart_adr, line_cnt, newlen_err_sts, and length_err are cleared. ★ During expansion, newlen, Tx, yAT, sdrst, Lc, and invalid_code_sts are also cleared. After a reset, it becomes reset=0. Be sure to reset at the page start.

(13) abort

Aborts processing.

- During compression

  Executing this library with abort=1 causes an abort (forced termination). An abort (forced termination) only accepts the start of an image data buffer, performs a flush (discharge of compressed code), outputs an ABORT marker to the compressed coded data buffer, and stops compression. After an abort (forced termination), abort=0 is set.

- During expansion

  An abort is not used for expansion of this library.
  The ABORT processing for expansion is shown below.

  - During detection of an ABORT marker code, it is returned as a function return value (0×10).
  - During detection of an ABORT marker code, expansion is aborted.

(14) sdrst

Sets (for compression) or indicates (for expansion) the state after 1-stripe processing ends. sdrst=1 is SDRST and sdrst=0 is SDNORM.

- During compression

  After 1-stripe processing ends, the following marker codes are output to the compressed coded data buffer.

```
FFH, 03H              FFH, 02H
 ↑    ↑                ↑    ↑
ESC  SDRST            ESC  SDNORM
```

  These are referenced after 1-stripe processing ends and if they are 1 (SDRST), the contents of the JBIG variable and Mps_St table are all cleared.

- During expansion

  ★ When an SDRST or SDNORM marker code is automatically detected, value 1 or 0 is set in sdrst. These statuses are cleared by reset (reset=1).
  If sdrst is 1 (SDRST) upon completion of stripe processing, the contents of the JBIG variable and Mps_St table are all cleared. If sdrst is 1 (SDRST) upon completion of stripe processing, the reference lines of the image data buffer are also cleared. Note that clearance of the reference lines depends on the state of the image data buffer. Refer to **2.1.2 (4) Reference line setting** for details.

(15) newlen_err_sts ★

If a NEWLEN marker error occurs, the following NEWLEN error status is indicated. If the NEWLEN marker is normal, 0 is returned. These error statuses are cleared by reset (reset=1).

0×01 : If newlen $\neq$ 0 during compression where Options: VLENGTH=0
If NEWLEN marker is detected during expansion where Options: VLENGTH=0
0×02 : If value specified by newlen during compression is less than the total number of line counts, or more than the number of pixels in the vertical direction

In addition to the above statuses, length_err is also returned if the NEWLEN marker error occurs.
The return value of a function (0×02) is returned during compression. During expansion, however, only the status is set and the processing is continued. If newlen_err_sts is returned during both compression and expansion, the length value is returned to length_err.

(16) invalid_code_sts ★

This member is meaningful only during expansion.
If 0×100 (marker code error) is returned as the return value of a function, the following error status is returned. These statuses are cleared by reset (reset=1).

0×01 : No END mark is found and a dummy byte (0x00) is detected at the end of a stripe.
0×02 : No END mark is found and a byte other than dummy byte is detected at the end of a stripe.
0×03 : A byte other than ATMOVE, NEWLEN, COMMENT, RESERVE, SDRST/SDNORM, ABORT, and staff byte is detected.

If 0×100 is returned as the return value of a function, the compressed code data buffer address and remaining space of the compressed code data buffer retain the values immediately before detection of the error code.
If invalid_code_sts is 0×01, the dummy byte can be processed if the user increments the compressed code data buffer address and decrements the size of the remaining space of the compressed code data buffer.
invalid_code_sts is cleared if an ESC code is detected next time.
If two or more dummy bytes exist, they can be processed as follows:

- Continue calling expansion until invalid_code_sts reaches 0.
- Increment the compressed code data address to the address of the ESC code, decrement the remaining space of the compressed code data buffer by the number of incremented addresses, and then call the expansion.

(17) dummy ★

This is a dummy byte for member word alignment.

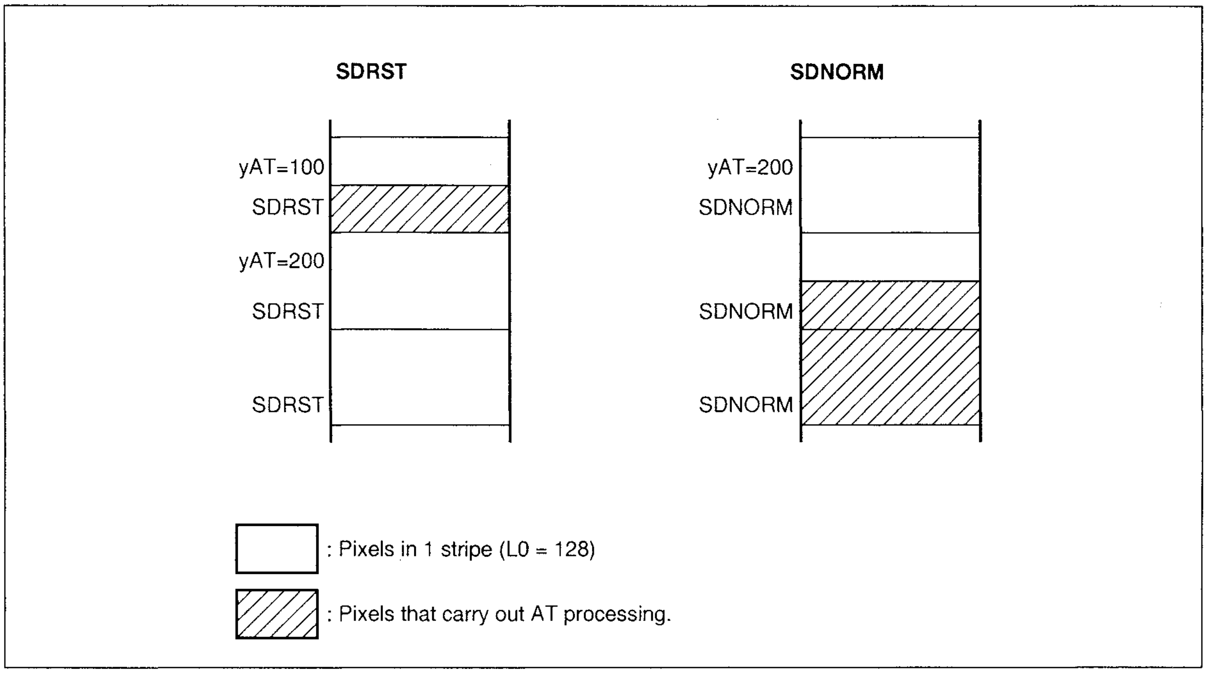
(18) Tx, yAT

Tx indicates the number of pixels per stripe involved in AT move in horizontal direction.
yAT indicates the start line of AT move. yAT=0 indicates the start of a stripe. These statuses are cleared ★ by reset (reset=1) during expansion only.
The relation between yAT and L0 (number of lines per stripe) is shown below.

When yAT > L0 is set

- For SDNORM, it accumulates the number of lines and starts AT processing from the next stripe.
- For SDRST, it does not start AT processing because it recounts the number of lines.

Figure 2-7. SDRST and SDNORM


SDRST
SDNORM
yAT=100
SDRST
yAT=200
SDRST
SDRST
yAT=200
SDNORM
SDNORM
SDNORM
: Pixels in 1 stripe (L0 = 128)
: Pixels that carry out AT processing.

- During compression

  AT can be moved once in a stripe up to 127 pixels in horizontal direction. It outputs the following marker codes to the compressed coded data buffer at the start of the stripe.

```
FFH,    06H,    XXH, XXH, XXH, XXH,    XXH,    00H
 ↑       ↑               ↑              ↑       ↑
ESC   ATMOVE            yAT             Tx    fixed
```

  If AT is the default (Tx=0), the marker code indicating Tx=0 is not output to the compressed coded data buffer. However, in SDNORM, the marker code indicating Tx=0 is output when AT processing returns from Tx≠0 to Tx=0.

- During expansion

  If an AT marker code is automatically detected, the AT move start line is set to yAT and the number of pixels involved in AT move in horizontal direction is set to Tx to perform expansion.

(19) newlen

Redefines the number of page lines.

The value of newlen is changed to a value specified by Yd and is processed as the number of new page lines.

- During compression ★

  The NEWLEN marker is output to the compressed code data buffer with the value of newlen as the number of lines of a page (length). After the NEWLEN marker has been output, 0 is returned as the value of newlen. Set a value to newlen at the start of a stripe or image buffer. If a value is set at the start of the code buffer, the operation is not guaranteed.

newlen = 0 : The NEWLEN marker is not output to the compressed code data buffer. At this time, clear newlen at the page start. It is not cleared by reset (reset=1).

newlen ≠ 0 : The NEWLEN marker is output if VLENGTH=1 and the value of newlen is within the permissible range (Yd ≥ length value ≥ total number of line counts).

If newlen is set at the start of the image buffer other than at the start of the last stripe, the END marker is output after the NEWLEN marker.

Figure 2-8. Output Data When newlen Is Set

(a) At start of stripe

ESC	NEWLEN	length	PSCD	ESC	SDNORM

(b) At start of image buffer other than at start of last stripe

PSCD	ESC	SDNORM	ESC	NEWLEN	length	ESC	SDNORM

The NEWLEN error occurs in the following cases. At this time, the NEWLEN marker error is assumed, the processing is aborted, the NEWLEN error status (newlen_err_sts) and length_err are set, and the return value of a function (0×20) is returned.

. If newlen ≠ 0 with Options: VLENGTH = 0. 0×01 is set to newlen_err_sts.

. If the length value is not within the permissible range (number of pixels in vertical direction < length value, or total number of line counts > length value), 0×02 is set to newlen_err_sts.

If the processing is aborted by the NEWLEN error, 0 is output as the value of newlen. The NEWLEN error status (newlen_err_sts) and the length value (length_err) in case of NEWLEN marker error are cleared by reset (reset=1).

★

- During expansion
  If the NEWLEN marker code is automatically detected, it is set to newlen as the number of lines of the new page. This code is cleared by reset (reset=1).
  If the NEWLEN marker is detected at the start of a stripe, it is set to newlen as the number of lines of a new page, and then expansion of the stripe is performed. If the END marker (SDNORM/SDRST) is detected in the middle of a stripe, expansion is performed as follows:

(1) Virtual code data: Expansion of one line is performed with 0×00 as the virtual code data, and the processing is temporarily stopped.
(2) Whether the next code of the END marker is a NEWLEN marker or not is checked.
(3) If it is not a NEWLEN marker, expansion is restarted with 0×00 as the virtual code data. In this case, the end condition is L0.
If the next code of END marker is a NEWLEN marker, VLENGTH is checked.
(4) If VLENGTH=0, 0×01 is set to the NEWLEN error status (newlen_err_sts) and length value is set to length_err, and the NEWLEN marker and length value are ignored. If a normal NEWLEN marker has already been detected, the number of lines of a new page at that time is assumed to be the end of a page and expansion is restarted. If the normal NEWLEN marker has not been detected, initial value Yd is assumed to be the end of a page and expansion is restarted.
(5) If the detected length value is within the permissible range (Yd ≥ length value ≥ total number of line counts), the number of lines of a new page is set to newlen, and expansion is restarted with this number of lines assumed as a new page end condition.
If the detected length value is not within the permissible range, 0×02 is set to the NEWLEN error status (newlen_err_sts), and the length value is set to length_err, and the NEWLEN marker and the length value are ignored. If a normal NEWLEN marker has been already detected, the number of lines of a new page at that time is assumed to be the end of the page and expansion is restarted. If the normal NEWLEN marker has not been detected, initial value Yd is assumed to be the end of the page and expansion is restarted.

The NEWLEN error occurs in the following cases. At this time, the NEWLEN error status (newlen_err_sts) and length_err are set, and expansion is continued.

- If the NEWLEN marker is detected with Options: VLENGTH=0. The NEWLEN marker is ignored, and 0×01 is set to newlen_err_sts.
- If the length value is not within the permissible range (number of pixels in vertical direction < length value, or total number of line counts > length value). 0×02 is set to newlen_err_sts.

The value of newlen does not change in case of a NEWLEN error. The NEWLEN error status (newlen_err_sts) and length value (length_err) in case of a NEWLEN marker error are cleared by reset (reset=1).

(20) length_err ★

The length value in case of the NEWLEN marker is abnormal is indicated. If the NEWLEN marker is normal, 0 is returned. The length value is cleared by reset (reset=1).
The return value (0×20) of a function is returned during compression. Only the length value is set during expansion, and the processing is continued. If a NEWLEN marker error occurs, newlen_err_sts is also returned with this member.

(21) Lc

Indicates the comment length of a comment marker.

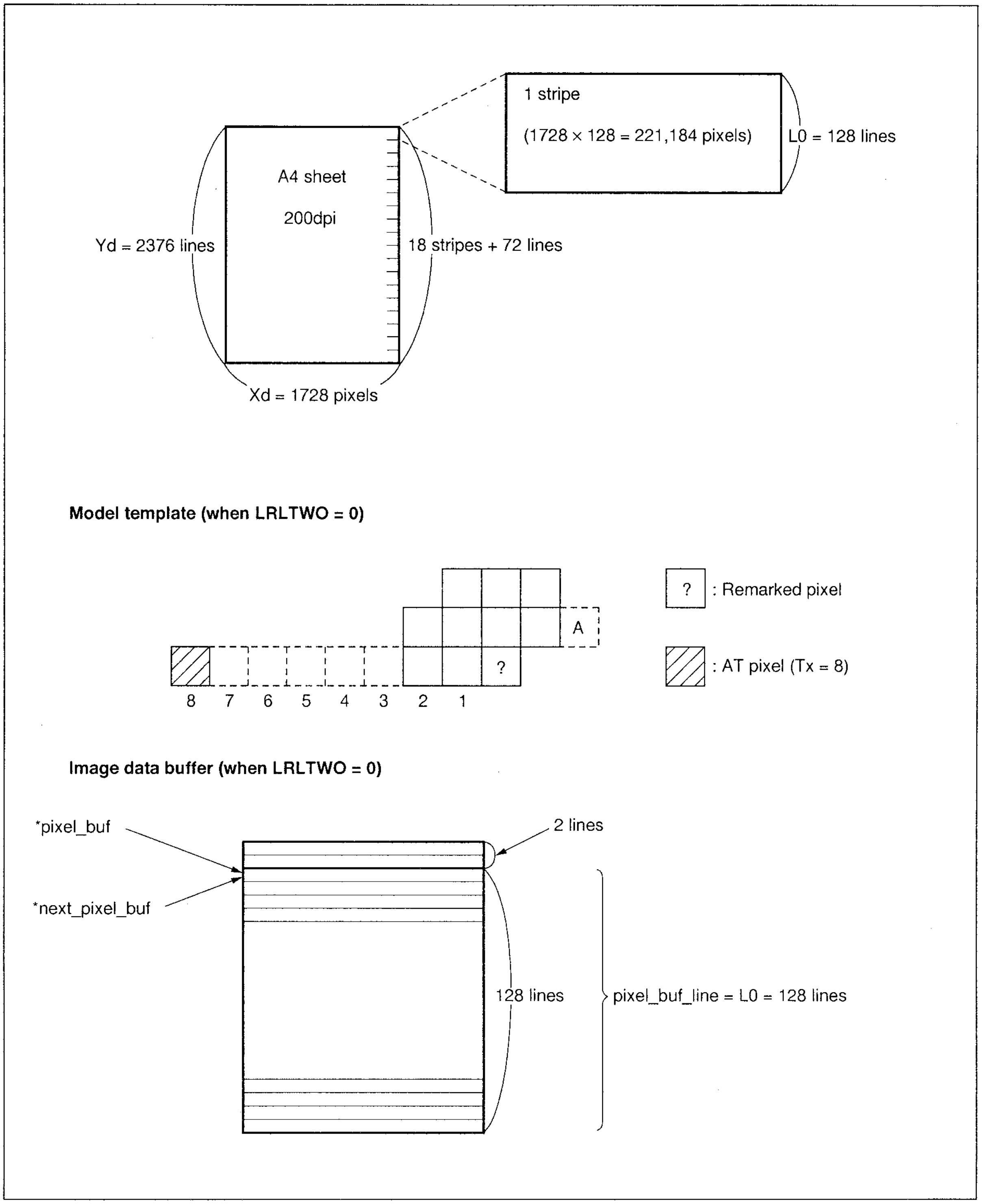
- During compression
  COMMENT marker output is not supported.

- During expansion
  If a COMMENT marker code is automatically detected, the comment length is set in Lc and the start address of a comment is set in code_buf. A function return value (0×40) is returned. This status is cleared by reset (reset=1). ★

(22) restart_adr

Indicates a flag necessary to re-execute this library without terminating processing for the specified lines. restart_adr is cleared by reset (reset=1) at the page start. ★

Figure 2-9. Parameter Examples



Model template (when LRLTWO = 0)

Image data buffer (when LRLTWO = 0)

2.2.2 External interface

(1) Compression system

(a) JBIG compression

- **Classification** : Compression system
- **Function name** : jbig_enc_m( ), jbig_enc_l( )
- **Function outline** : Performs compression from the specified image data buffer and JBIG free parameter, and outputs coded data to the compressed coded data buffer.
- **Format** : #include “jbig810.h” (V810 family)
  #include “jbig850.h” (V850 family) ★
  int jbig_enc_m(struct J_PARA *encdata)
- **Argument** : J_PARA

Table 2-4. Compression System

Member Name	Type	I/O	Description
*pixel_buf	unsigned char	io	Image data buffer address
*next_pixel_buf	unsigned char	i	Next image data buffer address
pixel_buf_line	unsigned int	io	Number of image data lines
*code_buf	unsigned char	io	Compressed coded data buffer address
code_buf_size	unsigned int	io	Compressed coded data buffer remaining size
*Mps_St_tbl	unsigned char	i	Mps_St table start address
Xd	unsigned int	i	Number of pixels in horizontal direction
Yd	unsigned int	i	Number of pixels in vertical direction
line_cnt	unsigned int	io	Line counter
L0	unsigned int	i	Number of lines per stripe
Options	unsigned char	i	JBIG BIH Options Byte
reset	unsigned char	io	1: Reset execution
abort	unsigned char	io	1: ABORT
sdrst	unsigned char	i	After 1-stripe processing ends, 1: SDRST/0: SDNORM
★ newlen_err_sts	unsigned char	o	NEWLEN error status
★ invalid_code_sts	unsigned char	–	Error marker code status
★ dummy	unsigned char	–	Dummy byte
Tx	unsigned char	i	Number of pixels per stripe involved in AT move in horizontal direction
yAT	unsigned int	i	AT move start line, 0: stripe start
newlen	unsigned int	io	Redefinition of number of page lines, number of new lines
★ length_err	unsigned int	o	length value when NEWLEN marker has an error
Lc	unsigned int	–	Comment length
restart_adr	unsigned int	io	Initialization/restart flag
reg_area [10]	unsigned int	io	Register save area
jbg_val [31]	unsigned int	io	JBIG variable save area (only for abort)

Remark i: Input, o: Output, io: Input/output, –: Unused

- **Return value**

Table 2-5 gives an explanation of the return value and Table 2-6 gives an explanation of each bit of the return value.

Table 2-5. Return Value (Compression System)

Return Value	Description
0x20	NEWLEN marker error[Note 1] ★
0x10	Abort (forced termination)[Note 2]
0x0C	1-stripe end (with some remaining image data buffer space)
0x0A	Page end (with some remaining image data buffer space)
0x04	1-stripe end (with no remaining image data buffer space)
0x02	Page end (with no remaining image data buffer space)
0x01	Abort by compressed coded data buffer full
0x00	Normal termination of compression for specified lines

Notes 1. Details of the error are stored in newlen_err_sts. At the same time, the setting value of newlen=0 and length_err=newlen are output. ★

2. Compression cannot be restarted. To restart compression, reset the return value and restart at the top of the page. ★

Table 2-6. Each Bit of Return Value (Compression System)

Bit Location	Description
31-6	Reserved: 0 fixed
5	NEWLEN marker error 0: Normal 1: Abnormal ★
4	Abort (forced termination) 0: No abort (forced termination) 1: Abort (forced termination)
3[Note 1]	Indicates state of image data buffer. 0: With no remaining buffer space 1: With remaining buffer space
2	1-stripe end 0: Not ended 1: End
1	Page end 0: Not ended 1: Ended
0	Indicates termination state. 0: Normal encoding termination[Note 2] 1: Abort[Note 3]

Notes 1. Whether or not there is some remaining space in the image data buffer matters only when 1 stripe ends or 1 page ends. ★

2. Specified image data line end, stripe end or page end

3. Abort due to compressed coded data buffer full

- **Function**

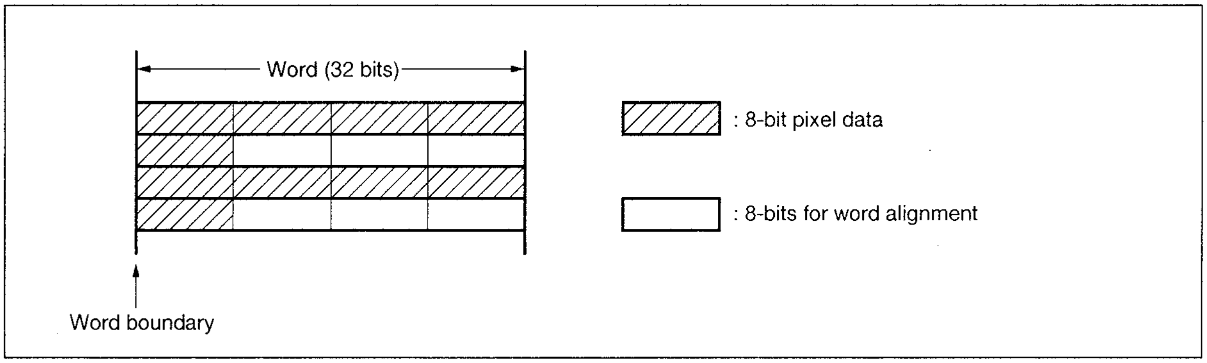
  Performs compression from the specified image data buffer and JBIG free parameter and outputs coded data to the compressed coded data buffer. Compresses data for the number of lines specified by the number of image buffer lines.

  Aborts operation when the remaining size of the compressed coded data buffer is 0. When 1-stripe processing ends, outputs an SDNORM/SDRST marker to the compressed coded data buffer and uses the status indicating 1-stripe end as a return value. For page end, also uses the status indicating page end as a return value.

(b) Image data

- When a page comes first, set all the reference lines to white.
  When LRLTWO=0: Set all to white with the previous 2 lines as reference lines.
  When LRLTWO=1: Set all to white with the previous 1 line as a reference line.
  When setting all reference lines to white, set "1" in reset.
- Set the number of pixels in horizontal direction to a multiple of 8.
- Place the image data so that the line start becomes the word boundary.

Figure 2-10. Line with 40 Pixels (Compression System)



- Place image data per line so that it is continuous on the word boundary.
- The number of image data lines changes by an abort due to stripe end, page end or compressed coded data buffer full.
- The recommended number of image buffer lines is the number of lines per stripe (L0) multiplied by a constant or a fraction. For example, suppose that the number of lines per stripe is 9, the recommended number of image buffer lines is its multiplication by a constant, i.e., 9, 18, 27, and so on, or its multiplication by a fraction, i.e., 1 and 3, and so on.

(2) Expansion system

(a) JBIG expansion

- **Classification** : Expansion system
- **Function name** : jbig_dec_m( ), jbig_dec_l( )
- **Function outline** : Performs expansion from the specified receive data buffer and JBIG free parameter, and outputs expanded image data to the image data buffer.
- **Format** : #include “jbig810.h” (V810 family)
  #include “jbig850.h” (V850 family) ★
  int jbig_dec_m(struct J_PARA *decdata)
- **Argument** : J_PARA

Table 2-7. Expansion System

Member Name	Type	I/O	Description
*pixel_buf	unsigned char	io	Image data buffer address
*next_pixel_buf	unsigned char	i	Next image data buffer address
pixel_buf_line	unsigned int	io	Number of image data lines
*code_buf	unsigned char	io	Compressed coded data buffer address
code_buf_size	unsigned int	io	Compressed coded data buffer remaining size
*Mps_St_tbl	unsigned char	i	Mps_St table start address
Xd	unsigned int	i	Number of pixels in horizontal direction
Yd	unsigned int	i	Number of pixels in vertical direction
line_cnt	unsigned int	io	Line counter
L0	unsigned int	i	Number of lines per stripe
Options	unsigned char	i	JBIG BIH Options Byte
reset	unsigned char	io	1: Reset execution
abort	unsigned char	–	Unused
sdrst	unsigned char	o	After 1-stripe processing ends, 1: SDRST/0: SDNORM
newlen_err_sts	unsigned char	o	NEWLEN error code status ★
invalid_code_sts	unsigned char	o	Error marker code status ★
dummy	unsigned char	–	Dummy byte ★
Tx	unsigned char	o	Number of pixels per stripe involved in AT move in horizontal direction
yAT	unsigned int	o	AT move start line, 0: stripe start
newlen	unsigned int	o	Redefinition of number of page lines, number of new lines
length_err	unsigned int	o	length value when NEWLEN marker has an error ★
Lc	unsigned int	o	Comment length
restart_adr	unsigned int	io	Initialization/restart flag
reg_area [10]	unsigned int	io	Register save area
jbg_val [31]	unsigned int	io	JBIG variable save area (only for abort)

Remark i: Input, o: Output, io: Input/output, –: Unused

- **Return value**

Table 2-8 gives an explanation of the return value and Table 2-9 gives an explanation of each bit of the return value.

★ **Table 2-8. Return Value (Expansion System)**

Return Value	Description
0x100	Marker code abnormality[Note 1]
0x80	Reserved marker detection
0x40	Comment marker detection
0x10	Abort (forced termination)[Note 2]
0x0C	1-stripe end (with some remaining image data buffer space)
0x0A	Page end (with some remaining image data buffer space)
0x04	1 stripe end (with no remaining image data buffer space)
0x02	Page end (with no remaining image data buffer space)
0x01	Abort by compressed coded data buffer empty
0x00	Normal termination of expansion for specified lines

Notes 1. Details of the error is stored in invalid_err_sts.
When invalid_err_sts=1 (dummy byte detected), expansion can be restarted after processing the dummy byte. In other case, expansion cannot be restarted. To restart expansion, reset the return value and restart at the top of the page.

2. Expansion cannot be restarted. To restart expansion, reset the return value and restart at the top of the page.

Table 2-9. Each Bit of Return Value (Expansion System) (1/2)

Bit Location	Description
31-9	Reserved: 0 fixed
8	Marker code abnormality 0: Normal 1: Abnormal[Note]
7	Reserved marker detection 0: Undetected 1: Detected
6	Comment marker detection 0: Undetected 1: Detected
★ 5	Not used

Note Other than ATMOVE, NEWLEN, COMMENT, RESERVE, SDRST/SDNORM, ABORT, stuff byte or unless the data following the one when a stripe ends is an END marker (SDRST/ ★ SDNORM). At this time, the compressed code data buffer address and the remaining space of the compressed code data buffer are not updated, and their values immediately before detection are retained. For details, refer to **2.2.1 (16) invalid_code_sts**.

Table 2-9. Each Bit of Return Value (Expansion System) (2/2)

Bit Location	Description
4	Abort (forced termination) 0: No abort (forced termination) 1: Abort (forced termination)
3[Note 1]	Indicates state of image data buffer 0: No remaining buffer space 1: With some remaining buffer space
2	1-stripe end 0: Not ended 1: Ended
1	1-page end 0: Not ended 1: Ended
0	Indicates termination state. 0: Normal termination of expansion[Note 2] 1: Abort[Note 3]

Notes 1. Whether or not there is some remaining space in the image data buffer matters only when 1 stripe ends or 1 page ends.

2. Specified image data line end, stripe end or page end

3. Abort by compressed coded data buffer empty

- **Function**

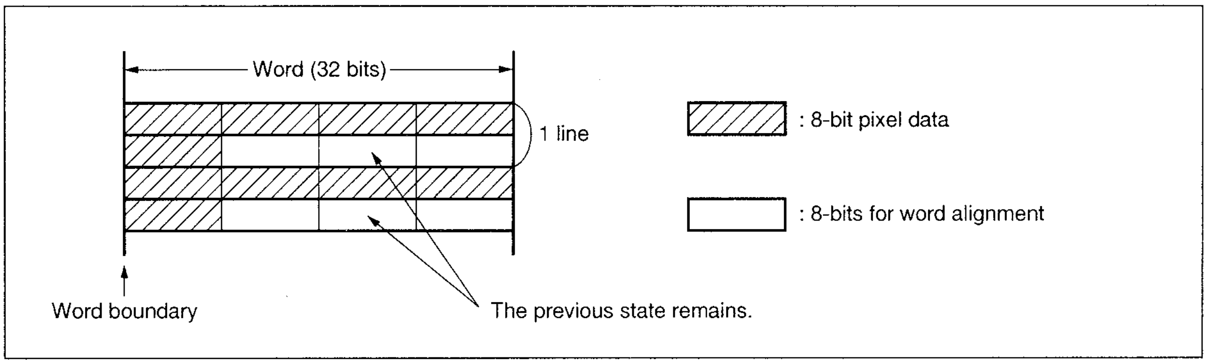
Performs expansion from the specified receive buffer and JBIG free parameter and outputs expanded image data to the image data buffer. Expands data for the number of lines specified by the number of image buffer lines.

Aborts operation when the remaining size of the compressed coded data buffer is 0. When 1-stripe processing ends, detects an SDNORM/SDRST marker and uses the status indicating one stripe end as a return value. For page end, also uses the status indicating page end as a return value.

(b) Image data

★
- When a page comes first, set all the reference lines to white.
  When LRLTWO=0: Set all to white with the previous 2 lines as virtual lines.
  When LRLTWO=1: Set all to white with the previous 1 line as a virtual line.
  When setting all reference lines to white, set "1" in reset.
- Place the image data so that the line start becomes the word boundary.
- Place image data per line so that it is continuous on the word boundary. Note that the 8 bits for word alignment shown below are not overwritten with image data but retain the previous state.

Figure 2-11. Line with 40 Pixels (Expansion System)



- The number of image data lines changes by an abort due to stripe end, page end or compressed coded data buffer empty.
- The recommended number of image buffer lines is the number of lines per stripe (L0) multiplied by a constant or a fraction.
  For example, suppose that the number of lines per stripe is 9, the recommended number of image buffer lines is its multiplication by a constant, i.e., 9, 18, 27, and so on, or its multiplication by a fraction, i.e., 1 and 3, and so on. If the number of image buffer lines and the number of lines per stripe are not obtained by multiplication by a constant or fraction, and it is SDRST (return value: 0×0C, sdrst=1), clear the reference lines when the next expansion is performed.

(c) ABORT processing

- When an ABORT marker code is detected, it is returned as a function return value (0×10).
- When an ABORT marker code is detected, expansion is aborted.

Remarks 1. When a RESERVE marker code is automatically detected, it is returned as a function return value (0×80). The next address of the RESERVE marker code is set to code_buf.

2. Stack bytes are automatically discarded.

CHAPTER 3 INSTALLATION

3.1 Link Procedure

The names of sections used in this library are shown below.

Table 3-1. Section

Section Name	Attribute	Function
.JBETEXT	text	JBIG compression program
.JBDTEXT	text	JBIG expansion program
.JBDATA	data	JBIG compression/expansion table

The link procedure is as follows.

(1) V810 family

(a) NEC version (ca732: Ver.2.00 or higher)

```
ld732 -D <link directive><..object file>.../libjbig.a -o <output file>
```

(b) GHS version (ELF version: Ver.1.8.7B or higher)

```
lx188 -o <output file> -sec <map file><..object file> -L<dir> -ljbig
```

(2) V850 family ★

(a) NEC version (ca850: Ver.1.00 or higher)

```
ld850 -D <link directive> <..object file>.../libjbig.a -o <output file>
```

(b) GHS version (ELF version: Ver.1.8.7B or higher)

```
lx188 -o <output file> -sec <map file><..object file> -L<dir> -ljbig
```

3.2 Sample Creation Procedure

The following NEC-version and GHS-version makefiles are executed.

3.2.1 V810 family

(1) NEC version (ca732: Ver.2.00 or higher)

- makefile

```
CC = ca732
AS = as732
LD = ld732

all: enc_main.elf dec_main.elf
enc_main.elf: crt810.o enc_main.o jbig_enc.lnk
    $(LD) -D jbig_enc.lnk crt810.o enc_main.o libjbig.a -o enc_main.elf
dec_main.elf: crt810.o dec_main.o jbig_dec.lnk
    $(LD) -D jbig_dec.lnk crt810.o dec_main.o libjbig.a -o dec_main.elf

enc_main.o: enc_main.c
    $(CC) -Wa,-cn -cpu 742 -c enc_main.c
dec_main.o: dec_main.c
    $(CC) -Wa,-cn -cpu 742 -c dec_main.c
crt810.o: crt810.s
    $(AS) crt810.s -o crt810.o
```

(a) NEC version linker options

-o <file name>
Specifies the name of an execution file to be generated.

-D <link directive>
Sets the start address of a section (.text, .data,...).
The contents of jbig_enc.lnk are shown below.

```
DATA: !LOAD ?RW V0×0 {.data = $PROGBITS ?AW;  JBDATA = $PROGBITS ?AW;};
TEXT: !LOAD ?RX V0×10000 {.text = $PROGBITS ?AX;  JBETEXT = $PROGBITS ?AX;};
_ _tp_TEXT @ %TP_SYMBOL;
_ _gp_DATA @ %GP_SYMBOL &_ _tp_TEXT;
```

The contents of jbig_dec.lnk are shown below.

```
DATA : !LOAD ?RW V0×0 {.data = $PROGBITS ?AW:  JBDATA = $PROGBITS ?AW:};
TEXT : !LOAD ?RX V0×10000 {.text = $PROGBITS ?AX:  JBDTEXT = $PROGBITS ?AX:};
_ _tp_TEXT @ %TP_SYMBOL;
_ _gp_DATA @ %GP_SYMBOL &_ _tp_TEXT;
```

(b) Sample main source compile

Example

```
ca732 -c enc_main.c
      |
      └────────── Compile only
```

Refer to respective user's manuals of the NEC linker and compiler for details.

(2) GHS version (ELF version Ver.1.8.7B or higher)

★ • makefile

```
CC = cc810e
AS = as800
LD = lx188

enc_main.elf: crt810.o enc_main.o
    $(LD) -o enc_main.elf -e__start -sec {.text 0x10000 : .JBETEXT:
       .data 0x0 : .JBDATA : .sdata : .sbss : .bss } crt810.o enc_main.o -L../../lib810
       -ljbig -M

dec_main.elf: crt810.o dec_main.o
    $(LD) -o dec_main.elf -e__start -sec {.text 0x10000 : .JBDTEXT:
       .data 0x0 : .JBDATA : .sdata : .sbss : .bss } crt810.o dec_main.o -L../../lib810
       -ljbig -M

enc_main.o: enc_main.c
    $(CC) -c -OA -G enc_main.c
dec_main.o: dec_main.c
    $(CC) -c -OA -G dec_main.c
crt810.o: crt810.s
    $(AS) -elf -cpu=V810 -o crt810.o crt810.s
```

(a) GHS linker options

-o <file name>
Specifies the name of an execution file to be generated.

-sec {section address [: section address...]}
Sets the start address of a section (.text, .data,..). Specification of each section is separated by ":". If an address is omitted, it is connected to the section specified immediately before.

(b) Sample main source compile

Specifies optimization option -OA and compiles.

Example

```
cc810e -c -OA -G enc_main.c
       |   |   |
       |   |   └──── Multi C source debugger
       |   └──────── Optimization (algorithm)
       └──────────── Compile only
```

See respective user's manuals of the GHS linker and compiler for details.

3.2.2 V850 family

(1) NEC version (ca850: Ver.1.00 or higher)

- makefile

```
CC = ca850
AS = as850
LD = ld850

all: enc_main.elf dec_main.elf
enc_main.elf: crt850.o enc_main.o jbig_enc.lnk
    $(LD) -D jbig_enc.lnk crt850.o enc_main.o libjbig.a -o enc_main.elf
dec_main.elf: crt850.o dec_main.o jbig_dec.lnk
    $(LD) -D jbig_dec.lnk crt850.o dec_main.o libjbig.a -o dec_main.elf

enc_main.o: enc_main.c
    $(CC) -cpu 3000 -c enc_main.c
dec_main.o: dec_main.c
    $(CC) -cpu 3000 -c dec_main.c
crt850.o: crt850.s
    $(AS) -cn -cpu 3000 crt850.s -o crt850.o
```

(a) NEC version linker options

-o <file name>

Specifies the name of an execution file to be generated.

-D <link directive>

Sets the start address of a section (.text, .data,...).
The contents of jbig_enc.lnk are shown below.

```
DATA1: !LOAD ?RW V0×1000 {.JBDATA = $PROGBITS ?AW;};
TEXT: !LOAD ?RX V0×2000 {.text = $PROGBITS ?AX;  JBETEXT = $PROGBITS ?AX;};
DATA2: !LOAD ?RW V0×100000 {.data = $PROGBITS ?AW;};
_ _tp_TEXT @ %TP_SYMBOL;
_ _gp_DATA @ %GP_SYMBOL & _ _tp_TEXT;
_ _ep_DATA @ %EP_SYMBOL;
```

The contents of jbig_dec.lnk are shown below.

```
DATA1 : !LOAD ?RW V0×1000 {JBDATA = $PROGBITS ?AW:};
TEXT : !LOAD ?RX V0×2000 {.text = $PROGBITS ?AX;  JBDTEXT = $PROGBITS ?AX;};
DATA2: !LOAD ?RW V0×100000 {.data = $PROGBITS ?AW;};
_ _tp_TEXT @ %TP_SYMBOL;
_ _gp_DATA @ %GP_SYMBOL & _ _tp_TEXT;
_ _ep_DATA @ %EP_SYMBOL;
```

(b) Sample main source compile

```
Example   ca850 -c enc_main.c
                 |
                 |_____________ Compile only
```

Refer to respective user's manuals of the NEC linker and compiler for details.

(2) GHS version (ELF version Ver.1.8.7B or higher)

- makefile

```
CC = cc850e
AS = as850e
LD = 1×188

all: enc_main.elf dec_main.elf
enc_main.elf: crt850.o enc_main.o
    $(LD) -o enc_main.elf -e_ _start -sec {.text 0×2000: .JBETEXT: .JBDATA 0×1000:
          .data 0×100000: .sdata: .bss: .sbss} crt850.o enc_main.o -L../../lib850 -ljbig -M

dec_main.elf: crt850.o dec_main.o
    $(LD) -o dec_main.elf -e_ _start -sec {.text 0×2000: .JBDTEXT: .JBDATA 0×1000:
          .data 0×100000: .sdata: .sbss: .bss} crt850.o dec_main.o -L../../lib850 -ljbig -M

enc_main.o: enc_main.c
    $(CC) -c -0A -G enc_main.c
dec_main.o: dec_main.c
    $(CC) -c -0A -G dec_main.c
crt850.o: crt850.s
    $(AS) -elf -o crt850.o crt850.s
```

(a) GHS linker options

-o <file name>

Specifies the name of an execution file to be generated.

-sec {section address [: section address...]}

Sets the start address of a section (.text, .data,..). Specification of each section is separated by ":". If an address is omitted, it is connected to the section specified immediately before.

(b) Sample main source compile

Specifies optimization option -OA and compiles.

Example cc850e -c -OA -G enc_main.c

Multi C source debugger

Optimization (algorithm)

Compile only

See respective user's manuals of the GHS linker and compiler for details.

3.3 Symbol Name Convention

All the symbols names used in this library are assigned “jbig_” at their beginning. “jbig_” is a middleware ID. A function name consists of middleware ID + function name. Pay attention to symbol names used in user applications.

CHAPTER 4 SYSTEM EXAMPLES

This chapter describes system examples of compression/expansion by JBIG. For main sources of the system examples, refer to **APPENDIX SAMPLE SOURCE LIST** (this sample does not satisfy all the specifications).

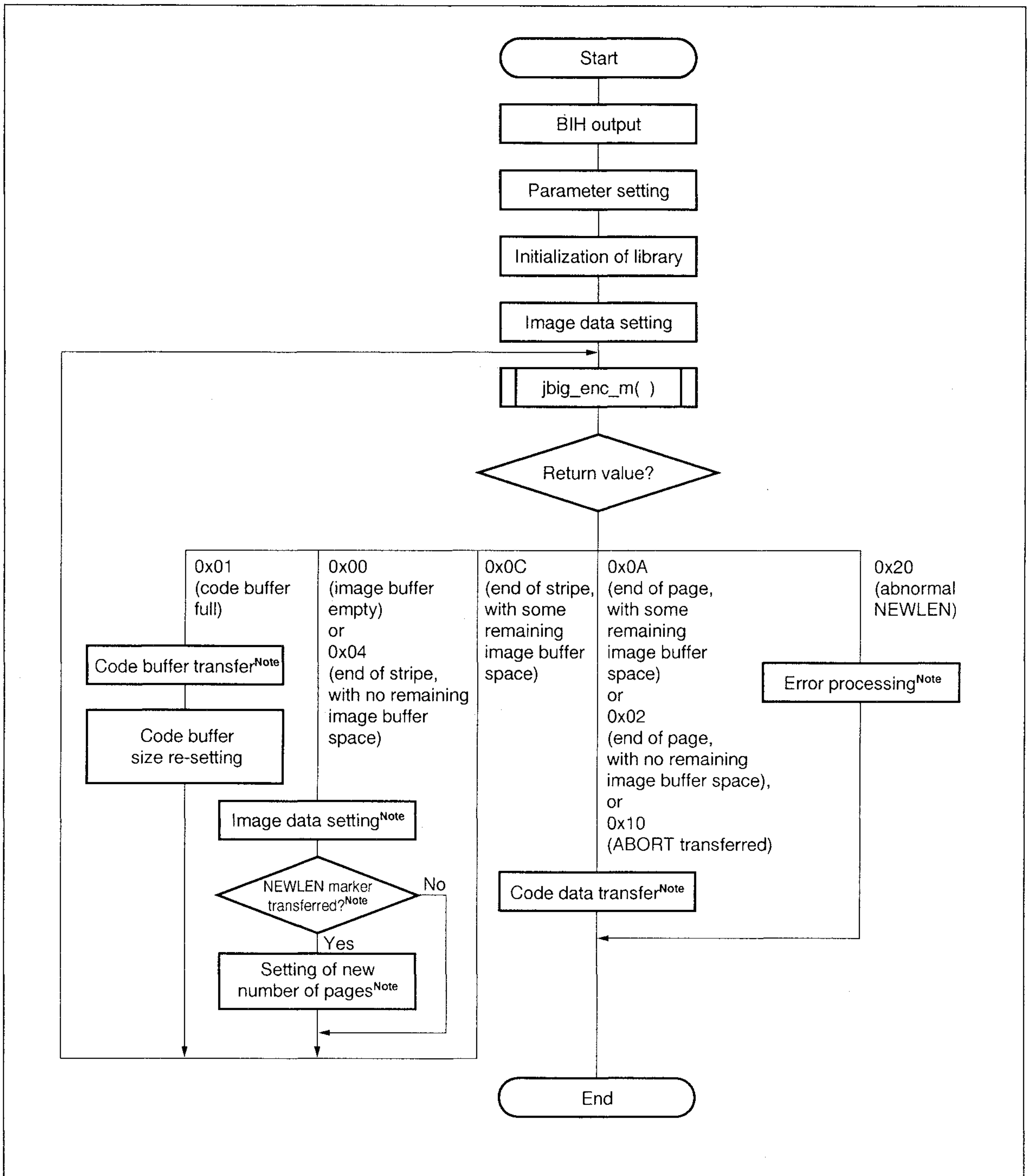
4.1 Compression System Example

Performs compression from the specified image data buffer and JBIG free parameter and outputs coded data to the compressed coded data buffer. Compresses data for the number of lines specified by the number of image buffer lines.

Aborts operation when the remaining size of the compressed coded data buffer is 0. When 1-stripe processing ends, outputs an SDNORM/SDRST marker to the compressed coded data buffer and uses the status indicating 1-stripe end as a return value.

★

Figure 4-1. Compression System Example



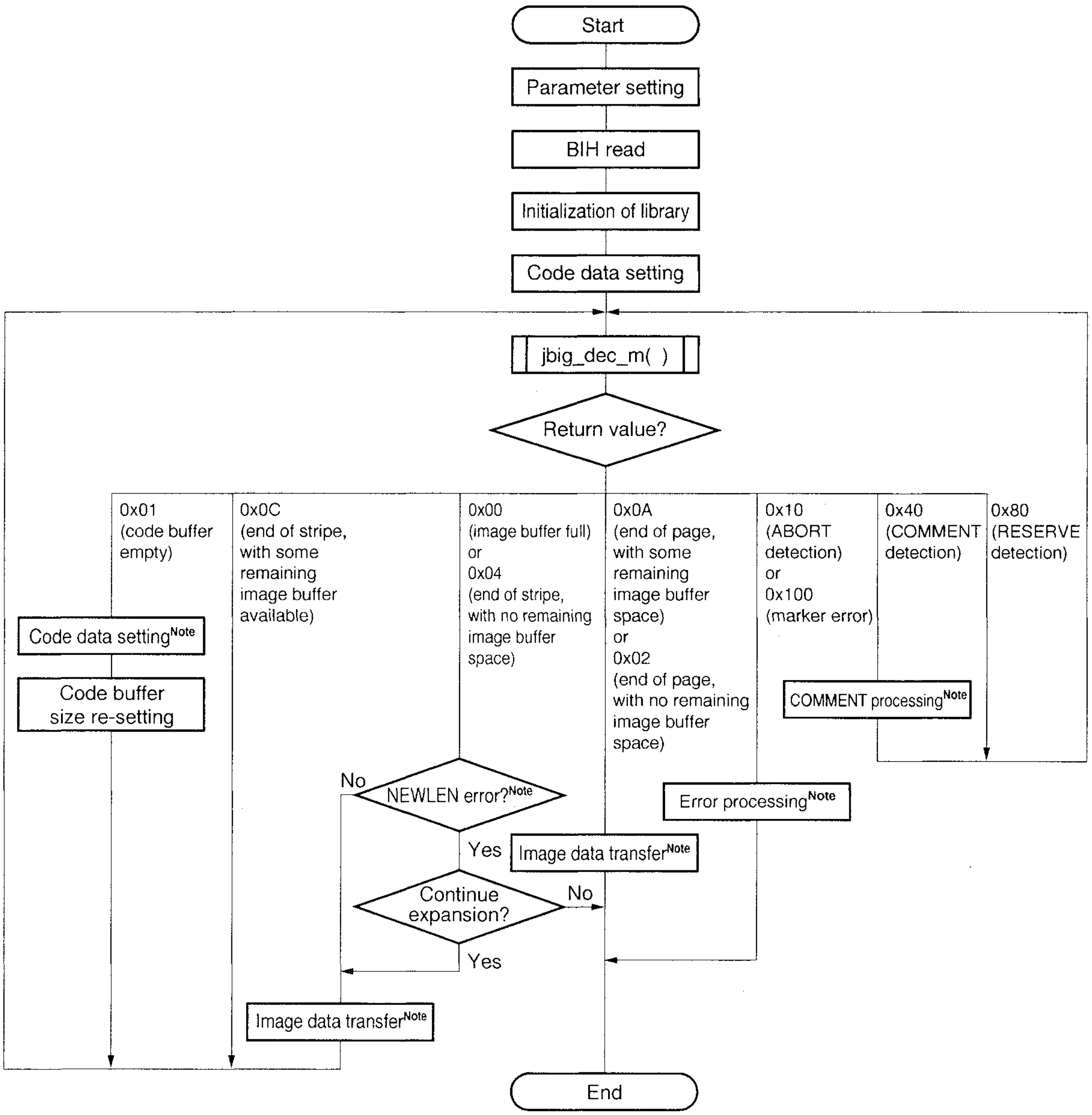
Note These conditional statements and processing are not included in the sample source.

4.2 Expansion System Example

Performs expansion from the specified receive buffer and JBIG free parameter and outputs expanded image data to the image data buffer. Expands data for the number of lines specified by the number of image data lines.

Aborts operation when the remaining size of the compressed coded data buffer is 0. When 1-stripe processing ends, detects an SDNORM/SDRST marker and uses the status indicating 1-stripe end as a return value.

Figure 4-2. Expansion System Example ★


Start
Parameter setting
BIH read
Initialization of library
Code data setting
jbig_dec_m( )
Return value?
0x01 (code buffer empty)
0x0C (end of stripe, with some remaining image buffer available)
0x00 (image buffer full) or 0x04 (end of stripe, with no remaining image buffer space)
0x0A (end of page, with some remaining image buffer space) or 0x02 (end of page, with no remaining image buffer space)
0x10 (ABORT detection) or 0x100 (marker error)
0x40 (COMMENT detection)
0x80 (RESERVE detection)
Code data setting^Note
Code buffer size re-setting
COMMENT processing^Note
No
NEWLEN error?^Note
Error processing^Note
Yes
Image data transfer^Note
Continue expansion?
No
Yes
Image data transfer^Note
End

Note These conditional statements and processing are not included in the sample source.

4.3 Memory Map Example

The followings are examples of sample program (compression/expansion) memory map.

Figure 4-3. V810 Family Memory Map Example

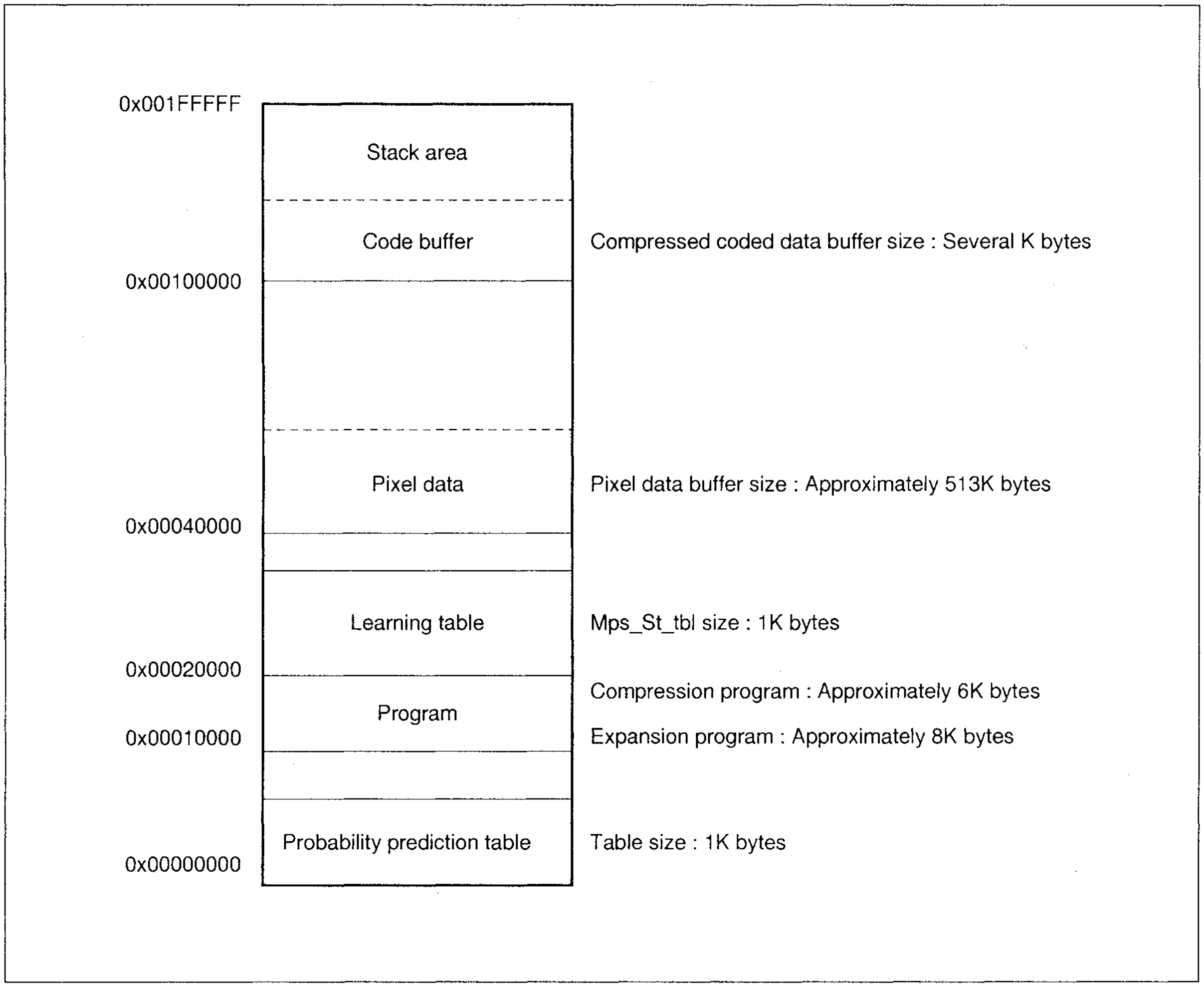
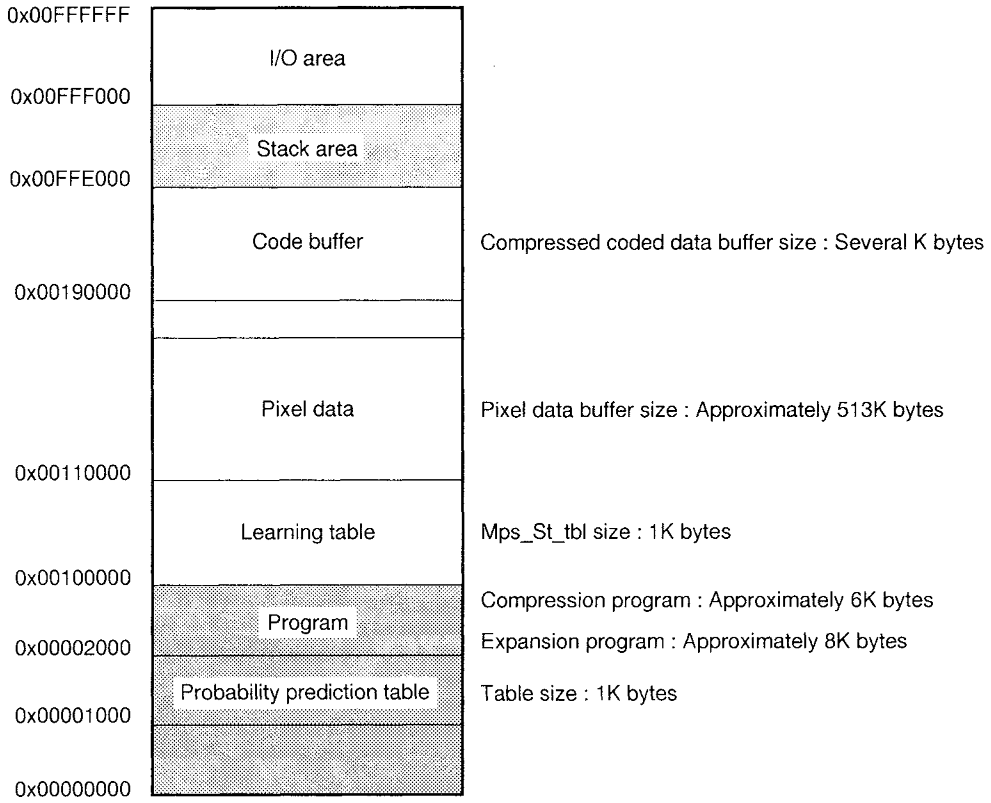

0x001FFFFF
Stack area
Code buffer
Compressed coded data buffer size : Several K bytes
0x00100000
Pixel data
Pixel data buffer size : Approximately 513K bytes
0x00040000
Learning table
Mps_St_tbl size : 1K bytes
0x00020000
Compression program : Approximately 6K bytes
Program
★ 0x00010000
Expansion program : Approximately 8K bytes
Probability prediction table
Table size : 1K bytes
0x00000000

Figure 4-4. V850 Family Memory Map Example (V851) ★


0x00FFFFFF
I/O area
0x00FFF000
Stack area
0x00FFE000
Code buffer
Compressed coded data buffer size : Several K bytes
0x00190000
Pixel data
Pixel data buffer size : Approximately 513K bytes
0x00110000
Learning table
Mps_St_tbl size : 1K bytes
0x00100000
Program
Compression program : Approximately 6K bytes
Expansion program : Approximately 8K bytes
0x00002000
Probability prediction table
Table size : 1K bytes
0x00001000
0x00000000

Remark ▨ : On-chip ROM area (0x00000000-0x000FFFFF)
░ : On-chip RAM area (0x00FFE000-0x00FFEFFF)

[MEMO]

APPENDIX SAMPLE SOURCE LIST

(1) JBIG compression sample source

```
#include "jbig810.h"

/* BIH */
#define BIH_DL     0
#define BIH_D      0
#define BIH_P      1
#define BIH_FILL   0
#define BIH_XD     1728
#define BIH_YD     2376
#define BIH_L0     128
#define BIH_MX     0x7F
#define BIH_MY     0
#define BIH_ORDER  0
#define VLENGTH    0x20
#define TPBON      0x08  /* TPBON */
#define LRLTWO     0x40  /* 3 LINE */

/* jbig_enc_m() */
#define PIXEL_BUF        0x40000L
#define PIXEL_BUF_LINE   72
#define CODE_BUF         0x100000L
#define CODE_BUF_SIZE    256
#define MPS_ST_BUF       0x20000L
#define PAGE1            2376
#define STRIPE1          128
#define XD               1728
#define NORMAL_END       0x00
#define CODE_FULL        0x01
#define PAGE_END_FULL    0x02
#define PAGE_END         0x0A
#define STRIPE_END_FULL  0x04
#define STRIPE_END       0x0C
#define ABORT_END        0x10
#define NEWLEN_ERR       0x20

main()
{

            struct J_PARA encdata;
            register unsigned int cnt;
            register unsigned int pixel_byte;
            register unsigned int status;

            encdata.code_buf = (unsigned char *)CODE_BUF;

            /* BIH output */
            *encdata.code_buf++ = (unsigned char)BIH_DL;
            *encdata.code_buf++ = (unsigned char)BIH_D;
            *encdata.code_buf++ = (unsigned char)BIH_P;
```

```
    *encdata.code_buf++ = (unsigned char)BIH_FILL;

    *encdata.code_buf++ = (unsigned char)(BIH_XD>>24);
    *encdata.code_buf++ = (unsigned char)(BIH_XD>>16);
    *encdata.code_buf++ = (unsigned char)(BIH_XD>>8);
    *encdata.code_buf++ = (unsigned char)BIH_XD;

    *encdata.code_buf++ = (unsigned char)(BIH_YD//24);
    *encdata.code_buf++ = (unsigned char)(BIH_YD//16);
    *encdata.code_buf++ = (unsigned char)(BIH_YD//8);
    *encdata.code_buf++ = (unsigned char)BIH_YD;

    *encdata.code_buf++ = (unsigned char)(BIH_L0>>24);
    *encdata.code_buf++ = (unsigned char)(BIH_L0>>16);
    *encdata.code_buf++ = (unsigned char)(BIH_L0>>8);
    *encdata.code_buf++ = (unsigned char)BIH_L0;

    *encdata.code_buf++ = (unsigned char)BIH_MX;
    *encdata.code_buf++ = (unsigned char)BIH_MY;
    *encdata.code_buf++ = (unsigned char)BIH_ORDER;
    *encdata.code_buf++ = (unsigned char)(TPBON|VLENGTH);

    /* jbig_enc_m( ) parameter set */
    encdata.pixel_buf = (unsigned char *)PIXEL_BUF;
    encdata.pixel_buf_line = PIXEL_BUF_LINE;
    encdata.code_buf_size = CODE_BUF_SIZE;
    encdata.Mps_St_tbl = (unsigned char *)MPS_ST_BUF;
    encdata.Xd = XD;
    encdata.Yd = PAGE1;
    encdata.line_cnt = 0;
    encdata.L0 = STRIPE1;
★   encdata.Options = (TPBON|VLENGTH);        /* TP=on, 3tmp    */
    encdata.newlen = 0;
    encdata.sdrst = 0;                        /* SDNORM         */
    encdata.abort = 0;                        /* ABORT          */
    encdata.Tx = 0;                           /* AT = default   */
    encdata.yAT = 0;
    encdata.restart_adr = 0;
    encdata.reset = 1;                        /* reset          */

    pixel_byte = PIXEL_BUF_LINE*(encdata.Xd/8);
    encdata.next_pixel_buf = (unsigned char *)(PIXEL_BUF + pixel_byte);

    /******************/
    /* pixel data set */
    /******************/
★   for(;;){
          switch(status = jbig_enc_m(&encdata)){
                  case CODE_FULL          :encdata.code_buf_size = CODE_BUF_SIZE;
                                           break;
                  case STRIPE_END         :break;
                  case PAGE_END           :goto page_end;
                  case ABORT_END          :if( encdata.code_buf_size != CODE_BUF_SIZE )
```

```
                                                 /*********************/
                                                 /* code data forward */
                                                 /*********************/
                                                 abort_int();
                          case STRIPE_END_FULL   :
                          case NORMAL_END        :/******************/
                                                  /* pixel data set */
                                                  /******************/
                                                 encdata.next_pixel_buf = (unsigned char *)
                                                 (encdata.pixel_buf + pixel_byte);
                                                 break;
                          case PAGE_END_FULL     :goto page_end;
                          case NEWLEN_ERR        :newlen_err();
            }
      }
page_end:;

}

abort_int(){
            exit(1);
}

newlen_err(){
            exit(1);
}
```

★ (2) JBIG expansion sample source

```
#include "jbig810.h"

/*jbig_dec_m()*/
#define PIXEL_BUF        0x40000L
#define PIXEL_BUF_LINE   72
#define CODE_BUF         0x100000L
#define CODE_BUF_SIZE    0x10000
#define MPS_ST_BUF       0x20000L
#define PAGE1            2376

#define     NORMAL_END       0x00
#define     CODE_BUF_FULL    0x01
#define     PAGE_END_FULL    0x02
#define     PAGE_END         0x0a
#define     STRIPE_END_FULL  0x04
#define     STRIPE_END       0x0c
#define     ABORT_END        0x10
#define     COMMENT_END      0x40
#define     RESERVE_END      0x80
#define     MARKER_ERR       0x100

main()
{
      struct J_PARA decdata;
      register unsigned int error, pixel_byte;
      register unsigned int pixel_buf_next = PIXEL_BUF;

      decdata.pixel_buf = (unsigned char *)PIXEL_BUF;
      decdata.pixel_buf_line = PIXEL_BUF_LINE;
      decdata.code_buf = (unsigned char*)CODE_BUF;
      decdata.code_buf_size = CODE_BUF_SIZE - 20;        /* BIH 20 byte          */
      decdata.pixel_buf = (unsigned char *)PIXEL_BUF;
      decdata.restart_adr = 0;
      decdata.Mps_St_tbl = (unsigned char *)MPS_ST_BUF;

      decdata.line_cnt = 0;
      decdata.code_buf += 4;

      decdata.Xd  = (*decdata.code_buf++) << 24;         /* read BIH (Xd)        */
      decdata.Xd |= ((*decdata.code_buf++) << 16);
      decdata.Xd |= ((*decdata.code_buf++) << 8);
      decdata.Xd |= *decdata.code_buf++;

      decdata.Yd  = (*decdata.code_buf++) << 24;         /* read BIH (Yd)        */
      decdata.Yd |= ((*decdata.code_buf++) << 16);
      decdata.Yd |= ((*decdata.code_buf++) << 8);
      decdata.Yd |= *decdata.code_buf++;

      decdata.L0  = (*decdata.code_buf++) << 24;         /* read BIH (L0)        */
      decdata.L0 |= ((*decdata.code_buf++) << 16);
      decdata.L0 |= ((*decdata.code_buf++) << 8);
      decdata.L0 |= *decdata.code_buf++;
      decdata.code_buf += 3;

      decdata.Options = (*decdata.code_buf++) & 0xff;    /* read BIH (Options)  */
      decdata.reset = 1;                                 /* reset               */

      pixel_byte = PIXEL_BUF_LINE * (jbig.Xd / 8);
      decdata.next_pixel_buf = (unsigned char *)(PIXEL_BUF + pixel_byte);

      for(;;){
            switch(status = jbig_dec_m(&decdata)){
                  case CODE_BUF_FULL    :decdata.code_buf_size = CODE_BUF_SIZE;break;
                  case STRIPE_END       :break;
                  case PAGE_END         :goto page_end;
                  case STRIPE_END_FULL  :
                  case NORMAL_END       :/**********************/
                                         /* pixel data forward */
                                         /**********************/
                                        decdata.next_pixel_buf = (unsigned char *)
                                        (decdata.pixel_buf + pixel_byte) ;
                                        break;
                  case PAGE_END_FULL    :goto page_end;
                  case ABORT_END        :abort_err();
                  case COMMENT_END      :comment_end();
                  case RESERVE_END      :break;
                  case MARKER_ERR       :marker_err();
            }
      }

page_end:;

}

abort_err(){
      exit(1);
}

comment_end()(
/**************************/
/* comment marker routine */
/**************************/
}

marker_err(){
      exit(1);
}
```

[MEMO]

Phase-out/Discontinued

NEC

Facsimile Message

From:

____
Name

____
Company

____
Tel. FAX

____
Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-889-1689

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: ______________________________

Document number: ______________________ Page number: __________

____

____

____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑