Federal Office
for Information Security

JAB Code (Just Another Bar Code) color bar code symbology specification

Authors: Huajian Liu, Waldemar Berchtold

Federal Office for Information Security
Post Box 20 03 63
D-53133 Bonn
Phone: +49 22899 9582-<Tel. Durchwahl>
E-Mail: <E-Mail>@bsi.bund.de
Internet: https://www.bsi.bund.de

Table of Contents

1 Introduction and Scope 5
2 Terms and definitions, abbreviations and symbols, mathematical and logical operations 6
2.1 Terms and definitions 6
2.1.1 Module 6
2.1.2 Finder pattern 6
2.1.3 Alignment pattern 6
2.1.4 Data interleaving 6
2.1.5 Color palette 6
2.1.6 Padding bit / Stuffing bit 6
2.1.7 Master symbol 6
2.1.8 Slave symbol 6
2.1.9 Host symbol 6
2.2 Abbreviations 7
2.3 Mathematical symbols 7
2.4 Mathematical and logical operations 7
3 Symbol description 8
3.1 Basic characteristics 8
3.2 Summary of additional features 9
3.3 Symbol structure 9
3.3.1 Square master symbol 9
3.3.2 Rectangle master symbol 10
3.3.3 Square slave symbol 10
3.3.4 Rectangle slave symbol 11
3.3.5 Symbol side size 11
3.3.6 Finder pattern 14
3.3.7 Alignment pattern 14
3.3.8 Color palette 18
3.3.9 Metadata 18
3.3.10 Encoded data 18
3.4 Metadata structure 18
3.4.1 Metadata of master symbol 19
3.4.2 Metadata of slave symbol 22
3.4.3 Metadata error encodation 23
3.4.4 Reserved modules for metadata and color palette 24
3.5 Symbol Cascading 30
3.5.1 Symbol docking rules 30
3.5.2 Symbol decoding order 30
4 Symbol generation 34
4.1 Encode procedure overview 34
4.2 Data analysis 34
4.3 Encoding modes 36
4.3.1 Uppercase mode 36
4.3.2 Lowercase mode 36
4.3.3 Numeric mode 37
4.3.4 Punctuation mode 37
4.3.5 Mixed mode 38

4.3.6 Alphanumeric mode ... 38
4.3.7 Byte mode ... 38
4.3.8 Extended Channel Interpretation (ECI) mode ... 38
4.3.9 FNC1 mode ... 39
4.4 Error correction ... 39
4.4.1 Selectable error correction levels ... 40
4.4.2 Stuffing Bits ... 40
4.4.3 Generating the error correction stream ... 40
4.5 Data interleaving ... 41
4.6 Metadata module reservation ... 41
4.7 Data module encodation and placement ... 42
4.8 Data masking ... 43
4.8.1 Data mask patterns ... 43
4.8.2 Evaluation of data masking results ... 44
4.9 Metadata generation and module placement ... 45
5 User Guidelines ... 46
5.1 Dimensions ... 46
5.2 Use selection of module color ... 46
5.3 User selection of error correction level ... 46
5.4 User selection of symbol and code shape ... 46
5.5 Guidelines for symbol print and scan ... 47
6 Reference decode algorithm ... 48
6.1 Decoding procedure overview ... 48
6.2 Locating the finder patterns ... 48
6.3 Decoding the metadata ... 49
6.4 Locating the alignment patterns and establishing the sampling grid ... 50
6.5 Constructing the color palettes ... 51
6.6 Decoding the data message ... 51
6.7 Locating and decoding slave symbols ... 52
Annex A: Error detection and correction ... 53
Annex B: Matrix generation for metadata ... 55
Annex C: JAB Code symbol encoding example ... 56
Annex D: Optimization of bit stream length ... 58
Annex E: Interleaving algorithm ... 60
Annex F: Guidelines for module color selection and color palette construction ... 61

1 Introduction and Scope

JAB Code is a color two-dimensional matrix symbology whose basic symbols are made of colorful square modules arranged in either square or rectangle grids. JAB Code has two types of basic symbols, named as master symbol and slave symbol. A JAB Code contains one master symbol and optionally multiple slave symbols. Master symbol contains four finder patterns located at the corners of the symbol, while slave symbol contains no finder pattern. A slave symbol can be docked to a master symbol or another docked slave symbol in either horizontal or vertical direction. JAB Code can encode from small to large amount of data correlated to user-specified percentages of error correction.

This specification defines the requirements for the symbology known as JAB Code. It specifies the JAB Code symbology characteristics, symbol structure, symbol dimensions, symbol cascading rules, data character encodation, error correction rules, user-selectable application parameters, print quality requirements and a reference decode algorithm.

2 Terms and definitions, abbreviations and symbols, mathematical and logical operations

2.1 Terms and definitions

2.1.1 Module

A module is a single cell in a matrix symbology which is elemental entity used to encode data. In JAB Code a module is the basic encoding entity which is a square in one color.

2.1.2 Finder pattern

Finder pattern is a fixed reference pattern at predefined positions in a matrix symbology, which enables the decode software to locate the JAB symbol in an image.

2.1.3 Alignment pattern

Alignment pattern is a fixed reference pattern at predefined positions in a matrix symbology, which enables the decode software to resynchronize the coordinate mapping of the modules in the event of moderate amounts of distortion of the image.

2.1.4 Data interleaving

Data interleaving is a procedure which pseudo-randomly arranges the data in a matrix symbology.

2.1.5 Color palette

Color palette is a set of reference modules of used colors in the symbol, which is located at predefined positions in a matrix symbology.

2.1.6 Padding bit / Stuffing bit

Padding/stuffing bits are the bits which are used to fill empty positions of the available encoding capacity after the final bit of the encoded data. Padding/stuffing bits do not represent data.

2.1.7 Master symbol

Mater symbol in JAB Code is the main symbol which contains finder patterns and is used to locate the whole JAB code.

2.1.8 Slave symbol

Slave symbol in JAB Code is appending symbols which may be used to encode more data with a lower overhead in terms of auxiliary modules.

2.1.9 Host symbol

Host symbol is the symbol in a JAB Code which docks slave symbols on its horizontal or vertical sides. Either master symbol or slave symbol may be a host symbol.

2.2 Abbreviations

JAB Just Another Bar code

LDPC Low-Density Parity-Check code

SS in master symbol: symbol shape flag / in slave symbol: same shape and size flag

VF side-version fag

MSK masking reference

SF slave position flag

SE same error correction level

V side-version

E error correction parameter

S slave positions

m the raw data bits

c transmitted codeword

r received codeword

L number of iterations

2.3 Mathematical symbols

For the purposes of this document, the following mathematical symbols apply:

N_c the module color mode indicating the number of module colors in the symbol

C the symbol capacity in number of bits

P_n the symbol net payload (the number of raw data bits)

P_g the symbol gross payload (the number of encoded data bits)

K the number of error correction bits in the symbol, equal to P_g-P_n

H the parity check matrix of LDPC code

w_r the number of 1's in each row in H

w_c the number of 1's in each column in H

2.4 Mathematical and logical operations

For the purposes of this document, the following mathematical and logic operations apply.

max(x,y) is the greater of x and y

div is the integer division operator

mod is the remainder after division

XOR is the exclusive-or logic function that outputs one only when the two inputs differ.

3 Symbol description

3.1 Basic characteristics

a. Encodable character set

(1) numeric data (digits 0 – 9; space, two punctuation characters: , .);

(2) uppercase letters (A – Z; space);

(3) lowercase letters (a – z; space);

(4) punctuation marks (32 characters, see Table 12);

(5) mixed characters (seven Germanic umlauts Ä Ö Ü ä ö ü ß; two other marks; control characters and combinations, see Table 12);

(6) alphanumeric data (digits 0 – 9; uppercase letters A – Z; lowercase letters a – z; space);

(7) byte data (default interpretation: ISO/IEC 8859-15, corresponding to ECI 000017).

b. Symbol type

(1) In JAB Code there are two types of symbols: master symbol and slave symbol.

(2) A JAB Code may contain one master symbol and optionally multiple slave symbols.

c. Symbol shape

(1) Master symbol and slave symbol can be of either square or rectangle form.

(2) Master symbol and slave symbol in a JAB Code may be of different shapes.

d. Symbol size

(1) The smallest size of JAB Code symbol side (master or slave) is 21 and the largest is 145, namely minimal 21×21 modules square and the maximal 145×145 modules.

(2) No quiet zone is required outside the bounds of the symbol.

e. Module color

(1) The number of module colors is configurable in eight modes.

(2) The minimal number of module colors is 4 and the maximal number of colors is 256.

(3) Guidelines for color selection are given in Annex F.

f. Representation of data

(1) A module represents $\log_2(N_c)$ binary bits. See Section 4.7.

(2) The binary bits that a module represents correspond to the index value of the module color in the color palette.

g. Data capacity

(1) The data capacity of JAB Code depends on the symbol size, the number of module colors, and the error correction level.

(2) The capacity of a single-symbol square code is listed in Table 1.

h. Selectable error correction

(1) User-selectable error correction levels are supported.

(2) In one JAB code, different error correction levels may be configured in each symbol.

i. Symbol cascading

(1) Slave symbols can be docked to the side of a master symbol or other slave symbols.

(2) JAB Code may have an arbitrary form by cascading master and slave symbols in horizontal and vertical directions.

j. Code type: Matrix

k. Orientation independent: Yes

3.2 Summary of additional features

The use of the following additional features is optional in JAB Code:

a. Mirror Imaging: When JAB Code is obtained in mirror reversal, it is still possible to achieve a valid decode of a symbol with the standard reader. Refer to Section 6.3.

b. Extended Channel Interpretation: The ECI mechanism enables data using character sets other than the default encodable set (e.g. Arabic, Chinese, Cyrillic, Greek, Hebrew, etc.) and other data interpretations or industry-specific requirements to be represented.

3.3 Symbol structure

3.3.1 Square master symbol

The structure of a square JAB Code master symbol is shown in Figure 1. A square master symbol shall consist of function patterns including finder pattern, alignment pattern (from Side-Version 6), color palette, metadata and encoded data region. Four finder patterns are located at the four symbol corners respectively, with one module between the outermost layer and the border. No quiet zone surrounding the symbol is required. The master symbol illustrated in Figure 1 is a square symbol of Side-Version 2, whose width and height are 25 modules.

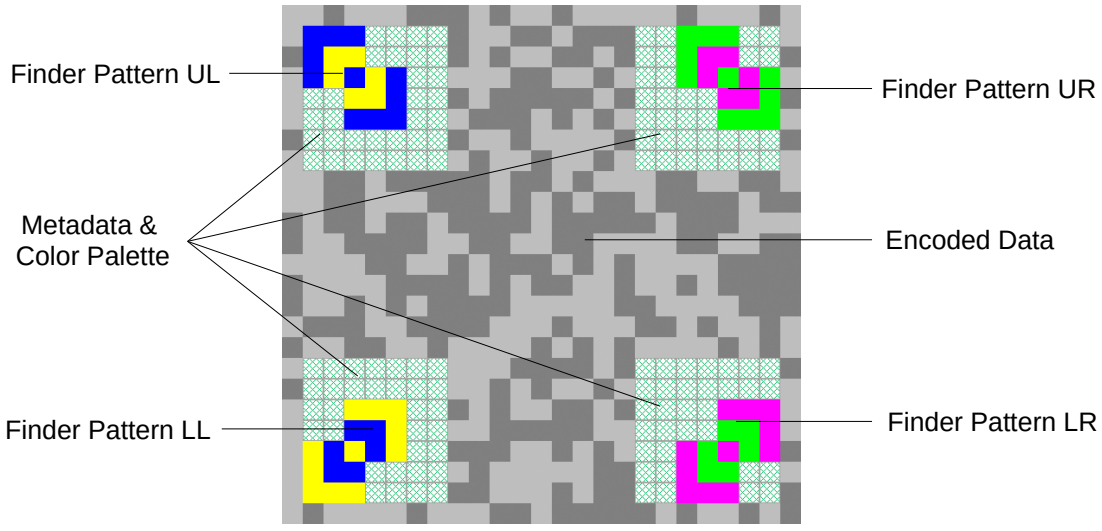

Figure 1: Structure of a square master symbol

3.3.2 Rectangle master symbol

The structure of a rectangle JAB Code master symbol is shown in Figure 2. The structure of a rectangle master symbol is the same as a square master symbol, except that the horizontal and vertical distance between the finder patterns are not equal. Like square symbols, no quiet zone is required for rectangle master symbols. The master symbol illustrated in Figure 2 is a rectangle symbol of combination of Side-Version 5 and 2, of which the width is 37 modules and the height is 25 modules.

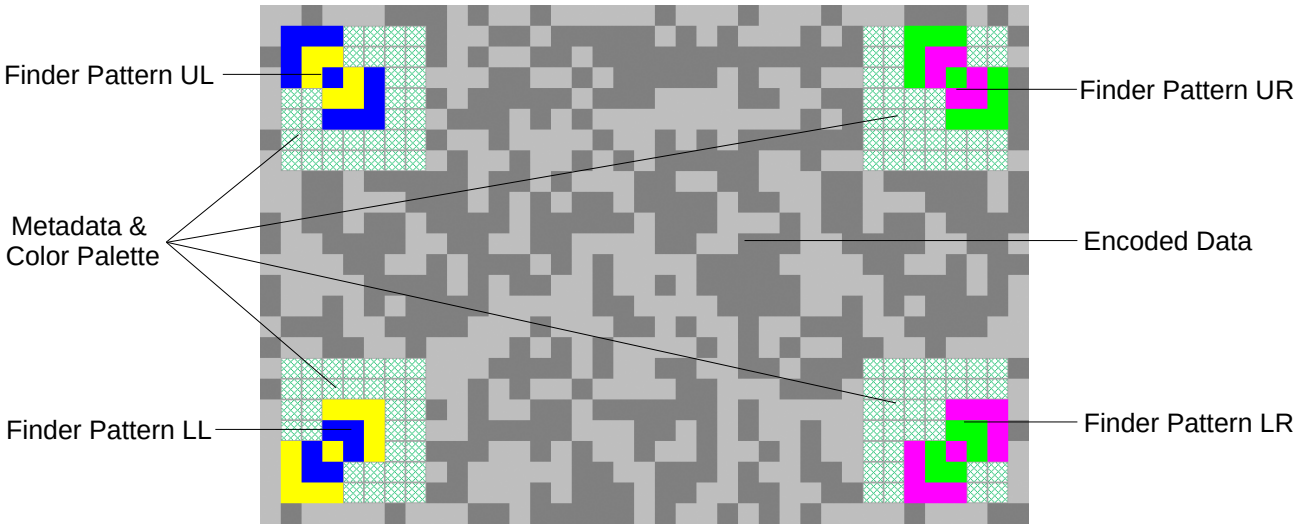

Figure 2: Structure of a rectangle master symbol

3.3.3 Square slave symbol

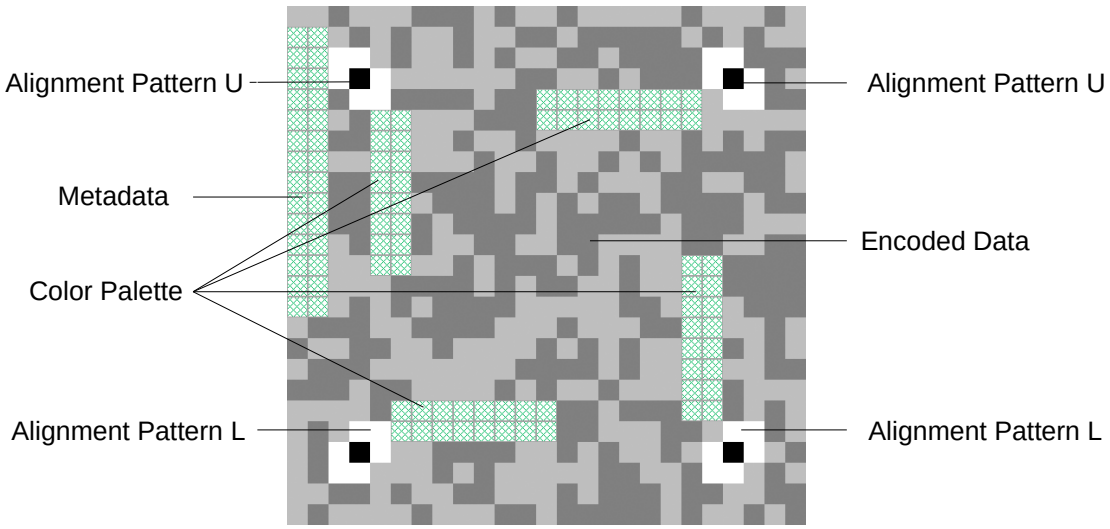

Figure 3: Structure of a square slave symbol

The structure of a square JAB Code slave symbol is shown in Figure 3. Except finder patterns, slave symbols contain the same function patterns as the master symbol, including alignment patterns, metadata regions

and encoded data region. In slave symbols, the four finder patterns are replaced by four alignment patterns. Like the master symbol, no surrounding quiet zone is required for slave symbols. The slave symbol illustrated in Figure 3 is a square symbol of Side-Version 2, whose width and height are 25 modules.

3.3.4 Rectangle slave symbol

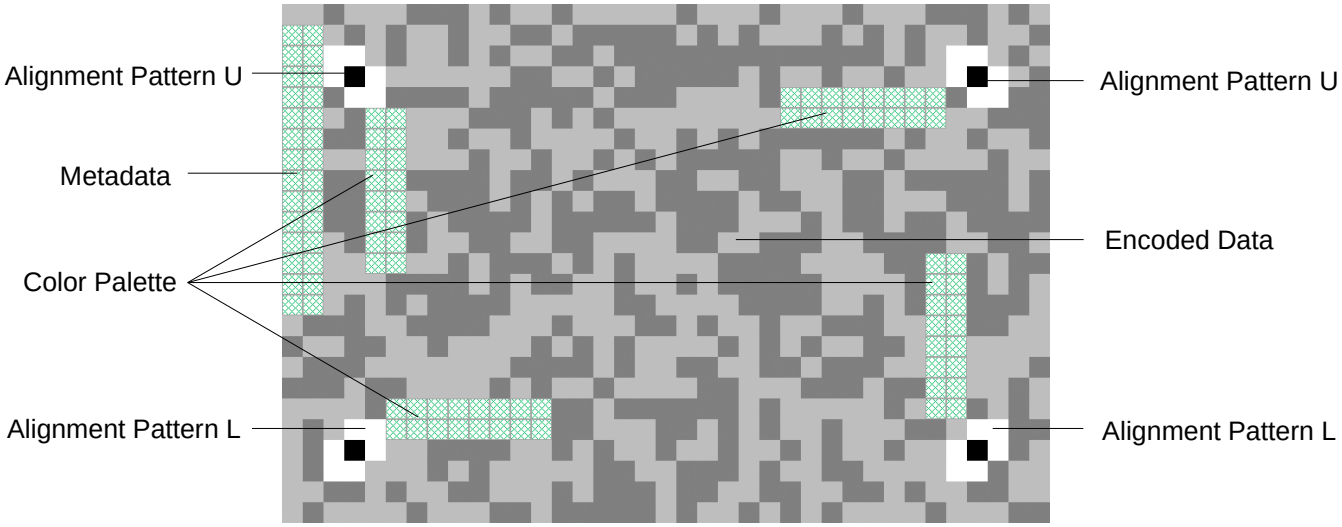

Figure 4: Structure of a rectangle slave symbol

The structure of a rectangle JAB Code slave symbol is shown in Figure 4. The structure of the rectangle slave symbol is the same as the rectangle master symbol, except that the finder patterns are replaced by four alignment patterns. Similarly, no quiet zone is required for rectangle slave symbols. The slave symbol illustrated in Figure 4 is a rectangle symbol of combination of Side-Version 5 and 2, of which the width is 37 modules and the height is 25 modules.

3.3.5 Symbol side size

The side of a JAB Code symbol may have 32 different sizes referred to as Side-Version 1, Side-Version 2, ... Side-Version 32, as listed in Table 1. The side size increases in step of 4 modules from 21 modules in Side-Version 1 to 145 modules in Side-Version 32. A square symbol has the same Side-Version for both the horizontal and vertical sides, while a rectangle symbol may have any combination of two different side-versions for the horizontal and vertical sides. The smallest square symbol measures 21×21 modules and the largest square symbol measures 145×145 modules. The smallest rectangle symbol measures 21×25 modules and the largest rectangle symbol measures 141×145 modules. The rectangle symbol of 21×145 or 145×21 modules has the maximal proportion between the horizontal and vertical sides.

The capacities listed in Table 1 are based on the recommended error correction level 6 for square symbols. In codes with 8 colors, the metadata take 15 modules for version 1 to 4, 16 modules for version 5 to 8, 17 modules for version 9 to 16 and 20 modules for version 17 to 32 for the master symbol. The metadata take 2 modules for slave symbols with the same configuration as the master symbol. The number of data modules can be calculated as follows:

Distance of Finder pattern Center to Border: $DFCB=4$

Minimum Distance Between Alignment pattern: $MDBA=16$

Number of alignment pattern modules:	$a_x = MAX\left(0, \left\lfloor (SideSize_x - DFCB \times 2 + 1) / MDBA - 1 \right\rfloor\right)$ $a_y = MAX\left(0, \left\lfloor (SideSize_y - DFCB \times 2 + 1) / MDBA - 1 \right\rfloor\right)$ $a = ((a_x + 2) \times (a_y + 2) - 4) \times 7$
Number of color palette modules:	$C_{Palette} = MIN(64, NumberOfModuleColor) \times 2$
Number of finder pattern modules:	$F_{Master} = 4 \times 17$; $F_{Slave} = 4 \times 7$
Number of data modules in master:	$SideSize_x \times SideSize_y - a - C_{Palette} - F_{Master} - Metadata$
Number of data modules in slave:	$SideSize_x \times SideSize_y - a - C_{Palette} - F_{Slave} - Metadata$

where MAX(. , .) and MIN(. , .) are the maximum and minimum function.

Table 1: Symbol side versions and square symbol capacity of JAB Code (N_c=000, 001, 010, w_c=4, w_r=7)

Side-Version	Side size (in modules)	Number of data modules						Symbol net payload P_n (in bits)					
		Square Master			Square Slave			Square Master			Square Slave		
		2	4	8	2	4	8	2	4	8	2	4	8
1	21	346	364	363	424	423	416	148	312	466	181	362	534
2	25	530	548	547	608	607	600	227	469	703	260	520	771
3	29	746	764	763	824	823	816	319	654	981	353	705	1049
4	33	994	1012	1011	1072	1071	1064	426	867	1299	459	918	1368
5	37	1270	1290	1290	1352	1351	1344	544	1105	1658	579	1158	1728
6	41	1582	1602	1602	1664	1663	1656	678	1373	2059	713	1425	2129
7	45	1926	1946	1946	2008	2007	2000	825	1668	2502	860	1720	2571
8	49	2302	2322	2322	2384	2383	2376	986	1990	2985	1021	2042	3054
9	53	2704	2727	2728	2792	2791	2784	1158	2337	3507	1196	2392	3579
10	57	3144	3167	3168	3232	3231	3224	1347	2714	4073	1385	2769	4145
11	61	3616	3639	3640	3704	3703	3696	1549	3119	4680	1587	3174	4752
12	65	4120	4143	4144	4208	4207	4200	1765	3551	5328	1803	3606	5400
13	69	4656	4679	4680	4744	4743	4736	1995	4010	6017	2033	4065	6089
14	73	5224	5247	5248	5312	5311	5304	2238	4497	6747	2276	4552	6819
15	77	5824	5847	5848	5912	5911	5904	2496	5011	7518	2533	5066	7590
16	81	6456	6479	6480	6544	6543	6536	2766	5553	8331	2804	5608	8403
17	85	7114	7140	7142	7208	7207	7200	3048	6120	9182	3089	6177	9257
18	89	7810	7836	7838	7904	7903	7896	3347	6716	10077	3387	6774	10152
19	93	8538	8564	8566	8632	8631	8624	3659	7340	11013	3699	7398	11088
20	97	9298	9324	9326	9392	9391	9384	3984	7992	11990	4025	8049	12065
21	101	10090	10116	10118	10184	10183	10176	4324	8670	13008	4364	8728	13083
22	105	10914	10940	10942	11008	11007	11000	4677	9377	14068	4717	9434	14142
23	109	11770	11796	11798	11864	11863	11856	5044	10110	15168	5084	10168	15243
24	113	12658	12684	12686	12752	12751	12744	5424	10872	16310	5465	10929	16385
25	117	13578	13604	13606	13672	13671	13664	5819	11660	17493	5859	11718	17568
26	121	14530	14556	14558	14624	14623	14616	6227	12476	18717	6267	12534	18792
27	125	15514	15540	15542	15608	15607	15600	6648	13320	19982	6689	13377	20057
28	129	16530	16556	16558	16624	16623	16616	7084	14190	21288	7124	14248	21363
29	133	17578	17604	17606	17672	17671	17664	7533	15089	22636	7573	15146	22710
30	137	18684	18684	18686	18752	18751	18744	7996	16014	24024	8036	16072	24099
31	141	19770	19796	19798	19864	19863	19856	8472	16968	25454	8513	17025	25529
32	145	20914	20940	20942	21008	21007	21000	8963	17948	26925	9003	18006	27000

3.3.6 Finder pattern

There are four types of finder patterns in JAB Code, i.e. Finder Pattern UL, Finder Pattern UR, Finder Pattern LR, Finder Pattern LL, located at the upper left, the upper right, the lower right and lower left corners respectively as illustrated in Figure 5. Each finder pattern contains two square references made up of 3×3 modules, connected with each other by an overlapping module (core module).

The finder patterns have different orientations. The core module of Finder Pattern UL and Finder Pattern UR shall be the lower right module of the upper reference and the upper left module of the lower reference respectively. In Finder Pattern LR and Finder Pattern LL, the core shall be the lower left module of the upper reference and the upper right module of the lower reference.

Each square reference in a finder pattern is constructed of three layers, which are of the same width of one module. The layers in the two references are symmetric with respect to the core module, as illustrated in Figure 5. Finder Pattern UL and Finder Pattern LL consist of blue and yellow layers, while Finder Pattern UR and Finder Pattern LR are made up of green and magenta layers. The color of each layer is different from its adjacent layers and the core module of each type of finder pattern is in a unique color, i.e. Finder Pattern UL has a blue core, Finder Pattern UR has a green core, Finder Pattern LR has a magenta core and Finder Pattern LL has a yellow core.

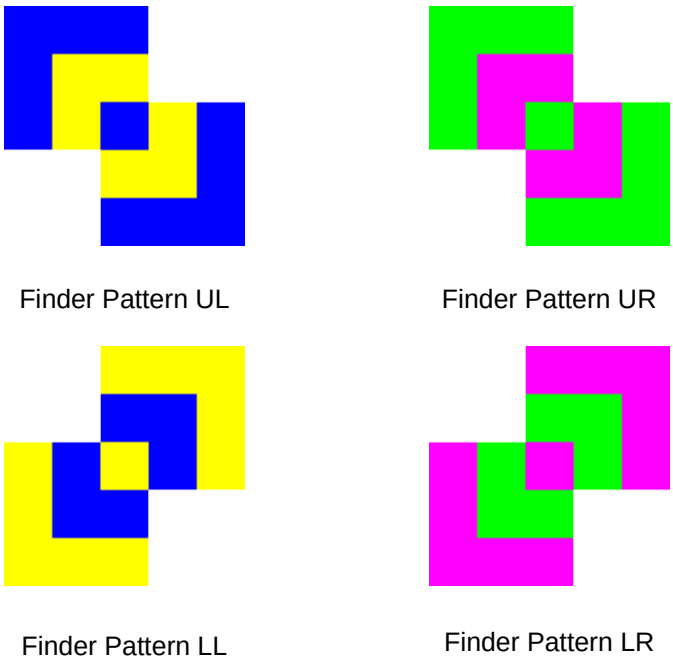

Figure 5: Finder patterns

3.3.7 Alignment pattern

There are four types of alignment patterns in JAB Code, i.e. Alignment Pattern U, Alignment Pattern L, Alignment Pattern X0 and Alignment Pattern X1, as illustrated in Figure 6. Each alignment pattern is constructed of two square references (2×2 modules) consisting of two layers, connected by an overlapping core module. Alignment Pattern U and Alignment Pattern L have white outer layer and black core, while Alignment Pattern X0 and Alignment Pattern X1 have black outer layer and white core.

Alignment Pattern U shall be placed in slave symbols at the same positions as Finder Pattern UL and Finder Pattern UR in master symbols. Alignment Pattern L shall be placed in slave symbols at the same positions as Finder Pattern LR and Finder Pattern LL in master symbols. Alignment Pattern X0 and X1 shall be placed between finder patterns in master symbols and between Alignment Pattern U and Alignment Pattern L in slave symbols.

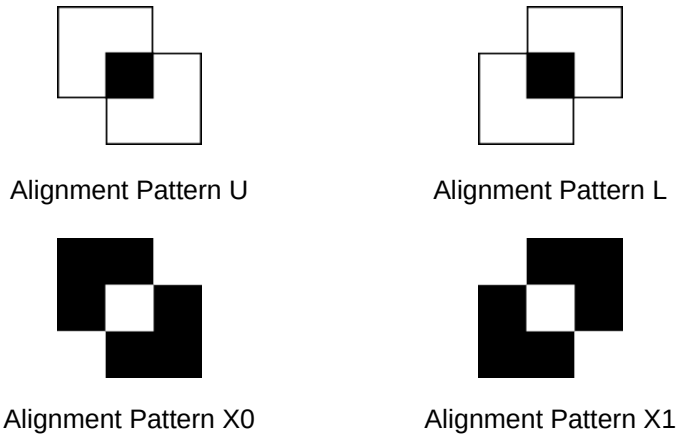

Figure 6: Alignment patterns (left: Alignment Pattern UL, right: Alignment Pattern LR)

Alignment Pattern X0 and Alignment Pattern X1 are present only in JAB Code symbols which has Side-Version 6 or larger. The number of alignment patterns depends on the side-version of each symbol side and the alignment patterns are spaced as evenly as possible. For each side-version, the number of alignment patterns and the column/row coordinates of the core module of each alignment pattern are specified in Table 2, where the coordinate of the top-left module in the symbol is defined as (1, 1).

In either master or slave symbols, alignment patterns shall be placed on the intersections of columns and rows of the coordinates listed in Table 2 except the positions where a finder pattern is located. For example, Table 2 indicates the coordinates 4, 20, 37 and 54 for Side-Version 10. Therefore, in a square master symbol of Side-Version 10, the 12 alignment patterns are centered at (column, row) positions (4, 20), (4, 37), (20, 4), (20, 20), (20, 37), (20, 54), (37, 4), (37, 20), (37, 37), (37, 54), (54, 20), (54, 37).

The first alignment pattern, which is located next to Finder Pattern UL in master symbols or next to Alignment Pattern U at the top-left corner in slave symbols in either horizontal or vertical direction, shall be Alignment X1. At the following placement positions of alignment patterns, Alignment Pattern X0 and Alignment Pattern X1 shall be placed alternatively in both horizontal and vertical directions, as illustrated in Figure 7.

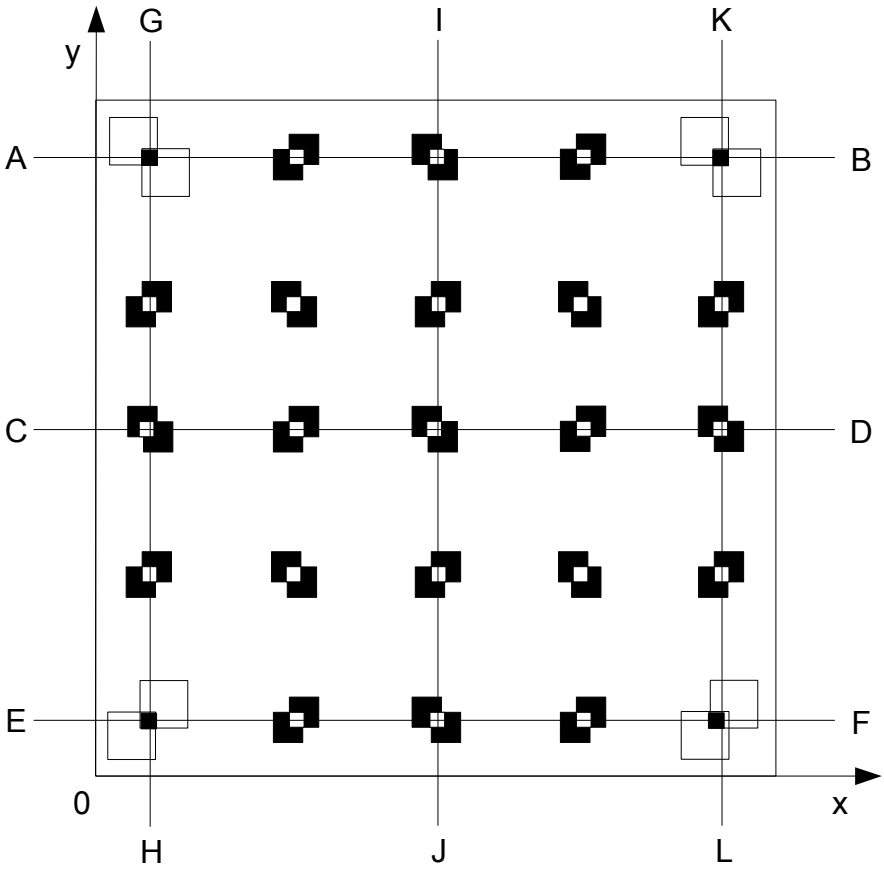

Figure 7: Placement of alignment patterns

Table 2: Positions of alignment patterns

Side-Version	Side size (in modules)	Number of alignment patterns	Column/Row (x/y) coordinates of core module								
1	21	0	-								
2	25	0	-								
3	29	0	-								
4	33	0	-								
5	37	0	-								
6	41	5	4	21	38						
7	45	5	4	23	42						
8	49	5	4	25	46						
9	53	5	4	27	50						
10	57	12	4	20	37	54					
11	61	12	4	22	40	58					
12	65	12	4	23	42	62					
13	69	12	4	24	45	66					
14	73	21	4	20	37	53	70				
15	77	21	4	21	39	56	74				
16	81	21	4	22	41	59	78				
17	85	21	4	23	43	62	82				
18	89	32	4	20	36	53	69	86			
19	93	32	4	21	38	55	72	90			
20	97	32	4	22	40	58	76	94			
21	101	32	4	22	41	60	79	98			
22	105	45	4	20	36	53	69	85	102		
23	109	45	4	21	38	55	72	89	106		
24	113	45	4	21	39	57	74	92	110		
25	117	45	4	22	40	59	77	95	114		
26	121	60	4	20	36	52	69	85	101	118	
27	125	60	4	20	37	54	71	88	105	122	
28	129	60	4	21	38	56	73	91	108	126	
29	133	60	4	22	40	58	76	94	112	130	
30	137	77	4	20	36	52	69	85	101	117	134
31	141	77	4	20	37	54	71	87	104	121	138
32	145	77	4	21	38	55	73	90	107	124	142

3.3.8 Color palette

The color palette provides reference module color values for symbol decoding. As listed in Table 5, JAB Code supports 8 module color modes, allowing minimally 4 and maximally 256 colors to be used in a symbol, hence the color palette has a minimal size of 4 and maximal size of 256, containing up to 256 colors indexed from 0 to 255. Table 3 shows an example of 8-color palette, corresponding to module color mode 2.

However, in either master or slave symbols, the color palettes embedded in the symbol contains only up to 64 colors indexed from 0 to 63. If there are less than or equal to 64 module colors in the symbol, all available colors shall be included in the embedded color palette. If there are more than 64 module colors, the embedded color palette shall contain only 64 selected colors. In addition, for metadata decoding the 8 colors available in module color mode 2 shall be always placed in the first 8 entries in the embedded color palette. Refer to Annex F for guidelines for module color specifications and color palette construction.

In either master or slave symbols, two color palettes shall be placed, which are located in different reserved regions. The module placement of color palettes is specified in Section 3.4.4.

Table 3: Color palette for 8-color symbols

Index	0	1	2	3	4	5	6	7
Color	Black	Blue	Green	Cyan	Red	Magenta	Yellow	White

3.3.9 Metadata

The metadata defines the symbol properties, which provide necessary parameters for symbol decoding, including the number of module colors, the symbol shape and size, the error correction parameters, the masking type and the code structure. The metadata structure and placement is defined in Section 3.4, which is different for master and slave symbols.

The metadata of either master or slave symbols are encoded in specific reserved regions which are defined in Section 3.4.4. The metadata in master symbols are encoded in the modules around the four finder patterns, as illustrated in Figure 8, while the metadata in slave symbols are encoded along the side docked to the host symbol, as illustrated in Figure 9 and Figure 15. Because the length of metadata in both master and slave symbols is variable, the number of modules that are used to encode these metadata also varies.

Metadata are secured by error correction codes. See Section 3.4.3.

3.3.10 Encoded data

Except the modules used for finder patterns, alignment patterns, color palette and metadata, all the remaining modules shall be used to encode data, including the error correction codewords.

The placement of encoded data is specified in Section 4.7.

3.4 Metadata structure

The metadata of a JAB Code symbol defines the number of module colors, the symbol shape, the symbol size, the error correction level, the positions of docked slave symbols and the masking type. The metadata is divided into three parts: Part I, Part II and Part III. The data in the preceding parts determine the encoding modes and the length of variables in the following parts.

Dividing metadata into parts enables customizable metadata requirements in different symbol sizes. To maximize the data capacity in small symbols, less metadata is required, while in large symbols, more metadata can be specified to achieve high encoding flexibility. The total length of master symbol metadata varies from 22 to 40 bits, while it varies from 3 to 27 bits in slave symbols.

Table 4: Metadata structure of master symbol

Metadata					
Part I		**Part II**		**Part III**	
Variable	Length (Bits)	Variable	Length (Bits)	Variable	Length (Bits)
N_c	3	SS	1	V	variable (2-10), depending on SS and VF
		VF	2		
		MSK	3	E	variable (10-16), depending on VF
		SF	1	S	variable (0 or 4), depending on SF
Sum	3	Sum	7	Sum	12 to 30
Total length: variable from 14 to 42 bits					
N_c - module color mode		SS - symbol shape flag		VF - side-version flag	
MSK - masking reference		SF - slave flag		V - symbol side-version	
E - error correction parameters		S - positions of docked slave symbols			

3.4.1 Metadata of master symbol

The metadata structure of master symbol is shown in Table 4. Part I defines the module color mode (N_c) indicating the number of module colors, Part II defines a series of flags, including symbol shape flag (SS), side-version flag (VF), masking reference (MSK) and slave flag (SF), and Part III defines symbol side-version (V), error correction parameters (E) and positions of docked slave symbols (S).

Part I, Part II and Part III are concatenated to form the final metadata in the following order. The bit length of variables in Part I and Part II is fixed and the bit-length of variables in Part III is determined by the flag values in Part II.

Part I	Part II				Part III		
N_c	SS	VF	MSK	SF	V	E	S
3 bits	1 bit	2 bits	3 bits	1 bit	2-10 bits	10-16 bits	0-4 bits

3.4.1.1 Module color mode

The variable N_c in Part I defines the module color mode which indicates the number of available module colors in the symbol, whose possible values are listed in Table 5. It takes 3 bits and defines 8 color modes. N_c shall be encoded in two-color mode which uses only the first module color and the last module color in the color palette, namely black and white in all color modes except mode 001. In mode 001, blue and yellow shall be used. A black/blue module represents a binary zero and a white/yellow module represents a binary one. Except N_c other metadata shall be encoded in multi-color mode using up to 8 colors. In the color modes

containing more than 8 colors, the metadata shall be encoded using the colors available in color mode 2, namely only the first 8 colors in the placed color palette shall be used to encode the metadata. Refer to Annex F for the specification of color palette construction.

Color mode 0 is reserved for future extensions, which can also be used for user-defined color modes.

Table 5: Module color modes in Part I of metadata of master symbol

Variable	Value (binary)	Color mode	Number of module colors
N_c	000	0	reserved
	001	1	4 module colors
	010	2	8 module colors
	011	3	16 module colors
	100	4	32 module colors
	101	5	64 module colors
	110	6	128 module colors
	111	7	256 module colors

3.4.1.2 Symbol shape

The symbol shape flag (SS) in Part II indicates whether the symbol is square or rectangle. As shown in Table 6, a binary zero stands for a square symbol and a binary one for a rectangle symbol.

3.4.1.3 Symbol size

The symbol size is specified by a combination of two variables: the side-version flag VF and the side-version value V. As shown in Table 6, VF indicates the range of the side-version and V gives the version number. The length of V is determined by SS and VF, varying from 2 to 10 as shown in Table 7. For square symbols, only one side-version needs to be encoded in V. The side-version is encoded in the segments defined by VF. For rectangle symbols, the horizontal and vertical side-versions have to be encoded respectively. In addition, the horizontal and vertical side-versions may lie in different segments, therefore full encoding is required for each side-version. Table 7 lists the required bit length for all cases.

3.4.1.4 Error correction level

The error correction level is determined by the variable E, which specifies the error correction parameters. The length of E is determined by the maximal symbol size in each side-version segment indicated by VF, varying from 10 to 16 bits as shown in Table 7. The first half part of E stores the parameter w_c-3 and the second half stores the parameter w_r-4 as defined in Section 4.4.1. The variable length of E enables customizable error correction in corresponding symbol sizes.

3.4.1.5 Positions of docked slaves

The flag SF in Part II indicates whether there are docked slave symbols as shown in Table 6. If there are docked slave symbols, the four bits in variable S in part III represent the positions of the docked slave symbols. The first to the fourth bits of S stand for the upper side, the right side, the lower side and the left side of the master symbol, respectively. A binary one indicates there is a docked slave symbol on the corresponding side, while a binary zero indicates there is no docked symbol.

3.4.1.6 Masking type

The flag MSK in Part II contains the data mask pattern reference from Table 20.

This flag exists only in master symbols and all the slave symbols in a JAB Code share the same mask type as the master symbol.

Table 6: Flag values in Part II of metadata of master symbol

Flag	Value (binary)		Description
SS	0		Square symbol
	1		Rectangle symbol
VF	SS=0	00	Side-Version 1-4
		01	Side-Version 5-8
		10	Side-Version 9-16
		11	Side-Version 17-32
	SS=1	00	Side-Version 1-4
		01	Side-Version 1-8
		10	Side-Version 1-16
		11	Side-Version 1-32
MSK	000 - 111		Mask pattern reference. See Table 20.
SF	0		No docked slave symbol
	1		Positions of docked slaves are specified by S.

Table 7: Variable length in Part III of metadata of master symbol

Flag	Value (binary)		Length (Bits)		
			V	E	
VF	SS=0	00	2	10	S
		01	2	12	
		10	3	14	
		11	4	16	
	SS=1	00	4	10	
		01	6	12	
		10	8	14	
		11	10	16	
SF	0				0
	1				4

3.4.2 Metadata of slave symbol

The metadata structure of slave symbol is shown in Table 8. Part I defines three variables. The first one (SS) indicates whether the symbol shape and size are identical to the host symbol. The second one (SE) indicates whether the slave symbol shares the same error correction level as the host symbol. The third one (SF) indicates whether there are further docked slaves. Part II defines two flags, including symbol side-version (V) and positions of docked slaves (S). Part III contains only one variable, defining error correction level (E).

Part I, Part II and Part III are concatenated to form the final metadata in the following order. The bit length of variables in Part I is fixed and the bit-length of variables in Part II and Part III is determined by the variable values in Part I.

Part I			Part II		Part III
SS	SE	SF	V	S	E
1 bit	1 bit	1 bit	0 or 5 bits	0 or 3 bits	0-16 bits

As shown in Table 8, the metadata in a slave symbol may have a minimal length of bits, which provides additional encoding space with low overhead.

Table 8: Metadata structure of slave symbol

Metadata					
Part I		**Part II**		**Part III**	
Variable	Length (Bits)	Variable	Length (Bits)	Variable	Length (Bits)
SS	1	V	variable (0 or 5), depending on SS	E	variable (0 or 10-16), depending on SE and V
SE	1	S	variable (0 or 3), depending on SF		
SF	1				
Sum	3	Sum	0 to 8	Sum	0 to 16
Total length: variable from 3 to 30 bits					
SS – same shape and size flag		SE – same error correction level flag		SF – slave position flag	
V – symbol side-version		S – positions of docked slave symbols		E – error correction parameters	

3.4.2.1 Symbol shape and size

The slave symbol may have a different shape from the host symbol. However, the docking side, which is docked to the host symbol, must have the same size as the corresponding side of the host symbol. In other words, the slave symbol has only one customizable side.

The symbol shape and size flag (SS) in Part I indicates whether the shape and the size of the slave symbol is identical to the host symbol. As shown in Table 9, SS takes 1 bit and when SS is binary zero, no further data is required to specify the symbol shape and size, therefore the length of V is 0. The corresponding metadata from the host symbol shall be used in the decoding. When SS is binary one, the variable V takes 5 bits to specify the side-version of the customizable side.

3.4.2.2 Error correction level

The slave symbol may either share the same error correction level as the host symbol or use its own error correction level to encode data. The flag SE in Part I indicates whether a different error correction level is specified in the slave symbol, as shown in Table 9.

If a different error correction level is specified, similar to the definition of error correction level in master symbol, the variable E in Part III defines the error correction parameters. The length of E is determined by the larger side version between the side versions in x and y directions as listed in Table 10.

3.4.2.3 Positions of docked slaves

Slave symbols may dock further slave symbols at the three free sides. The flag SF in Part I indicates whether there are further docked slaves. As shown in Table 9, if there are further docked slaves, the variable S in Part III contains 3 bits which represent the positions of docked slave symbols. The first to the third bits stand for the three free sides in the same order as defined in Section 3.4.1.5, skipping the docking side to the host symbol.

Table 9: Flag values in Part I and variable length in Part II and Part III of metadata of slave symbol

Flag	Value	Length (Bits)	Description
SS	0	V: 0	The slave symbol has the identical shape and size to the host symbol.
	1	V: 5	The side of the slave symbol that is not docked to the master has different side-version specified by V.
SE	0	E: 0	The slave symbol shares the same error correction level as the master.
	1	E: 10 - 16	The slave symbol uses a different error correction level specified by E.
SF	0	S: 0	No docked slave symbol.
	1	S: 3	Positions of docked slaves are specified by S.

Table 10: Length of variable E in Part III of metadata of slave symbol

Side-Version	Value	Length of E (Bits)
max(side_version_x, side_version_y)	1-4	10
	5-8	12
	9-16	14
	17-32	16

3.4.3 Metadata error encodation

Each part of the metadata is encoded using the LDPC code separately, which results in a doubled bit length. Refer to Annex B for more details of error correction encoding of each part of the metadata.

The error correction bits for each metadata part shall be calculated as described in Annex B and appended to the metadata bits. Table 11 lists the possible bit length for each metadata part in master and slave symbols. In master symbols, the total length of the final encoded metadata varies from 44 to 80 bits, while in slave symbols, it varies from 6 to 54 bits.

Table 11: Lengths of encoded metadata parts

Symbol type	Metadata part	Original length (Bits)	Encoded length (Bits)	Sum
Master	Part I	3	6	44 - 80
	Part II	7	14	
	Part III	12 - 30	24 - 60	
Slave	Part I	3	6	6 - 54
	Part II	0, 3, 5, 8	0, 6, 10, 16	
	Part III	0, 10, 12, 14, 16	0, 20, 24, 28, 32	

3.4.4 Reserved modules for metadata and color palette

The metadata and the color palette shall be encoded using the modules at predefined positions. If the metadata are encoded using only two colors, which have the lowest encoding capacity per module, up to 80 modules are needed to encode the metadata in master symbols and up to 54 modules are needed in slave symbols. In addition, up to 128 modules are needed to store the two color palettes in either master and slave symbols. Therefore, totally up to 208 modules in master symbols and 182 modules in slave symbols shall be reserved for metadata and color palettes.

The placement order for the maximal metadata and color palettes in master and slave symbols is defined in Figure 8 and Figure 9-12, where the modules with a black index number are reserved for metadata, while the modules with a red number are for color palettes. The metadata placement position in slave symbols depends on the position of the host symbol. As shown in Figure 9-12, the metadata in slave symbols are placed along the side which is docked to the host symbol.

Since the metadata length and the palette size are both variable and the encoding capacity of each module is determined by the number of available module colors, the actually required modules for metadata and color palettes in a specific symbol may vary greatly. As the metadata can be encoded using up to 8 colors, in symbols with more than 8 module colors, the maximal number of required modules for metadata in master symbols are reduced to 31 and in slave symbols to 18.

The string of metadata bits including the error correction bits, from the most significant bit to the least significant bit, shall be encoded into the reserved modules in master or slave symbols following the placement order shown in Figure 8 and Figure 9. Each module contains one or more bits. The remaining bits in the last used module shall be filled with binary zeros. If the number of required modules are less than the reserved ones, in master symbols the unused modules shall be used for color palette placement or data encoding. In slave symbols the unused modules shall be used for data encoding.

In either master or slave symbols, two color palettes shall be placed. In master symbols, the first 16 colors shall be placed in the modules next to the finder patterns, numbered from 0 to 15, and the other 48 colors, when available, shall be placed in the regions around finder patterns following the last used metadata module, as shown in Figure 8. The position of the last used metadata module may be located in any of the four metadata regions around finder patterns, depending on the actual metadata length. For example, if the metadata encoding ends at the reserved module with index number 50, the color palette placement shall start from the reserved module with index number 51 with the 16^{th} color in the color palette. Each color in the palette shall be placed twice in the two diagonally opposite regions.

In salve symbols, the color palette shall be placed in the reserved modules close to Alignment Patter U and Alignment Pattern L as shown in Figure 9. The first 8 colors in the color palette shall be placed in the reserved modules close to the metadata (denoted as region A) and in the diagonally opposite reserved modules (denoted as region B), following the placement order indicated by the index numbers. If there are more than 8 module colors, the colors in the first half of the palette shall be placed in region A and region B,

while the colors in the second half of the palette shall be placed in the other two regions, following the placement order indicated by the index numbers in each region, respectively.

In case there are less available module colors than the reserved modules, the unused modules shall be used for data encoding.

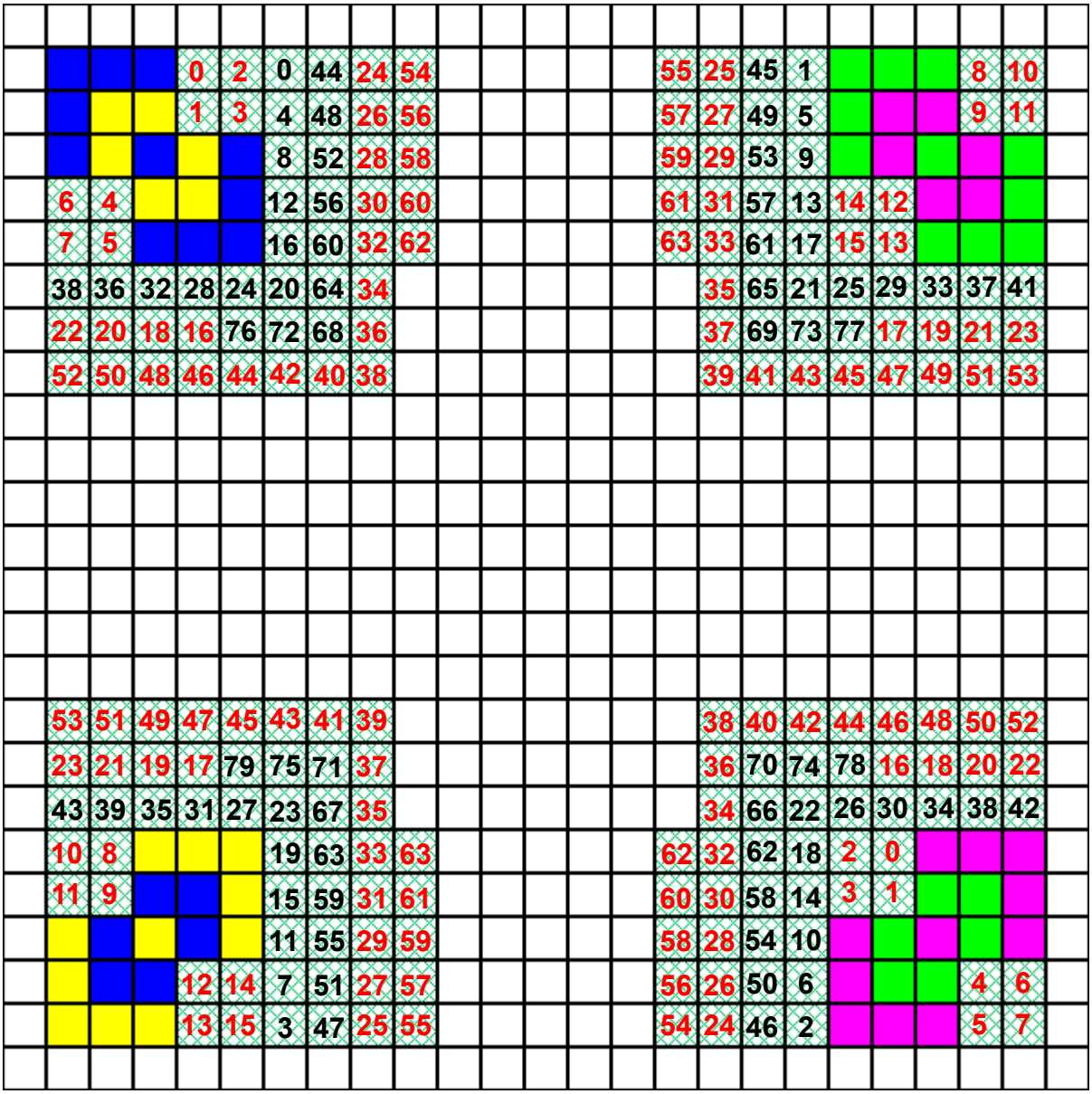

Figure 8: Metadata and color palette module placement in master symbol

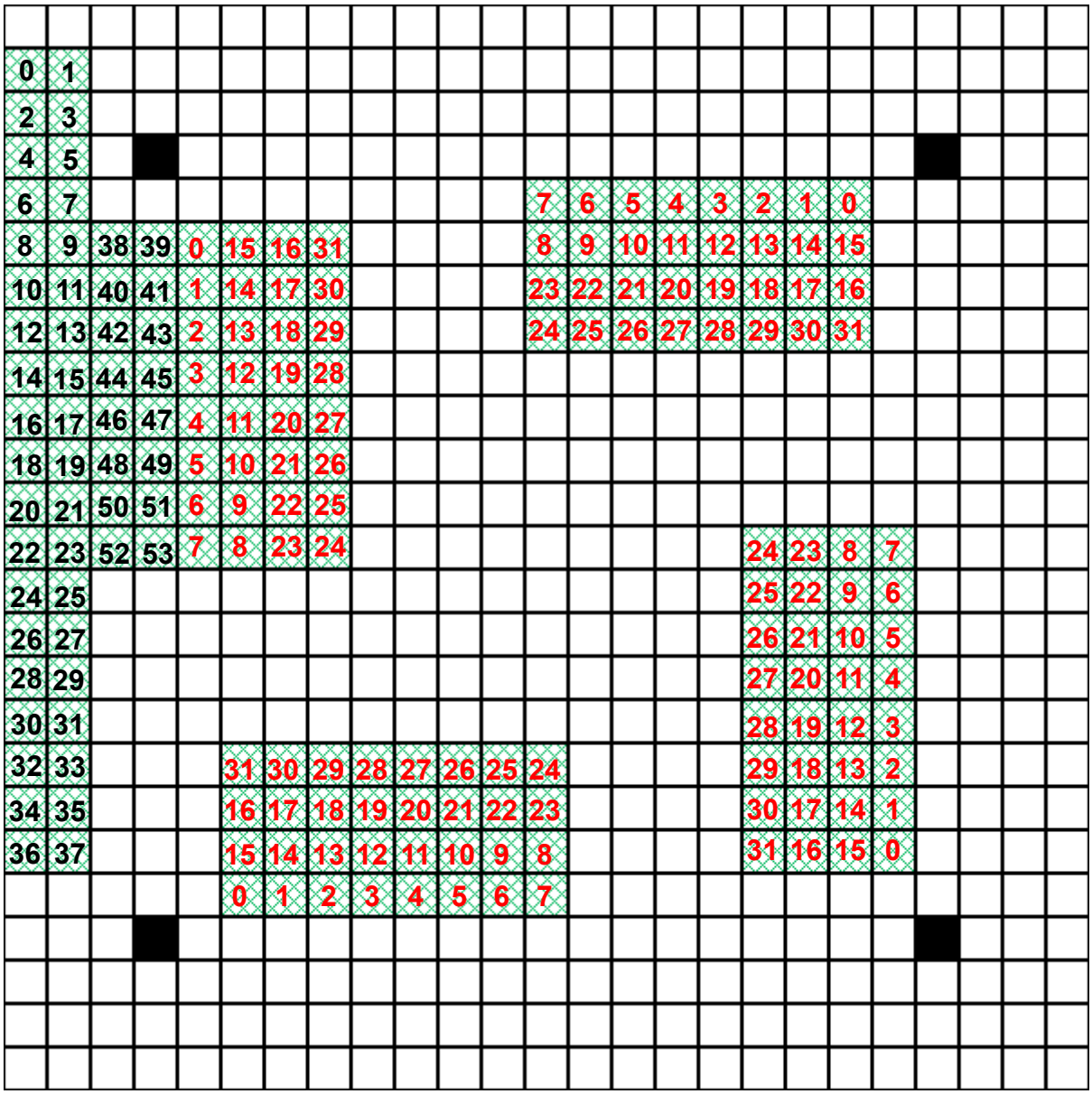

Figure 9: Metadata and color palette module placement in slave symbols with left side docked to the host symbol

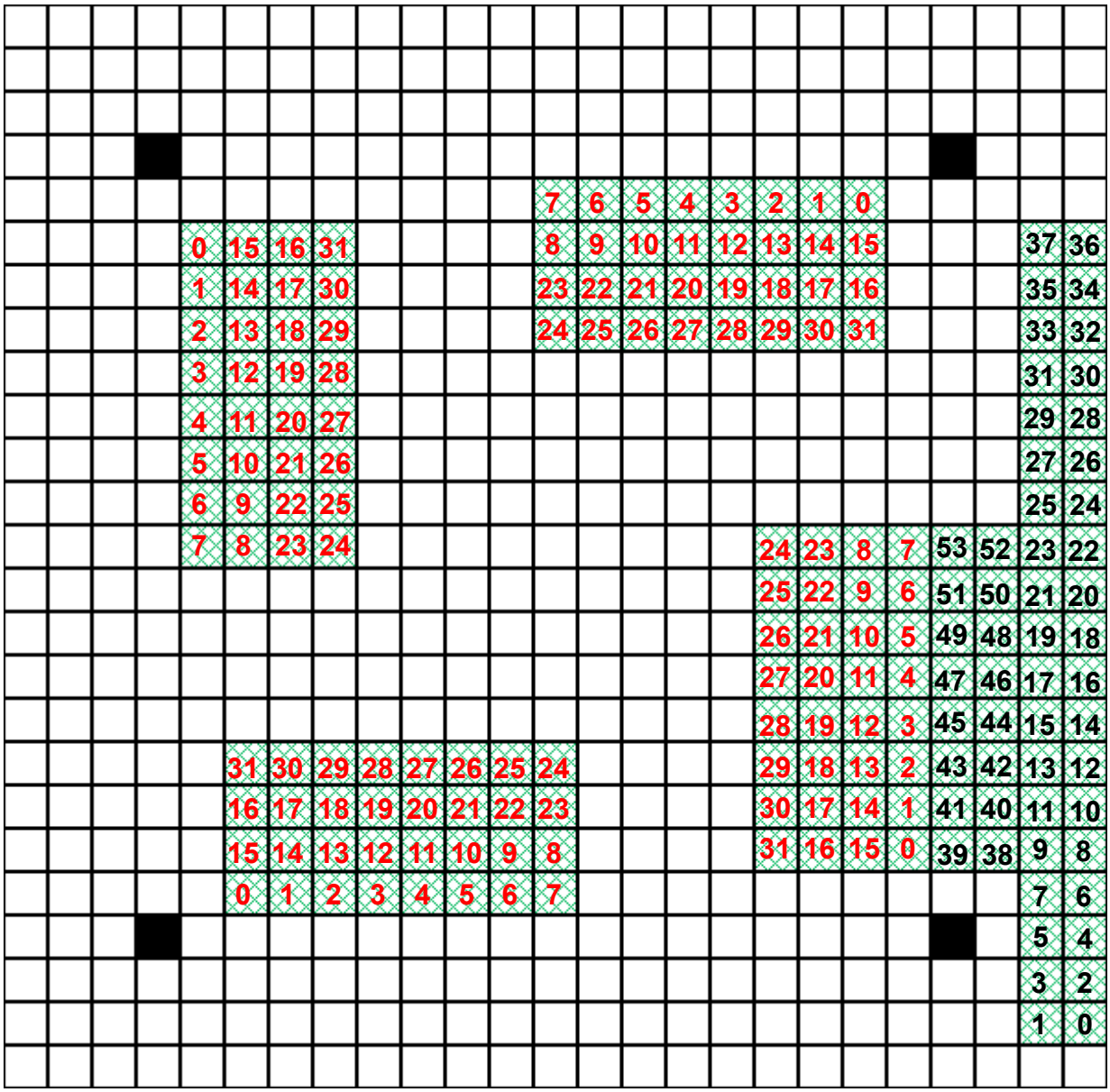

Figure 10: Metadata and color palette module placement in slave symbol with right side docked to the host symbol

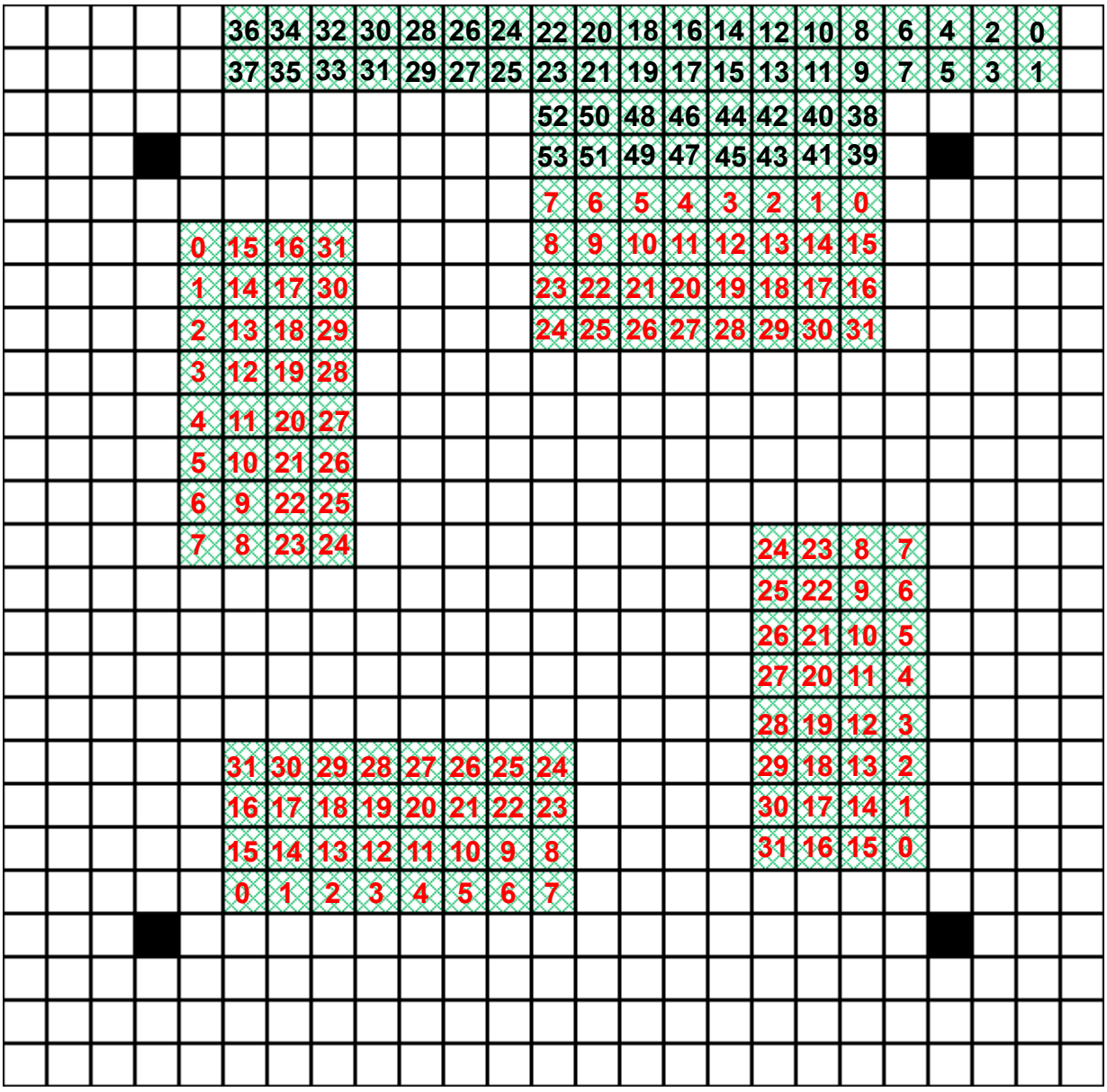

Figure 11: Metadata and color palette module placement in slave symbol with top side docked to the host symbol

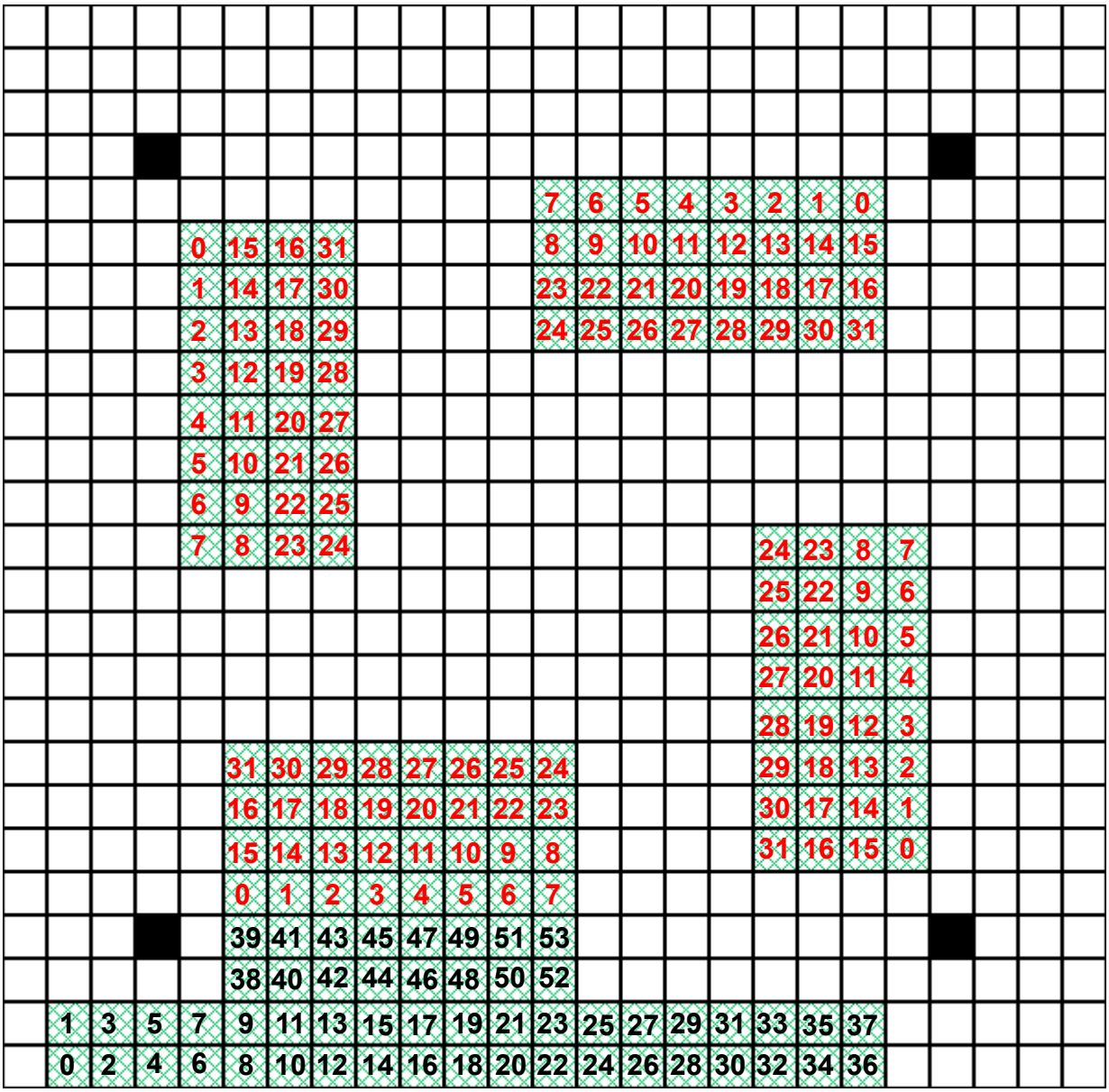

Figure 12: Metadata and color palette module placement in slave symbol with bottom side docked to the host symbol

3.5 Symbol Cascading

3.5.1 Symbol docking rules

JAB Code may have arbitrary forms by cascading master and slave symbols in horizontal and vertical directions. A JAB Code shall contain one and only one master symbol and may optionally have multiple slave symbols. Slave symbols shall be docked to the master symbol or the other slave symbols.

The master and slave symbols in a JAB Code may be of different shapes, square or rectangle, namely, they may have different Side-Versions for horizontal and vertical sides. Nevertheless, the docking side between two adjacent symbols must share the same Side-Version. It is recommended that the master symbol in a JAB Code possesses the largest symbol size.

Figure 13, Figure 14 and Figure 15 illustrate three example of JAB Code with cascaded symbols. The master and slaves symbols in Figure 13 have the same shape and size. Figure 14 shows a code containing master and slave symbols of different shapes and sizes. Figure 15 gives an example code with recursive symbol docking, in which the slave symbol docked to the master has a further docked slave symbol.

3.5.2 Symbol decoding order

The JAB Code decoding shall always start from the master symbol. If more than one slave symbols are docked to the master symbol, the decoding shall follow the order: top-bottom-left-right. If the slave symbols docked to the master have further docked slave symbols, the decoding shall follow the order defined below.

1. The slave symbols that are directly docked to the master symbol shall be first decoded according to the top-bottom-left-right order, which are denoted as the first layer.
2. According to the top-bottom-left-right order, the slave symbols in the first layer shall be checked in turn. If there are further docked slave symbols, they shall be decoded according to the top-bottom-left-right order. These slave symbols are denoted as the second layer.
3. Apply the same order to the other docked slave symbols in further layers until all the slave symbols are decoded.

According to the decoding order defined by the rules above, the positions of the first 60 slave symbols are defined in Figure 16. The slave symbols with smaller position numbers shall be decoded first.

Figure 17 illustrates an example of the decoding order of cascaded symbols. The symbols indices indicate the decoding order. The symbols with a smaller index number shall be decoded first.

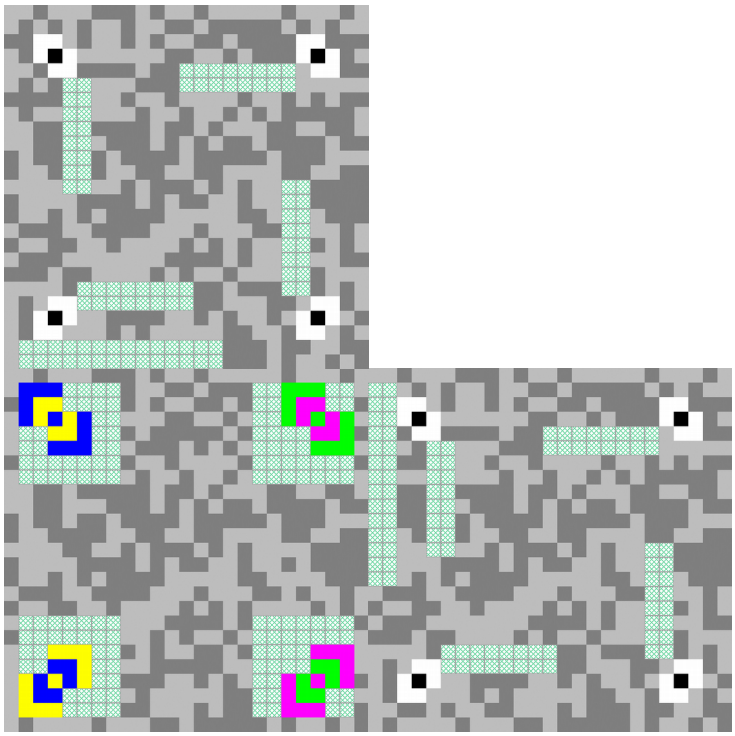

Figure 13: JAB Code with one square master and two square slave symbols

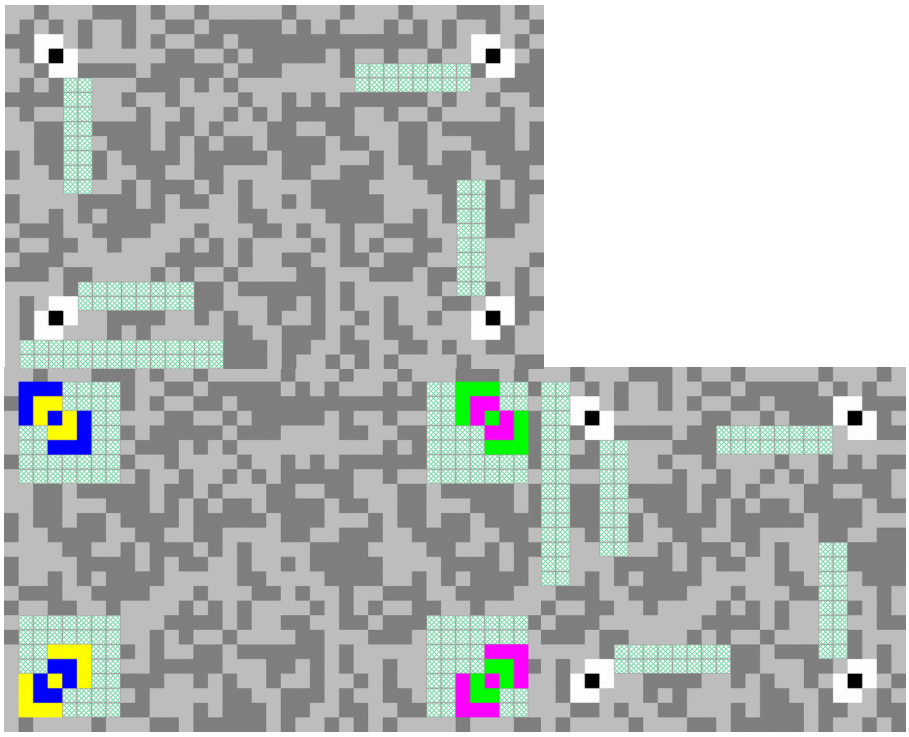

Figure 14: JAB Code with one rectangle master symbol and two square slave symbols

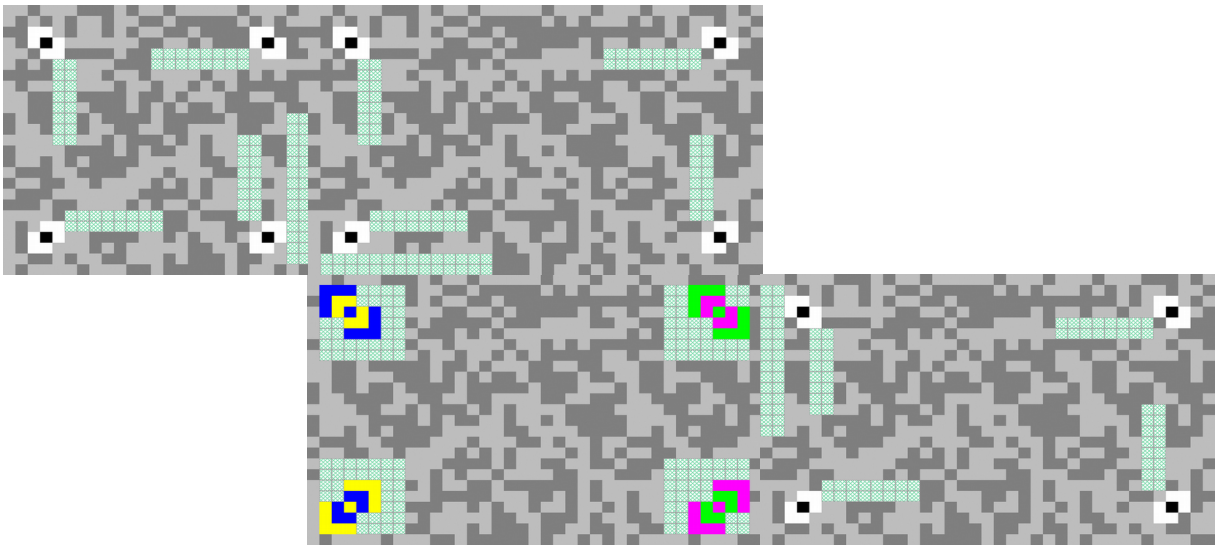

Figure 15: JAB Code with one rectangle master symbol and multiple square and rectangle slave symbols

					41					
				42	25	43				
			44	26	13	27	45			
		46	28	14	5	15	29	47		
	48	30	16	6	1	7	17	31	49	
59	39	23	11	3	0	4	12	24	40	60
	57	37	21	9	2	10	22	38	58	
		55	35	19	8	20	36	56		
			53	33	18	34	54			
				51	32	52				
					50					

Figure 16: Decoding order of cascaded master and slave symbols

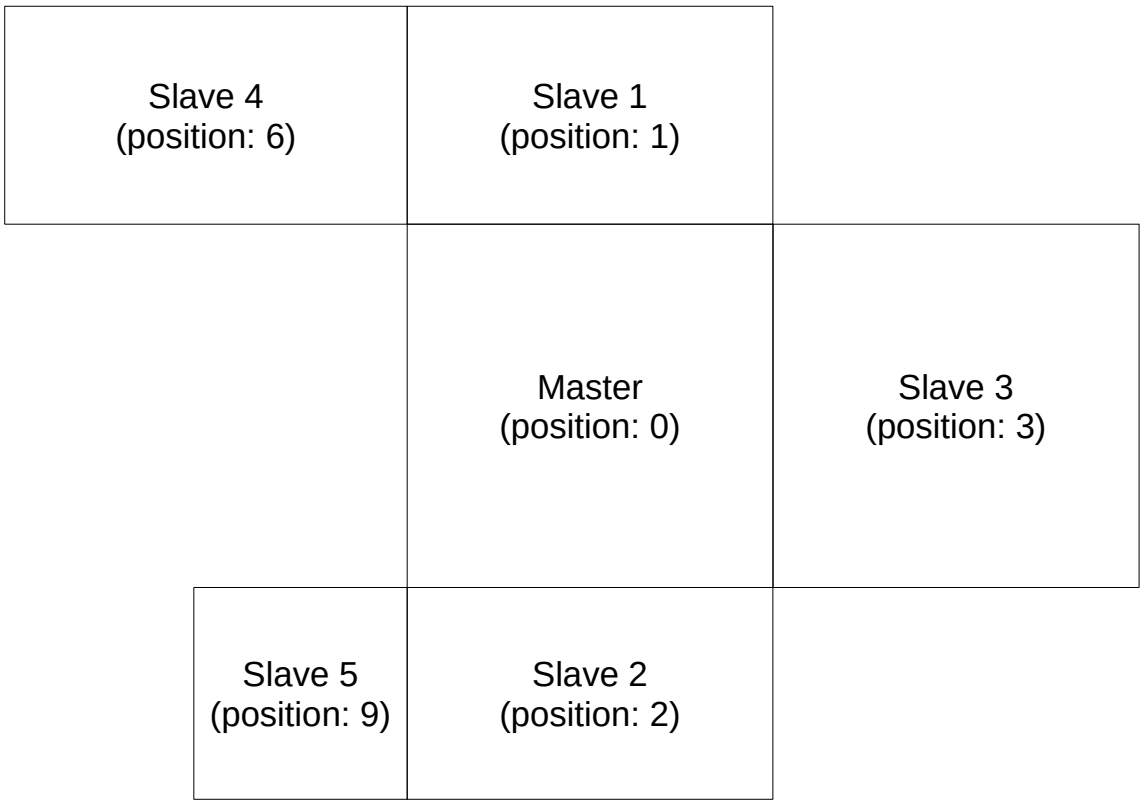

Figure 17: Example of decoding order of cascaded master and slave symbols

4 Symbol generation

4.1 Encode procedure overview

The following steps are required to convert input data to a JAB Code symbol.

(1) Data analysis

Analyze the input data to identify the most efficient modes to encode the characters. JAB Code supports seven default encoding modes to convert input data into a binary string, which can be switched to each other as needed by intermediary mode switch. Further character sets are enabled by supporting Extended Channel Interpretation (ECI) and FNC1 encoding. See Section 4.3.

(2) Data encoding

Convert the data characters into a binary stream using selected encoding modes as specified in Section 4.3.

(3) Error correction coding

Encode the binary stream using systematic LDPC for error correction and append the parity data to the end of the source binary stream, as specified in Section 4.4.

(4) Data interleaving

Interleave the encoded data in each symbol as specified in Section 4.5 and add padding bits if necessary.

(5) Metadata module reservation

Based on the code parameters, including the number of module colors, symbol shape, symbol size, error correction level and code structure, calculate the actual metadata size and reserve the modules which are required to accommodate the metadata, following the sequence specified in Section 3.4.4.

(6) Data module placement

First, place the finder patterns, the alignment patterns and the color palettes in the matrix. Second, place the data modules (including the modules for error correction bits) in the remaining area of the matrix (skipping the reserved modules for metadata) according to the interleaved data stream as specified in Section 4.7.

(7) Data masking

Apply every available data mask pattern on the data modules and evaluate the masking results. Select the most appropriate masking pattern which results in the most balanced module color distribution and minimizes the occurrence of undesirable patterns, as specified in Section 4.8.

(8) Metadata generation and module placement

According to the code parameters, including the number of module colors, symbol shape, symbol size, error correction level, masking type and code structure, generate the metadata information for each symbol as defined in Section 3.4.1 and 3.4.2 and encode the generated metadata as specified in Section 3.4.3. Finally, place the encoded metadata information into the reserved modules, following the sequence specified in Section 3.4.4.

4.2 Data analysis

Analyze the character types of the input data and choose appropriate encoding modes in order to encode the input data with the shortest bit stream. Annex D presents an algorithm for selecting the most appropriate encoding modes to minimize the bit stream length.

Table 12: JAB Code character encoding modes

	Upper		Lower		Numeric		Punct		Mixed			Alphanumeric				
Value	Char	ISO	Char	ISO	Char	ISO	Char	ISO	Char	ISO	Value	Char	ISO	Value	Char	ISO
0	SP	32	SP	32	SP	32	!	33	#	35	0	SP	32	32	V	86
1	A	65	a	97	0	48	"	34	*	42	1	0	48	33	W	87
2	B	66	b	98	1	49	$	36	+	43	2	1	49	34	X	88
3	C	67	c	99	2	50	%	37	<	60	3	2	50	35	Y	89
4	D	68	d	100	3	51	&	38	=	61	4	3	51	36	Z	90
5	E	69	e	101	4	52	'	39	>	62	5	4	52	37	a	97
6	F	70	f	102	5	53	(	40	[	91	6	5	53	38	b	98
7	G	71	g	103	6	54	)	41	\	92	7	6	54	39	c	99
8	H	72	h	104	7	55	,	44	]	93	8	7	55	40	d	100
9	I	73	i	105	8	56	-	45	^	94	9	8	56	41	e	101
10	J	74	j	106	9	57	.	46	_	95	10	9	57	42	f	102
11	K	75	k	107	,	44	/	47	`	96	11	A	65	43	g	103
12	L	76	l	108	.	46	:	58	{	123	12	B	66	44	h	104
13	M	77	m	109	P/S		;	59	|	124	13	C	67	45	i	105
14	N	78	n	110	U/L		?	63	}	125	14	D	68	46	j	106
15	O	79	o	111	MS		@	64	~	126	15	E	69	47	k	107
16	P	80	p	112					HT	9	16	F	70	48	l	108
17	Q	81	q	113					LF	10	17	G	71	49	m	109
18	R	82	r	114					CR	13	18	H	72	50	n	110
19	S	83	s	115					CR LF	10, 13	19	I	73	51	o	111
20	T	84	t	116					, SP	44, 32	20	J	74	52	p	112
21	U	85	u	117					. SP	46, 32	21	K	75	53	q	113
22	V	86	v	118					: SP	58, 32	22	L	76	54	r	114
23	W	87	w	119					€	164	23	M	77	55	s	115
24	X	88	x	120					§	167	24	N	78	56	t	116
25	Y	89	y	121					Ä	196	25	O	79	57	u	117
26	Z	90	z	122					Ö	214	26	P	80	58	v	118
27	P/S		P/S						Ü	220	27	Q	81	59	w	119
28	L/L		U/S						ß	223	28	R	82	60	x	120
29	N/L		N/L						ä	228	29	S	83	61	y	121
30	A/L		A/L						ö	246	30	T	84	62	z	122
31	MS		MS						ü	252	31	U	85	63	MS	

4.3 Encoding modes

There are nine encoding modes in JAB Code: uppercase mode, lowercase mode, numeric mode, punctuation mode, mixed mode, alphanumeric mode, byte mode, ECI mode and FNC1 mode. The first six character encoding modes are defined in Table 12. In JAB Code, the default interpretation for values 0 to 31 (control characters) is in accordance with the U.S. national version of ISO 646 and for values 32 to 255 in accordance with the ISO/IEC 8859-15 character set, corresponding to ECI 000017.

Encoding can switched from one mode to another mode as often as needed in two ways: shift and latch. Shift indicates a temporary switch only for the next character, e.g. shift to punctuation mode (P/S) and shift to uppercase mode (U/S), while latch indicates a permanent switch for the following characters until another switch is encountered, e.g. latch to lowercase mode (L/L), latch to numeric mode (N/L).

4.3.1 Uppercase mode

Uppercase mode encodes 27 characters, including 26 capital letters (A-Z) and SPACE, at 5 bits per character. Each character is assigned a character value from 0 to 26 according to Table 12.

The remaining five values are used for mode switch, as defined in Table 13. The first four values from 27 to 30 define direct switch to punctuation, lowercase, numeric and alphanumeric modes. The last value 31 defines an extension of more switches, which indicates four more mode switches by appending two bits at the end of $(11111)_{BIN}$, for example, $(1111100)_{BIN}$ indicates shifting into byte mode and $(1111101)_{BIN}$ indicates shifting into mixed mode.

Data encoding starts by default in uppercase mode.

Table 13: Mode switch in uppercase mode

	Value	**Char**	**Mode Switch**		
Direct	27	P/S	shift to punctuation mode for the next character		
	28	L/L	latch to lowercase mode for the following characters		
	29	N/L	latch to numeric mode for the following characters		
	30	A/L	latch to Alphanumeric mode for the following characters		
Extended	31	MS	more switches by appending bits	00	shift to byte mode
				01	shift to mixed mode
				10	latch to ECI mode
				11	latch to FNC1 mode

4.3.2 Lowercase mode

Lowercase mode encodes 27 characters, including 26 small letters (a-z) and SPACE, at 5 bits per character. Each character is assigned a character value from 0 to 26 according to Table 12.

The remaining five values are used for mode switch, as defined in Table 14. The first four values from 27 to 30 define direct switch to punctuation, uppercase, numeric and alphanumeric modes. The last value 31 defines an extension of more switches, which indicates three more mode switches and an end-of-message flag (EOM) by appending two bits at the end of $(11111)_{BIN}$, for example, $(1111100)_{BIN}$ indicates shifting into byte mode and $(1111111)_{BIN}$ indicates the end of message bits.

Table 14: Mode switch in lowercase mode

	Value	Char	Mode Switch		
Direct	27	P/S	shift to punctuation mode for the next character		
	28	U/S	shift to uppercase mode for the next character		
	29	N/L	latch to numeric mode for the following characters		
	30	A/L	latch to Alphanumeric mode for the following characters		
Extended	31	MS	more switches by appending bits	00	shift to byte mode
				01	shift to mixed mode
				10	latch to uppercase mode
				11	End of message (EOM)

4.3.3 Numeric mode

Numeric mode encodes 13 characters, including 10 digits (0-9), SPACE and two punctuation marks, at 4 bits per character. Each character is assigned a character value from 0 to 12 according to Table 12.

The remaining three values are used for mode switch, as defined in Table 15. The first two values from 13 to 14 define direct switch to punctuation and uppercase modes. The last value 15 defines an extension of more switches, which indicates four more mode switches by appending two bits at the end of $(1111)_{BIN}$, for example, $(111100)_{BIN}$ indicates shifting into byte mode and $(111101)_{BIN}$ indicates shifting into mixed mode.

Table 15: Mode switch in numeric mode

	Value	Char	Mode Switch		
Direct	13	P/S	shift to punctuation mode for the next character		
	14	U/L	latch to uppercase mode for the following characters		
Extended	15	MS	more switches by appending bits	00	shift to byte mode
				01	shift to mixed mode
				10	shift to uppercase mode
				11	latch to lowercase mode

4.3.4 Punctuation mode

Punctuation mode encodes 16 commonly used punctuation characters, at 4 bits per character. Each character is assigned a character value from 0 to 15 according to Table 12.

Punctuation mode has a fixed run-length of one character, after which encoding reverts to the mode from which punctuation mode was invoked.

4.3.5 Mixed mode

Mixed mode encodes Germanic umlauts, more punctuation characters and other marks, control characters and combinations, at 5 bits per character. Each character is assigned a character value from 0 to 31 according to Table 12.

Like punctuation mode, mixed mode also has a fixed run-length of one character, after which encoding reverts to the mode from which punctuation mode was invoked.

4.3.6 Alphanumeric mode

Alphanumeric mode encodes 63 characters, including 26 capital letters (A-Z), 26 small letters (a-z), 10 digits (0-9) and SPACE, at 6 bits per character. Each character is assigned a character value from 0 to 62 according to Table 12.

The remaining value 63 is used for mode switch, as defined in Table 16, which defines an extension of four mode switches by appending two bits at the end of $(111111)_{BIN}$, for example, $(11111100)_{BIN}$ indicates shifting into byte mode and $(11111110)_{BIN}$ indicates shifting into punctuation mode.

Table 16: Mode switch in alphanumeric mode

	Value	**Char**	**Mode Switch**		
Extended	15	MS	more switches by appending bits	00	shift to byte mode
				01	shift to mixed mode
				10	shift to punctuation mode
				11	latch to uppercase mode

4.3.7 Byte mode

Byte mode encodes any 8-bit characters at 8 bits per character. Byte mode starts with a 4-bit binary value, which, if non-zero, encodes the number of bytes (1-15) that follow, but if zero then the next 13 bits encode the number of bytes less 15. Thus, byte mode can encode any ASCII characters that are not included in Table 12 and long strings of binary data, possibly filling the whole symbol. At the end of the byte string, encoding returns to the mode from which byte mode was invoked.

4.3.8 Extended Channel Interpretation (ECI) mode

ECI mode enables data interpretation different from the default character set. ECI mode starts with a 6-digit ECI assignment number which is encoded as an 8-bit, 16-bit or 22-bit binary string, as defined in Table 17. The preceding indicating bits determines the length of the binary string.

- If it begins with a 0 bit, it contains 8 bits.
- If it begins with “10”, it contains 16 bits.
- If it begins with “11”, it contains 22 bits.

In each case, the following bits after the indicating bits are the binary representation of the ECI assignment number, after which encoding returns to the mode from which ECI mode was invoked.

In the input data to be encoded, the ECI assignment number is represented as a backslash character, $(5C)_{HEX}$, followed by a 6-digit number, i.e. "\nnnnnn". When ECI protocol applies, if the input data contains a backslash character, it shall be doubled as two $(5C)_{HEX}$ characters.

Data in an ECI sequence shall be handled as 8-bit byte values, which can be encoded using any encoding modes irrespective of their significance. For example, a sequence of bytes in the range $(30)_{HEX}$ to $(57)_{HEX}$ would be most efficiently encoded in the Numeric mode even if the sequence might not actually represent numeric data.

Any ECI invoked shall apply until the end of the encoded data, or until another ECI is encountered.

Table 17: Encoding ECI assignment number

ECI Assignment Number	Encoded value
000000 to 000127	**0**bbbbbbb
000000 to 016383	**10**bbbbbb bbbbbbbb
000000 to 999999	**11**bbbbbb bbbbbbbb bbbbbb
where b...b is the binary value of the ECI assignment number	

4.3.9 FNC1 mode

FNC1 mode is used for messages containing specific data formats. When FNC1 precedes the first message character, it designates data formatted in accordance with GS1 General Specifications. When FNC1 immediately follows a single upper or lower case letter or two digits at the beginning of the message, it designates data formatted in accordance with a specific industry application previously agreed with AIM Inc., identified by the preceding data. FNC1 mode applies to the entire symbol. When FNC1 occurs at any other location in the data stream, it serves as a field separator and causes an ASCII 29 (<GS>) to be inserted in its place in the output data string.

4.4 Error correction

The error correction coding is performed by the LDPC code and operates on binary data. After encoding the message data to a binary stream as specified in Section 4.3, the LDPC code adds check bits to the binary stream in order to enable the symbol to to remain decodable in case of damage. The LDPC code can correct codewords with misdecoded bits caused by damage. The error correction level shall be selectable between 0 and 10. The default error correction level shall be 6.

Table 18 shows the recovery capability of the bit errors in more than 95% of cases. More errors can be detected by the error correction code but with a probability less than 95%. The error correction level shall be determined by the application requirements and expected symbol quality. The higher error recovery capacity is achieved at the cost of larger stream size, leading to larger symbol size. The increase of the stream size is indicated by the code rate R, which is defined as $R=P_n/P_g$.

Table 18: The approximated amount of bit error recovery capacity in %

Level	Recovery capacity in %	Code rate R
0	3	0.67
1	4	0.63
2	5	0.57
3	6	0.55
4	7	0.50
5	8	0.43
6	9	0.34
7	10	0.25
8	11	0.20
9	12	0.17
10	14	0.14

4.4.1 Selectable error correction levels

In JAB Code, 11 error correction levels are defined as listed in Table 18. Based on the input data and the symbol capacity, the best combination of the two parameters w_c and w_r shall be determined. The value of w_c shall be an integer between 3 and 8, w_r an integer between 4 and 9.

Given the symbol capacity and the net payload, the relation $w_r/w_c=C/(C-P_n)$ holds. The parameters, w_c and w_r, shall be determined by the code rate $R= 1- w_c /w_r$ which is closest to but smaller than the corresponding code rate of the specified error correction level.

With these two parameters w_c and w_r the number of error correction bits and the gross payload are specified by

$$K=\lfloor\frac{C\times w_c}{w_r}\rfloor \quad \text{and} \quad P_g=P_n+K \quad .$$

With Pg and K the size of the matrix H is determined and the matrix H shall be generated as defined in Section 4.4.3.

4.4.2 Stuffing Bits

The number of stuffing bits S_{Bit} is $S_{bit}=C-P_g$. The stuffing bits are a sequence consisting of alternate '0's and '1's starting with '0', filling up the unused capacity at the end of the error correction stream.

4.4.3 Generating the error correction stream

The error correction stream c is generated by multiplying the message m with the generator matrix G in the GF(2) with $c=m\otimes G$. There are four steps to obtain the generator matrix:

Step 1: Construct a matrix A_0 with K/w_c rows and P_g columns:

$$A_0 = \begin{bmatrix} \underbrace{1111}_{w_r} & ...0... & ...0... \\ ...0... & \underbrace{1111}_{w_r} & ...0... \\ ...0... & ...0... & \underbrace{1111}_{w_r} \end{bmatrix}$$

Step 2: Form the matrix A with K rows and P_g columns by stacking w_c permutations:

$$A = \begin{bmatrix} \pi_1(A_0) \\ \pi_2(A_0) \\ \pi_3(A_0) \\ . \\ . \\ . \end{bmatrix}$$

where π is the permutation. The permutation is performed by using the random number generator R(seed, A_0) as defined in Annex E with the initial seed 785465 for the message data and the seed 38545 for the metadata with π_1=R(seed, A_0), π_2=R(seed, π_1), π_3=R(seed, π_2), ...

Step 3: Generate the matrix H by using the Gauss-Jordan elimination for the matrix A to obtain $H(C^T|I) \in (K \, x \, P_g)$.

Step 4: The generator matrix is created by $G(I|C) \in (P_n \, x \, P_g)$.

An example is given in Annex A.1.

4.5 Data interleaving

The final sequence of encoded data shall be constructed following the steps below.

1. Calculate the remaining capacity in the selected symbol, which is equal to C-P_g.
2. Fill the unused capacity with stuffing bits to get the final sequence. The stuffing bits shall be a binary string containing alternating 0 and 1 and starting with 0, e.g. 0101010...
3. Interleave all bits in the final sequence using the random permutation algorithm with an initial seed of 226759 as defined in Annex E.

4.6 Metadata module reservation

In JAB Code, the metadata in master and slave symbols are variable-length data. The actual metadata length and the number of modules required to accommodate the metadata are determined by the following five code parameters which shall be specified by user input:

- the number of module colors
- symbol shape
- symbol size
- error correction level
- code structure

If the number of module colors is not specified by user input, the color mode “010” (8 module colors) shall be used.

If no symbol shape is specified by user input, square symbols shall be used.

If no symbol size is specified by the user input, the smallest square symbol, which will accommodate the encoded data with the given error correction level, shall be used. However, in case of rectangle symbols, the horizontal and vertical symbol sizes must be specified by user input, respectively.

If error correction level is not specified by user input, the error correction level, which will achieve the highest error correction rate with the input data in the symbol in use, shall be used.

If neither symbol size nor error correction level is specified by user input, the smallest square symbol, which will accommodate the input data encoded with the default error correction level, shall be used.

If no code structure is specified by user input, one single master symbol without docking slave symbols shall be used.

Based on the length of the final metadata information and the error correction encodation, the number of required metadata modules shall be determined and reserved in each symbol, following the placement order defined in Section 3.4.4.

4.7 Data module encodation and placement

Before placing the modules for data message, the modules for finder patterns, alignment patterns and color palettes shall be first placed in the matrix.

The bits of encoded data message, including the error correction bits, are graphically encoded using each color module in the color palette to represent $\log_2(N_c)$ bits. The $\log_2(N_c)$ binary bits are encoded using the index value of module color in the palette. For example, if there are eight module colors in a symbol, i.e. $\log_2(N_c)=3$, the first module color in the color palette represents 000, the second one represents 001, and so forth, as defined in Table 19.

The interleaved final sequence of encoded data shall be placed in the remaining modules, starting from the most upper left available module, running downwards from left to right, to the most lower right available module, skipping over the modules occupied by finder patterns, alignment patterns, metadata and color palettes, as shown in Figure 18. The data module placement in slave symbols follows the same way as in master symbols.

Table 19: Bit encoding using eight module colors

Module color	Color index	Binary bits
black	0	000
magenta	1	001
yellow	2	010
cyan	3	011
red	4	100
green	5	101
blue	6	110
white	7	111

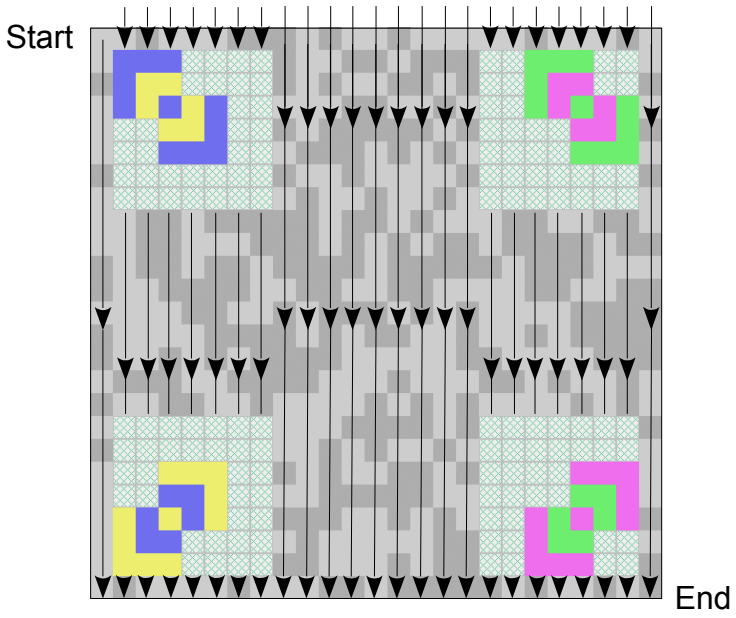

Figure 18: Data module placement

4.8 Data masking

For reliable JAB Code reading, the distribution of color modules shall preferably meet the following two conditions:

1. The color modules should be arranged in a well-balanced manner in the symbol.
2. The occurrences of patterns similar to finder patterns and alignment patterns in other regions of the symbol should be avoided as much as possible.

In order to meet the above conditions, data masking shall be applied as following.

1. Data masking is only applied to data modules, not to modules for finder pattern, alignment patterns, metadata and color palettes.
2. Apply each data mask pattern to data modules in turn. The masking result of each module is calculated by the XOR operation between the module color and the mask pattern color, as defined in Section 4.8.1.
3. Evaluate the masking results by charging penalties for undesirable features.
4. Select the data mask pattern with the lowest penalty point score.

4.8.1 Data mask patterns

JAB Code has eight data mask patterns as listed in Table 20. The binary reference values are used in metadata to identify the masking type. Each data pattern covers only the modules in the data encoding region in a symbol, excluding modules for finder pattern, alignment patterns, metadata and color palettes. The color of each module in a data mask pattern is determined by the pattern module color generators as defined in Table 20. The result of each generator indicates the index in the color palette of the symbol. In the generators, x refers to the horizontal position of the module and y refers to its vertical position, with (x, y) = (0, 0) for the upper left module in the symbol.

The data masking is applied to a data module through the bitwise XOR operation between the color index of the data module and the color index of the corresponding module in the mask pattern. For example, in case of N_c=3, if the module has the color index of 5, $(101)_{BIN}$, and the corresponding module in the mask pattern has the color index 3, $(011)_{BIN}$, then the resulting module has the color index 6, $(110)_{BIN}$.

Table 20: Data mask pattern generation conditions

Data mask pattern reference	Pattern module color generator
000	$(x+y) \bmod 2^{Nc+1}$
001	$x \bmod 2^{Nc+1}$
010	$y \bmod 2^{Nc+1}$
011	$((x \text{ div } 2) + (y \text{ div } 3)) \bmod 2^{Nc+1}$
100	$((x \text{ div } 3) + (y \text{ div } 2)) \bmod 2^{Nc+1}$
101	$((x+y) \text{ div } 2 + (x+y) \text{ div } 3) \bmod 2^{Nc+1}$
110	$(((x \times x \times y) \bmod 7) + ((2 \times x \times x + 2 \times y) \bmod 19)) \bmod 2^{Nc+1}$
111	$(((x \times y \times y) \bmod 5) + ((2 \times x + y \times y) \bmod 13)) \bmod 2^{Nc+1}$

4.8.2 Evaluation of data masking results

The masking results using each data mask pattern listed in Table 20 shall be evaluated by scoring penalty points for each occurrence of the following undesirable features listed in Table 21. The higher the penalty points, the less acceptable the masking result. In Table 21, each undesirable feature is scored by a weighting factor, which is defined as W1=100, W2=3 and W3=3.

In the evaluation, all the modules are taken into account although the data masking shall be only applied to the data modules. Furthermore, all the symbols in JAB Code, including the master symbol and the slave symbols (if exist), shall be evaluated together.

After result evaluation according to Table 21, the data mask pattern that results in the lowest penalty score shall be selected for the code.

Table 21: Scoring of data masking results

Feature	Scoring condition	Penalty points
Pattern with the same color and structure as any finder pattern in row/column	Existence of the pattern	W1
Block of modules in same color	block_size = m × n	W2 × (m − 1) × (n − 1)
Adjacent modules in row/column in same color	Number of modules = (5+k), k>0	W3 + k

4.9 Metadata generation and module placement

The final metadata information shall be constructed for each symbol as defined in Section 3.4.1 and 3.4.2 and encoded using LDPC as specified in Section 3.4.3, including the number of module colors, symbol shape, symbol size, data mask pattern type, error correction parameters and the code structure.

The bits of metadata including the error correction bits are graphically encoded into the reserved modules. As defined in Section 3.4.1.1, up to 8 module colors shall be used to encode metadata, except the first part of metadata in master symbols, which shall be encoded in two-color mode.

- N_c shall be encoded in two-color mode.
- In case of module color modes 1 and 2, corresponding to 4 and 8 module colors, the metadata shall be encoded using all available colors.
- In case of other module color modes, which contain more than 8 modules colors, only the colors available in color mode 2 shall be used to encode the metadata.

The encoded metadata bit stream shall be placed into the modules which are reserved in Section 4.6, following the placement order specified in Section 3.4.4. If the length of metadata is not an exact multiple of N, where N is the number of bits a metadata module represents, up to N-1 padding bits (all zeros) shall be appended to the metadata bit stream to fill up the final used module.

5 User Guidelines

5.1 Dimensions

JAB Code symbols (both master and slave symbols) shall conform to the following dimensions:

X dimension: the width of a module shall be specified by the application, taking into account the technologies used to produce and scan the symbol;

Y dimension: the height of a module shall be equal to the X dimension.

No limit is placed on the module size in this specification. However, all modules in the symbols of a JAB Code shall be of the same size.

5.2 Use selection of module color

In JAB Code, eight module color modes are specified, which allow a symbol to contain up to 256 different module colors. Using more module colors in a symbol allows higher data capacity, but it also puts higher requirements upon the technologies used to produce and read the symbol. The selection of module colors should be determined in relation to:

- the required data payload according to the application requirements;
- the expected symbol size according to the application requirements;
- the capability of the technologies used to produce and scan the symbol.

See Annex F for guidelines for module color specifications.

5.3 User selection of error correction level

An appropriate error correction level should be defined by the user according to the application requirements. As specified in Section 3.4.1.4, JAB Code allows customizable error correction levels. For a given message length, a higher level of error correction will lead to some increase in symbol size. The recommended error correction level for normal use should be enforced as the default level in the encoder and decoder.

If the symbol size is fixed in the application regardless of the message length, the highest possible error correction level shall be used, with which the whole encoded data can be hold by the symbol, in order to achieve the best robustness.

5.4 User selection of symbol and code shape

JAB Code allows square, rectangle symbols and arbitrary code shapes by symbol cascading. The user should define the appropriate symbol shape and code structure to suit the application requirements.

The symbol shape should be determined by the shape of the space to place the symbol. In case of non-square placing space, rectangle symbols can be used to accommodate more data than square symbols by making the most of the available space.

With the same symbol size, slave symbols may accommodate more data than master symbols, as they have lower overhead thanks to absence of finder patterns and shorter metadata. However, symbol cascading increases the reading complexity of JAB Code and may consequently decrease decoding reliability. Therefore, symbol cascading should only be used in the following cases:

- the data message can not be accommodated by a single master symbol;

- the available space to place the code has an irregular shape, which can not be fully utilized by a single square or rectangle symbol.
- small symbols (small side-version) are preferred due to the application requirements.

5.5 Guidelines for symbol print and scan

Any JAB Code application must be viewed as a total system solution. All the symbology encoding/decoding components (surface marker or printer, labels, readers) making up an application need to operate together as a system. A failure in any link of the chain, or a mismatch between them, could compromise the performance of the overall system.

While compliance with the specifications is one key to assuring overall system success, other considerations come into play which may influence performance as well. The following guidelines suggest some factors to keep in mind when specifying or implementing bar code systems:

- Select a print density which will yield tolerance values that can be achieved by the marking or printing technology being used. Ensure that the module dimension is an integer multiple of the print head pixel dimension.
- Choose a reader with a resolution suitable for the symbol density and quality produced by the printing technology.
- Ensure that the optical properties of the printed symbol are compatible with the wavelength of the scanner light source or sensor.
- Ensure that the lighting condition is consistent over the whole symbol when scanning the printed symbol.
- Verify symbol compliance in the final label or package configuration. Overlays, show-through, and curved or irregular surfaces can all affect symbol readability.

6 Reference decode algorithm

This reference decode algorithm finds the symbols in an image and decodes them. This algorithm may be used by practical reader implementations.

6.1 Decoding procedure overview

Decoding JAB Code from a captured image involves the following steps:

1. Preprocess the image which contains a JAB Code. When necessary, the captured image containing the symbols should be denoised to remove isolated pixels which have colors greatly differing from their neighbors.
2. Locate the master symbol within the image by finding the finder patterns.
3. Decode the metadata of master symbol and determine the code parameters, including module color mode, side-versions, error correction parameters, mask pattern reference and docking positions of slave symbols.
4. Locate the alignment patterns and establish the sampling grid.
5. Extract and construct the color palettes.
6. Decode the data modules by determining their color index in the color palette.
7. Release the data masking using the mask pattern corresponding to the decoded mask pattern reference.
8. Deinterleave the data stream.
9. Detect and correct errors in the data stream.
10. Decode the data stream into the original message in accordance with the encoding modes in use.
11. If exist, locate the slave symbols docked to the master symbol and decode the metadata and the data stream in the slave symbols.
12. If exist, locate and decode further docked slave symbols recursively, according to the symbol decoding order.

6.2 Locating the finder patterns

The four finder patterns in JAB Code, Finder Pattern UL, Finder Pattern UR, Finder Pattern LR and Finder Pattern LL, consist of five layers with a core module in unique color, located at the four corners of the master symbol as described in Section 3.3.6. All of them have a distinct characteristic which may make them easy to be identified in the image: layer widths in each finder pattern form a *p*C1-C2-C1-C2-*q*C1 sequence, where C1 and C2 represent the layer colors and the relative widths of each layer are *p*:1:1:1:*q* in all scanlines running through the center of the core module. In this reference algorithm, the tolerance for each of these widths is 50%, corresponding to a range of 0.5 to 1.5).

When a candidate scanline with the above-mentioned characteristic is detected, note the position A where the scanline intersects the border between the first and the second layer and the position B where the scanline intersects the border between the fourth layer and the fifth layer, as shown in Figure 19. Repeat this by scanning the adjacent pixel lines in the image until all pixel lines crossing the core module in the horizontal direction have been identified.

Apply the last step on pixel columns to identify the pixel lines crossing the core module in the vertical direction. Locate the center of the finder pattern by finding the midpoint between A and B in horizontal

direction and between C and D in vertical direction, which is shown as point E in Figure 19. The orientation of the finder pattern can be determined by repeating the line scanning step in diagonal directions.

All of the four finder patterns can be located by repeating these steps. After all finder patterns are located, identify the type of each finder pattern by the detected core module color C1. Subsequently, determine the rotational orientation of the symbol and possible mirroring by analyzing the coordinates of the four finder patterns.

The horizontal and vertical module size in each finder pattern can be calculated by the distance between A and B as below.

$$M_X = d_{AB}/3$$

$$M_Y = d_{CD}/3$$

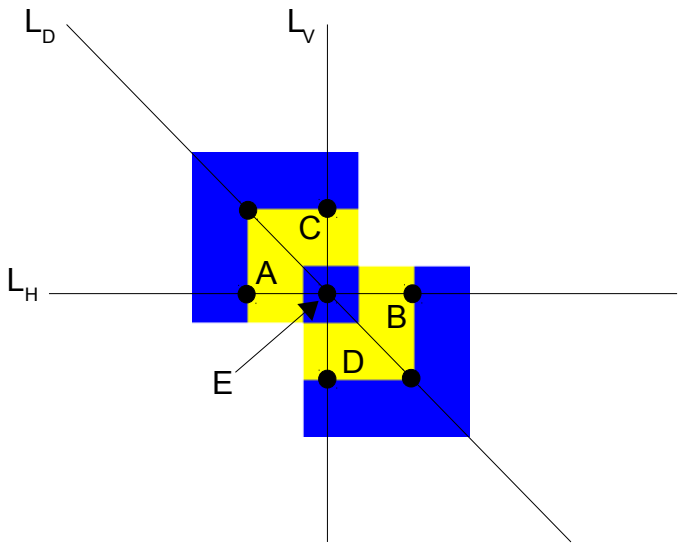

Figure 19: Scanlines in finder pattern

6.3 Decoding the metadata

After the finder patterns are identified, find the guide lines AB, CD, EF and GH, which pass through the centers of the core modules of the four finder patterns as shown in Figure 20. The guide lines AB and CD are perpendicular to EF and GH.

The sampling grid for each module center in the reserved metadata area around the finder patterns can be determined based on the lines parallel to the guide lines, the central coordinates of each finder pattern and the module size M_X and M_Y of each finder pattern.

The modules that encode the metadata Part I shall be first decoded which are located at the reserved positions as defined in Section 3.4.4. These modules are encoded in two-color mode as described in Section 3.4.1.1. The modules with lower luminance are decoded as dark modules, representing binary zero, and the modules with higher luminance are decoded as light modules, representing binary one. Detect and correct errors in the extracted 6-bit metadata to get the module color mode N_c.

After the used module color mode is identified, the reference colors used for metadata can be sampled in the same way as metadata at the predefined positions. In this step, up to 8 colors in the color palette shall be decoded. Based on the sampled reference colors, decode other parts of the metadata step by step as defined

in Section 3.4.1 to determine the code parameters, including the side-versions, the error correction parameters, the masking pattern reference and the docking positions of slave symbols.

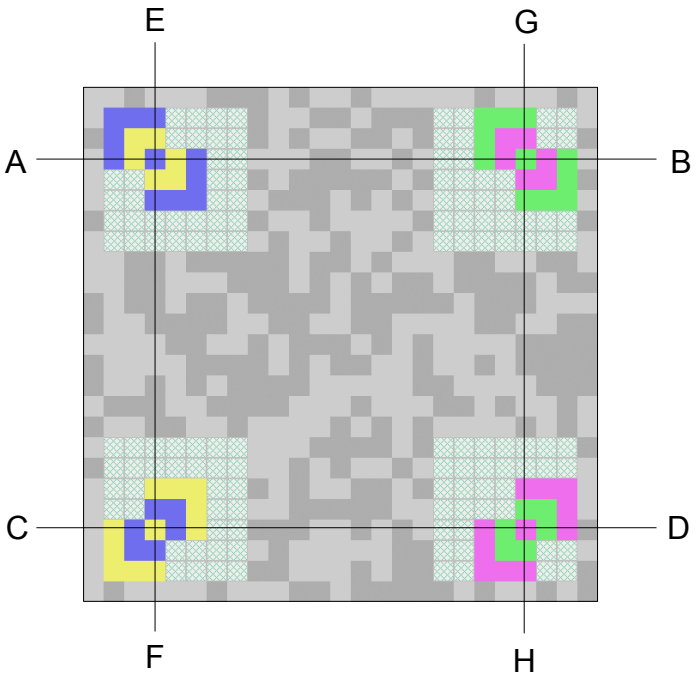

Figure 20: Finder patterns and guide lines

6.4 Locating the alignment patterns and establishing the sampling grid

For Side-Version 6 or larger symbols, determine the provisional central coordinate of each alignment pattern, Alignment Pattern X0 or X1, from the coordinates defined in Table 2, based on the central coordinate of each finder pattern, the module size and the lines parallel to the guide lines AB, CD, EF and GH, i.e. the lines A1B1, A2B2, C1D1, E1F1, E2F2 and G1H1 as shown in Figure 21. Scan the pixel lines around the provisional central coordinate to find the actual central coordinate of each alignment pattern.

The sampling grid for each area surrounded by four alignment patterns in the symbol can be determined based on the central coordinates of the four alignment patterns, the lines parallel to the guide lines connecting the alignment pattern centers and the average module size. The average module size can be calculated by the center-to-center distance of the alignment patterns and the number of modules between the alignment pattern centers as defined in Table 2. Thus, the sampling grind can be established with lines equidistantly spaced between the centers of the alignment patterns. In the areas at the four corners of the symbol, one alignment pattern shall be replaced by the finder pattern at the corresponding coordinate to determine the sampling grid.

For the symbols with side-version less than 6, in which there exists no alignment pattern, the sampling grid is established solely based on the centers of the four finder patterns and the lines parallel to the guide lines AB, CD, EF and GH.

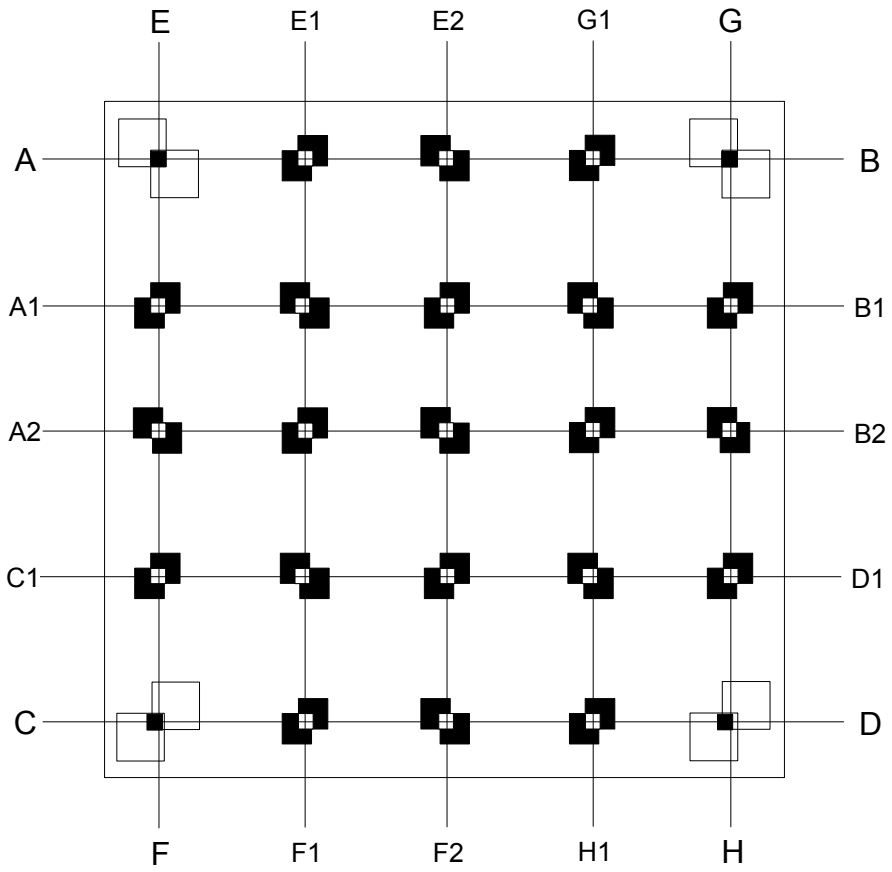

Figure 21: Finder patterns and alignment patterns with guide lines

6.5 Constructing the color palettes

After the sampling grid is established, the modules which encode the reference colors can be sampled to obtain the reference colors. For each reference color module located at the reserved positions as defined in Section 3.4.4, sample an area of 3×3 image pixels centered on the intersection of the grid lines and store the sampled color into the corresponding color palette. Repeat this step until the two embedded color palettes are assembled.

For symbols that contain more than 8 module colors, rearrange the color entries in the two assembled color palettes to obtain the original color palette. For symbols that contain more than 64 module colors, after entry rearrangement, interpolate the assembled palettes to restore the absent reference colors and obtain the original full-size color palettes. Refer to Annex F for the specification of color palette interpolation and construction.

6.6 Decoding the data message

First, based on the sampling grids, sample an area of 3×3 image pixels centered on each intersection of the grid lines and determine the color of the module based on the nearest color palette in the symbol. Map the module color into the corresponding bit string according to its index in the color palette.

Second, based on the decoded mark reference from the metadata, apply the used masking pattern to the decoded data modules to release the data masking and restore the original encoded bit strings. Sequentially assemble the bit string of each data module to get the data stream. Subsequently deinterleave the data stream to obtain the encoded data in the original sequence.

Third, based on the decoded error correction parameters from the metadata, detect errors in the data stream and correct them if there exist errors using the LDPC decoder.

Finally, decode the data message into the original message in accordance with the encoding mode in use, as described in Section 4.3.

6.7 Locating and decoding slave symbols

Based on the decoded metadata, if there exist slave symbols docked to the master symbol, locate and decode them according to the symbol decoding order defined in Section 3.5.2.

1. Determine the provisional central coordinates of the two Alignment Pattern U or Alignment L close to the docking side of the master symbol based on the central coordinates of the two finder patterns at the docking side, the corresponding guide lines and the distance between the two finder patterns and the two adjacent alignment patterns in the slave symbol which is fixed at 7 modules. For horizontally docked slave symbols, the two Alignment Pattern U or L are located on the extension lines of AB and CD. For vertically docked slave symbols, they are located on the extension lines of EF and GH. Scan the pixel lines around the provisional central coordinate to find the actual central coordinate of each alignment pattern.
2. Determine the sampling grid for the metadata and the color palette in the slave symbol based on the central coordinates of the two finder patters and the detected alignment patterns, the lines parallel to the guide lines connecting the finder pattern and alignment pattern centers and the side-version of the docking side.
3. Sample the modules for the reference colors used for slave symbol metadata at the reserved positions as defined in Section 3.4.4. Decode the metadata in the slave symbol at the reserved positions step by step as defined in Section 3.4.2 to determine the slave symbol parameters, including the side-version of the other side, the error correction parameters and the docking positions of further slave symbols.
4. Determine the provisional central coordinates of the other two Alignment Pattern U or Alignment L far from the docking side based on the central coordinates of the two detected Alignment Pattern U or L, the extension lines of the guide lines connecting the finder patterns at the docking side and the adjacent Alignment Pattern U or L and the side-version of the non-docking side of the slave symbol. Scan the pixel lines around the provisional central coordinate to find the actual central coordinate of each alignment pattern.
5. Locate the other alignment patterns, Alignment Pattern X0 and X1, if exist, establish the sampling grid, construct the color palettes and decode the data in the same way as the master symbol.

Repeat step 1-5 until all the slave symbols docked to the master symbol are decoded. If there exist further docked symbols to the decoded slave symbols, locate and decode all the symbols recursively according to the decoding order defined in Section 3.5.2.

Annex A: Error detection and correction

A.1 Error encoding

The Low-Density Parity Check Code (LDPC) shall be used for error correction. The parity check matrix $H_{P_n}(C^T|I) \in (K \, x \, P_g)$ in systematic form for each length of the message length P_n is defined by the three parameter w_c and w_r and K. The parameter w_c describes the number of '1's in each column and w_r the number of '1's in each rows. With w_c and w_r the rate R of the code is determinable with $R=1-w_c/w_r$. One way to generate the matrix H is to randomly fill the columns with w_c '1's such that w_r holds and it comprise the identity matrix I. It is recommended to select $w_c \geq 3$ and $w_r \geq w_c+1$.

The generator matrix $G_{P_n}(I|C) \in (P_n \, x \, P_g)$ is obtained by the matrix H and only shown for the first two metadata length. The Codeword c is obtained by matrix multiplication in the GF(2) $c = m \otimes G$.

Example:

Given the parameters $w_c = 3, w_r = 6, K = 5$, the message $m = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \end{bmatrix}$, the generated matrix H and G are as follows:

$$H_5 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow G_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

The final codeword is obtained by $c = m \otimes G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$.

A.2.1 Error detection and correction with soft decision

The error correction for the metadata in JAB Code shall be computed using iterative Log Likelihood decoding algorithm for binary LDPC Codes. After releasing the masking the error correction shall be performed.

The Log Likelihood decoding requires the matrix $H_{P_n}(C^T|I) \in (K \, x \, P_g)$ used for encoding, the received codeword r (after releasing the masking) and the maximum number of iterations L.

The minimal Hamming distance d_{min} induced by the used H matrix gives the number of detectable errors and $\lfloor (d_{min}-1)/2 \rfloor$ the number of correctable errors.

The decoding algorithm requires initializing the parameter $\eta_{u,v}^{[0]} = 0$ for all (u,v) with H(u,v)=1, $\lambda_v^{[0]} = r_v$ and the loop counter $l = 1$.

For each (u,v) with H(u,v)=1 the algorithm computes $\eta_{u,v}^{[l]} = -2 \tanh^{-1}\left(\prod_{j \in V_{u,v}} \tanh\left(\frac{-\lambda_j^{[l-1]} - \eta_{u,j}^{[l-1]}}{2} \right) \right)$ and $\lambda_v^{[l]} = r_v + \sum_{u \in U_v} \eta_{u,v}^{[l]}$. The algorithm makes a tentative decision with $\hat{c}_v = 1$ if $\lambda_v^{[l]} > 0$, else $\hat{c}_v = 0$.

If $H\hat{c}=0$, the algorithm found the correct codeword, else the algorithm update the loop counter l, $\eta_{u,v}^{[l]}$ and $\lambda_v^{[l]}$ as long as $l<L$. If $l=L$ and $H\hat{c}\neq 0$ the decoder declares a decoding failure and stops. It is recommended to use L=25.

Example:
Use matrix H from Annex A.1. Received vector $\lambda^{[0]}$ is:

$$\lambda^{[0]}=\begin{bmatrix}1.2 & -1.1 & 1.3 & -1.5 & 1.9 & 0.2 & -1.5 & -0.08 & -1.7 & -1.3\end{bmatrix}$$

$$\eta^{[1]}=\begin{bmatrix}-0.21 & -0.17 & -0.19 & 0 & 0 & 0.16 & -0.18 & 0 & 0 & 0.22\\ 0.027 & 0 & 0.024 & 0 & -0.15 & -0.02 & 0 & -0.026 & 0.03 & 0\\ 0 & 0 & -0.0013 & -0.021 & 0.0083 & 0 & -0.0013 & 0 & -0.0017 & 0.0016\\ 0 & -0.0018 & 0 & -0.03 & 0.012 & 0.0016 & 0 & 0.0021 & 0 & 0.0023\\ 0.01 & 0.0083 & 0 & 0.14 & 0 & 0 & 0.0088 & -0.0097 & 0.011 & 0\end{bmatrix}$$

and

$$\lambda^{[1]}=\begin{bmatrix}1.4 & -1.1 & 1.3 & -1.7 & 2 & 0.072 & -1.6 & 0.011 & -1.8 & -1.4\end{bmatrix}$$

$$\hat{c}=\begin{bmatrix}1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0\end{bmatrix},\quad H\hat{c}\neq 0$$

One further iteration brings the decoder to the result such that $H\hat{c}=0$.

A.2.2 Error detection and correction with hard decision

The error correction for the message data in JAB Code shall be computed using hard decision decoding algorithm for binary LDPC Codes. After releasing the masking the error correction shall be performed.

The decoding algorithm requires initializing the parameter $\eta_{u,v}^{[0]}=0$ for all (u,v) with H(u,v)=1, $\lambda_v^{[0]}=0$ and the loop counter $l=1$.

For each (u,v) with H(u,v)=1 the algorithm computes $\eta_{u,v}^{[l]}=\left(\hat{c}\wedge\eta_{u,j}^{[l-1]}\right) for\ j\in V_{u,v}$ and $\lambda_v^{[l]}=\sum_{u\in U_v}\eta_{u,v}^{[l]}$.
The algorithm flip those bits with the maximum $\lambda_v^{[l]}>0$ in each iteration step l.

If $H\hat{c}=0$, the algorithm found the correct codeword, else the algorithm update the loop counter l, $\eta_{u,v}^{[l]}$ and $\lambda_v^{[l]}$ as long as $l<L$. If $l=L$ and $H\hat{c}\neq 0$ the decoder declares a decoding failure and stops. It is recommended to use L=25.

Annex B: Matrix generation for metadata

The Low-Density Parity Check Code (LDPC) shall be used for error correction of metadata. The parity check matrix $H_{P_n}(C^T|I) \in (K \, x \, P_g)$ in is generated with P_n as the metadata length. The generator matrix $G_{P_n}(I|C) \in (P_n \, x \, P_g)$ is obtained by the matrix H and only shown for the first two metadata length. The Codeword c is obtained by matrix multiplication in the GF(2) $c = m \otimes G$ (see Annex A). The algorithm shall create the matrix H as follows:

1. Set $w_c = 2$ if the metadata length is shorter than 36 bits, else $w_c = 3$.

2. Set the number of '1' in each row of matrix H to: $\lfloor C \times K / w_c + 3 \rfloor / K$.

3. The '1's in each row shall be equal distributed. Matrix H will be obtained by using the interleaving algorithm listed in Annex E to specify the position of the '1's in each row of the matrix.

Annex C: JAB Code symbol encoding example

For instance the message “JAB Code 2016!” shall be encoded into the JAB Code symbol.

C.1 Creating the message bit stream

The characters are taken one after each other and processed according to Annex D. The output is shown in Table 22:

Table 22: Message encoding for "JAB Code 2016!"

Input Message	Encoding mode	Value	Binary Stream
J	Uppercase	10	01010
A	Uppercase	1	00001
B	Uppercase	2	00010
SP	Uppercase	0	00000
C	Uppercase	3	00011
L/L	Uppercase	28	11100
o	Lowercase	15	01111
d	Lowercase	4	00100
e	Lowercase	5	00101
N/L	Lowercase	29	11101
SP	Numeric	0	0000
2	Numeric	3	0011
0	Numeric	1	0001
1	Numeric	2	0010
6	Numeric	5	0101
P/S	Numeric	13	1101
!	Punctuation	0	0000

C.2 Selecting symbol

The resulting binary message length is 78 bits. Using the default error correction level 2 and 8 colors the metadata length is 10 modules. The gross message length P_g and the required error correction bits K are obtained according to Section 4.4.1, P_g=182 and K=104. According to Table 1 the symbol Side-Version 1 is selected with the side size 21. According to Section 4.4.2 and 4.4.3 the stuffing bits and the error correction bits are added. Afterward the interleaving for the Side-Version 1 is applied according to Section 4.5 and the masking according to Section 4.8.

C.3 Assembling the Metadata

The metadata are selected according to Section 3.4.1. The first part of the metadata is 010 and encoded separately by LDPC according to Annex B and requires 6 modules. Part II of the metadata is SS=0, VF=00,

MSK=000 and SF=0, together 0000000. Part II is also encoded into 14 bits according Annex B. Part III is V=00, E=0010100101 and encoded according Annex B. After encoding the length of part III is 24 bits. Part II and Part III together require 13 modules.

C.4 Assembling the Symbol

The assembled symbol is shown in Figure 22.

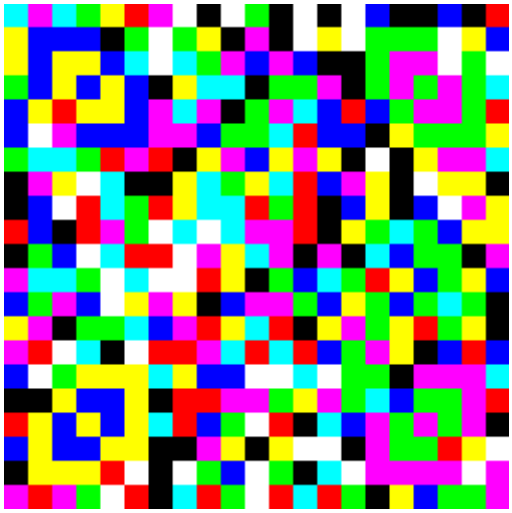

Figure 22: Sample symbol

Annex D: Optimization of bit stream length

As described in Section 4.3, JAB Code offers various encoding modes. The modes differs in the number of required bits to represent given data characters. Since there is an overlap between the character sets of the modes, it is necessary to choose the most appropriate mode to encode this data in the shortest sequence of bits possible. The algorithm described in the following processes the input data characters one after each other and output a trellis graph. After the last character is processed, the shortest sequence is known.

First some tables are set up to show the length of bits required to latch or shift from one mode to another. The seven encoding modes are abbreviated by:

Encoding Mode	Abbreviation	Character Size in Bit per Character
Uppercase	U	5
Lowercase	L	5
Numeric	N	4
Punctuation	P	4
Mixed	M	5
Alphanumeric	A	6
Byte	B	8

The following table present the length of bits required to shift from one to another mode, where 10^6 indicates a not possible shift.

From \ To	U	L	N	P	M	A	B
U	10^6	10^6	10^6	5	7	10^6	11
L	5	10^6	7	5	7	10^6	11
N	6	10^6	10^6	4	6	10^6	10
P	0	0	0	10^6	10^6	0	10^6
M	0	0	0	10^6	10^6	0	10^6
A	10^6	10^6	10^6	8	8	10^6	12
B	0	0	0	10^6	10^6	0	10^6

The following table present the length of bits required to latch from one to another mode, where 10^6 indicates a not possible latch.

From \ To	U	L	N	P	M	A	B
U	0	5	5	10^6	10^6	5	10^6
L	7	0	5	10^6	10^6	5	10^6
N	4	6	0	10^6	10^6	10^6	10^6
P	10^6	10^6	10^6	10^6	10^6	10^6	10^6
M	10^6	10^6	10^6	10^6	10^6	10^6	10^6
A	8	10^6	10^6	10^6	10^6	0	10^6
B	10^6	10^6	10^6	10^6	10^6	10^6	0

To optimize the bit stream length the algorithm use four varaiables. One variable with 14 integers $CharSize_C$[EncodingMode(Latch), EncodingMode(Shift)] for each character and set the character size according to the first table in this Annex if the character is encoded by the appropriate encoding mode (See Table 12), otherwise set the value to 10^6. A further variable SwitchMode contain all the bit length to latch or switch from one into another mode (see the second and third table of this Annex). The third variable contains also 14 integer values and hold the information of the current binary sequence length CurrSeqLen[EncodingMode(Latch), EncodingMode(Shift)]. The fourth variable is 14-character long for each character and keep the information about the previous mode PrevMode, which induce the shortest length for each mode.

Step 1. Initialize $CharSize_0$[0 10^6 10^6 10^6 10^6 10^6 10^6 0 10^6 10^6 10^6 10^6 10^6 10^6] because the initial mode is the Uppercase mode.

Step 2. Accept the first character.

Step 3. Set CurrSeqLen to the number of bits required to encode this character according to the encoding mode.

Step 4. Use the $CharSize_C$ and SwitchMode variable to update the CurrSeqLen variable with the minimum length. Set the previous mode which induce the shortest length to PrevMode.

Step 5. Accept the next character and move on with Step 3 while no character remains.

Step 6. The shortest path through the trellis graph is known by picking the smallest number in CurrSeqLen and go the whole graph back by means of PrevMode. Set all unused values in $CharSize_C$ to 10^6.

Annex E: Interleaving algorithm

The following in-place random permutation algorithm is used to interleave a bit sequence in JAB Code.

Denote a bit sequence as $S=\{b_0, b_1, b_2, ..., b_{N-1}\}$ containing N bits.

1. Give an initial seed for the random number generator.
2. Set the variable L=N.
3. Generate a random number R between 0 and L-1.
4. Swap the bit b_R at index R and the bit b_{L-1} at index L-1.
5. Update L=L-1.
6. Repeat the steps 3-5 until L=0.

The following C routine is used to generate random numbers in step 3.

```
#include <inttypes.h>
uint32_t temper(uint32_t x)
{
	x ^= x>>11;
	x ^= x<<7 & 0x9D2C5680;
	x ^= x<<15 & 0xEFC60000;
	x ^= x>>18;
	return x;
}
uint32_t lcg64_temper(uint64_t* seed)
{
	*seed = 6364136223846793005ULL * *seed + 1;
	return temper(*seed >> 32);
}
```

Annex F: Guidelines for module color selection and color palette construction

F.1 Module color selection

JAB Code supports eight color modes and up to 256 module colors are allowed to use in a symbol. In order to optimize the decoding of JAB Code, the used colors shall be so distinguishable as possible. Therefore, the used colors shall keep a distance from each other in the RGB color space cube as shown in Figure 23.

(a) In case of 4-color mode, blue, green, magenta and yellow shall be used.

(b) In case of 8-color mode, each color channel of R, G and B takes two values, 0 and 255, which will generate the 8 colors at the vertexes of the cube, i.e. black, blue, green, cyan, red, magenta, yellow and white. These 8 colors are included in all the following color modes.

(c) In case of 16-color mode, the color channel R takes 4 values, 0, 85, 170 and 255, and the channel G and B take two values, 0 and 255, which will totally generate 16 colors as listed in Table 23.

(d) In case of 32-color mode, the color channel R and G take four values, 0, 85, 170 and 255, and the color channel B takes two values, 0 and 255, which will totally generate 32 colors.

(e) In case of 64-color mode, each color channel of R, G and B takes four values, 0, 85, 170 and 255, which will totally generate 64 colors.

(f) In case of 128-color mode, the color channel R takes eight values, 0, 36, 73, 109, 146, 182, 219 and 255, and the color channel G and B take four values, 0, 85, 170 and 255, which will totally generate 128 colors.

(g) In case of 256-color-mode, the color channel R and G take eight values, 0, 36, 73, 109, 146, 182, 219 and 255, and the color channel B takes four values, 0, 85, 170 and 255, which will totally generate 256 colors.

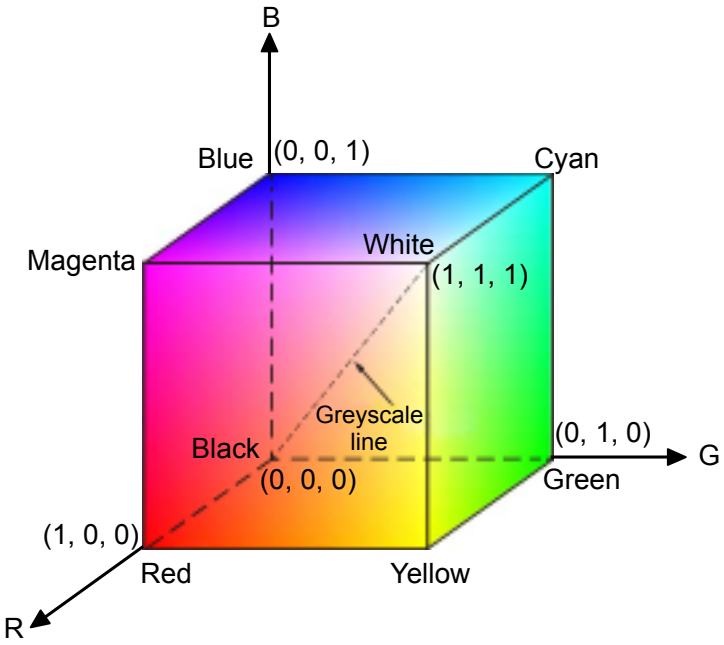

Figure 23: RGB color space cube

Table 23: Used colors in 16-color mode

Color index	R	G	B	Binary bits
0	0	0	0	0000
1	0	0	255	0001
2	0	255	0	0010
3	0	255	255	0011
4	85	0	0	0100
5	85	0	255	0101
6	85	255	0	0110
7	85	255	255	0111
8	170	0	0	1000
9	170	0	255	1001
10	170	255	0	1010
11	170	255	255	1011
12	255	0	0	1100
13	255	0	255	1101
14	255	255	0	1110
15	255	255	255	1111

F.2 Construction of the embedded color palette in the symbol

In either master and slave symbols, there are 128 modules reserved for two color palettes. Therefore, each color palette can contain up to 64 colors.

(a) In case of 4-color to 64-color modes, all available colors shall be included in the embedded color palettes.

(b) In case of 128-color mode, the colors whose R channel values are 0, 73, 182 or 255 shall be included in the embedded color palettes.

(c) In case of 256-color mode, the colors whose R and G channel values are 0, 73, 182 or 255 shall be included in the embedded color palettes.

In symbols containing more than 8 colors, the metadata are encoded using only the 8 colors available in the 8-color mode, namely the 8 colors at the vertexes of the RGB color cube. In order to enable metadata decoding in early decoding steps, in which only the first part of the color palette is decoded, the color entries in the embedded color palette containing more than 8 colors shall be rearranged. The 8 colors at the vertexes shall be always moved to the first 8 entries, followed by the other selected colors in their original sequence.

In the symbol decoding, the extracted color palettes shall be first inversely arranged into their original sequence. For symbols containing 128 and 256 colors, the original full-size color palette shall be reconstructed by interpolating each color channel of the colors in the extracted color palettes. For example, the colors whose original R channel value is 36 shall be restored by interpolating the colors whose original R channel value is 0 and 73.