基础知识

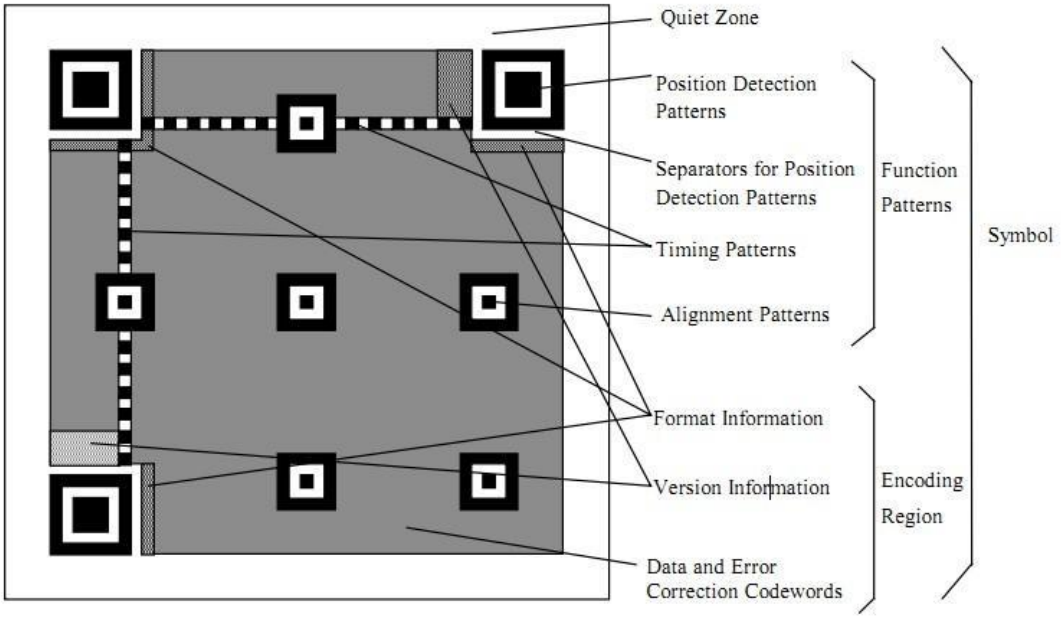
二维码一共有 40 个尺寸，Version1 是21 × 21的矩阵，Version2 是25 × 25的矩阵，Version3 是29 × 29的尺寸，Version 每增加 1，尺寸就对应加 4。

公式为：

$(V-1)*4+21$,

其中 V 为版本号，V 最大值为 40。

二维码样例：



（1）定位图案

•**Position Detection Pattern:** 定位图案，用于标记二维码的矩形大小。这三个定位图案有白边叫 Separators for Position Detection Patterns。之所以三个而不是四个意思就是三个就可以标识一个矩形了。

•**Timing Patterns:** 也是用于定位的。原因是二维码有 40 种尺寸，尺寸过大了后需要有根标准线，不然扫描的时候可能会扫歪了。

•**Alignment Patterns:** 只有 Version2 以上（包括 Version2）的二维码需要这个东东，同样是为了定位用的。

（2）功能性数据

•**Format Information:** 存在于所有的尺寸中，用于存放一些格式化数据的。

•**Version Information:** 在 Version7 以上（包括 Version7），需要预留两块3 × 6的区域存放一些版本信息。

（3）数据码和纠错码

除了上述的那些地方,剩下的地方存放数据码(Data Code)和纠错码(Error Correction Code)。

数据编码

二维码（即 QR 码）支持如下的编码：

（1）数字编码（Numeric mode）

从 0 到 9。如果需要编码的数字的个数不是 3 的倍数，那么，最后剩下的 1 或 2 位数会被转成 4 或 7bits，则其它的每 3 位数字会被编成 10，12，14bits，编成多长还要看二维码的尺寸。

（2）字符编码（Alphanumeric mode）

包括 0-9，大写的 A 到 Z（没有小写），以及符号$ % * + - . / : 包括空格。这些字符会映射成一个字符索引表。如下所示:（其中的 SP 是空格，Char 是字符，Value 是其索引值）编码的过程是把字符两两分组，然后转成下表的 45 进制，然后转成 11bits 的二进制，如果最后有一个落单的，那就转成 6bits 的二进制。而编码模式和字符的个数需要根据不同的 Version 尺寸编成 9,11 或 13 个二进制。

Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value
0	0	6	6	C	12	I	18	O	24	U	30	SP	36	.	42
1	1	7	7	D	13	J	19	P	25	V	31	$	37	/	43
2	2	8	8	E	14	K	20	Q	26	W	32	%	38	:	44
3	3	9	9	F	15	L	21	R	27	X	33	*	39		
4	4	A	10	G	16	M	22	S	28	Y	34	+	40		
5	5	B	11	H	17	N	23	T	29	Z	35	-	41		

（3）字节编码（Byte mode）

可以是 0-255 的 ISO-8859-1 字符。有些二维码的扫描器可以自动检测是否是 UTF-8 的编码。

（4）日文编码（Kanji mode）

双字节编码。同样，也可以用于中文编码。日文和汉字的编码会减去一个值。如：在 0X8140 到 0X9FFC 中的字符会减去 8140，在 0XE040 到 0XEBBF 中的字符要减去 0XC140，然后把结果前两个 16 进制位拿出来乘以 0XC0，然后再加上后两个 16 进制位，最后转成 13bit 的编码。如下图示例：

Input character	“点”	“茗”
(Shift JIS value):	**935F**	**E4AA**
1. Subtract 8140 or C140	**935F - 8140 = 121F**	**E4AA - C140 = 236A**
2. Multiply m.s.b. by C0	**12 × C0 = D80**	**23 × C0 = 1A40**
3. Add l.s.b.	**D80 + 1F = D9F**	**1A40 + 6A = 1AAA**
4. Convert to 13 bit binary	**0D9F → 0 1101 1001 1111**	**1AAA →1 1010 1010 1010**

（5）Extended Channel Interpretation (ECI) mode

主要用于特殊的字符集。并不是所有的扫描器都支持这种编码。

（6）Structured Append mode

用于混合编码，也就是说，这个二维码中包含了多种编码格式。

（7）FNC1 mode

这种编码方式主要是给一些特殊的工业或行业用的。比如 GS1 条形码之类的。

下面给出两张表：

• **Table2：** 各个编码格式的“编号”，这个东西要写在 Format Information 中。

注：中文是 1101。

• **Table3：** 表示不同版本（尺寸）的二维码，对于数字、字符、字节和 Kanji 模式下，对于单个编码的 2 进制的位数。（在二维码的规格说明书中，有各种各样的编码规范表）

Table 2 — Mode indicators

Mode	Indicator
ECI	**0111**
Numeric	**0001**
Alphanumeric	**0010**
8-bit Byte	**0100**
Kanji	**1000**
Structured Append	**0011**
FNC1	**0101** (First position) **1001** (Second position)
Terminator (End of Message)	**0000**

Table 3 — Number of bits in Character Count Indicator

Version	Numeric Mode	Alphanumeric Mode	8-bit Byte Mode	Kanji Mode
1 to 9	**10**	**9**	**8**	**8**
10 to 26	**12**	**11**	**16**	**10**
27 to 40	**14**	**13**	**16**	**12**

下面我们看几个示例：

示例一：数字编码

在 Version1 的尺寸下，纠错级别为 H 的情况下，编码：01234567。

① 把上述数字分成三组：012 345 67；

② 把他们转成二进制：012 转成 0000001100，345 转成 0101011001，67 转成 1000011；

③ 把这三个二进制串起来：0000001100 0101011001 1000011；

④ 把数字的个数转成二进制（version1-H 是 10bits ）：

8 个数字的二进制是 0000001000；

⑤ 把数字编码的标志 0001 和第④步的编码加到前面：

0001 0000001000 0000001100 0101011001 1000011；

示例二：字符编码

在 Version1 的尺寸下，纠错级别为 H 的情况下，编码：AC-42。

① 从字符索引表中找到 AC-42 这五个字条的索引（10，12，41，4，2）；

② 两两分组：（10，12）、（41，4）、（2）；

③ 把每一组转成 11bits 的二进制：

（10，12）10*45+12 等于 462 转成 00111001110，

（41，4）41*45+4 等于 1849 转成 11100111001，

（2）等于 2 转成 000010；

④ 把这些二进制连接起来：00111001110 11100111001 000010；

⑤ 把字符的个数转成二进制（Version1-H 为 9bits）：5 个字符，5 转成 000000101；

⑥ 在头上加上编码标识 0010 和第⑤步的个数编码：

0010 000000101 00111001110 11100111001 000010。

结束符和补齐符

假如我们有个 HELLO WORLD 的字符串要编码，根据上面的示例二，我们可以得到下面的编码：

编码	字符数	HELLO WORLD的编码
0010	000001011	01100001011 01111000110 10001011100 10110111000 10011010100 001101

我们还要加上结束符：

编码	字符数	HELLO WORLD的编码	结束
0010	000001011	01100001011 01111000110 10001011100 10110111000 10011010100 001101	0000

按 8bits 重排：

如果所有的编码加起来不是 8 的倍数我们还要在后面加上足够的 0，比如上面一共有 78 个 bits，所以，我们还要加上 2 个 0，然后按 8 个 bits 分好组：

```
00100000  01011011  00001011  01111000  11010001  01110010  11011100  01001101
01000011  01000000
```

补齐码（Padding Bytes）：

最后，如果还没有达到我们最大的 bits 数的限制，我们还要加一些补齐码（Padding Bytes），Padding Bytes 就是重复下面的两个 bytes：11101100 00010001（这两个二进制转成十进制是 236 和 17）。关于每一个 Version 的每一种纠错级别的最大 Bits 限制，可以参看 QR Code Spec 的第 28 页到 32 页的 Table-7 一表。

假设我们需要编码的是 Version1 的 Q 纠错级，那么，其最大需要 104 个 bits，而我们上面只有 80 个 bits，所以，还需要补 24 个 bits，也就是需要 3 个 Padding Bytes，我们就添加三个，于是得到下面的编码：

```
00100000 01011011 00001011 01111000 11010001 01110010 11011100 01001101 01000011
01000000 11101100 00010001 11101100
```

上面的编码就是数据码了，叫 Data Codewords，每一个 8bits 叫一个 codeword，我们还要对这些数据码加上纠错信息。

纠错码

上面我们说到了一些纠错级别，Error Correction Code Level，二维码中有四种级别的纠错，这就是为什么二维码有残缺还能扫出来，也就是为什么有人在二维码的中心位置加入图标。

错误修正容量	
L水平	7%的字码可被修正
M水平	15%的字码可被修正
Q水平	25%的字码可被修正
H水平	30%的字码可被修正

那么，QR 是怎么对数据码加上纠错码的？首先，我们需要对数据码进行分组，也就是分成不同的 Block，然后对各个 Block 进行纠错编码，对于如何分组，我们可以查看 QR Code Spec 的第 33 页到 44 页的 Table-13 到 Table-22 的定义表。注意最后两列：

- **Number of Error Code Correction Blocks:** 需要分多少个块；
- **Error Correction Code Per Blocks:** 每一个块中的 code 个数，所谓的 code 的个数，也就是有多少个 8bits 的字节。

5	134	L	26	1	(134,108,13)
		M	48	2	(67,43,12)
		Q	72	2 2	(33,15,9) (34,16,9)
		H	88	2 2	(33,11,11) (34,12,11)
6	172	L	36	2	(86,68,9)
		M	64	4	(43,27,8)
		Q	96	4	(43,19,12)
		H	112	4	(43,15,14)

[a] (c, k, r): c = total number of codewords
k = number of data codewords
r = number of error correction capacity

[b] Error correction capacity is less than half the number of error correction codewords to reduce the probability of misdecodes.

举个例子：

上述的 Version 5 + Q 纠错级：需要 4 个 Blocks（2 个 Blocks 为一组，共两组），头一组的两个 Blocks 中各 15 个 bits 数据 + 各 9 个 bits 的纠错码。

注：表中的 codewords 就是一个 8bits 的 byte；

最后一例中的（c，k，r ）的公式为：c = k + 2 * r，因为纠错码的容量小于纠错码的一半。

下图给一个 5-Q 的示例（可以看到每一块的纠错码有 18 个 codewords，也就是 18 个 8bits 的二进制数）：

组	块	数据	对每个块的纠错码
1	1	67 85 70 134 87 38 85 194 119 50 6 18 6 103 38	213 199 11 45 115 247 241 223 229 248 154 117 154 111 86 161 111 39
	2	246 246 66 7 118 134 242 7 38 86 22 198 199 146 6	87 204 96 60 202 182 124 157 200 134 27 129 209 17 163 163 120 133
2	1	182 230 247 119 50 7 118 134 87 38 82 6 134 151 50 7	148 116 177 212 76 133 75 242 238 76 195 230 189 10 108 240 192 141
	2	70 247 118 86 194 6 151 50 16 236 17 236 17 236 17 236	235 159 5 173 24 147 59 33 106 40 255 172 82 2 131 32 178 236

注：二维码的纠错码主要是通过 Reed-Solomon error correction（里德-所罗门纠错算法）来实现的。

参见：https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction

最终编码

穿插放置：

如果你以为我们可以开始画图，你就错了。二维码的混乱技术还没有玩完，它还要把数据码和纠错码的各个 codewords 交替放在一起。如何交替呢，规则如下：

对于数据码：把每个块的第一个 codewords 先拿出来按顺度排列好，然后再取第一块的第二个，如此类推。比如上述示例中的 Data Codewords 如下：

块 1	67	85	70	134	87	38	85	194	119	50	6	18	6	103
块 2	246	246	66	7	118	134	242	7	38	86	22	198	199	146
块 3	182	230	247	119	50	7	118	134	87	38	82	6	134	151
块 4	70	247	118	86	194	6	151	50	16	236	17	236	17	236

我们先取第一列的：67、246、182、70；

然后再取第二列的：67、246、182、70、85、246、230、247；

如此类推：67、246、182、70、85、246、230、247,…，38、6、50、17、7、236；

对于纠错码，也是一样：

块 1	213	199	11	45	115	247	241	223	229	248	154	117	154	11
块 2	87	204	96	60	202	182	124	157	200	134	27	129	209	17
块 3	148	116	177	212	76	133	75	242	238	76	195	230	189	10
块 4	235	159	5	173	24	147	59	33	106	40	255	172	82	2

和数据码取的一样，得到：213、87、148、235、199、204、116、159，…,39、133、141、236。

然后，再把这两组放在一起（纠错码放在数据码之后）得到：

67, 246, 182, 70, 85, 246, 230, 247, 70, 66, 247, 118, 134, 7, 119, 86, 87, 118, 50, 194, 38, 134, 7, 6, 85, 242, 118, 151, 194, 7, 134, 50, 119, 38, 87, 16, 50, 86, 38, 236, 6, 22, 82, 17, 18, 198, 6, 236, 6, 199, 134, 17, 103, 146, 151, 236, 38, 6, 50, 17, 7, 236, 213, 87, 148, 235, 199, 204, 116, 159, 11, 96, 177, 5, 45, 60, 212, 173, 115, 202, 76, 24, 247, 182, 133, 147, 241, 124, 75, 59, 223, 157, 242, 33, 229, 200, 238, 106, 248, 134, 76, 40, 154, 27, 195, 255, 117, 129, 230, 172, 154, 209, 189, 82, 111, 17, 10, 2, 86, 163, 108, 131, 161, 163, 240, 32, 111, 120, 192, 178, 39, 133, 141, 236

这就是我们的数据区。

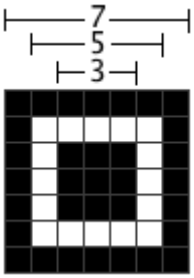
Remainder Bits:

最后再加上 Reminder Bits，对于某些 Version 的 QR，上面的还不够长度，还要加上 Remainder Bits，比如：上述的 5Q 版的二维码，还要加上 7 个 bits，Remainder Bits 加零就好了。关于哪些 Version 需要多少个 Remainder bit，可以参看 QR Code Spec 的第 15 页的 Table-1 的定义表。

画二维码图

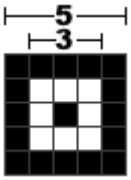
（1）Position Detection Pattern

首先，先把 Position Detection 图案画在三个角上。（无论 Version 如何，这个图案的尺寸就是这么大）



（2）Alignment Pattern

然后，再把 Alignment 图案画上。（无论 Version 如何，这个图案的尺寸就是这么大）

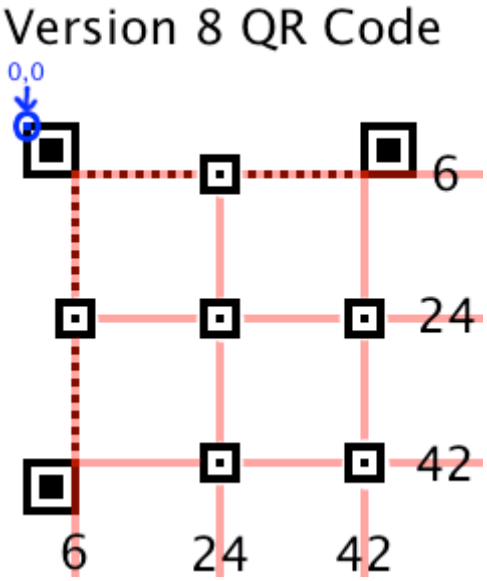


关于 Alignment 的位置，可以查看 QR Code Spec 的第 81 页的 Table-E.1 的定义表（下表是不完全表格）：

Table E.1 — Row/column coordinates of center module of Alignment Patterns

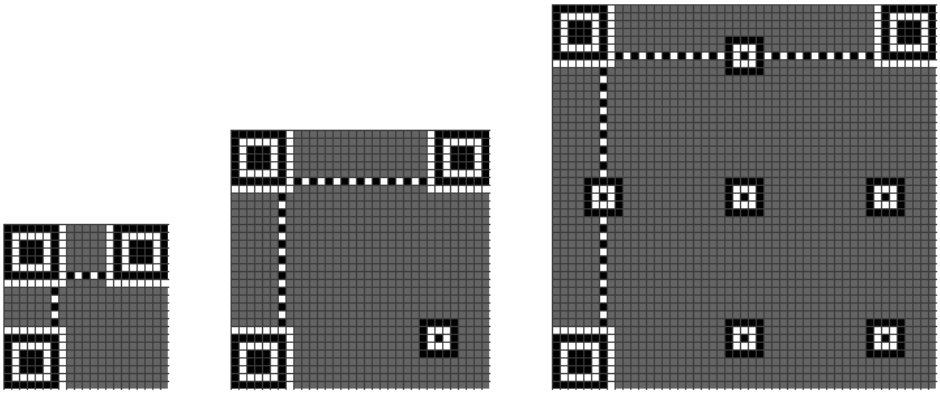
Version	Number of Alignment Patterns	Row/Column coordinates of center module						
1	0	-						
2	1	6	18					
3	1	6	22					
4	1	6	26					
5	1	6	30					
6	1	6	34					
7	6	6	22	38				
8	6	6	24	42				
9	6	6	26	46				
10	6	6	28	50				

下图是根据上述表格中的 Version8 的一个例子（6，24，42）：



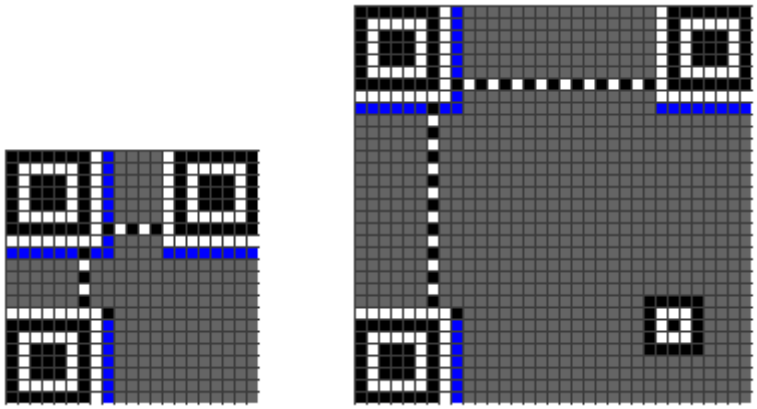
(3) Timing Pattern

接下来是 Timing Pattern 的线。

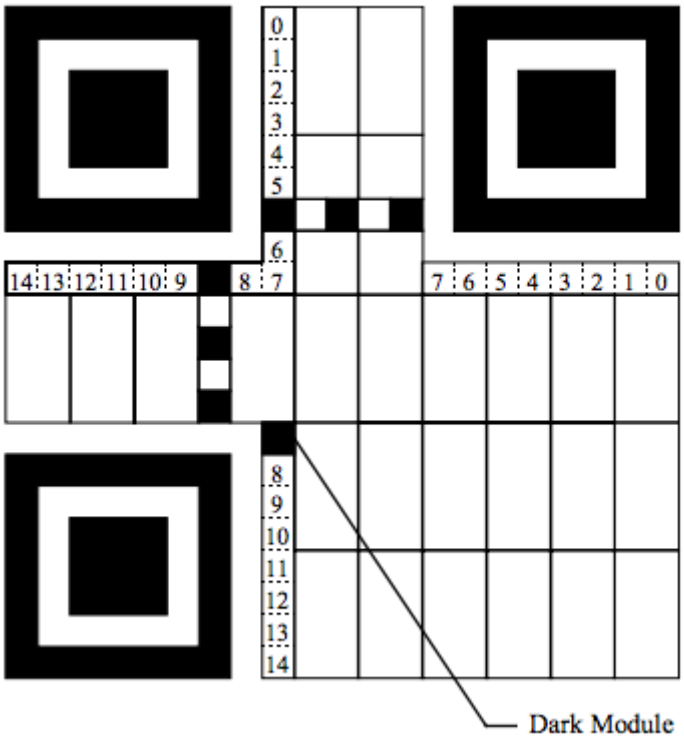


(4) Format Information

再接下来是 Formation Information，下图中的蓝色部分：



Format Information 是一个 15 个 bits 的信息，每一个 bit 的位置如下图所示：（注意图中的 Dark Module，那是永远出现的）



这 15 个 bits 中包括：

- **5 个数据 bits：** 其中，2 个 bits 用于表示使用什么样的 Error Correction Level，3 个 bits 表示使用什么样的 Mask；
- **10 个纠错 bits：** 主要通过 BCH Code 来计算。

然后 15 个 bits 还要与 101010000010010 做 XOR（异或）操作。这样就保证不会因为我们选用了 00 的纠错级别和 000 的 Mask，从而造成全部为白色，这会增加我们的扫描器的图像识别的困难。

下面是一个示例：

Assume Error Correction Level M:	**00**
and Mask Pattern Reference:	**101**
Data:	**00101**
BCH bits:	**0011011100**
Unmasked bit sequence:	**001010011011100**
Mask pattern for XOR operation:	**101010000010010**
Format Information module pattern:	**100000011001110**

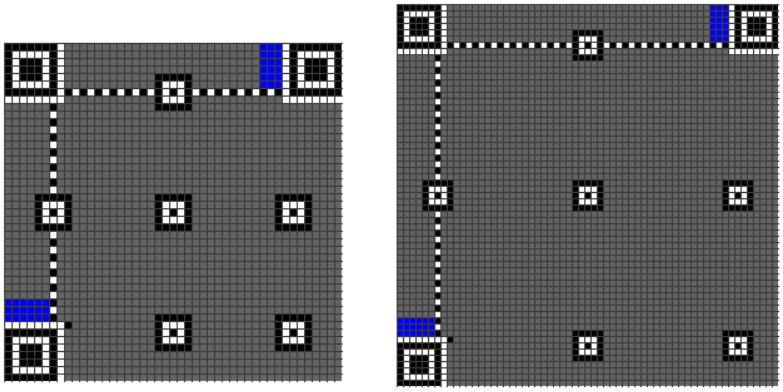
关于 Error Correction Level 如下表所示：

Error Correction Level	Binary indicator
L	01
M	00
Q	11
H	10

关于 Mask 图案如后面的 Table 23 所示。

（5）Version Information

再接下来是 Version Information（版本 7 以后需要这个编码），下图中的蓝色部分：



Version Information 一共是 18 个 bits，其中包括 6 个 bits 的版本号以及 12 个 bits 的纠错码，下面是一个示例：

Version number:	**7**
Data:	**000111**
BCH bits:	**110010010100**
Format Information module pattern:	**000111110010010100**

而其填充位置如下：

0	3	6	9	12	15
1	4	7	10	13	16
2	5	8	11	14	17

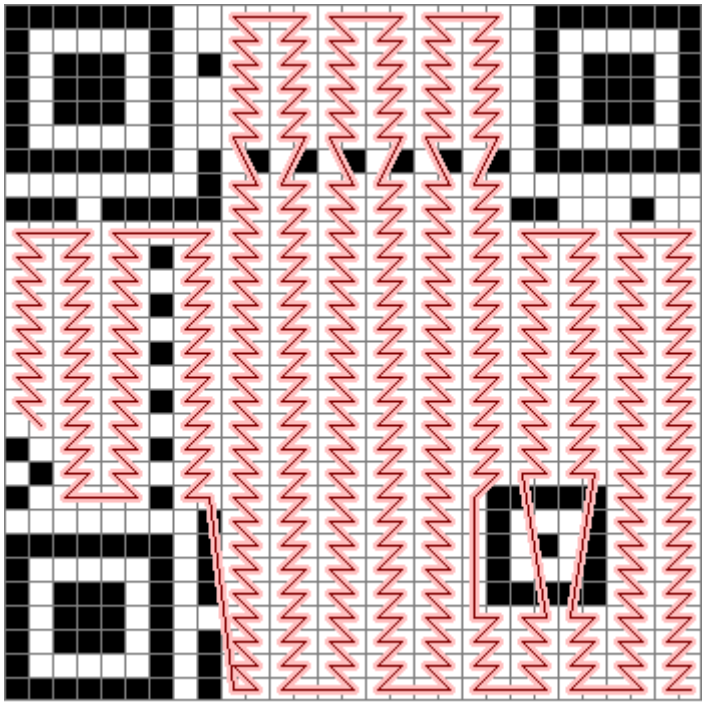
Version Information in lower left

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14
15	16	17

Version Information in upper right

（6）数据和数据纠错码

然后是填接我们的最终编码，最终编码的填充方式如下：从左下角开始沿着红线填我们的各个 bits，1 是黑色，0 是白色。如果遇到了上面的非数据区，则绕开或跳过。



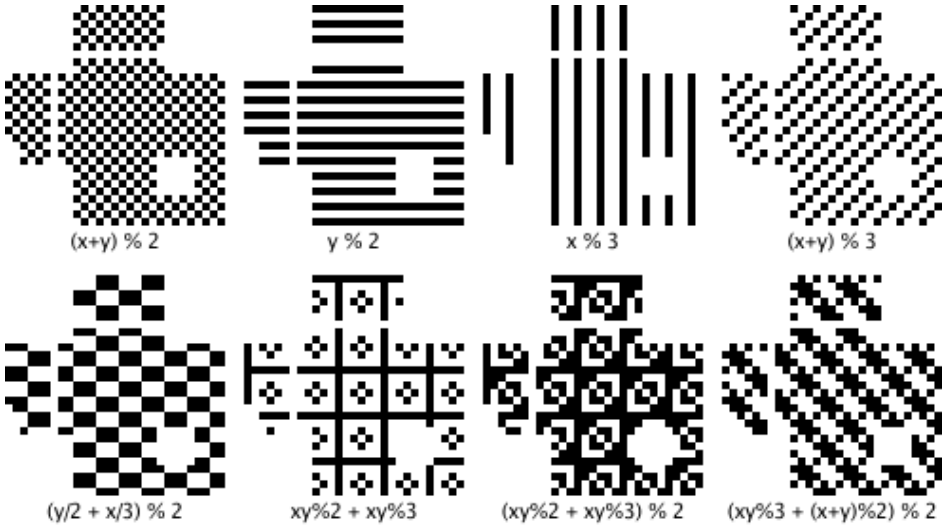
QR v3 order, from bottom right

（7）掩码图案

这样下来，我们的图就填好了，但是，也许那些点并不均衡，如果出现大面积的空白或黑块，会告诉我们扫描识别的困难。所以，我们还要做 Masking 操作。QR 的 Spec 中说了，QR 有 8 个 Mask 你可以使用，如下所示。

其中，各个 mask 的公式在各个图下面。所谓 mask，说白了，就是和上面生成的图做 XOR 操作。Mask 只会和数据区进行 XOR，不会影响功能区。

注：选择一个合适的 Mask 也是有算法的。

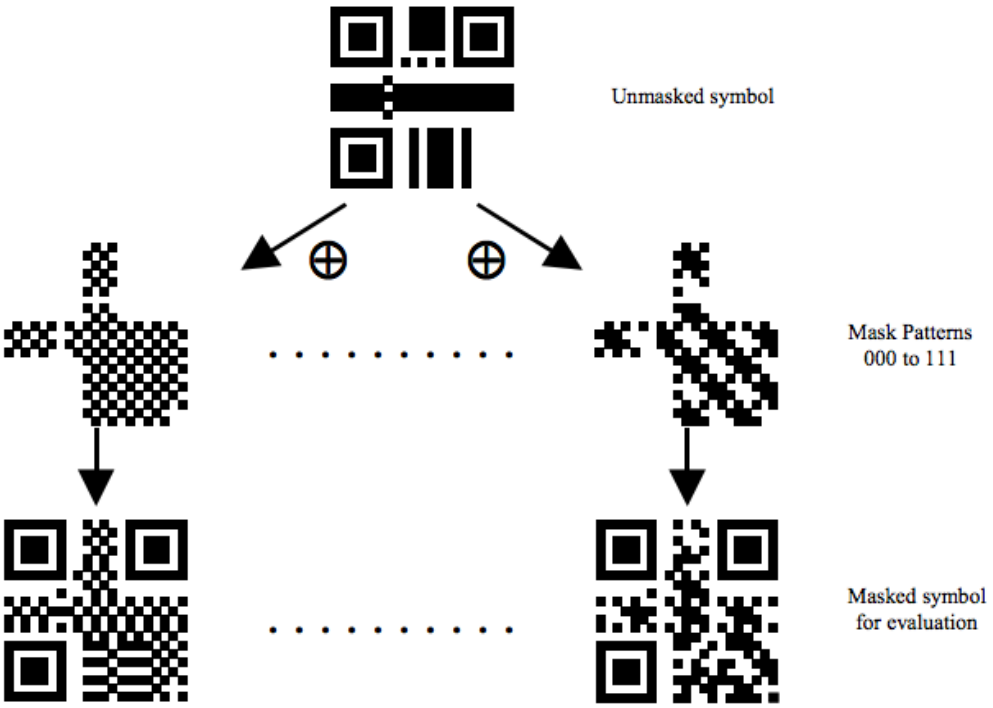


其 Mask 的标识码如下所示：（其中的 i, j 分别对应于上图的 x, y）

Table 23 — Mask pattern generation conditions

Mask Pattern Reference	Condition
000	$(i + j) \bmod 2 = 0$
001	$i \bmod 2 = 0$
010	$j \bmod 3 = 0$
011	$(i + j) \bmod 3 = 0$
100	$((i \text{ div } 2) + (j \text{ div } 3)) \bmod 2 = 0$
101	$(i j) \bmod 2 + (i j) \bmod 3 = 0$
110	$((i j) \bmod 2 + (i j) \bmod 3) \bmod 2 = 0$
111	$((i j) \bmod 3 + (i+j) \bmod 2) \bmod 2 = 0$

下面是 Mask 后的一些样子，我们可以看到被某些 Mask XOR 了的数据变得比较零散了。



Mask 过后的二维码就成最终的图了。