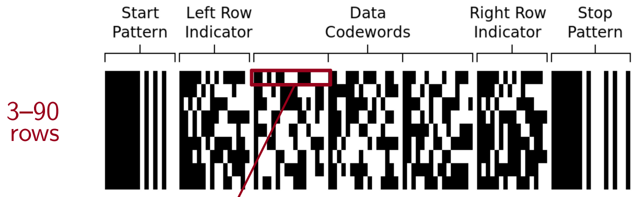
7. Applications of RS Codes: Some Examples

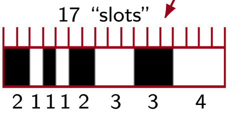
Application: PDF417 bar code

PDF417: A multi-row, 1D bar code (*PDF: Portable Data File*).

PDF417 bar code structure



In bar-code jargon, the whole array is referred to as a *symbol*



Codeword: an alternating pattern of 4 *bars* and 4 *spaces*, of varying widths, satisfying some constraints (e.g. width ≤ 6). Total width: 17; 417 comes from $4+17$.

- Basic global parameters (height, width, ECC level, etc.) are encoded in the *left* and *right row indicators*. A form of *repetition coding* (one copy per row).
- Consecutive rows use different sets of bar/space patterns (codewords). Each set has 929 codewords; 3 disjoint sets are used cyclically.
- Number of rows: $3 \leq h \leq 90$. Number of codewords per row: $1 \leq w \leq 30$ (all rows have the same number of codewords).
- Total number of codewords (all rows): $n \leq 928$.
- Using fixed tables, each codeword is mapped to a number in $\{0, 1, \ldots, 928\}$, and *interpreted as an element of* $\mathrm{GF}(929)$ (929 is prime).

PDF417: Codeword mapping

Table H1. The Bar-Space Sequence Table. Cluster 0

bsbsbsbs	val	bsbsbsbs	val	bsbsbsbs	val	bsbsbsbs	val	bsbsbsbs	val	bsbsbsbs	val
31111136	0	41111144	1	51111152	2	31111235	3	41111243	4	51111251	5
21111326	6	31111334	7	21111425	8	11111516	9	21111524	10	11111615	11
21112136	12	31112144	13	41112152	14	21112235	15	31112243	16	41112251	17
11112326	18	21112334	19	11112425	20	11113136	21	21113144	22	31113152	23
11113235	24	21113243	25	31113251	26	11113334	27	21113342	28	11114144	29
21114152	30	11114243	31	21114251	32	11115152	33	51116111	34	31121135	35
41121143	36	51121151	37	21121226	38	31121234	39	41121242	40	21121325	41
31121333	42	11121416	43	21121424	44	31121432	45	11121515	46	21121523	47
11121614	48	21122135	49	31122143	50	41122151	51	11122226	52	21122234	53
31122242	54	11122325	55	21122333	56	31122341	57	11122424	58	21122432	59
11123135	60	21123143	61	31123151	62	11123234	63	21123242	64	11123333	65
21123341	66	11124143	67	21124151	68	11124242	69	11124341	70	21131126	71
31131134	72	41131142	73	21131225	74	31131233	75	41131241	76	11131316	77
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

PDF417: Error correction

- An *error correction level*, s, $0 \leq s \leq 8$, is defined.
- The sequence of codewords (all rows) is interpreted as a *code block* in a $[k+r,\ k,\ r+1]$ shortened Reed Solomon code over $\mathrm{GF}(929)$, where
  - k is the number of codewords used for actual data.
    - Raw data is mapped to codewords using various efficient modes depending on whether the data is numeric, text, binary, or mixed.
    - One bar code can encode more than 1100 raw bytes, 1800 ASCII characters, or 2700 decimal digits, depending on the mode.
  - $r = 2^{s+1}$, so $r \in \{2, 4, 8, 16, 32, 64, 128, 256, 512\}$.
  - $k + r \leq 928$.
- 2 check digits are reserved for *detection*; the rest (if any) are used for *erasure* and *full error* correction.
- The *generator polynomial* of the RS code is

$$g(x) = \prod_{i=1}^{r} (x - 3^i)\,,$$

3 *is primitive in* $\mathrm{GF}(929)$.

Application: QR codes

Version 1: 21×21

Version 3: 29×29

Version 10: 57×57

Version 40: 177×177

A truly 2D, highly versatile bar code (array referred to as a *symbol*)

Application: QR codes

Widespread use

- Product or part tracking (original motivation)
- Web links
- Restaurant menus
- Tickets
- Document verification
- ... etc.

Robust ECC allows for data recovery under significant damage, and also for graphic art customization.

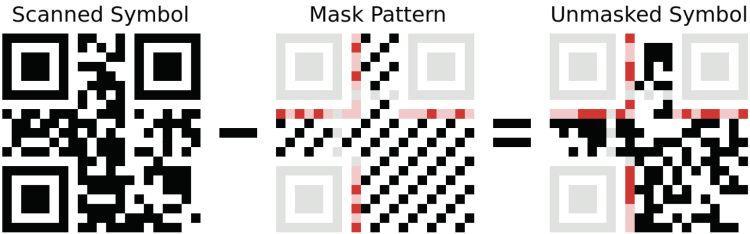
Fully recoverable symbols

QR codes: Versions (= Sizes)

Version	Size	Capacity	Version	Size	Capacity	Version	Size	Capacity	Version	Size	Capacity
M1	11	4½	8	49	242	19	93	991	30	137	2185
M2	13	10	9	53	292	20	97	1085	31	141	2323
M3	15	16½	10	57	346	21	101	1156	32	145	2465
M4	17	24	11	61	404	22	105	1258	33	149	2611
1	21	26	12	65	466	23	109	1364	34	153	2761
2	25	44	13	69	532	24	113	1474	35	157	2876
3	29	70	14	73	581	25	117	1588	36	161	3034
4	33	100	15	77	655	26	121	1706	37	165	3196
5	37	134	16	81	733	27	125	1828	38	169	3362
6	41	172	17	85	815	28	129	1921	39	173	3532
7	45	196	18	89	901	29	133	2051	40	177	3706

Capacity = number of main data bytes (including ECC)

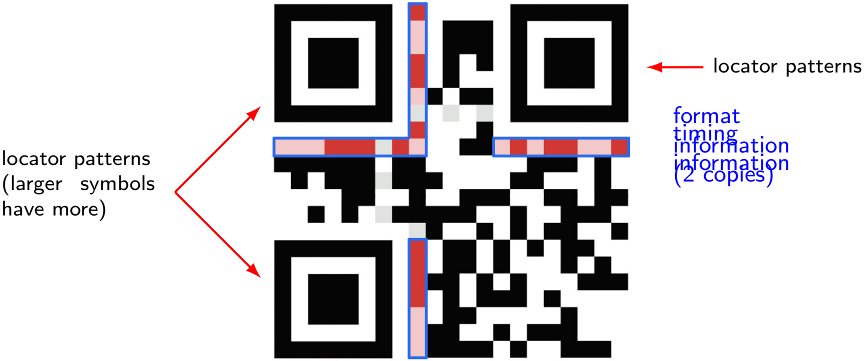
QR codes: masking



- An XOR mask is applied by the encoder to the raw data to minimize undesirable features (large areas of the same color, etc.).
- Several masks are tried, and the resulting array is scored for bad features. Mask with the best score is chosen.
- The choice is encoded in the symbol.

QR codes: structure

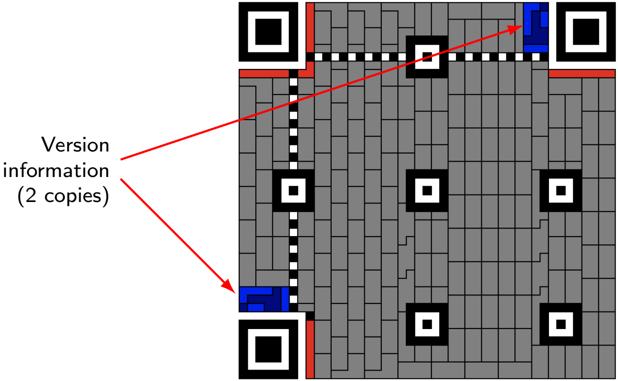
Version 1 symbol: 21×21



Format areas (2 copies): 5 bits of information, encoded with a $[15, 5, 7]$ binary BCH code (small code, exhaustive decoding possible). Format info (5 bits):

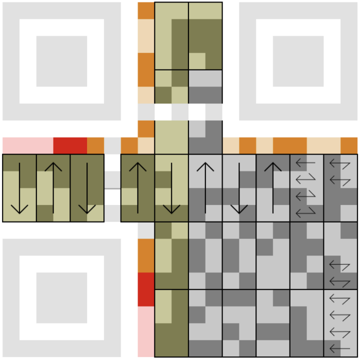
- 2 bits: error correction level (4 levels: L, M, Q, H).
- 3 bits: masking pattern.

QR codes: structure



Larger symbols (Version 7: 45×45 and higher) also carry *version information*: 6 bits, encoded with a binary $[18, 6, 8]$ code.
The code is derived from the $[23, 12, 7]$ (perfect) Golay code by taking the even codewords ($[23, 11, 8]$) and shortening.
As with format information, two copies are written.

QR codes: main data with error correction



Data is encoded using *shortened RS codes* over $\mathrm{GF}(256)$.

ECC Level	$n, n-k$ for 21×21 symbol	redundancy in general case
L	$26,\ 7$	$\approx 14\,\%$
M	$26, 10$	$\approx 30\,\%$
Q	$26, 13$	$\approx 50\,\%$
H	$26, 17$	$\approx 60\,\%$

For larger symbols:

- Data is broken up into multiple RS blocks (41×41 and larger)
- RS block length is limited so that $n-k \leq 30$ (complexity)
- RS blocks are *interleaved*

Examples:

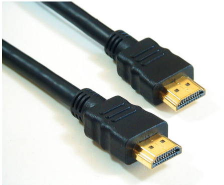
vers.	array size	ECC level	message bytes	num. blocks $\times(n, n-k)$	ECC level	message bytes	num. blocks $\times(n, n-k)$
10	57×57	L	274	$2\times(86, 18)$ $2\times(87, 18)$	Q	154	$6\times(43, 24)$ $2\times(44, 24)$
40	177×177	L	2956	$19\times(148, 30)$ $6\times(149, 30)$	Q	1666	$34\times(54, 30)$ $34\times(55, 30)$

Other applications

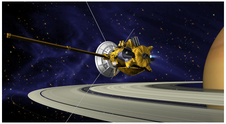
≈ 1mm-wide cut
CD still plays normally

Other applications

Magnetic tape data storage



HDMI protocol



Space communication

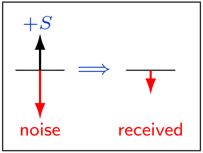
... and many more ...

Is coding any good?

- Coding protects bits against noise, but ...
  - We need to send more bits, which are also exposed to noise.
  - If we have a limited energy budget, each bit gets less energy than in the uncoded case, which makes it more vulnerable to noise.
  - Is the trade-off worth it?

- Simple physical model for BSC channel
  - Each bit is sent as an electrical signal of amplitude S: $\begin{array}{ccc} +S & \rightarrow & 0 \\ -S & \rightarrow & 1 \end{array}$
  - The signal is affected by additive Gaussian noise of zero mean and variance $\sigma^2 = 1$ (by choosing appropriate scaling for S).
  - A bit is flipped by the channel if the noise exceeds S in the "wrong direction". This has probability

$$p_{\text{bit}} = \frac{1 - \text{erf}(S/\sqrt{2})}{2}.$$



  - The *signal to noise ratio* is given by

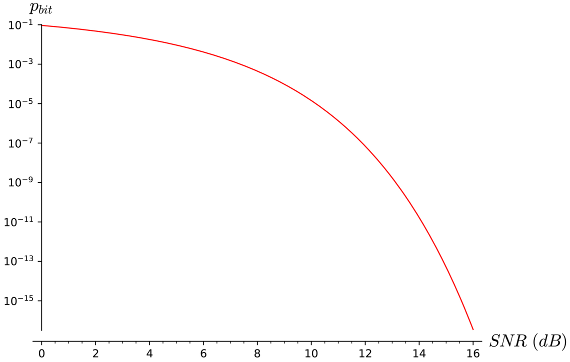
$$\text{SNR} = 10 \log_{10} \frac{S^2}{2R} \text{ dB},$$

where R is the code rate. This takes into account the "energy dilution" due to coding ($S^2 = 2R{\cdot}10^{\text{SNR}/10}$, lower $R \implies$ lower energy/bit).

Is coding any good?

$$p_{\mathsf{bit}} = \frac{1 - \mathsf{erf}(S/\sqrt{2})}{2}, \quad \mathsf{SNR} = 10 \log_{10} \frac{S^2}{2R}\ \mathsf{dB}.$$

These equations allow us to express p_{bit} as a function of SNR.



Is coding any good?

- Say we use an $[n, n-r, r+1]$ RS code over F_{2^m}, and we use a decoder that corrects up to $t = \lfloor r/2 \rfloor$ symbol errors (this is often not the best code to use for the BSC, but good enough for the point we are trying to make).
- The probability of a symbol being hit by noise is $p_{\text{symb}} = 1 - (1 - p_{\text{bit}})^m$.
- The probability of a code block *not* being decoded correctly is

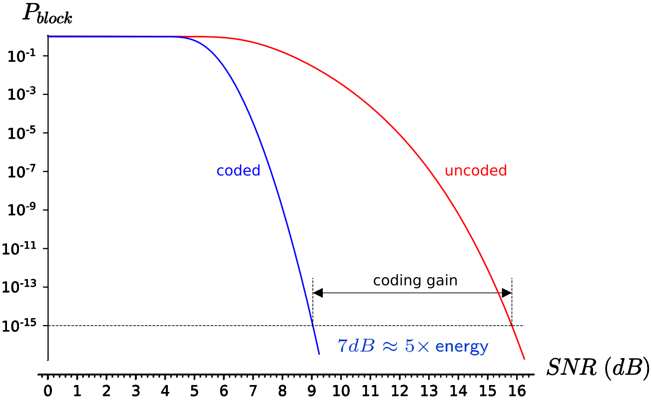
$$P_{\text{block}} = \sum_{i=t+1}^{n} \binom{n}{i} p_{\text{symb}}^i (1 - p_{\text{symb}})^{n-i} = 1 - \sum_{i=0}^{t} \binom{n}{i} p_{\text{symb}}^i (1 - p_{\text{symb}})^{n-i}.$$

  These sums may be tricky to compute numerically. Suggestions:
  - Compute terms in the log domain, exponentiate before adding.
  - Use gamma and log-gamma functions for binomial coefficients ($\Gamma(n+1) = n!$).
- As before, we can express p_{symb} and P_{block} as functions of SNR.
- In the *uncoded* case, we send raw blocks of $k = n - r$ symbols each, and the probability of a block being hit is

$$P_{\text{block}}^{\mathsf{U}} = 1 - (1 - p_{\text{symb}})^k.$$

Is coding any good?

- Example: RS code with $n = 128,\ r = 16$ over $\mathbb{F}_{2^8}$.



Yes, coding is very good!