Dependability and Fault Tolerance
Zuverlässigkeit und Fehlertoleranz

Chapter 4: **Code-Based Methods for Error Detection and Correction**

H. T. Vierhaus

Winter Semester 2018 / 2019

Error Correction Coding

Error correction coding (ECC)

Has found wide application for example for the protection of memory blocks from transient and from permanent errors.

Conditions: Relatively few error bits (e. g. 1 per byte or 1-2 for a 64 bit double word, often even 1-2 per 128 bit quad-word), but with only 1-2clock cycles for error detection and correction. Mainly using double error detection (DED) / single error correction (SEC) codes.

Forward error correction (FEC)

Finds very wide application in wireless communication.

Conditions: Errors are plenty and will often occur in clusters. Hence codes that facilitate multiple error detection (MED) and correction (MEC) are needed. Typically, timing constraints allow to spend hundreds or thousands of clock cycles on a single word or „symbol“.

Error Correction Coding

A large area in practical research since more than 50 years.

Application:

- **Hardware- Supervision**: Mostly by codes, which can detect single and double bit errors and can repair single bit errors rapidly.

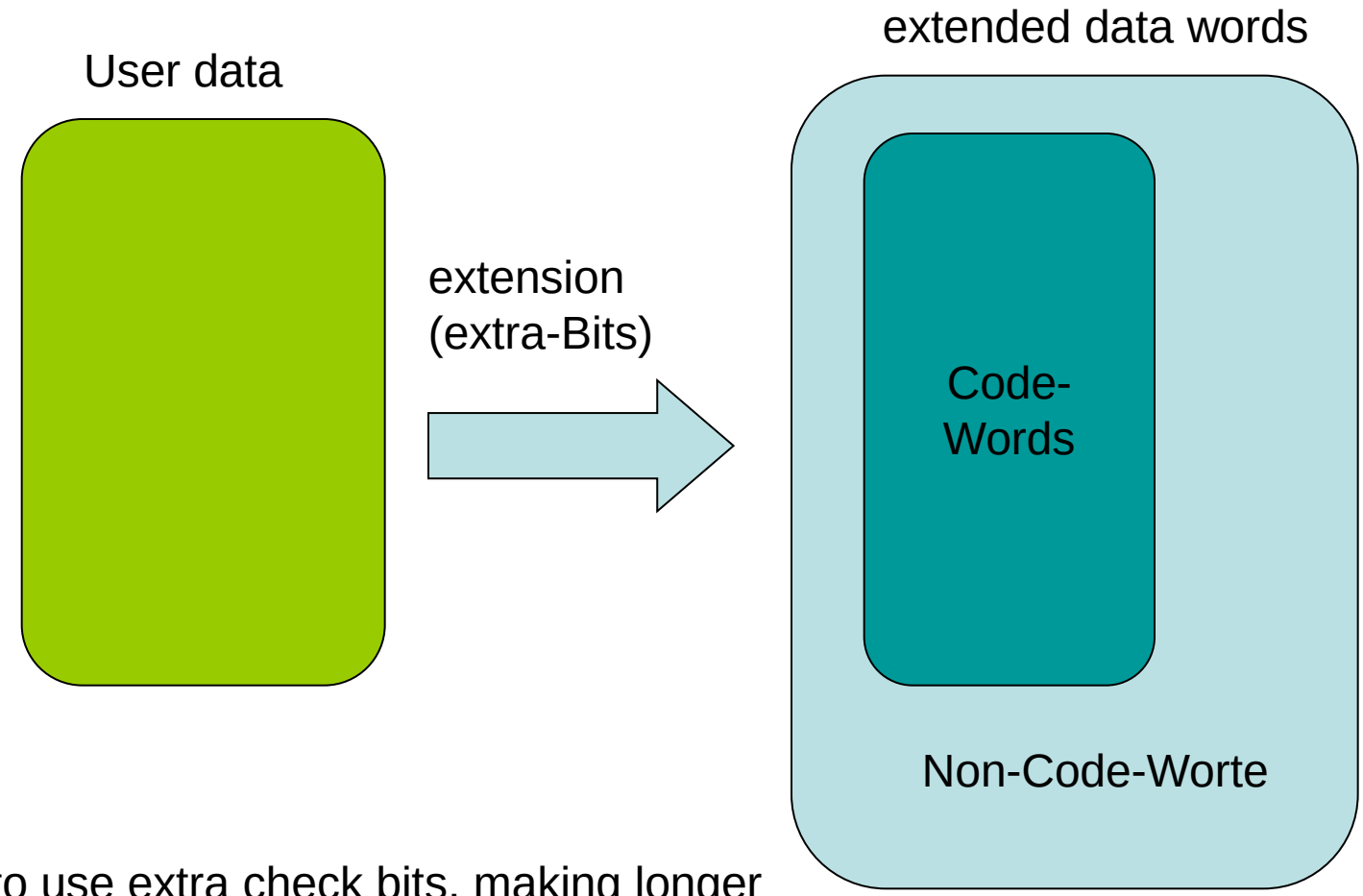
- **Memory-Supervsion**: Memory interface supervison, typically to detect single/ double bit errors, correct single bit errors. Double bit repair coming up !

Coding / decoding by hardware in nanoseconds !

- **Wireless Communication**: Typically requires codes that can perform multiple bit error detection / correction. Also for scratches on CDs / DVSs ! Mostly takes much longer time than a single clock cycle. Often implemented in software. May yield results that are mostly, but not always correct !

Coding / decoding may be in software in milliseconds !

Basics about Coding



We need to use extra check bits, making longer data words. But then only a fraction of the possible longer data words are valid code words !

Codes for Error Detection / Correction

A large area in practical research since more than 50 years.

Application:

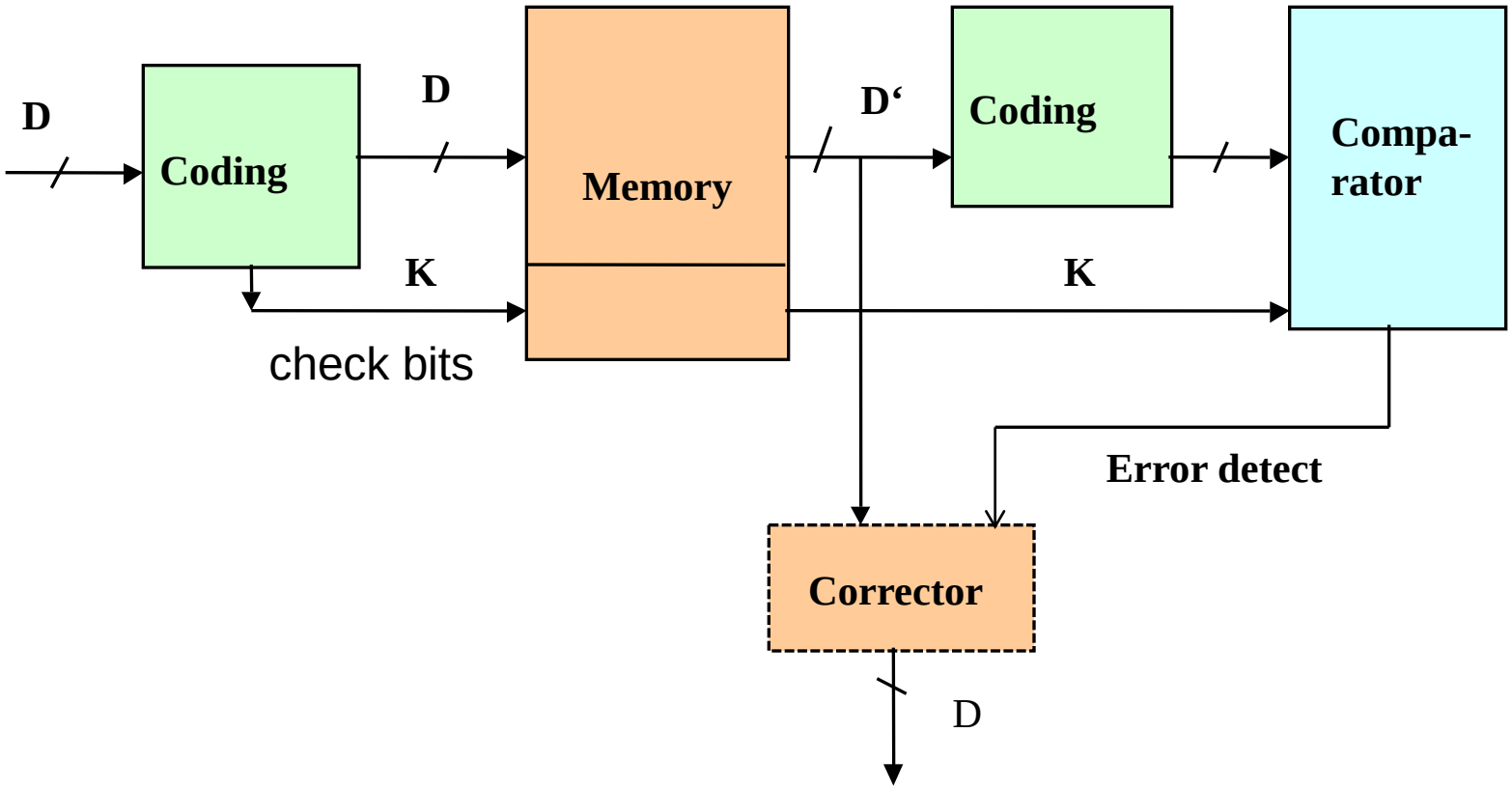
- **Hardware- Supervision**: Mostly by codes, which can detect single and double bit errors and can repair single bit errors rapidly.
- **Memory-Supervsion**: Memory interface supervison, typically to detect single/ double bit errors, correct single bit errors. Double bit repair coming up !

Coding / decoding by hardware in nanoseconds !

- **Wireless Communication**: Typically requires codes that can perform multiple bit error detection / correction. Also for scratches on CDs / DVSs ! Mostly takes much longer time than a single clock cycle. Often implemented in software. May yield results that are mostly, but not always correct !

Coding / decoding may be in software in milliseconds !

Error Detection by Coding



Error Correction for What ?

- Hardware faults that occur as transient faults or as permanent faults in operation. Fault density is relatively low, in most cases single bit faults.

  Problem: Error detection and correction must be done within very short time intervals, typically in the same clock cycle or (pipeline stalls) with 1-3 extra clock cycles.

  Typically „hard“ correction to deliver deterministic results.

- Errors due to noisy, disturbed communication channels. Occur rather frequently, most likely affecting multiple bits. Often errors occuring in clusters (such as scratches on CDs / DVSs).

  Problem: High numbers and high density of errors. May „overload“ the repair capabilities of a given code. False correction is always likely. Typically has time intervals of microseconds or milliseconds. Then done in software.

  Often „soft“ correction, best results achieved with „approximative correction“, based on majority votes.

Forward Error Correction (FEC)

Single bit error correction (SEC)
Double bit error detection (DED)

Hamming Code
Extended Hamming Code
Hsiao Code

May work in a single step of encoding / decoding, perform deterministic repair

Often used in memory interfaces

Multi bit error detection (MEC)
Multi bit error correction (MED)

Reed-Solomon code
BCH code
Turbo code
Low-densitiy parity code

Typically need multi time-steps for decoding, often using comparison and best guess

Often used in wireless communication

FEC is an old and well established science with decades of experience !

Block-Codes and Error Vector

We start with a volume X of digital vectors of equal length, e. g. 8 bits. There is a sub-volume Ψ of vectors which are code words. Vectors, which are in X but not in Ψ are not code words. A code is a block code, if all vectors in X and Ψ have the same length.

Correct data word: $v = v_1, v_2, ...v_n$

Faulty data word: $v' = v'_1, v'_2, ... v`_n$

The Error Vector is defined as:

$e = (e_1,.....e_n) = (v \oplus v') = (v_1 \oplus v'_1), ..., (v_n \oplus v'_n)$

For $e_i = 1$ the i-th components of v_i has a false value, for $e_i=0$ the i-th component is correct.

The number of 1-positions in the error vector is the number of bit errors.

Hamming-Weight and Hamming-Distance

The number w (v) of 1- positions in a binary vector $v = (v_1, ... v_n)$

$$w(v) = \sum_{i=1}^{n} v_i$$

is called the „Hamming Weight“.

The Hamming-Distance d (v, v‘) between 2 Data-Vectors $v = (v_1, ..., v_n)$ and $v' = (v'_1, ..., v'_n)$ is defined as the number of different bit positions between these 2 vectors.

$$d(v, v') = \sum_{i=1}^{n} (v_i \oplus v_n),$$

Hamming weight and Hamming Distance: $d(v, v') = w(v \oplus v')$

Parity and Hamming-Distance

Even Parity Bit

Good case :

P

0 1 1 0 0 1 1 0 1 | 1 | Number of 1-positions is odd (5)

Error case :

0 1 1 0 0 1 0 0 1 | 1 | Hamming-Distance 2

Should be:

0 1 1 0 0 1 0 0 1 | 0 | Number of 1s is even (4)

Error detection will possibly fail (not always!), if the number of error bits is equal to or higher than the Hamming distance.

Error Detection by Codes

An error e having the Hamming weight w (e) is defined as faking, w (e) bits of the affected code word..

For a given Code C the Hamming-Distance between any pairs of Code-words can be defined.

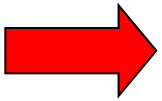
The smallest Hamming distance between 2 Code-words d_{min} is defined as the „Hamming-Distance“ of the code.

If a code has a distance of d_{min}, then no error having a Hamming-weight of $w(e) < d_{min}$ can change a Code-Word into an other Code-Word.
An error with equal or even larger w(e) can do so !

An error is not detectable, if it changes a code word into an other code word. By chance, even an error with a w(e) that is equal to or exceeds d_{min} may be detected „by chance“, if it ends up in a non-code-word.

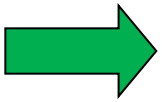
Error Detection

An erroneous date word is mapped to a valid code word:



Non error detecton !

An erroneous data word is mapped to a non-code word:



Error detection !

Note: Even an erroneous data word which contains too many faulty bits for the code has a chance to be mapped to a non-code word !

Rule: If N_c is the total number of Code-words, then there is a total of N_c-1 error cases, which are not detected. These are just those errors that change a code word into an other code word.

Hamming-Distance

The „Hamming weight“ **v** of a code word is the number of „1“ positions in the code word.

The „Hamming distance“ **d** between two code words is the minimum of bit positions, in which 2 code words differ from each other.

Every code has a minimum Hamming distance $\mathbf{d_{min}}$ between two of its code words.

The number of correctable errors in a code t_{kor} is related to $\mathbf{d_{min}}$:

$\mathbf{d_{min}} \geq 2 * t_{kor} + 1$ ⇒ 3 for singlebit error correction

The number of detectable errors in a code t_{erk} is also related to $\mathbf{d_{min}}$:

$\mathbf{d_{min}} \geq t_{erk} + 1$ ⇒ 2 for single bit error detection

⇒ $d_{min} \geq t_{kor} + t_{erk} + 1$

.. valid only for $t_{erk} < t_{kor}$

User bits	t_{erk}	t_{kor}	d_{min}
8	2	1	4

can correct 1-bit errors and detect 2-bit errors !

Block-Codes and Error Vector

We start with a volume X of digital vectors of equal length, e. g. 8 bits. There is a sub-volume Ψ of vectors which are code words. Vectors, which are in X but not in Ψ are not code words. A code is a block code, if all vectors in X and Ψ have the same length.

Correct data word: $v = v_1, v_2, ...v_n$

Faulty data word: $v' = v'_1, v'_2, ... v`_n$

The Error Vector is defined as:

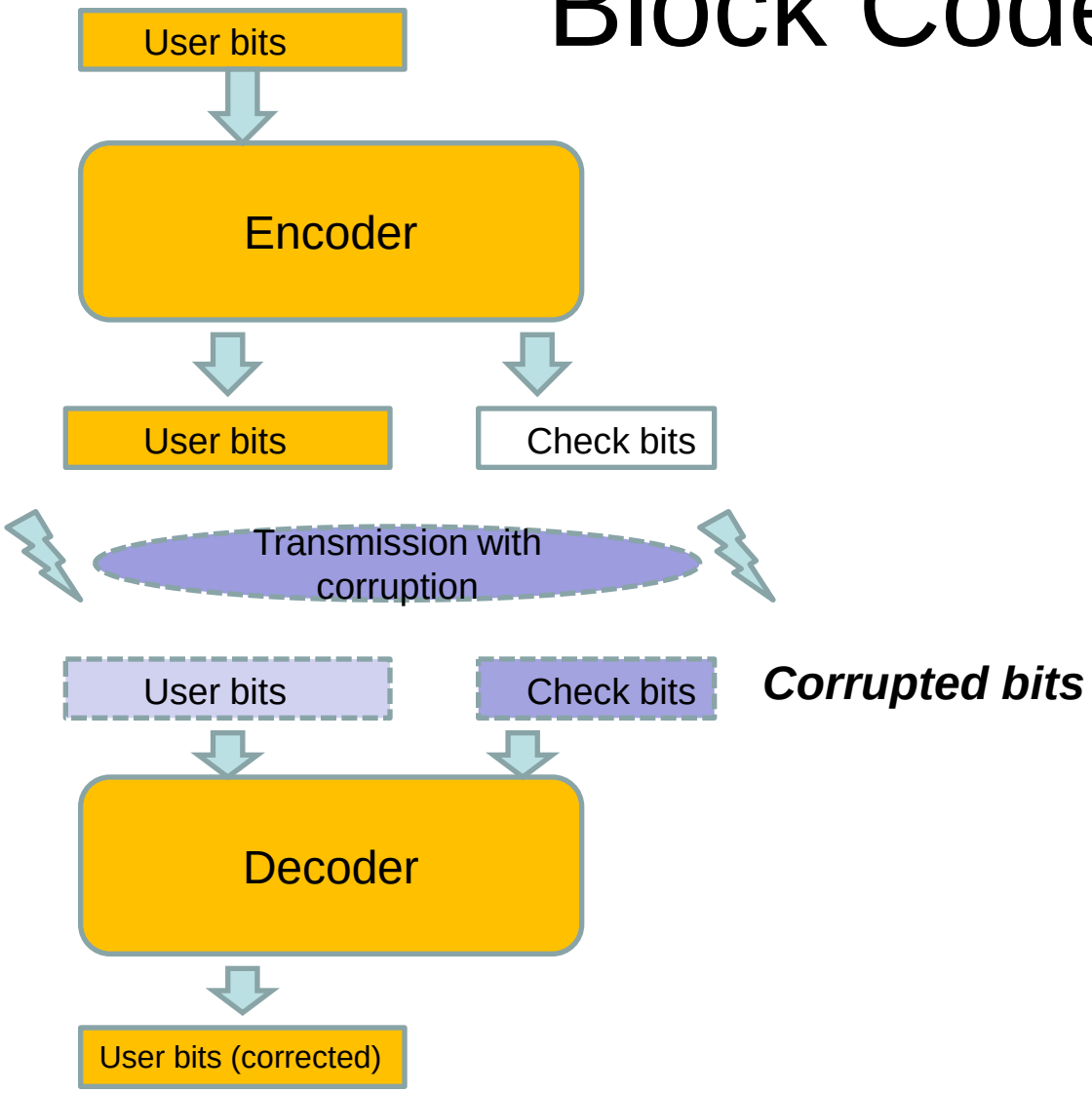
$e = (e_1,.....e_n) = (v \oplus v') = (v_1 \oplus v'_1), ..., (v_n \oplus v'_n)$

For $e_i = 1$ the i-th components of v_i has a false value, for $e_i=0$ the i-th component is correct.

The number of 1-positions in the error vector is the number of bit errors.

Most codes used in FEC are block codes !

Block Codes


User bits
Encoder
User bits
Check bits
Transmission with corruption
User bits
Check bits
Corrupted bits
Decoder
User bits (corrected)

Types of Codes

Linear Codes (parity, group parity, Hamming, BCH):

A specific type of error is definitly detectable or non-detectable.

Non-linear Codes: A specific error can be detectable or non-detectable, depending on what the rest of the code word looks like.

Linear Codes: The generation of check bits is done using the „linear“ XOR-function only.

Non-linear Codes: Check bits may be generated using AND, OR functions.

Information word: $u = (u_1, ...,u_k)$

Code-Wort: $v = (v_1,....., v_n)$

Generally: $n > k$

2 non-equal information words need to be mapped into 2 non-equal code words.

Systematic Block-Codes

(also named m- out of n-Codes)

Typical property: $u = (u_1,...,u_k)$ is the information word.

To generate the Code-Words, this word is expanded by the Check-Bits $c_1,...,c_l$.

The Code-Word of a systematic block-codes has the structure:
$v = (v_1,.... v_n) = (u_1,....u_k, c_1,....,c_l)$

Die Check-Bits $c_1,....c_l$ are derived from the information bits by Boolean functions $g_1,...., g_l$ of the length k.

$c_1 = g_1 (u_1,..., u_k)$, $c_2 =.....$, $c_l = g_l(u_1,...., u_k)$

with $v_1,....., v_k = u_1,....., u_k$

Linear Boolean function (with XOR, Inverter): linear codes

Non-linear Boolean function (AND, OR): non-linear codes

Parity-Codes

Parity-Bit: $c_p = u_1 \oplus u_2 \oplus ..... \oplus u_k$ for the even case

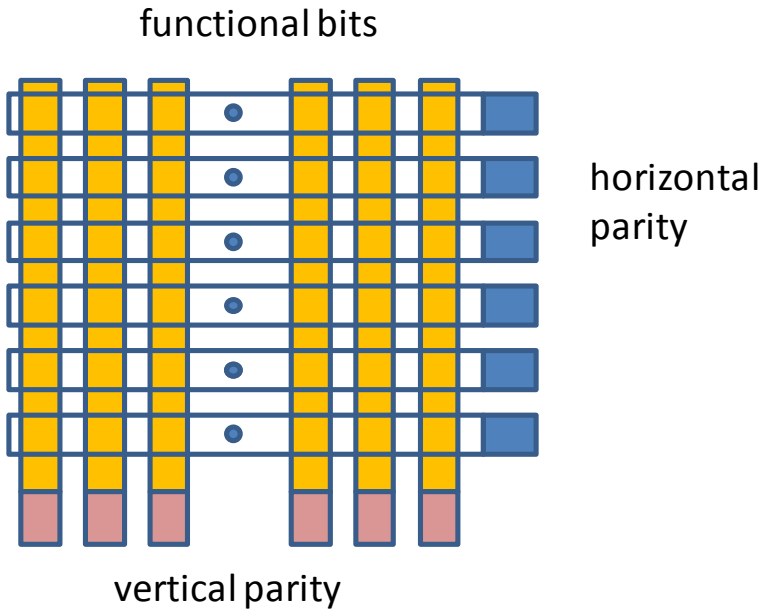
Parity-Bit: $c_p = \overline{u_1 \oplus u_2 \oplus ..... \oplus u_k}$ for the odd call

Even parity: All words of a length n = k+1 with an even number of 1s are Code-Words. All others are not.

Odd parity: Alle words of a lenght n = k+1 with an odd number of 1s are Code-Worte. All others are not.

Parity-Bits

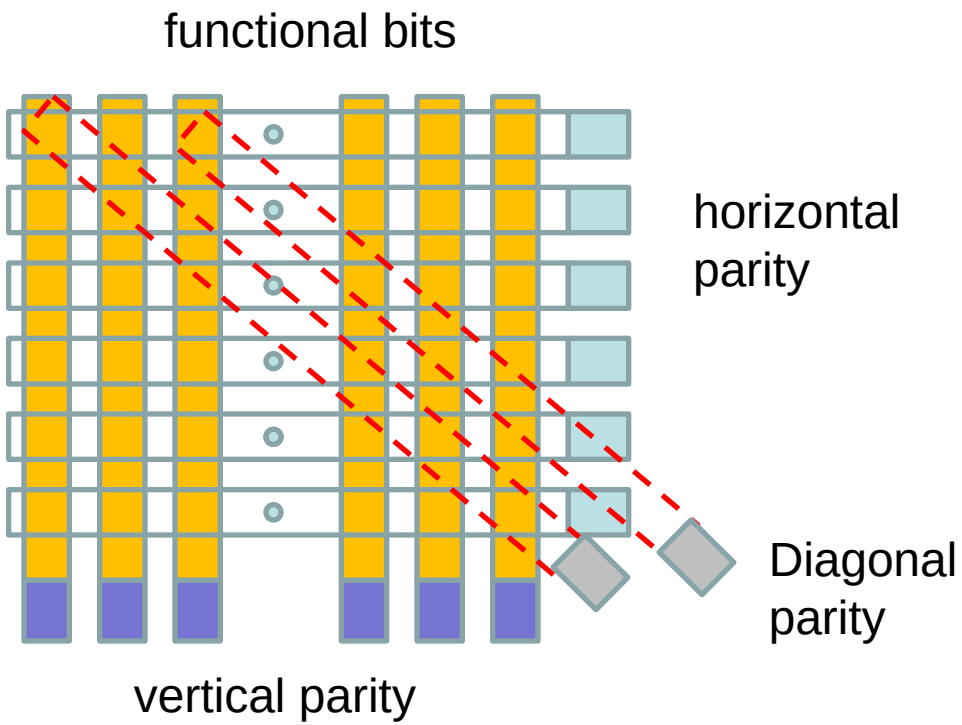
Using a single parity bit results in a Hamming distance of 2. This is enough only to detect single bit errors, but not enough to either detect 2-or more bit errors of to correct single bit errors !



A 2-dimensional scheme that used single bit error detection in 2 dimensions can detect double bit errors and correct single bit errors !

Rule: Hamming distances in 2 directions seem to multiply in such schemes !

Double Error Detection / Correction



By introducing a third „diagonal“ parity bit it is possible to detect and correct 2-bit errors and to detect 3-bit-errors !

Group-Parity-Codes

Information bits are structured into 2 or more groups.

Each of these groups gets ist own check bit.

$c_1 = u_{1,1} \oplus \ldots\ldots \oplus u_{1,k1}$

$\circ \quad \circ \quad \circ \quad \circ$

$c_l = u_{1,l} \oplus \ldots\ldots \oplus u_{l,kl}$

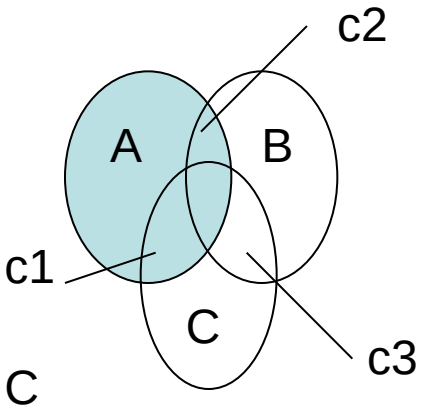
Each linear code is also a Group-Parity-Code. Mostly the term is used only if the groups are defined under specific conditions.

The groups may and very often will contain elements in common..

Example: Bytes of a larger data word may !

Group Parity Codes

A	B	C
1	0	0



Groups may be:

$c1 = A \oplus C$

$c2 = A \oplus B$

$c3 = B \oplus C$

$c4 = A \oplus B \oplus C$

Example: If c1, c2 indicate an error, not c3, A must be faulty
if c1, c3 indicate an error, not c2, C must be faulty
if c2, c3 indicate an error, not c1,B must be faulty

If any of c1, c2, c3 indicates an error, but not c4 (total parity), there is a 2-bit error !

Hamming Code

Check-Bits are calculated as follows (⊕ stands for XOR):

C1 = D1⊕ D2 ⊕ D4 ⊕ D5 ⊕ D7
C2 = D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D7
C4 = D2 ⊕ D3 ⊕ D4 ⊕ D8
C8 = ⊕ D5 ⊕ D6 ⊕ D7 ⊕ D8

Data-Bits: D1 bis D8
Check-Bits: C1, C2, C4, C8 stand for each „new“ bit position in binary data

Bit-position	12	11	10	9	8	7	6	5	4	3	2	1
Pos. Nr.	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C8				C4		C2	C1

Check-Bits in Hamming Code

Code Bits: C1, C2, C3,…. Cn is the total code word to be built.

Data-Bits: D1, D2, D3, …. Dk

Bit-Positions C1, C2, C4, C8, C16 are occupied by check bits.

Parity-Bits: P1= C1= D1 ⊕ D3 ⊕ D5 …… (alle odd data bits)
= C3 ⊕ C5 ⊕ C7 ⊕ C9….

P2 =C2 =C3 ⊕ C6 ⊕ C7 ⊕ C10 ⊕ C11 ⊕ C14 ⊕ C15 ⊕ C18 ⊕ C19….
= D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D7 ⊕ D10 ⊕ D11 ⊕ D13 ⊕ D14…

Rule: Start with the code bit next right from P2, that is C3. Leave the next 2 code bits, then take 2, leave 2….

P3 =C4 = C5⊕C6 ⊕C7⊕ C12 ⊕ C13 ⊕ C14 ⊕ C15 ⊕ C20

Rule: Start with the next 3 code bits right from P3, that is C5, C6, C7. Leave out the next 4, take next 4…

The parity bit P_i is built from bit positions C_j of the code word, which contain a „1“ in the binary coding of the index. C-bits, which are themselves check bits, are not taken.

Hamming Correction

Bit-position	12	11	10	9	8	7	6	5	4	3	2	1
Pos. Nr.	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Datenbit	D8	D7	D6	D5		D4	D3	D2		D1		
Check-Bit					C8				C4		C2	C1
Wort gesp.	0	0	1	1	0	1	0	0	1	1	1	1
Wort gelesen	0	0	1	1	0	1	1	0	1	1	1	1
Check bit fehlerfrei.					0				1		1	1
Check Bit bei Fehler					0				0		0	1

XOR

0		1		1	0	

Basic Scheme of Hamming Code

Bit-position	12	11	10	9	8	7	6	5	4	3	2	1
Pos. Nr.	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1

C12 C11 C10 C9 C8 C7 C6 C5 C4 C3 C2 C1

Separable Hamming-Code

In „serable Hamming code“ the code bits are separately appended at the end. This means a modificaton of the H- Matrix that controls the creation of the code wor, but the properties of the code remain.

For the „extended Hamming code“ there is an additional „total parity“ bit taken over all code bits (including the check bits).

If there is one or more group check bits set, but the overall parity bit is not triggered, this indicates a double-bit-error..

Hamming Overhead

User Bits	Hamming Bits	Ext. Hamming Bits	Overhead
8	4	5	50 / 62.5 %
16	5	6	31.25 / 37.5 %
32	6	7	18.75 / 21.8%
64	7	8	10.94 / 12.5%
128	8	9	6.25 / 7.03%
256	9	10	3.5 / 3.9 %

With a rising number of user bits in the word, the number of check bits grows only on a logarithmic scale. It is $\log_2 (k) + 1$ or + 2. However, with larger data word widths, the probability of an error grows on a linear scale.

Other codes that allow for the detection and correction of multiple bit errors are necessary, now also for memory interfaces !

Summary Hamming Code

- Hamming code is a linear group-parity block code that can detect and correct single bit errors. The Hamming distance is 3.
- With one single extra overall parity bit, the „extended“ Hamming code can detect double bit errors, but not correct them. Hamming distance is 4.
- Such a code is also named as a „Single Error Correcting“ (SEC)-„Double Error Detecting“-(DED)-code.
- If there are multi-bit errors, they may not be detected, but even false corrections are quite likely !
- SEC-DED-codes are popular, because they can be encoded or decoded in just one clock cycle for word lengths in hardware.

Simplified Hsiao Code

Hsiao-code is, just as the extended Hamming code, a SEC- DED-code.
It is also a linear group-parity block code.
It has become popular, because it is even faster than Hamming code in de-coding.
Therefore it is widely used on processor / memory interfaces (ECC-Memories).

Properties:
Deriving the check-bits (P1….Pk) from the user-bits (C0…Cn) is determined by the H matrix:

Check bits	User bits C0	C1	C2	C3	C4	C5	C6	C7
P1	1	0	0	0	1	0	1	1
P2	0	1	0	0	1	1	0	1
P3	0	0	1	0	1	1	1	0
P4	0	0	0	1	0	1	1	1

The H-Matrix shows, which user bit C0…Cn is used to generate which check bit P1… Pk.

$P1 = C0 \oplus C4 \oplus C6 \oplus C7$

$P2 = C1 \oplus C4 \oplus C5 \oplus C7$

$P3 = C2 \oplus C4 \oplus C5 \oplus C6$

$P4 = C3 \oplus C5 \oplus C6 \oplus C7$

Rules defining the Hsiao code:

The number of 1-positions in each column of the H-matrix must be odd (1, 3, 5…).

The number of 1-positions in the rows of the H-matrix must not differ by more than 1.
The total number of 1-positions should be a minimum.

Real Hsiao-Code

	C0	C1	C2	C3	C4	C5	C6	C7
P1	1	0	0	0	1	0	1	1
P2	0	1	0	0	1	1	0	1
P3	0	0	1	0	1	1	1	0
P4	0	0	0	1	0	1	1	1

P1 = C0⊕C4 ⊕C6 ⊕C7

P2 = C1⊕C4 ⊕C5 ⊕C7

P3 = C2⊕C4 ⊕C5 ⊕C6

P4 = C3⊕C5 ⊕C6 ⊕C7

Single bit errors in the user bits C0…C7 are mapped on 1 or 3 check bits..

Double bit errors in the user bits will trigger either 2 or 4 check bits and therefore they become detectable.

But now single bit errors on check bits P1… P4.may trigger false rapair actions !

We need to modify the H-Matrix such that every single bit error will activate exactly 3 check-bits.

	C0	C1	C2	C3	C4	C5	C6	C7	
P1	1	0	0	1	1	1	1	0	5
P2	1	1	0	1	0	0	0	1	4
P3	1	1	1	0	1	0	1	0	5
P4	0	1	1	0	1	1	0	1	5
P5	0	0	1	1	0	1	1	1	5
Sum	3	3	3	3	3	3	3	3	

We need one parity bit more, but then single bit errors on parity bits can no longer trigger false repaits. Neither can 2-bit errors.
Problem: 3-bit- errors can not even be detected safely and can still trigger false repairs !

Hsiao Code Matrix Including Check Bits

	User Bits								Check Bits				
	C0	C1	C2	C3	C4	C5	C6	C7	P1	P2	P3	P4	P5
P1	1	0	0	1	1	1	1	0	1	0	0	0	0
P2	1	1	0	1	0	0	0	1	0	1	0	0	0
P3	1	1	1	0	1	0	1	0	0	0	1	0	0
P4	0	1	1	0	1	1	0	1	0	0	0	1	0
P5	0	0	1	1	0	1	1	1	0	0	0	0	1

Hsiao Code / Triple Errors

Triple error false mapping for 32 bit vectors

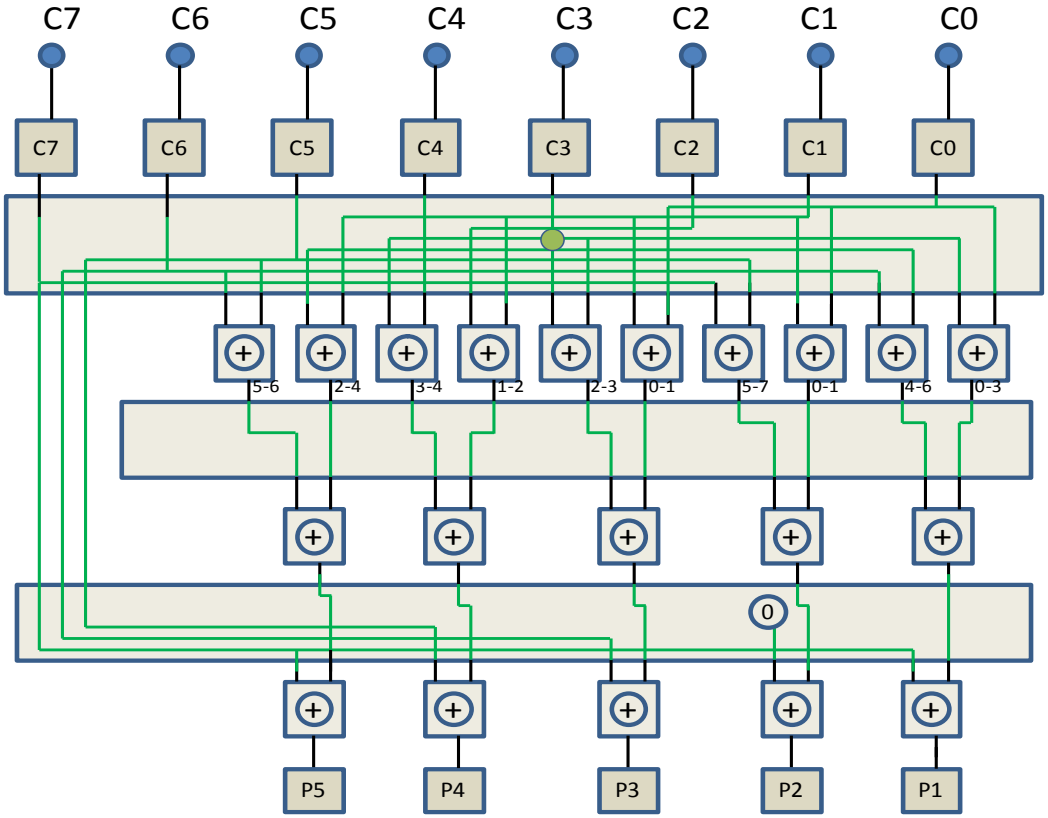
Parity bits per funct. bit	Error detected (%) at no. of error bits					Errors (%) Corrected	False corrections (%) at no. of error bits			
	1	2	3	4	5	1-bit	2	3	4	5
3	100	100	100	98	100	100	0	60	0	11
5	100	100	100	99	100	100	0	30	0	6

Full triple error detection for 8 user bits:

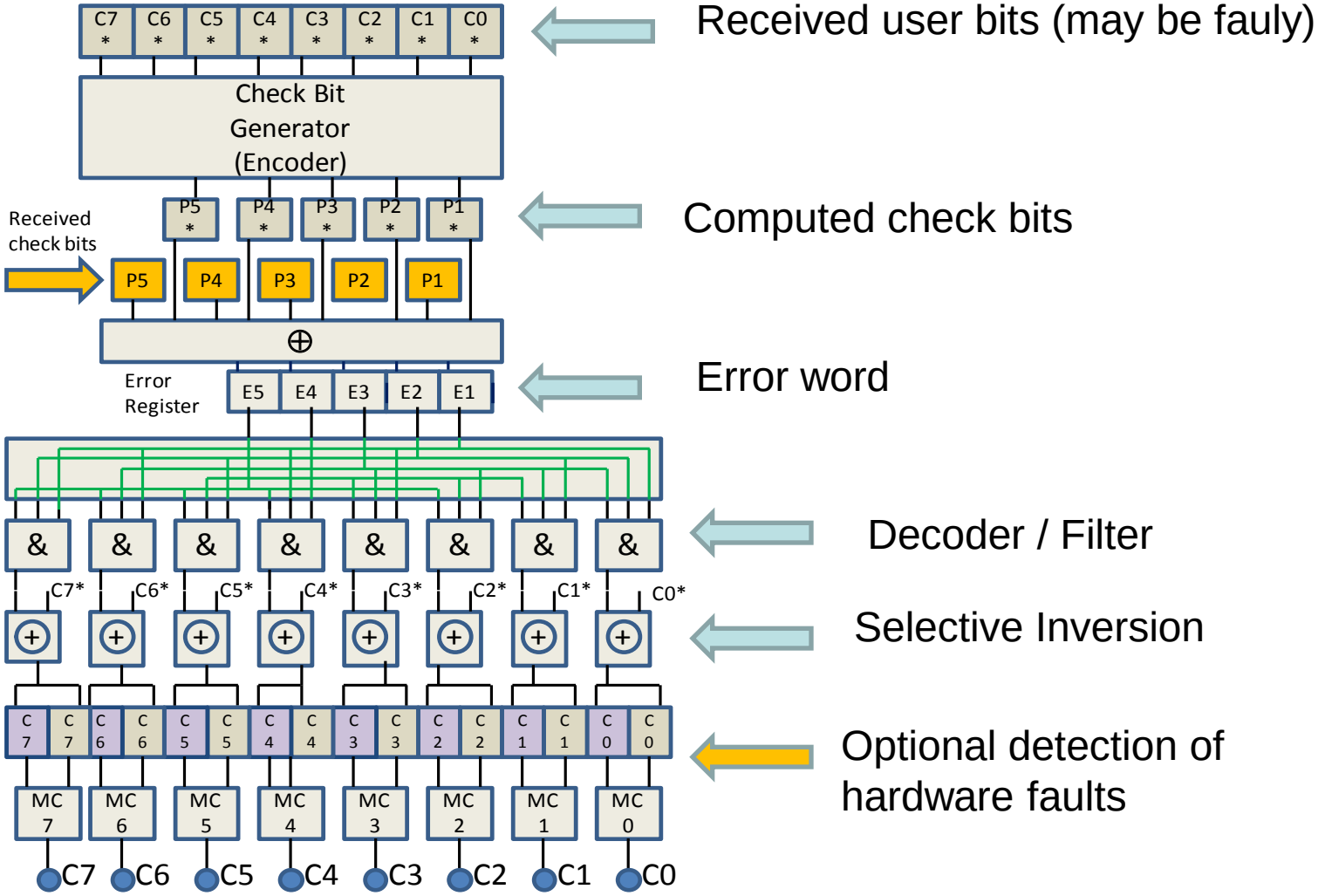
- 5 parity bits, 3 „1“s per column (3 parity bits needed for single bit correction)
- 4 extra parity bits (even, odd, special groups)

Looks *like full triple error detection* needs 4 extra parity bits in the general case beyond (extended) Hsiao !

Regular Hsiao Encoder



Regular Hsiao Decoder



Hardware Faults in Encoders / Decodes

- Hardware faults in encoders will emerge as single or multiple bit errors in the user bits or the check bits. They may partially be correctable, but they will reduce error correction capabilities in the decoder a lot.
- Hardware faults in decoders may be treated as if they were single or multiple bit errors in the user bits and the check bits.
- Properly designed decoders may detect and correct their own hardware faults, but may fail under multiple fault conditions.
- Hsiao-code encoder / decoder can be designed to become self testing and (mostly) self-correcting with a reasonable overhead !

Practically Important Codes

- Parity Codes
- Group-Parity-Codes
- Duplication Code — Duplication and Comparison
- Two-Rail-Codes — Duplication with inversion
- Berger Codes — Counts 0 / 1 positions in the code word
- Modulo p-Code
- m aus n codes. — Out of n bit-positions in the code word only m bit-positions can be „1“.

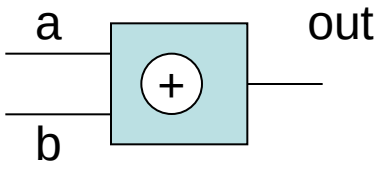
In modulo-p-codes the arithmetic value of r is used. This is the remainder of the division of the number of 1- or 0-positions n by p. Example: n=5 1-positions, p = 3 gives a remainder of 5 - 2*3 = 2, for n=13 and p =3 we get 13-4*3 = 1.

Non-Linear Codes

Non-linear codes have a much less elaborated mathematical theory than linear codes. Check bits can be generated using non-linear AND and OR functions instead of only XORs. Typically they it is not possible to prove that a k-bit error in a n-bit code word is safely detected / repaired, but that depends on the other bits of the code word as well.

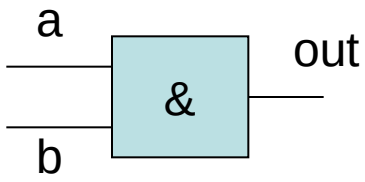
For non-linear codes, typically the „probability“ of error detection has to be given ! For known examples, this probability can become very high.

Linear



Out is always depending on a and b.

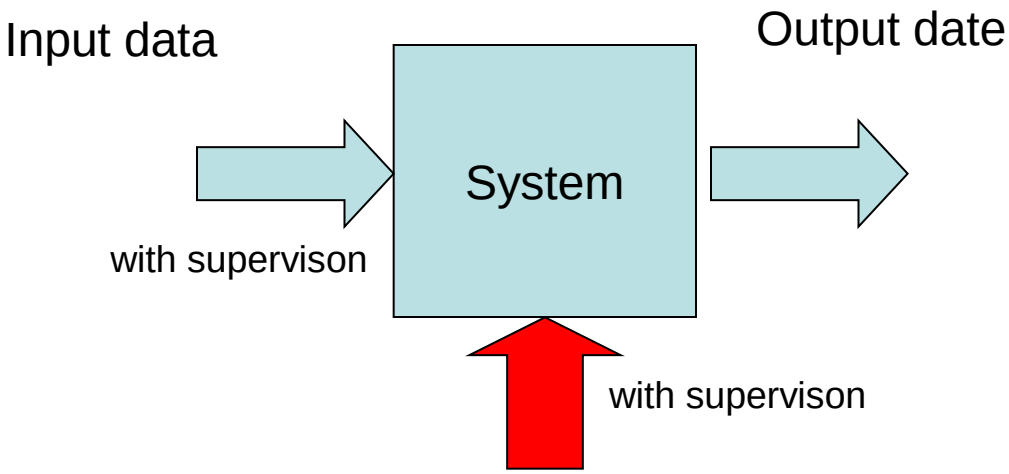
Non-linear



In case of a=0 we get out=0 independenltly from b

In case of b=0 we get out=0 independently from a

Code-Disjoint-Circuits

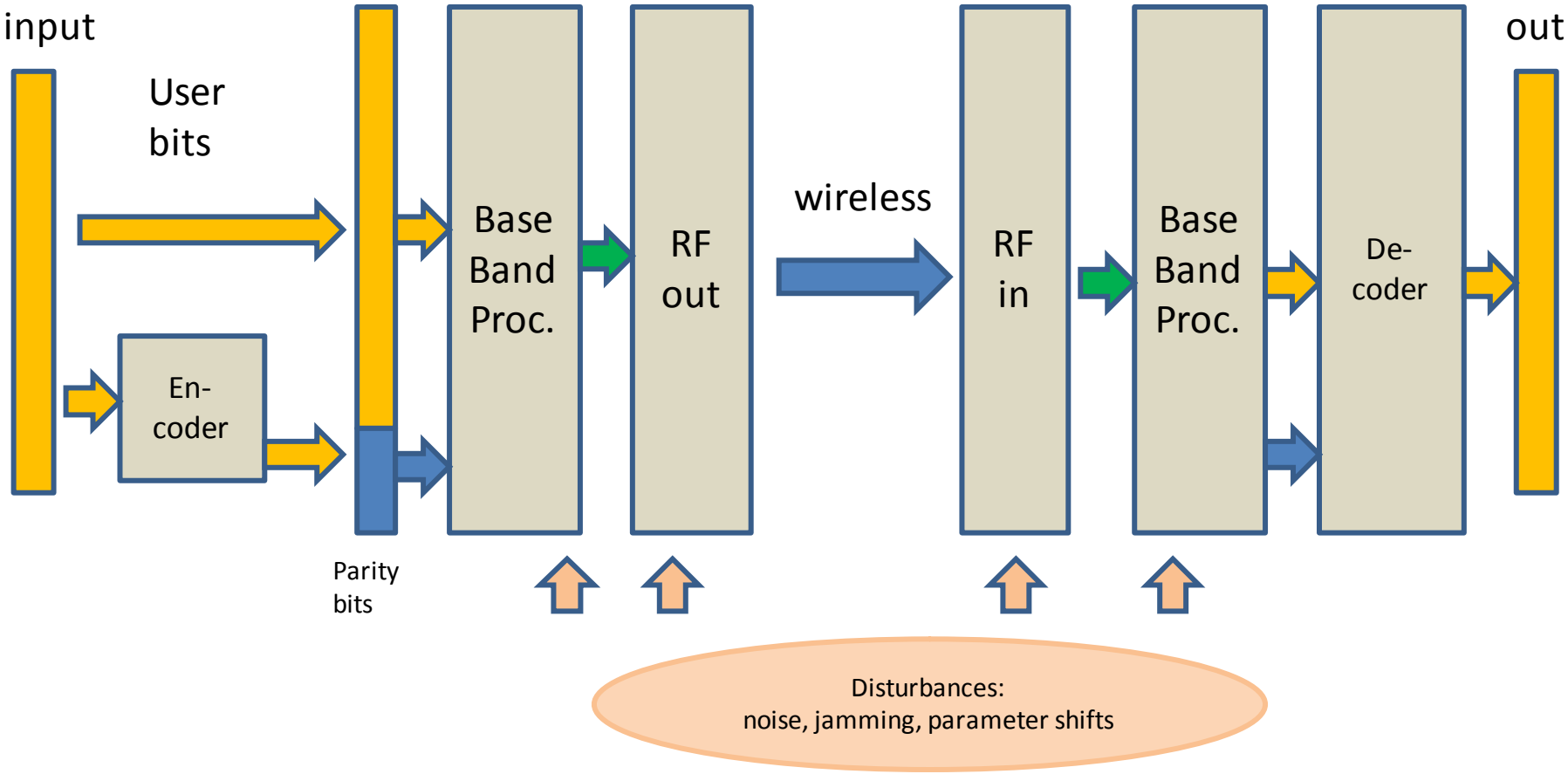


Code- disjoint methods are used to detect not only mal-functions of the system, but also faulty input data.
Input- and subsequently output data are encoded.

Mappings:

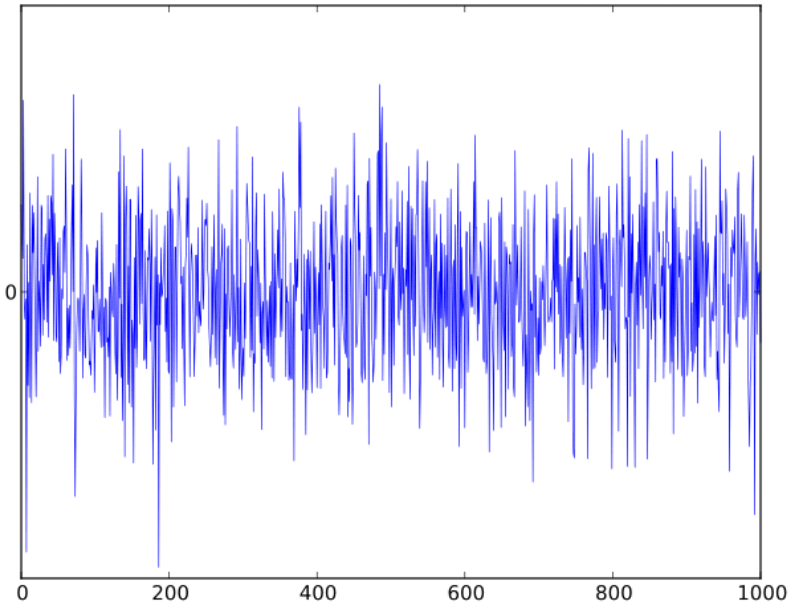
- *Input code words to output code words*
- *Input non-code words to output non-code words*

Wireless Communication Systems



… typically require multi-bit error detection / correction !!

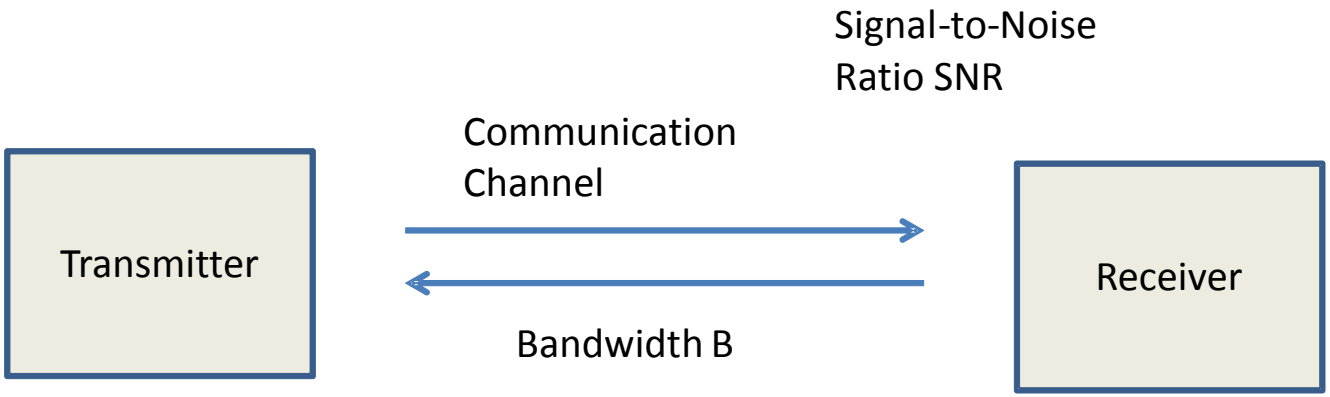
Noise



There are „irregular“ random electric signals in each type of hardware and (even more) on wireless communication channels. The sources are solar radiation, thermic radiation of materials, man-made signals.

In communications it is an important challenge to receive even signals which are „burried in noise“. In digital communications, some bits may be detected with the false logic value. And in many cases bits are of the type „don´t know“ !!

Shannon-Limit



Shannon Limit for Channel Capacity in Bit / s :
$C = B \log_2 (1+SNR)$

$SNR[db] = 10 \lg SNR$

This already requires the detection and correction of erroneous bits.

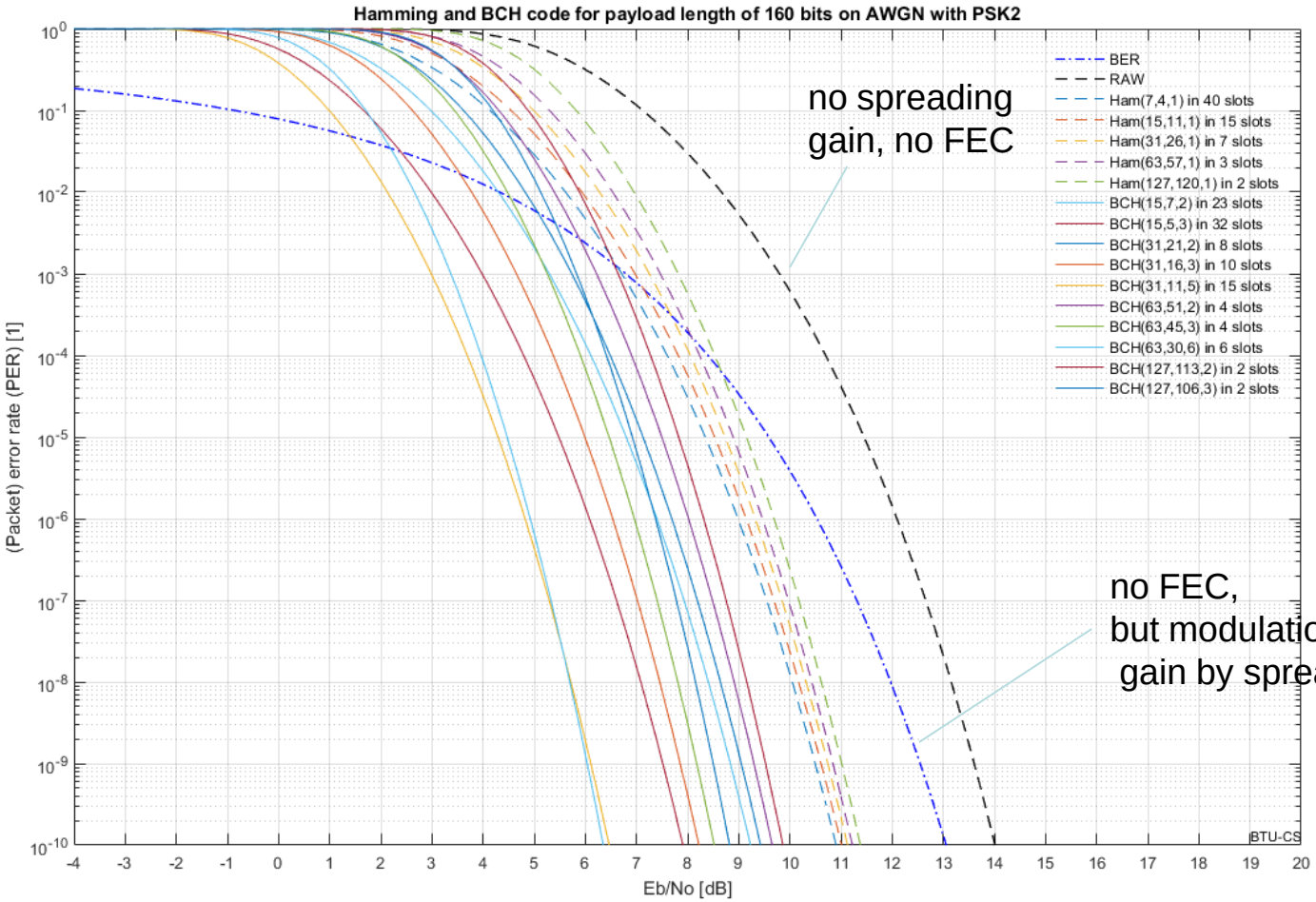
Known methods for error detection come quite close to this limit, but never reach it !

Known codes always have a limit with respect to the number of correctable errors. Therefore they produce errors inevitably in case of „too poor“ signal-to-noise-ratios.

Packet Error Rates

Assuming a symbol lenght of 256 bits that is en-/decoded as a whole or in slots.

error rate



no spreading gain, no FEC

no FEC, but modulation gain by spreading

Signal to noise ration (in db)

Codes (Hamming, BCH) and Overhead

Total bits	Data bits	Parity bits	Parity bits to data bits	Parity bits to packet length	Number of corrected bit errors	Note
15	11	4	36.4%	26.7%	1	(Hamming)
15	7	8	114.3%	53.3%	2	Limit for 128bit industrial packet fitted in one single PSSS255 Symbol
31	26	5	19.2%	16.1%	1	(Hamming)
31	21	10	47.6%	32.3%	2	
31	16	15	93.8%	48.4%	3	Limit for 128bit industrial packet fitted in one single PSSS255 Symbol
63	57	6	10.5%	9.5%	1	(Hamming)
63	51	12	23.5%	19.0%	2	
63	45	18	40.0%	28.6%	3	
63	39	24	61.5%	38.1%	4	
63	36	27	75.0%	42.9%	5	Limit for 128bit industrial packet fitted in one single PSSS255 Symbol
127	120	7	5.8%	5.5%	1	(Hamming)
127	113	14	12.4%	11.0%	2	
127	106	21	19.8%	16.5%	3	
127	99	28	28.3%	22.0%	4	
127	92	35	38.0%	27.6%	5	
127	85	42	49.4%	33.1%	6	
127	78	49	62.8%	38.6%	7	
127	71	56	78.9%	44.1%	9	
127	64	63	98.4%	49.6%	10	Limit for 128bit industrial packet fitted in one single PSSS255 Symbol
255	247	8	3.2%	3.1%	1	(Hamming)
255	239	16	6.7%	6.3%	2	
255	231	24	10.4%	9.4%	3	
255	223	32	14.3%	12.5%	4	
255	215	40	18.6%	15.7%	5	
255	207	48	23.2%	18.8%	6	
255	199	56	28.1%	22.0%	7	
255	191	64	33.5%	25.1%	8	
255	187	68	36.4%	26.7%	9	
255	179	76	42.5%	29.8%	10	
255	171	84	49.1%	32.9%	11	
255	163	92	56.4%	36.1%	12	
255	155	100	64.5%	39.2%	13	
255	147	108	73.5%	42.4%	14	
255	139	116	83.5%	45.5%	15	
255	131	124	94.7%	48.6%	18	Limit for 128bit industrial packet fitted in one single PSSS255 Symbol

Codes for Wireless Communication

***Hard decision methods*: Use algebraic methods for error detecion / correction**

BCH Codes: Describe data words and keys by polynomials.
Use characteristic features of linear feed-back shift registers (LFSRs) to perform polynomial division.
Detection and correction of multi-bit errors is possible by analyzing the remainder of a polynomial division.

Features: Fast encoding, *but lengthy and complex decoding* !

***Soft decison methods*: Identify „candidates“ for correct data words and decide by majority vote**

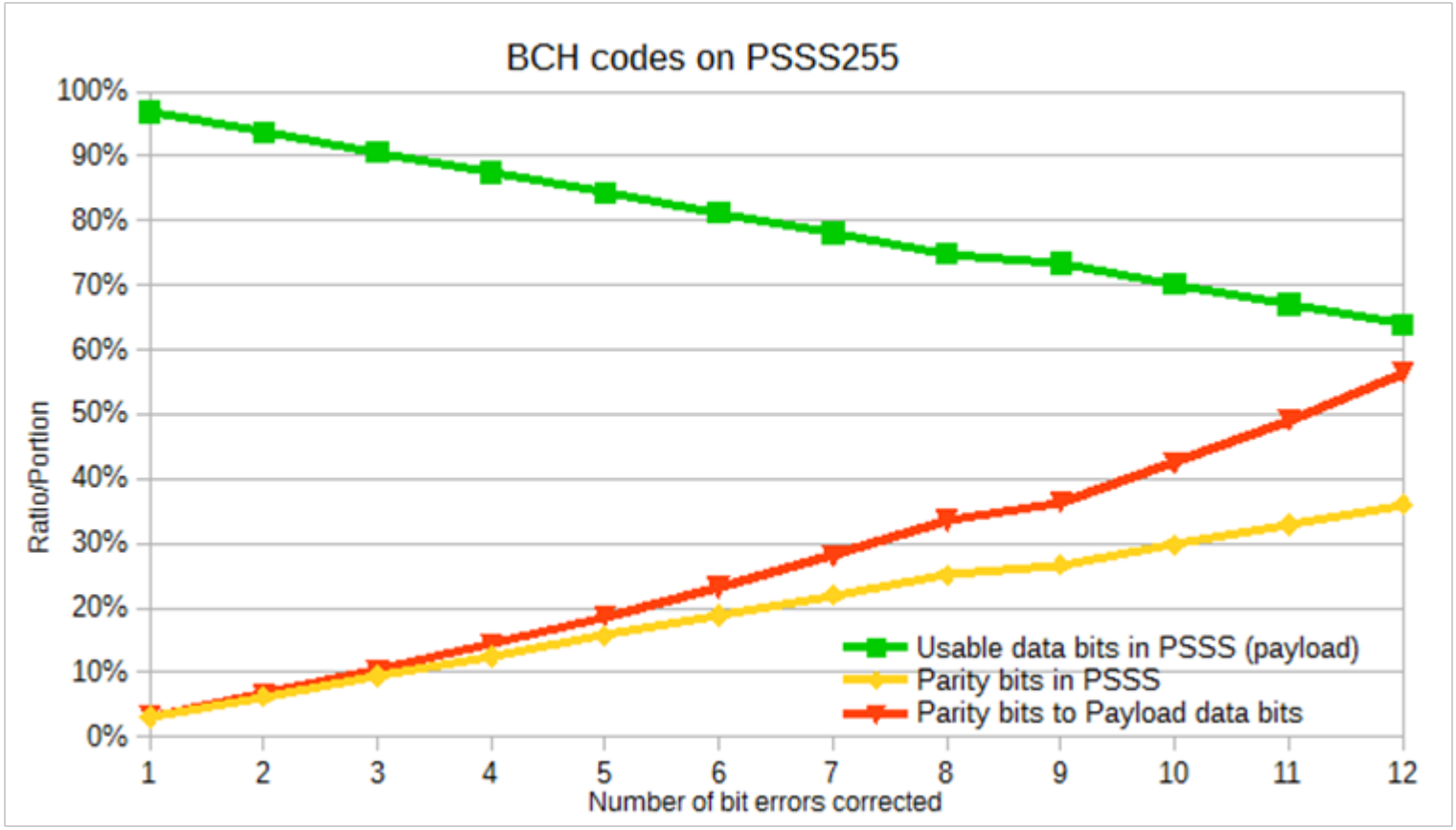
Examples: Turbo codes, low-density parity codes (LDPC).

Come close to the „Shannon Limit“ of retrieving information from a disturbed channel !

But decoders are slow (multiple clock cycles) and include lots of memory cells, which are susceptible to transient hardware faults !

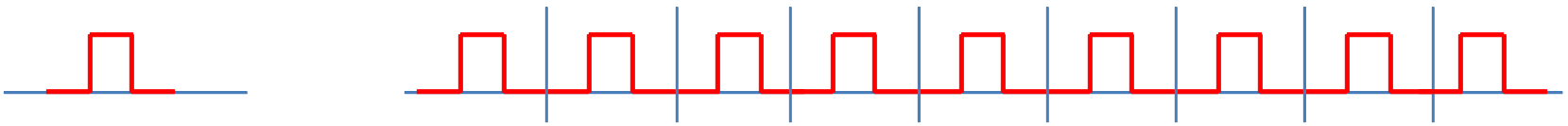
Cost of FEC

Total lenght of a symbol: 255 bits



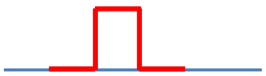
Bit Spreading / Gain

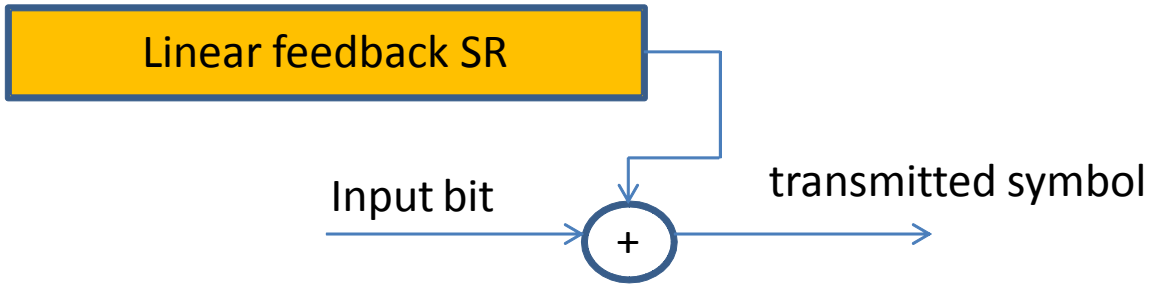
Simplest case:



A simple „1“ is transmitted as k „1“ values, possibly at a higher frequency. Decoding via majority vote makes a „spreading gain“.

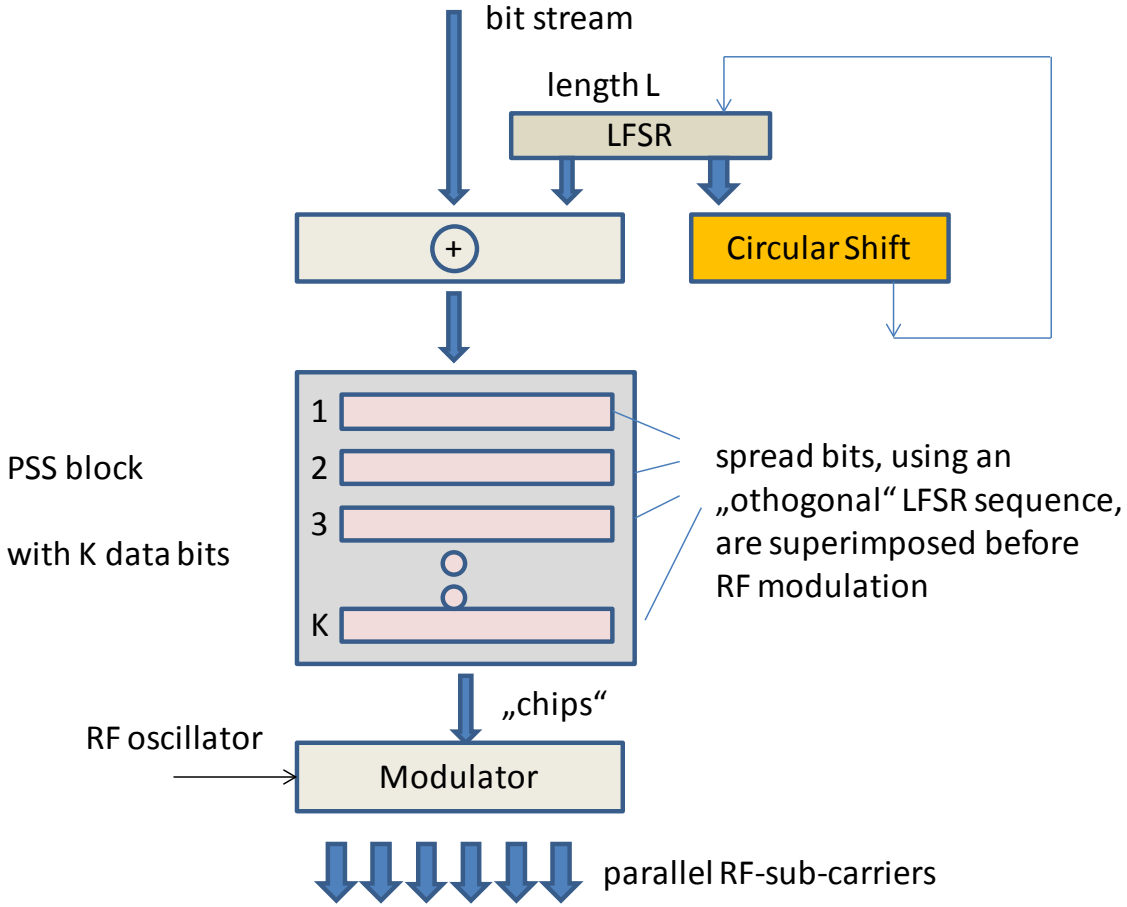
Sophisticated approach:





Each input bit is „XORed“ with the sequence produced by an LFSR. This sequence (or its inverse) can easily be detected by a correlation receiver.

Superposition of Patterns



This gives a much better usage of the RF carrier bandwith (user bits per Hz), but the modulation gain from spreading is lost !

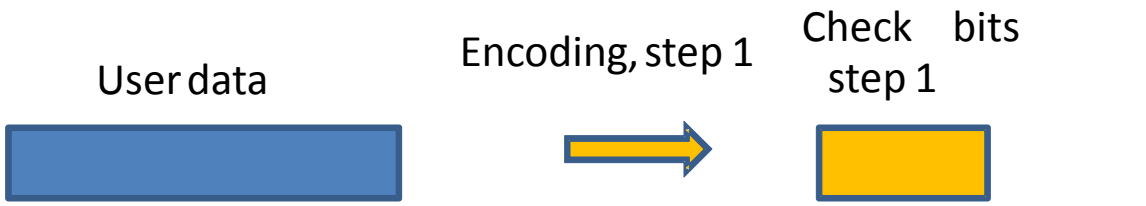
Modulation in Wireless Systems

Wireless communication systems have several partly contradictory requirements:

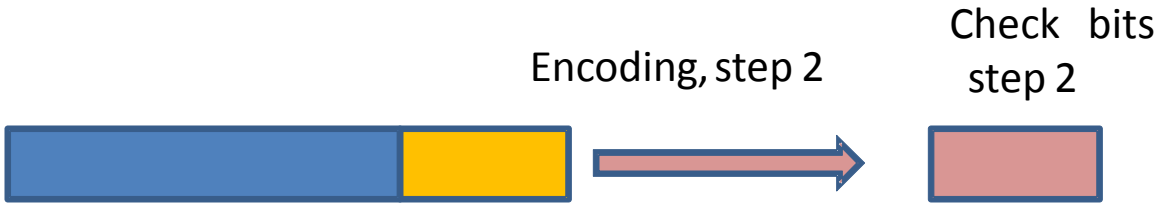
- Best usage of bandwidth indicated by the bits/ Hz ratio
- Transmission with a minimal error rate (i. e. 10^{-9})
- Not too complicated transmitters / receivers
- Good detection of „signals in noise“ using correlation receivers and FEC.

It has become a „standard“ to perform folding of each bit with a LFSR pattern and then superimpose several such patterns using i. e. analog adders. This requires that the LFSR sequences used have the property of being „orthogonal“. For wider LFSRs, only few of such LFSR sequences are known, and they are difficult to find. State-of-the-art methods achieve more than 20 bits / Hz. One such method is OFDM (orthogonal frequency domain multiplexing).

Product Codes



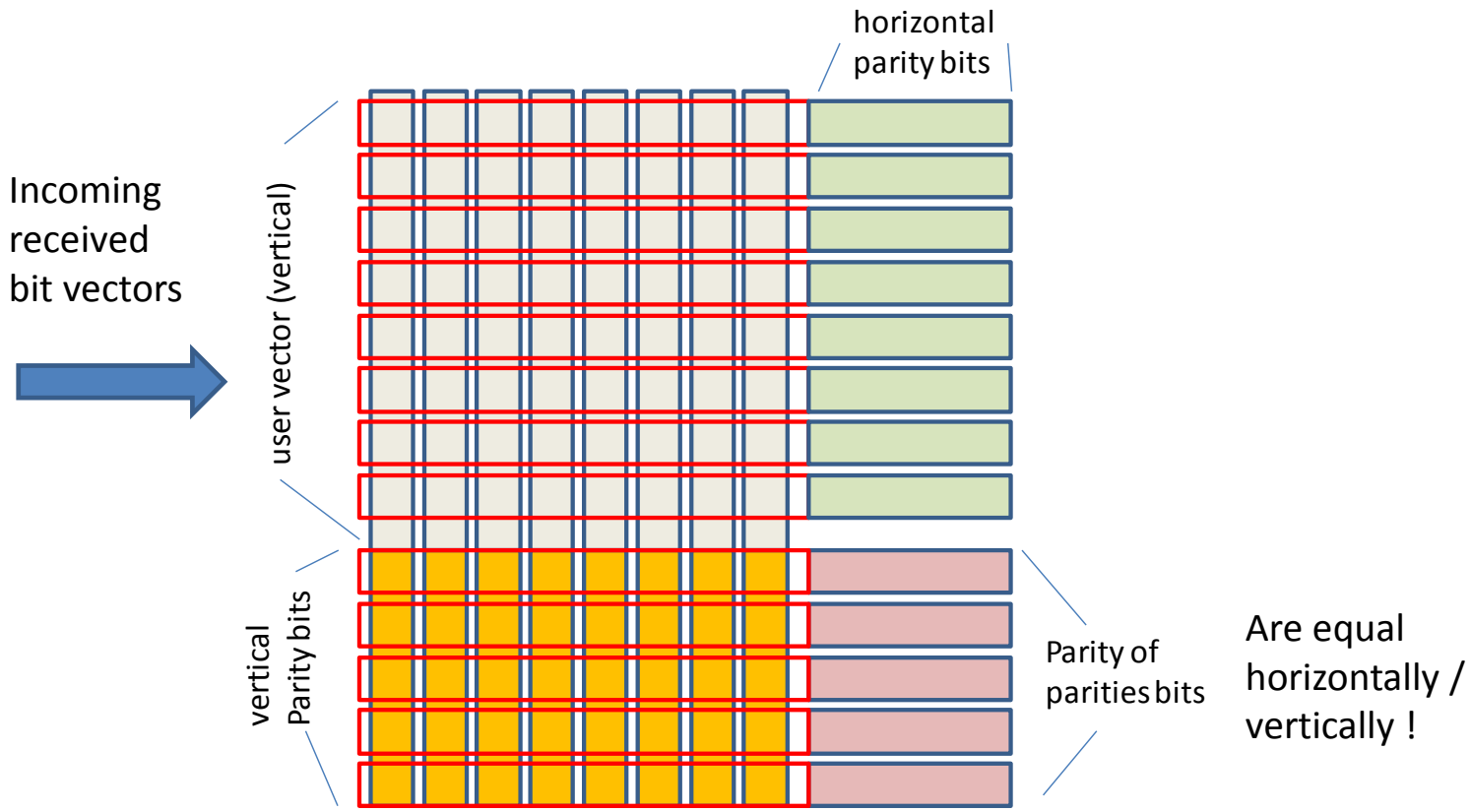
Hamming distance between code works: k



Hamming distance between code works: n

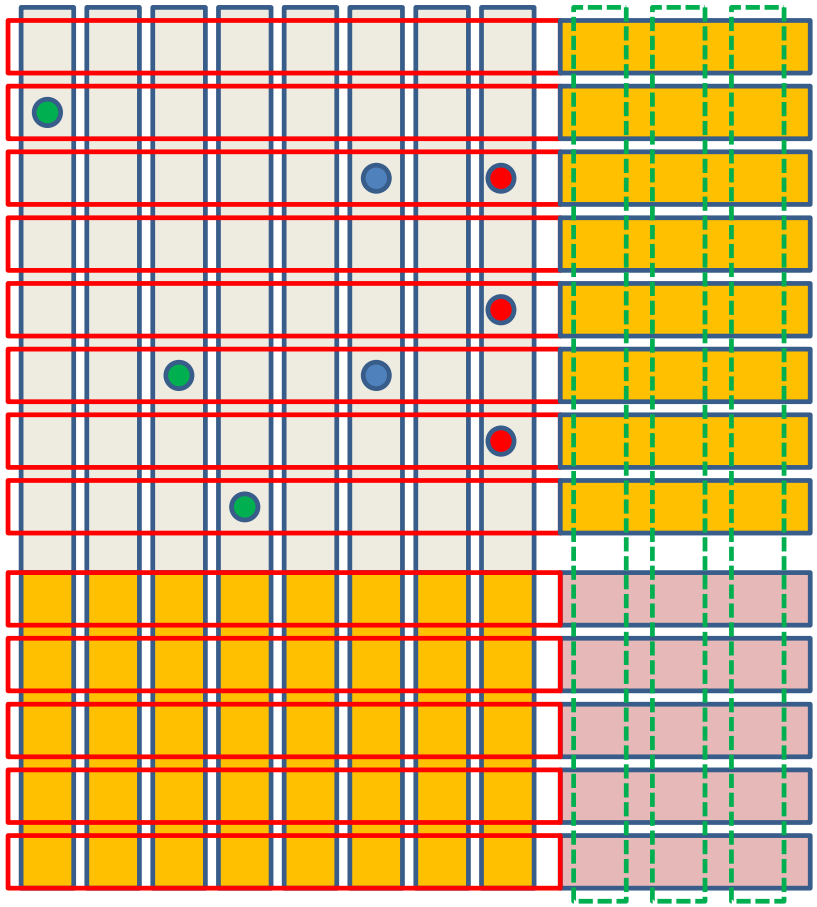
The resulting Hamming distance is H = k*n !!

2-Dimensional Matrix



- Iterative decoding of multiple errors by step-wize application of„simple“ (Hsiao) single-bit error correction. Use of „product“ Hamming distance.
- Relatively simple encoder / decoder circuitry which makes „addional“ consideration of hardware errors feasible.

Iterative Decoding



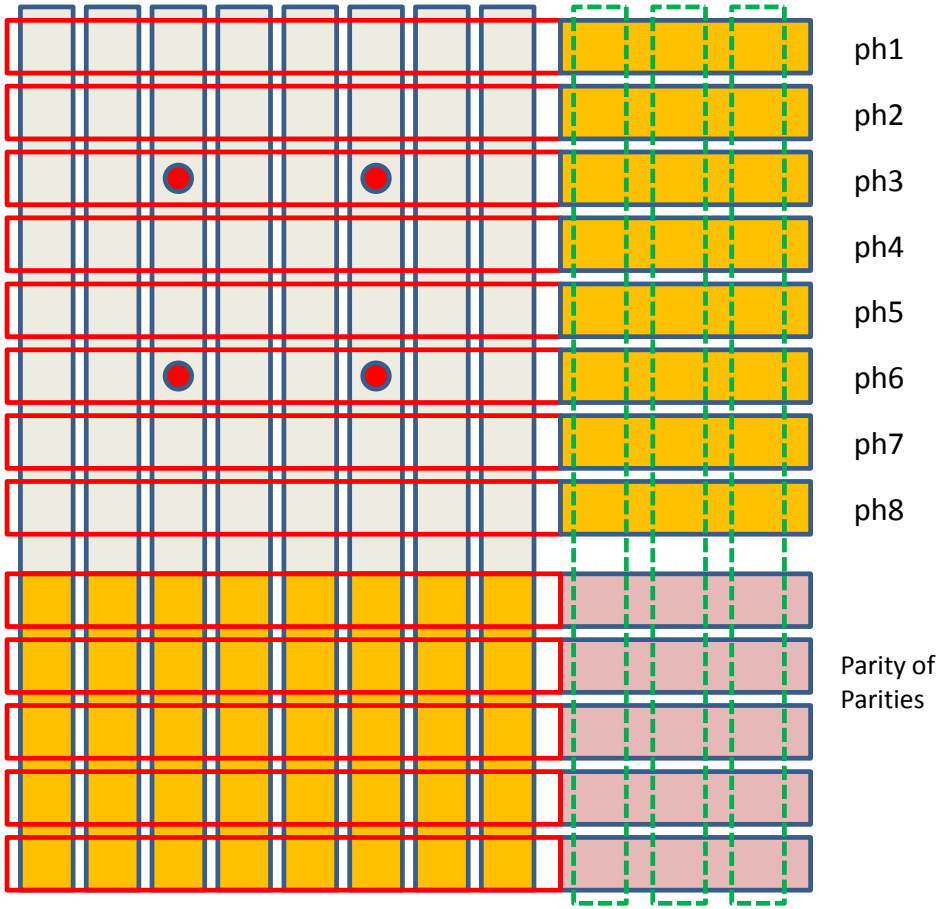
- Single errors in one vertical vector can be detected / corrected in one step.
- Double / multiple errors in a single vertical vector can be corrected if the corresponding horizontal vectors have a single error only.

The scheme can be applied iteratively using single bit error correction repeatedly !

Fast error correction in case of „few“ error bits, relatively low hardware overhead ?

Multi-Bit-errors in a vector must be detected to avoid miscorrection !

Where the Scheme Fails



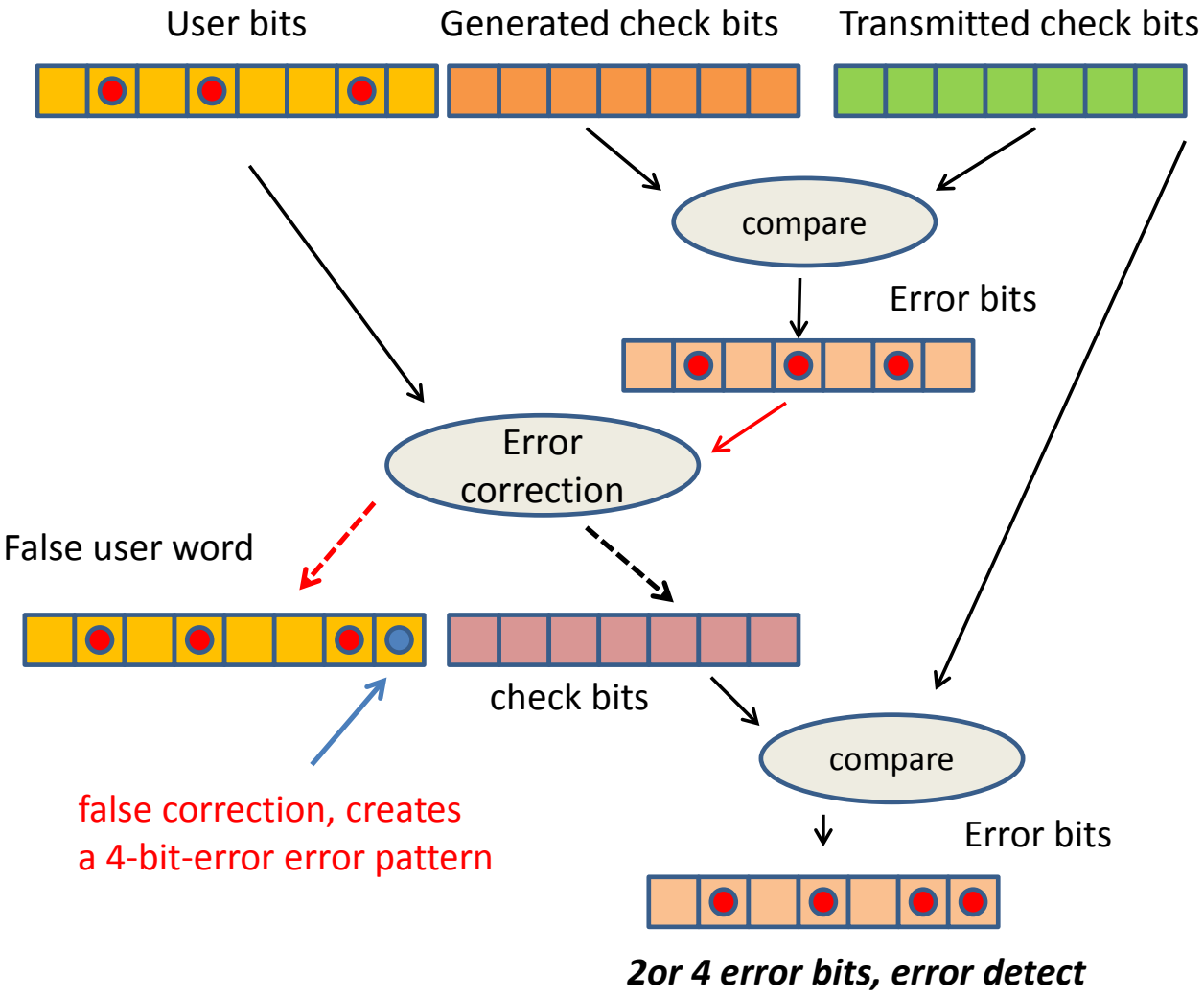
- Iterative decoding fails if 2 (or more) vertical vectors have 2 (or more) errors on the same vertical vectors !

Partial solution:

If there is only a single such pattern in the error map, it can be identified and corrected in a special step !

The scheme seems to fail if there are several such patterns !

Avoiding False Corrections



Standard Codes for FEC

There is no code that serves all requirements equally well.

- BCH-codes: Based on the theory of polynomials and Galoise fields. Formally well described and understood. Works for error clusters and for randomly distributed errors. Deterministic results. Decoding is slow and expensive.
- Reed-Solomon-codes: Similar to BCH codes, but optimized to handle errors in local clusters (such as scratches on a CD or DVD).
- Turbo codes: Combine fault detection by double to multiple encoding with permutation, soft detection by comparison, shows good results close to the Shannon limit for large vectors and relatively low fault density, slow in decoding.
- Low-density parity codes (LDPC). Advanced methods of finding the best coding matrix based on graphs. Good performance versus the Shannon limit. Good and relatively fast hardware solutions exist.

H-Matrix

Check Bits **Code Bits**

	C0	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**….Ck**
P1	H10	H11	H12	H13	H14	H15	H16	H17	H18	H19	H1k
P2	H20	H21	H22	H23	H24	H25	H26	H27	h28	H29	H2k
P3	H30	H31	H32	H33	H34	H35	H36	H37	H38	H39	H3k
P4	H40	H41	H42	H43	H44	H45	H46	H47	H48	H49	H4k
•	•	•	•	•	•	•	•	•	•	•	•
Pn	Hn0	Hn1	Hn2	Hn3	Hn4	Hn5	Hn6	Hn7	Hn8	Hn9	Hnk

The H-matrix defines which check bit is generated from which user bit.

An error pattern is derived in the decoder by comparing, for a specific user bit vector, the check bits sent with the user bits and the check bits derived from the actually received user vector. The difference is an error bit pattern. By using suitable decoding algorithms, the faulty bit positions can be found.

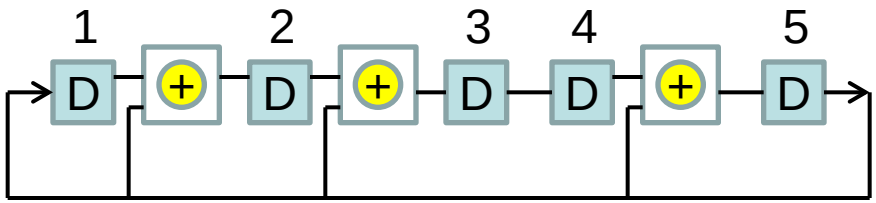
Using Polynome Arithmetics

The content of a register can be described by a polynome:

10011011

$1*x^7 + 0*x^6+0*x^5+1*x^4+1*x^3+0*x^2+1*x+1$

The feedback characteristic of a feedback shift register can also be described by a polynome:



$X^5 + X^4 + X^2 + 1$

If the content of the register is fed into the „polynome“ register serially by an EXOR, this works like a polynome division.

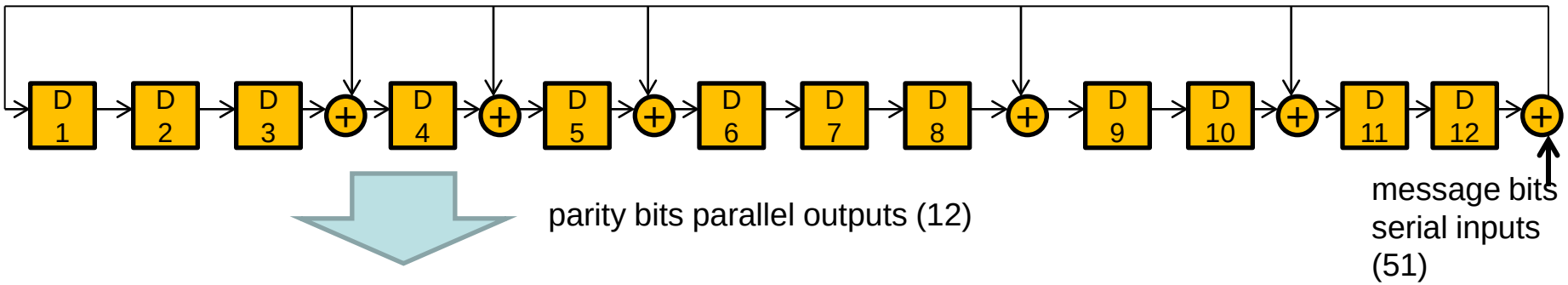
BCH-Code

Block code: User bits | Check bits

- For a certain configuration of total word length (63), a number of user bits (51) and the number of detectable and repairable errors (2) define a suitable generator polynomial :

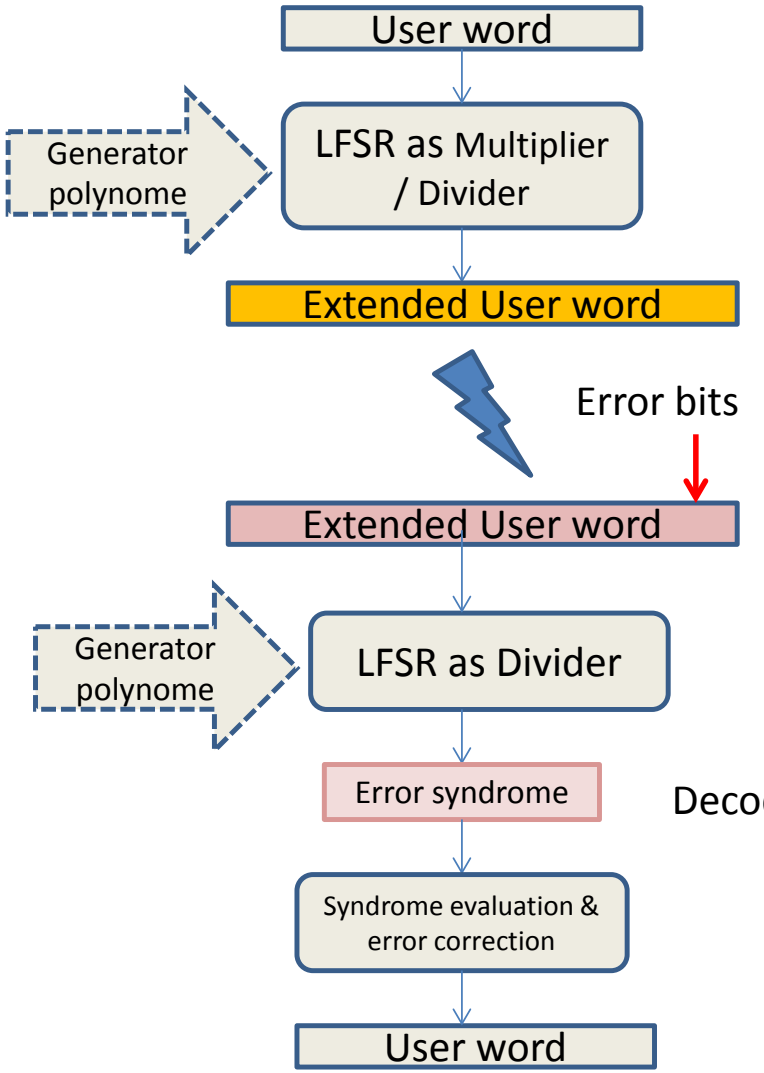
$$G(x) = 1 + x^3 + x^4 + x^5 + x^9 + x^{10} + x^{12}$$ implying 12 check bits.

This polynomial is mapped into a matching hardware structure, that is a linear feedback shift register, whose feedbacks are defined by the polynomial.



The parity bits notify the remainder from dividing the message polynomial by the generator polynomial. With (max. 2) error bits the transmitted message polynomial is again divided by the generator giving a „syndrome“. From this syndrome an „error locator polynomial“ can be derived. Its remainder after polynomial divisions indicate error bits in the received code word.

BCH Encoder / Decoder



Remainder is zero in case of no errors. Otherwize, any specific error pattern creates a specific „remainder“.

Decoding is slow and complex !

BCH Cost / Overhead / Time

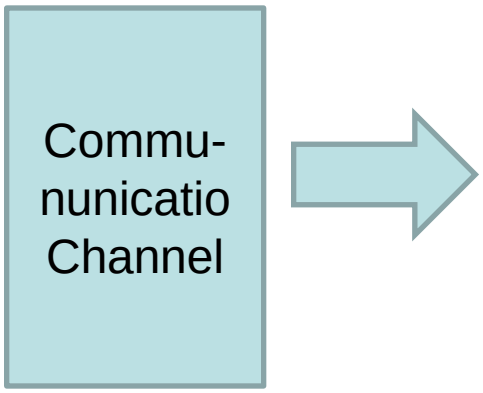
BCH Codes, total word lenght 255 bits, overhead and timing

Total bits	Data bits	Parity bits	Code rate (payload to total length)	Parity bits to data bits	Parity bits to packet length	Number of corrected bit errors	200 MHz clock Encoder	200 MHz clock Decoder
255	247	8	0.97	3.2%	3.1%	1	1.28 µs	2.52 µs
255	239	16	0.94	6.7%	6.3%	2	1.28 µs	2.48 µs
255	231	24	0.91	10.4%	9.4%	3	1.28 µs	2.44 µs
255	223	32	0.87	14.3%	12.5%	4	1.28 µs	2.40 µs
255	215	40	0.84	18.6%	15.7%	5	1.28 µs	2.36 µs
255	207	48	0.81	23.2%	18.8%	6	1.28 µs	2.32 µs
255	199	56	0.78	28.1%	22.0%	7	1.28 µs	2.28 µs
255	191	64	0.75	33.5%	25.1%	8	1.28 µs	2.24 µs
255	187	68	0.73	36.4%	26.7%	9	1.28 µs	2.22 µs
255	179	76	0.70	42.5%	29.8%	10	1.28 µs	2.18 µs
255	171	84	0.67	49.1%	32.9%	11	1.28 µs	2.14 µs
255	163	92	0.64	56.4%	36.1%	12	1.28 µs	2.10 µs
255	155	100	0.61	64.5%	39.2%	13	1.28 µs	2.06 µs
255	147	108	0.58	73.5%	42.4%	14	1.28 µs	2.02 µs
255	139	116	0.55	83.5%	45.5%	15	1.28 µs	1.98 µs
255	131	124	0.51	94.7%	48.6%	18	1.28 µs	1.94 µs

← Hamming

..typically used in wireless communication with „relaxed“ timing constraints !

Codes with Soft Error Detection



Evaluation of a bit

Conventional	Extended
0	hard 0
	soft 0
X	unknown
	soft 1
1	hard 1

Advanced codes like „turbo codes“ make use of such additional information to get closer to the Shannon Limit in channel communication capacity.
But results are certainly „not guaranteed“, and calculation is slow.

Possibly the best existing scheme used so far is „Low Density Parity Codes“, (LDPC) also with hardware implementations.

Low-Density Parity Codes (1)

$\mathbf{a}_l$ oder **a*** is a user code word that needs to be transmitted

a is the encoded word transmitted over the channel

$\mathbf{a}_k$ is the redundant part of **a** needed for error correction

b is the received code word, possibly containing errors

H is a generator matrix, needed to generate **a** from a_l

n is the total lenght of the code word

l is the mumber of user bits in a code word

The coding scheme involves a sequence of vectors a^T and a generator matrix H that satisfies the equation: $H * a^T = 0$

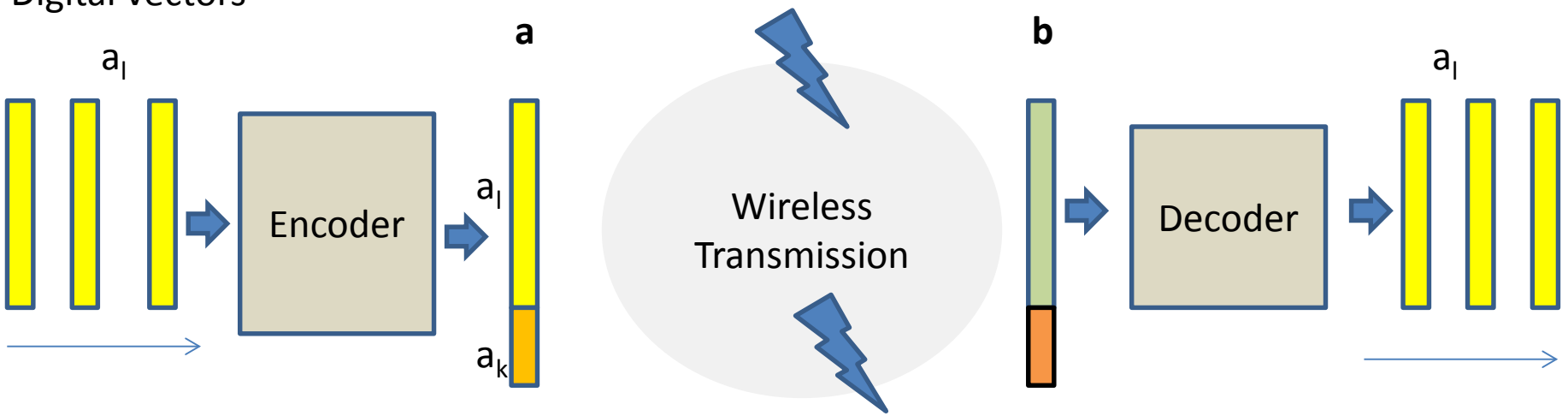
Then each vector $a = [a_k, a_l]$ consists of redundant check bits a_k and user bits a_l. Also the generator matrix H is comprised from a section H_k affecting the control bits and a section H_l affecting the user bits: $H = [H_k, H_l]$.

Then the check bits can be derived from: $a_k^T = H_k^{-1} * H_l * a_l$

Then the received bit sequence b should satisfy the condition: $H * b^T = 0$

Low-Density Parity Checks

Digital vectors



Input vectors have a „user“ section a_l and get a „redundant“ part a_k, making a total vector a. These a- vectors in total form a data packet **a** which is a matrix. Then $\mathbf{a}^T$ is the transposed matrix of **a**.
Instead of the correct matrix **a** at the output we receive the matrix **b** which contains errors.

The basic rule for finding an appropriate set of a_k-values is:
$H * a^T = 0$

Low-Density Parity Codes (2)

So for LDPC-coding a generator matrix H has to be defined, which depends on the word length and the number of error bits assumed. If the number of „1“s in a column is constant, we have a „regular LDPC code“.

The generator matrix has to be known both to the encoder and the decoder. As it contains only few „1“ positions, it gives the name to „low density parity check“.

With a sequence of vectors **b** collected and stored, the decoder has to make sure that the condition: $\mathbf{H} * \mathbf{b}^T = 0$ is met. Bits in **b** have to be selected accordingly.

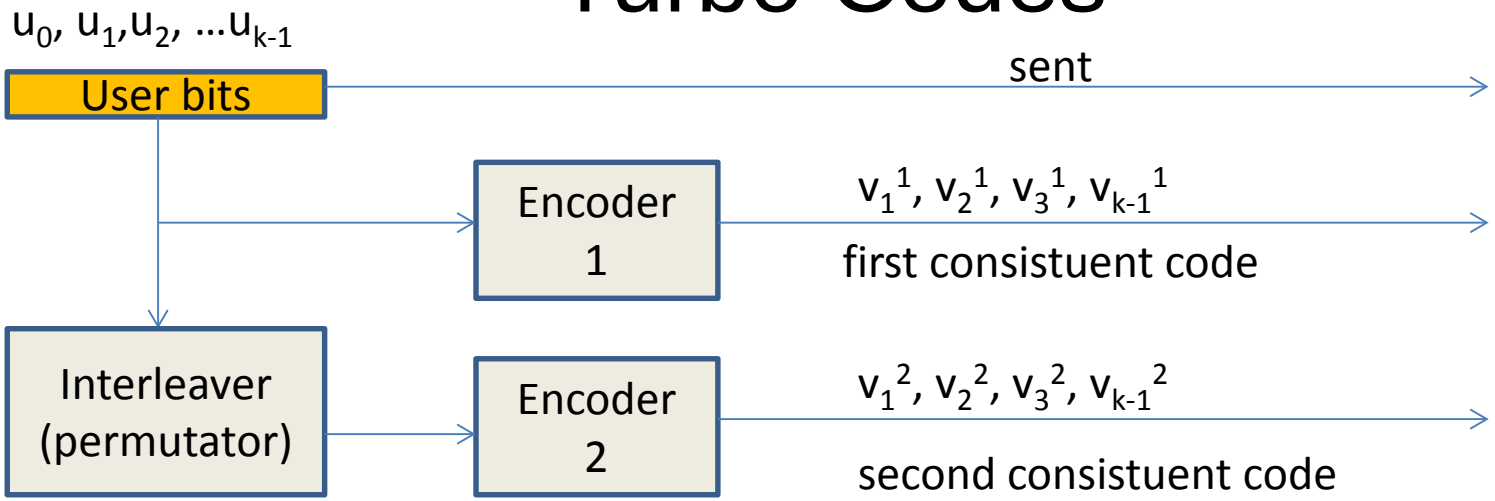
This is actually a set of equations, including several up to many unknown „X“ bits, for which valid solutions need to be found.

LDPC de-coders can solve such equations in an iterative manner in several steps. Thereby it is possible to make more / less likely assumption on possible values of „unknown“ bits, also using trial and error.

This is apparently the property which makes LDPC-codes the best choice when trying to reach the „Shannon Limit“.

For shorter word widths, there are good and relatively fast hardware solutions for decoders, but definitly none working in 1-10 clock cycles

Turbo Codes



Code length: n = 3 k, code rate: R = 1 /3

Schemes with double or multiple interleaving are possible !

In de-coding, we can identify „questionable“ bits from each of the encoding steps.
Then it is possible to identify „most favorable“ values for such bits from comparison.

Good performance in case of long bit symbols and low error rates !

Favorable implementation in software !

Summary

Error detection by coding is a very specific art.

Most methods have been developed and are still used for signal restauration in wireless communication.

Such methods always assume that inputs and outputs are equal. That is valid also for memory circuits, but not for logic.

Code-based methods are much less easy to apply if outputs differ from inputs, such as in logic and in arithmetic units.

Then „code bit predictors“ are needed, which may become as expensive as the arithmetic / logic units themselves.