Digital Communication Systems ECS 452

Asst. Prof. Dr. Prapun Suksompong

prapun@siit.tu.ac.th

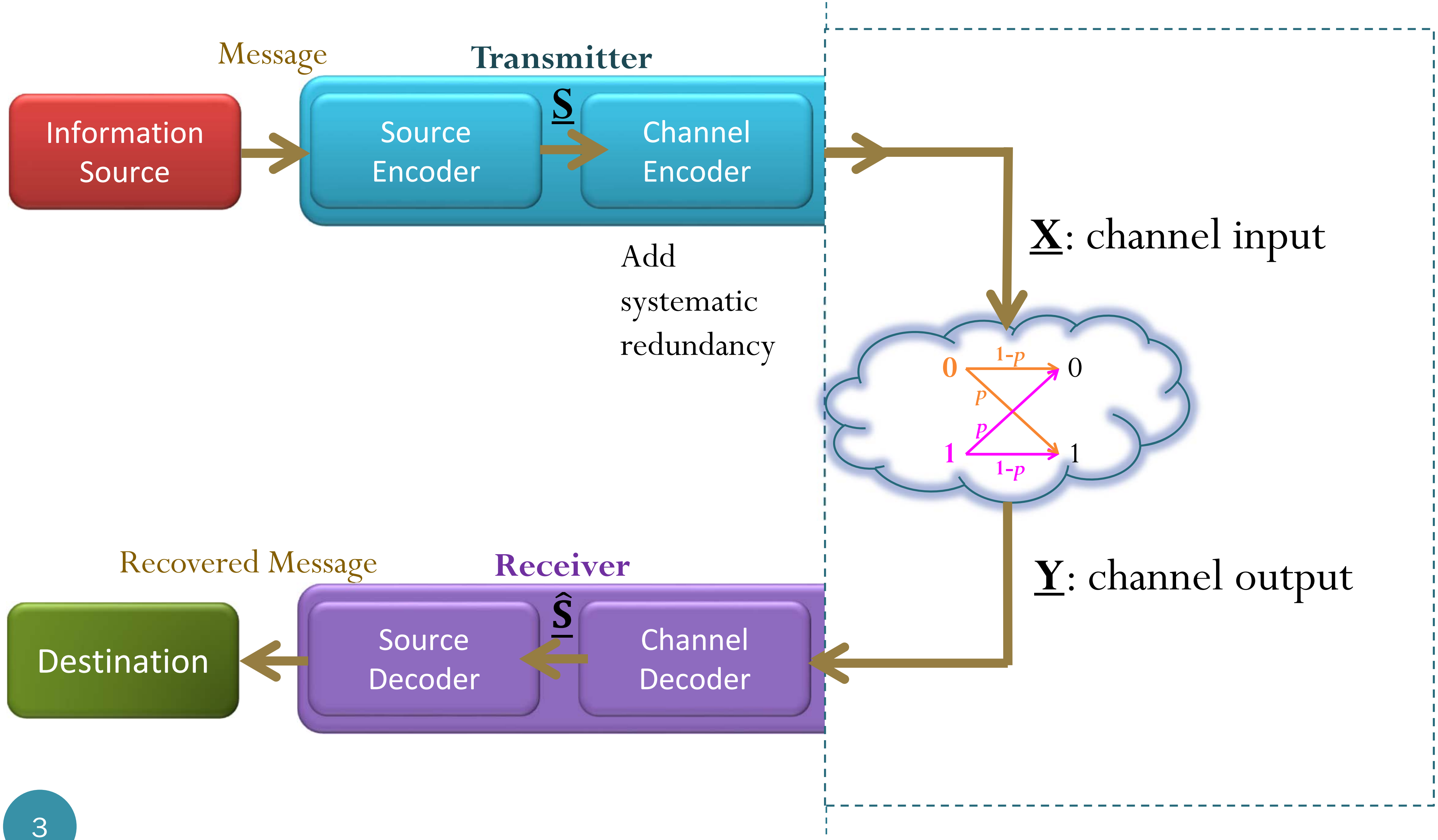
5.1 Binary Linear Block Codes

Office Hours:

BKD, 6th floor of Sirindhralai building

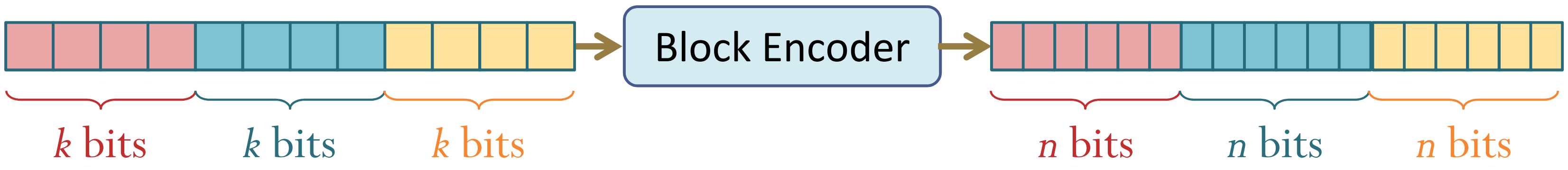
Monday	10:00-10:40
Tuesday	12:00-12:40
Thursday	14:20-15:30

Review: Channel Encoder and Decoder



Review: Block Encoding

- We mentioned the general form of channel coding over BSC.
- In particular, we looked at the general form of **block codes**.



Code length

"Dimension" of the code

- **(n,k) codes**: n-bit blocks are used to conveys k-info-bit blocks

codewords

"messages"

- **Assume $n > k$**
- **Rate**: $R = \frac{k}{n}$.

Max. achievable rate

Recall that the capacity of BSC is $C = 1 - H(p)$.
For $p \in (0,1)$, we also have $C \in (0,1)$.
Achievable rate is < 1.

$\mathcal{C}$

- $\mathcal{C}$ = the collection of all codewords for the code considered
- Each n-bit block is selected from $\mathcal{C}$.
- The message (data block) has k bits,
  so there are 2^k possibilities.
- A reasonable code would not assign the same codeword to different messages.
- Therefore, there are 2^k (distinct) codewords in $\mathcal{C}$.
- Ex. Repetition code with $n = 3$

GF(2)

- The construction of the codes can be expressed in matrix form using the following definition of **addition** and **multiplication** of bits:

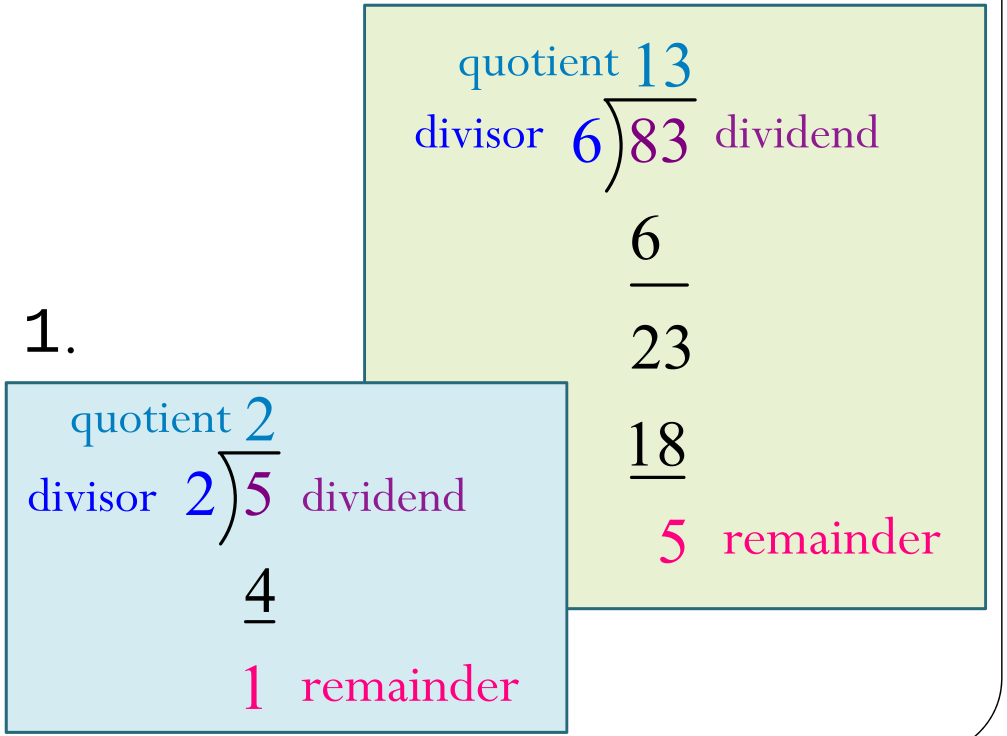
$\oplus$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

- These are **modulo-2** addition and **modulo-2** multiplication, respectively.
- The operations are the same as the **exclusive-or** (**XOR**) operation and the **AND** operation.
  - We will simply call them addition and multiplication so that we can use a matrix formalism to define the code.
- The two-element set {0, 1} together with this definition of addition and multiplication is a number system called a **finite field** or a **Galois field**, and is denoted by the label **GF(2)**.

Modulo operation

- The **modulo operation** finds the **remainder** after division of one number by another (sometimes called **modulus**).
- Given two positive numbers, a (the **dividend**) and n (the **divisor**),
- $\boldsymbol{a}$ **modulo** $\boldsymbol{n}$ (abbreviated as $\boldsymbol{a}$ **mod** $\boldsymbol{n}$) is the remainder of the division of a by n.
- "83 mod 6" = 5
- "5 mod 2" = 1
  - In MATLAB, `mod(5,2) = 1`.
- **Congruence relation**
  - $5 \equiv 1 \pmod{2}$



quotient 13

divisor 6)83 dividend

6

23

18

5 remainder

quotient 2

divisor 2)5 dividend

4

1 remainder

GF(2) and modulo operation

- Normal addition and multiplication (for 0 and 1):

+	0	1
0	0	1
1	1	2

×	0	1
0	0	0
1	0	1

- Addition and multiplication in GF(2):

⊕	0	1
0	0	1
1	1	0

•	0	1
0	0	0
1	0	1

GF(2)

- The construction of the codes can be expressed in matrix form using the following definition of addition and multiplication of bits:

$\oplus$	0	1
0	0	1
1	1	0

$\bullet$	0	1
0	0	1
1	1	0

- Note that

$$x \oplus 0 = x$$

$$x \oplus 1 = \overline{x}$$

$$x \oplus x = 0$$

The above property implies $\underbrace{-x} = x$

By definition, "*-x*" is something that, when added with *x*, gives 0.

- Extension: For vector and matrix, apply the operations to the elements the same way that addition and multiplication would normally apply (except that the calculations are all in GF(2)).

Examples

- Normal matrix multiplication:

$$\big(7\times(-2)\big)+(4\times 3)+\big(3\times(-7)\big)=-14+12+(-21)$$

$$\begin{bmatrix}7 & 4 & 3\\ 2 & 5 & 6\\ 1 & 8 & 9\end{bmatrix}\begin{bmatrix}-2 & 4\\ 3 & -8\\ -7 & 6\end{bmatrix}=\begin{bmatrix}-23 & 14\\ -31 & 4\\ -41 & -6\end{bmatrix}$$

- Matrix multiplication in GF(2):

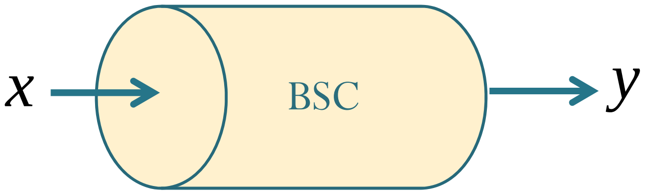
$$(1\cdot 1)\oplus(0\cdot 0)\oplus(1\cdot 1)=1\oplus 0\oplus 1$$

$$\begin{bmatrix}1 & 0 & 1\\ 0 & 0 & 1\\ 1 & 1 & 1\end{bmatrix}\begin{bmatrix}1 & 1\\ 0 & 1\\ 1 & 0\end{bmatrix}=\begin{bmatrix}0 & 1\\ 1 & 0\\ 0 & 0\end{bmatrix}$$

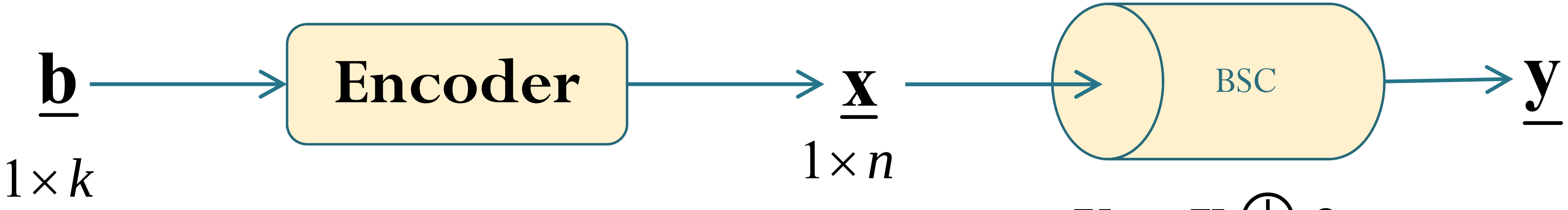
Alternatively, one can also apply normal matrix multiplication first, then apply "mod 2" to each element:

$$\begin{bmatrix}1 & 0 & 1\\ 0 & 0 & 1\\ 1 & 1 & 1\end{bmatrix}\begin{bmatrix}1 & 1\\ 0 & 1\\ 1 & 0\end{bmatrix}=\begin{bmatrix}2 & 1\\ 1 & 0\\ 2 & 2\end{bmatrix}\xrightarrow{\text{mod } 2}\begin{bmatrix}0 & 1\\ 1 & 0\\ 0 & 0\end{bmatrix}$$

BSC and the Error Pattern



- Again, to transmit k information bits, the channel is used n times.



$$\underline{\mathbf{y}} = \underline{\mathbf{x}} \oplus \underline{\mathbf{e}}$$

error pattern

Its nonzero elements mark the positions of transmission error in $\underline{\mathbf{y}}$

Review: Block Decoding

- In this chapter, we assume the use of **minimum distance decoder**.
  - $\hat{\underline{\mathbf{x}}}(\underline{\mathbf{y}}) = \arg\min_{\underline{\mathbf{x}}} d(\underline{\mathbf{x}}, \underline{\mathbf{y}})$
- Recall
  1. The **MAP decoder** is the optimal decoder.
  2. When the codewords are equally-likely, the **ML decoder** the same as the MAP decoder; hence it is also **optimal**.
  3. When the **crossover probability** of the BSC p is $< \mathbf{0.5}$,
     ML decoder is the same as the **minimum distance decoder**.
- Also, in this chapter, we will focus
  - less on probabilistic analysis,
  - but more on explicit codes.

Vector Notation

$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_i \\ \vdots \\ v_n \end{pmatrix}$$

$\vec{0}, \underline{0}$: the zero vector (the all-zero vector)

$\vec{1}, \underline{1}$: the one vector (the all-one vector)

- $\vec{\mathbf{v}}$: column vector
- $\underline{\mathbf{r}}$: row vector $(r_1, r_2, \ldots, r_i, \ldots r_n)$
- Subscripts represent element indices inside individual vectors.
  - v_i and r_i refer to the i^{th} elements inside the vectors $\vec{\mathbf{v}}$ and $\underline{\mathbf{r}}$, respectively.
- When we have a list of vectors, we use superscripts in parentheses as indices of vectors.
  - $\vec{\mathbf{v}}^{(1)}, \vec{\mathbf{v}}^{(2)}, \ldots, \vec{\mathbf{v}}^{(M)}$ is a list of M column vectors
  - $\underline{\mathbf{r}}^{(1)}, \underline{\mathbf{r}}^{(2)}, \ldots, \underline{\mathbf{r}}^{(M)}$ is a list of M row vectors
  - $\vec{\mathbf{v}}^{(i)}$ and $\underline{\mathbf{r}}^{(i)}$ refer to the i^{th} vectors in the corresponding lists.

Linear Block Codes

- Definition: $\mathcal{C}$ is a **(binary) linear (block) code** if and only if $\mathcal{C}$ forms a vector (sub)space (over GF(2)).

In case you forgot about the concept of vector space,…

  - Equivalently, this is the same as requiring that

$$\text{if } \underline{\mathbf{x}}^{(1)} \text{ and } \underline{\mathbf{x}}^{(2)} \in \mathcal{C}, \text{ then } \underline{\mathbf{x}}^{(1)} \oplus \underline{\mathbf{x}}^{(2)} \in \mathcal{C}.$$

  - Note that any (non-empty) linear code $\mathcal{C}$ must contain $\underline{\mathbf{0}}$.

- Ex. The code that we considered in **Problem 5 of HW3** is

$$\mathcal{C} = \{00000, 01000, 10001, 11111\}$$

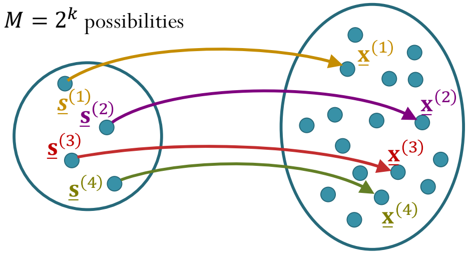
Is it a linear code?

Linear Block Codes: Motivation (1)

- Why linear block codes are popular?
- Recall: General block **encoding**
  - Characterized by its codebook.
    - The table that lists all the 2^k mapping from the k-bit info-block $\underline{\mathbf{s}}$ to the n-bit codeword $\underline{\mathbf{x}}$ is called the **codebook**.
    - The M info-blocks are denoted by $\underline{\mathbf{s}}^{(1)}, \underline{\mathbf{s}}^{(2)}, \dots, \underline{\mathbf{s}}^{(M)}$. The corresponding M codewords are denoted by $\underline{\mathbf{x}}^{(1)}, \underline{\mathbf{x}}^{(2)}, \dots, \underline{\mathbf{x}}^{(M)}$, respectively.

[See p. 51 in Ch. 3 of the lecture notes.]

index i	info-block $\underline{\mathbf{s}}$	codeword $\underline{\mathbf{x}}$
1	$\underline{\mathbf{s}}^{(1)} = 000\dots0$	$\underline{\mathbf{x}}^{(1)} =$
2	$\underline{\mathbf{s}}^{(2)} = 000\dots1$	$\underline{\mathbf{x}}^{(2)} =$
$\vdots$	$\vdots$	$\vdots$
M	$\underline{\mathbf{s}}^{(M)} = 111\dots1$	$\underline{\mathbf{x}}^{(M)} =$


$M = 2^k$ possibilities
$\underline{\mathbf{s}}^{(1)}$
$\underline{\mathbf{s}}^{(2)}$
$\underline{\mathbf{s}}^{(3)}$
$\underline{\mathbf{s}}^{(4)}$
$\underline{\mathbf{x}}^{(1)}$
$\underline{\mathbf{x}}^{(2)}$
$\underline{\mathbf{x}}^{(3)}$
$\underline{\mathbf{x}}^{(4)}$

Choose $M = 2^k$ from 2^n possibilities to be used as codewords.

  - Can be realized by combinational/combinatorial circuit.
    - If lucky, can used K-map to simplify the circuit.

Linear Block Codes: Motivation (2)

- Why linear block codes are popular?
- Linear block encoding is the same as matrix multiplication.
  - See next slide.
  - The matrix replaces the table for the codebook.
  - The size of the matrix is only $k \times n$ bits.
    - Compare this against the table (codebook) of size $2^k \times (k+n)$ bits for general block encoding.
- Linearity $\Rightarrow$ easier implementation and analysis
- Performance of the class of linear block codes is similar to performance of the general class of block codes.
  - Can limit our study to the subclass of linear block codes without sacrificing system performance.

Linear Block Codes: Generator Matrix

For any linear code, there is a matrix $\mathbf{G} = \begin{bmatrix} \boxed{\underline{\mathbf{g}}^{(1)}} \\ \boxed{\underline{\mathbf{g}}^{(2)}} \\ \vdots \\ \boxed{\underline{\mathbf{g}}^{(k)}} \end{bmatrix}_{k \times n}$

called the **generator matrix**

such that, for any codeword $\underline{\mathbf{x}}$, there is a message vector $\underline{\mathbf{b}}$ which produces $\underline{\mathbf{x}}$ by

mod-2 summation

$$\boxed{\underline{\mathbf{x}} = \underline{\mathbf{b}}\mathbf{G}} = \underbrace{\sum_{j=1}^{k} b_j \underline{\mathbf{g}}^{(j)}}$$

Note:

(1) Any codeword can be expressed as a linear combination of the rows of $\mathbf{G}$

(2) $C = \{\underline{\mathbf{b}}\mathbf{G} : \underline{\mathbf{b}} \in \{0,1\}^k\}$

Note also that, given a matrix $\mathbf{G}$, the (block) code that is constructed by (2) is always linear.

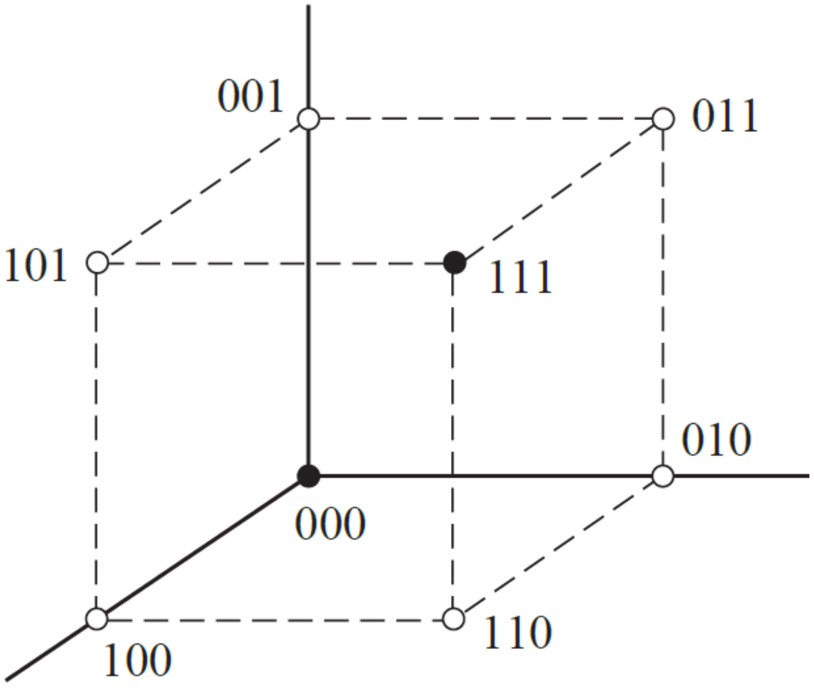
Linear Block Codes: Examples

- **Repetition code**: $\underline{\mathbf{x}} = [b \quad b \quad \cdots \quad b]$
  - $\mathbf{G} = [1 \quad 1 \quad \cdots \quad 1]$
  - $\underline{\mathbf{x}} = \underline{\mathbf{b}}\mathbf{G} = b\mathbf{G} = [b \quad b \quad \cdots \quad b]$
  - $R = \frac{k}{n} = \frac{1}{n}$

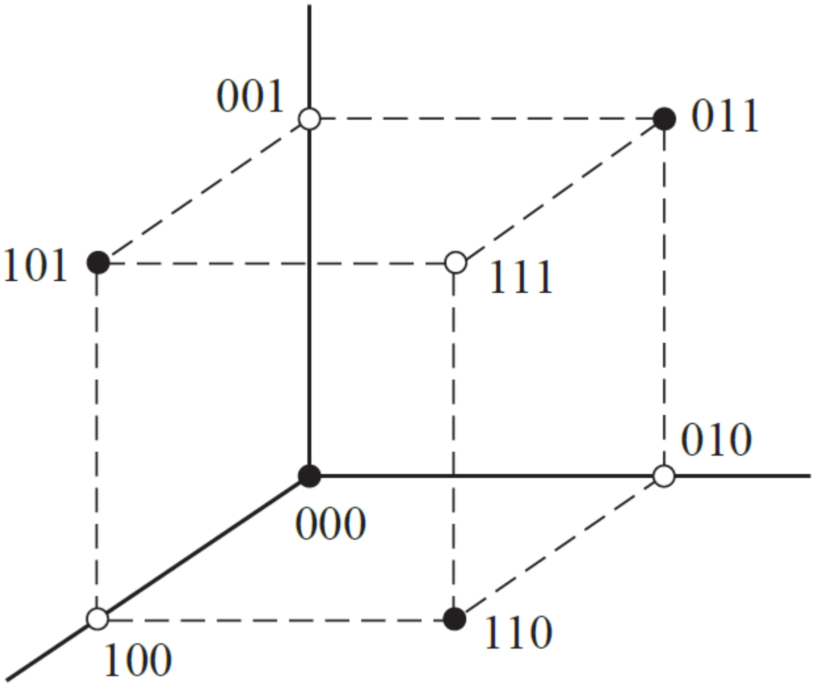
- **Single-parity-check code**: $\underline{\mathbf{x}} = \left[\boxed{\quad\underline{\mathbf{b}}\quad}; \underbrace{\sum_{j=1}^{k} b_j}_{\text{parity bit}}\right]$
  - $\mathbf{G} = [\mathbf{I}_{k\times k}; \underline{\mathbf{1}}^T]$
  - $R = \frac{k}{n} = \frac{k}{k+1}$

Vectors representing 3-bit codewords

Representing the codewords in the two examples on the previous slide as vectors:



Triple-repetition code



Parity-check code

Related Idea:

Even Parity vs. Odd Parity

- Parity bit checking is used occasionally for transmitting ASCII characters, which have 7 bits, leaving the 8th bit as a **parity bit**.
- Two options:
  - **Even Parity**: Added bit ensures an even number of 1s in each codeword.
    - A: 1000001**0**
  - **Odd Parity**: Added bit ensures an odd number of 1s in each codeword.
    - A: 1000001**1**

Even Parity vs. Odd Parity

- Even parity and odd parity are properties of a codeword (a vector), not a bit.
- Note: The generator matrix $\mathbf{G} = [\mathbf{I}_{k\times k}; \underline{\mathbf{1}}^T]$ previously considered produces even parity codeword

$$\underline{\mathbf{x}} = \left[\boxed{\quad\quad \underline{\mathbf{b}} \quad\quad}; \sum_{j=1}^{k} b_j \right]$$

- Q: Consider a code that uses odd parity. Is it linear?

Error Control using Parity Bit

- If an odd number of bits (including the parity bit) are transmitted incorrectly, the parity bit will be incorrect, thus indicating that a parity error occurred in the transmission.
- Ex.
  - Suppose we use even parity.
  - Consider the codeword $\underline{\mathbf{x}}$ = 10000010
- Suitable for *detecting* errors; *cannot correct* any errors

Error Detection

- Two types of **error control**:
  1. **error detection**
  2. **error correction**
- **Error detection**: the determination of whether errors are present in a received word.
- An error pattern is **undetectable** if and only if it causes the received word to be a valid codeword other than that which was transmitted.
  - Ex: In single-parity-check code, error will be undetectable when the number of bits in error is even.

Error Correction

- In **FEC** (**forward error correction**) system, when the decoder detects error, the arithmetic or algebraic **structure** of the code is used to determine which of the valid codewords was transmitted.

- It is possible for a detectable error pattern to cause the decoder to select a codeword other than that which was actually transmitted. The decoder is then said to have committed a **decoding error**.

Square array for error correction by parity checking.

- The codeword is formed by arranging *k* message bits in a square array whose rows *and* columns are checked by $2\sqrt{k}$ parity bits.
- A transmission error in one message bit causes a row and column parity failure with the error at the intersection, so single errors can be corrected.

$$\underline{\mathbf{b}} = [b_1, b_2, \ldots, b_9]$$

b_1	b_2	b_3	p_1
b_4	b_5	b_6	p_2
b_7	b_8	b_9	p_3
p_4	p_5	p_6	

$$\underline{\mathbf{x}} = [b_1, b_2, \ldots, b_9, p_1, p_2, \ldots, p_6]$$

[Carlson & Crilly, p 594]

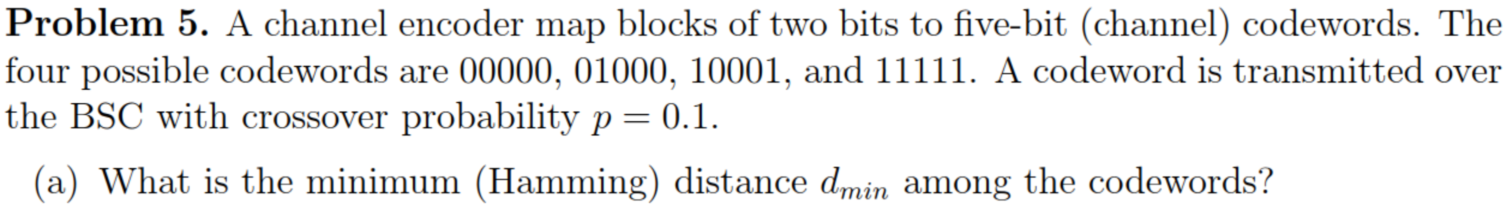
Weight and Distance

- The **weight** of a vector is the number of nonzero coordinates in the vector.
  - The weight of a vector $\underline{\mathbf{x}}$ is commonly written as $\boldsymbol{w}(\underline{\mathbf{x}})$.
  - Ex. $w(010111) =$
  - For BSC with cross-over probability $p < 0.5$, error pattern with smaller weight (less #1s) are more likely to occur.
- The **Hamming distance** between two n-bit blocks is the number of coordinates in which the two blocks differ.
  - Ex. $d(010111{,}011011) =$
  - Note:
    - The Hamming distance between any two vectors equals the weight of their sum.
    - The Hamming distance between the transmitted codeword $\underline{\mathbf{x}}$ and the received vector $\underline{\mathbf{y}}$ is the same as the weight of the corresponding error pattern $\underline{\mathbf{e}}$.

Review: Minimum Distance (d_{min})

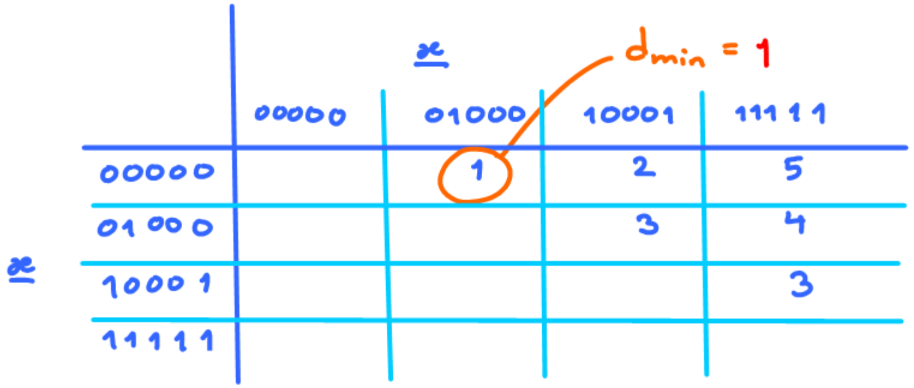
The **minimum distance** (d_{min}) of a block code is the minimum Hamming distance between all pairs of distinct codewords.

- Ex. **Problem 5 of HW3**:



Problem 5. A channel encoder map blocks of two bits to five-bit (channel) codewords. The four possible codewords are 00000, 01000, 10001, and 11111. A codeword is transmitted over the BSC with crossover probability $p = 0.1$.

(a) What is the minimum (Hamming) distance d_{min} among the codewords?



x	00000	01000	10001	11111
00000		1	2	5
01000			3	4
10001				3
11111				

$d_{min} = 1$

- Ex. Repetition code:

$d_{\min}$: two important facts

- For any linear block code, the **minimum distance** (***d*$_{\min}$**) can be found from minimum weight of its nonzero codewords.
  - So, instead of checking $\binom{2^k}{2}$ pairs, simply check the weight of the 2^k codewords.

- A code with minimum distance $d_{\min}$ can
  - detect all error patterns of weight w ≤ $d_{\min}$-1.
  - correct all error patterns of weight w ≤ $\left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$.

the floor function

$d_{\min}$ is an important quantity

- To be able to detect *all* w-bit errors, we need $d_{\min} \geq w + 1$.
  - With such a code there is no way that w errors can change a valid codeword into another valid codeword.
  - When the receiver observes an illegal codeword, it can tell that a transmission error has occurred.

- To be able to correct *all* w-bit errors, we need $d_{\min} \geq 2w + 1$.
  - This way, the legal codewords are so far apart that even with w changes the original codeword is still ***closer*** than any other codeword.

Example

Consider the code

$$\mathcal{C} \in \{0000000000,\ 0000011111,\ 1111100000, \text{ and } 1111111111\}$$

- Is it a linear code?

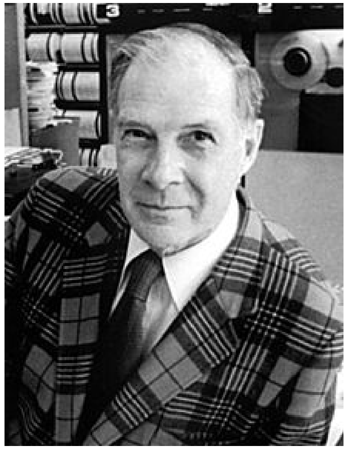
- $d_{\min} =$

- It can detect (at most) ____ errors.

- It can correct (at most) ____ errors.

Hamming codes

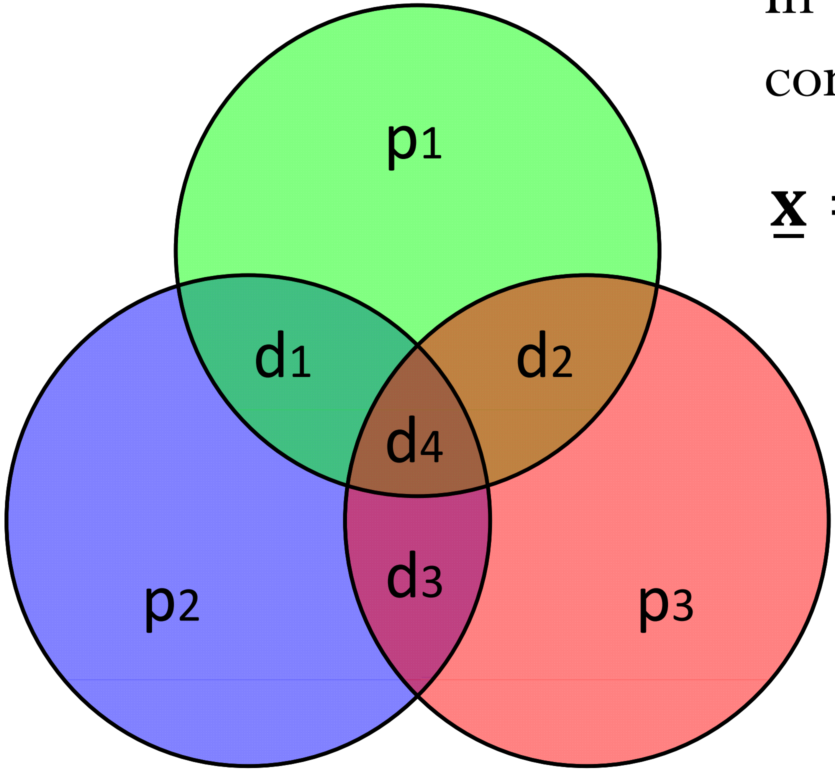
- One of the earliest codes studied in coding theory.
- Named after Richard W. Hamming
  - The IEEE Richard W. **Hamming Medal**, named after him, is an award given annually by Institute of Electrical and Electronics Engineers (IEEE), for "exceptional contributions to information sciences, systems and technology“.
    - Sponsored by Qualcomm, Inc
    - Some Recipients:
      - 1988 - Richard W. Hamming
      - 1997 - Thomas M. Cover
      - 1999 - David A. Huffman
      - 2011 - Toby Berger



- The simplest of a class of (algebraic) error correcting codes that **can *correct* one error in a block of bits**

Hamming codes: Ex. 1

This is an example of Hamming (7,4) code



In the video, the codeword is constructed from the data by

$$\underline{\mathbf{x}} = [p_1 \quad d_1 \quad p_2 \quad d_2 \quad p_3 \quad d_3 \quad d_4]$$

where

$$p_1 = d_1 \oplus d_2 \oplus d_4$$
$$p_2 = d_1 \oplus d_3 \oplus d_4$$
$$p_3 = d_2 \oplus d_3 \oplus d_4$$

- The message bits are also referred to as the data bits or information bits.
- The non-message bits are also referred to as parity check bits, checksum bits, parity bits, or check bits.

Generator matrix: a revisit

- Fact: The 1s and 0s in the j^{th} column of **G** tells which positions of the data bits are combined ($\oplus$) to produce the j^{th} bit in the codeword.
- For the Hamming code in the previous slide,

$$\underline{\mathbf{x}} = \begin{bmatrix} p_1 & d_1 & p_2 & d_2 & p_3 & d_3 & d_4 \end{bmatrix}$$

$$p_1 = d_1 \oplus d_2 \oplus d_4$$
$$p_2 = d_1 \oplus d_3 \oplus d_4$$
$$p_3 = d_2 \oplus d_3 \oplus d_4$$

$$= \begin{bmatrix} d_1 & d_2 & d_3 & d_4 \end{bmatrix} \underbrace{\begin{bmatrix} 1 & 1 & & & & & \\ 1 & 0 & & & & & \\ 0 & 0 & & & & & \\ 1 & 0 & & & & & \end{bmatrix}}_{\mathbf{G}}$$

Generator matrix: a revisit

- From $\underline{\mathbf{x}} = \underline{\mathbf{b}}\mathbf{G} = \sum_{j=1}^{k} b_j \underline{\mathbf{g}}^{(j)}$, we see that the *j* element of the codeword $\underline{\mathbf{x}}$ of a linear code is constructed from a linear combination of the bits in the message:

$$x_j = \sum_{i=1}^{k} b_i g_{ij} .$$

- The elements in the *j*th column of the generator matrix become the weights for the combination.
  - Because we are working in GF(2), g_{ij} has only two values: 0 or 1.
    - When it is 1, we use b_i in the sum.
    - When it is 0, we don't use b_i in the sum.
- Conclusion: For the *j*th column, the *i*th element is determined from whether the *i*th message bit is used in the sum that produces the *j*th element of the codeword $\underline{\mathbf{x}}$.

Codebook of a linear block code

$\underline{\mathbf{d}}$				$\underline{\mathbf{x}}$						
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	0	1	0	1
0	0	1	0	0	0	1	0	1	1	0
0	0	1	1	1	0	0	0	0	1	1
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	0	1	1	0	0	1
0	1	1	0	1	0	1	1	0	1	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	1	1	0	0	0	0
1	0	0	1	0	1	0	0	1	0	1
1	0	1	0	1	1	0	0	1	1	0
1	0	1	1	0	1	1	0	0	1	1
1	1	0	0	0	1	1	1	1	0	0
1	1	0	1	1	1	0	1	0	0	1
1	1	1	0	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	1	1

- Now that we have a sufficiently-large example of a codebook, let’s consider some important types of problems.
- Given a codebook, how can we check that the code is linear?
- Given a codebook, how can we find the corresponding generator matrix?

Codebook of a linear block code

$\underline{\mathbf{d}}$				$\underline{\mathbf{x}}$						
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	0	1	0	1
0	0	1	0	0	0	1	0	1	1	0
0	0	1	1	1	0	0	0	0	1	1
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	0	1	1	0	0	1
0	1	1	0	1	0	1	1	0	1	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	1	1	0	0	0	0
1	0	0	1	0	1	0	0	1	0	1
1	0	1	0	1	1	0	0	1	1	0
1	0	1	1	0	1	1	0	0	1	1
1	1	0	0	0	1	1	1	1	0	0
1	1	0	1	1	1	0	1	0	0	1
1	1	1	0	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	1	1

Note that

- Each bit of the codeword for linear code is either
  - the same as one of the message bits
    - Here, the second bit (x_2) of the codeword is the same as the first bit (b_1) of the message
  - the sum of some bits from the message
    - Here, the first bit (x_1) of the codeword is the sum of the first, second and fourth bits of the message.
- So, each column in the codebook should also satisfy the above structure (relationship).

“Reading” the structure from the codebook.

	d				**x**						
	0	0	0	0	0	0	0	0	0	0	0
d_4	0	0	0	1	1	0	1	0	1	0	1
d_3	0	0	1	0	0	0	1	0	1	1	0
	0	0	1	1	1	0	0	0	0	1	1
d_2	0	1	0	0	1	0	0	1	1	0	0
	0	1	0	1	0	0	1	1	0	0	1
	0	1	1	0	1	0	1	1	0	1	0
	0	1	1	1	0	0	0	1	1	1	1
d_1	1	0	0	0	1	1	1	0	0	0	0
	1	0	0	1	0	1	0	0	1	0	1
	1	0	1	0	1	1	0	0	1	1	0
	1	0	1	1	0	1	1	0	0	1	1
	1	1	0	0	0	1	1	1	1	0	0
	1	1	0	1	1	1	0	1	0	0	1
	1	1	1	0	0	1	0	1	0	1	0
	1	1	1	1	1	1	1	1	1	1	1

- One can “read” the structure (relationship) from the codebook.
- From $x_j = \sum_{i=1}^{k} d_i g_{ij}$, when we look at the message block with a single 1 at position i, then
  - the value of x_j in the corresponding codeword gives g_{ij}
  - $x_1 = d_1 \oplus d_2 \oplus d_4$
  - $x_3 = d_1 \oplus d_3 \oplus d_4$

“Reading” the generator matrix from the codebook.

	$\underline{\mathbf{d}}$				$\underline{\mathbf{x}}$							
	0	0	0	0	0	0	0	0	0	0	0	
d_4	0	0	0	1	1	0	1	0	1	0	1	$\underline{\mathbf{g}}^{(4)}$
d_3	0	0	1	0	0	0	1	0	1	1	0	$\underline{\mathbf{g}}^{(3)}$
	0	0	1	1	1	0	0	0	0	1	1	
d_2	0	1	0	0	1	0	0	1	1	0	0	$\underline{\mathbf{g}}^{(2)}$
	0	1	0	1	0	0	1	1	0	0	1	
	0	1	1	0	1	0	1	1	0	1	0	
	0	1	1	1	0	0	0	1	1	1	1	
d_1	1	0	0	0	1	1	1	0	0	0	0	$\underline{\mathbf{g}}^{(1)}$
	1	0	0	1	0	1	0	0	1	0	1	
	1	0	1	0	1	1	0	0	1	1	0	
	1	0	1	1	0	1	1	0	0	1	1	
	1	1	0	0	0	1	1	1	1	0	0	
	1	1	0	1	1	1	0	1	0	0	1	
	1	1	1	0	0	1	0	1	0	1	0	
	1	1	1	1	1	1	1	1	1	1	1	

- One can also “read” **G** from the codebook.
- From $\underline{\mathbf{x}} = \underline{\mathbf{b}}\mathbf{G} = \sum_{j=1}^{k} b_j \underline{\mathbf{g}}^{(j)}$, when we look at the message block with a single 1 at position i, then the corresponding codeword is the same as $\underline{\mathbf{g}}^{(j)}$.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} = \mathbf{G}$$

Checking linearity of a code

	$\underline{\mathbf{d}}$				$\underline{\mathbf{x}}$						
	0	0	0	0	0	0	0	0	0	0	0
d_4	0	0	0	1	1	0	1	0	1	0	1
d_3	0	0	1	0	0	0	1	0	1	1	0
	0	0	1	1	1	0	0	0	0	1	1
d_2	0	1	0	0	1	0	0	1	1	0	0
	0	1	0	1	0	0	1	1	0	0	1
	0	1	1	0	1	0	1	1	0	1	0
	0	1	1	1	0	0	0	1	1	1	1
d_1	1	0	0	0	1	1	1	0	0	0	0
	1	0	0	1	0	1	0	0	1	0	1
	1	0	1	0	1	1	0	0	1	1	0
	1	0	1	1	0	1	1	0	0	1	1
	1	1	0	0	0	1	1	1	1	0	0
	1	1	0	1	1	1	0	1	0	0	1
	1	1	1	0	0	1	0	1	0	1	0
	1	1	1	1	1	1	1	1	1	1	1

- Another technique for checking linearity of a code when the codebook is provided
  is to look at each column of the codeword part.
- Write down the equation by reading the structure from appropriate row discussed earlier.
  - For example, here, we read $x_1 = d_1 \oplus d_2 \oplus d_4$.
- Then, we add the corresponding columns of the message part and check whether the sum is the same as the corresponding codeword column.
- So, we need to check n summations.
  - Direct checking that we discussed earlier consider $\binom{M-1}{2}$ summations.

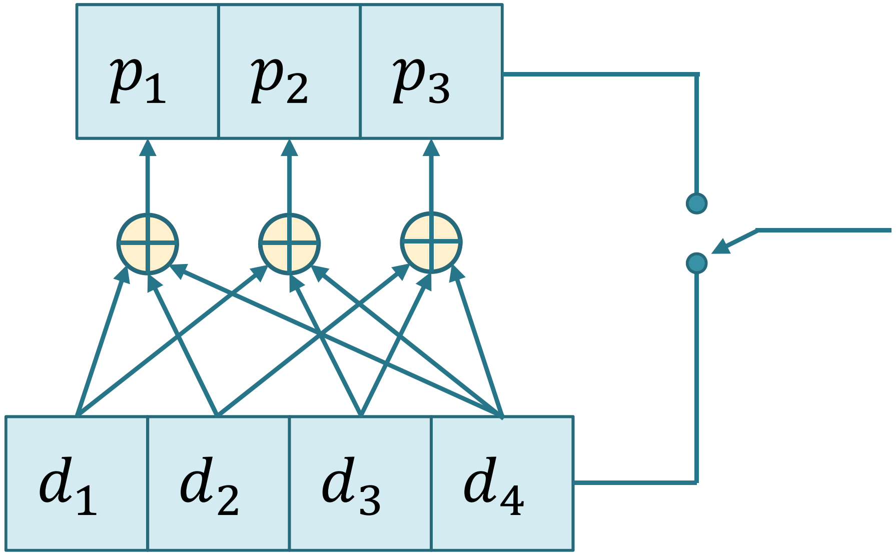
Checking linearity of a code

	$\underline{\mathbf{d}}$				$\underline{\mathbf{x}}$							
	0	0	0	0	0	0	0	0	0	0	0	
d_4	0	0	0	1	1	0	1	0	1	0	1	$\underline{\mathbf{g}}^{(4)}$
d_3	0	0	1	0	0	0	1	0	1	1	0	$\underline{\mathbf{g}}^{(3)}$
	0	0	1	1	1	0	0	0	0	1	1	
d_2	0	1	0	0	1	0	0	1	1	0	0	$\underline{\mathbf{g}}^{(2)}$
	0	1	0	1	0	0	1	1	0	0	1	
	0	1	1	0	1	0	1	1	0	1	0	
	0	1	1	1	0	0	0	1	1	1	1	
d_1	1	0	0	0	1	1	1	0	0	0	0	$\underline{\mathbf{g}}^{(1)}$
	1	0	0	1	0	1	0	0	1	0	1	
	1	0	1	0	1	1	0	0	1	1	0	
	1	0	1	1	0	1	1	0	0	1	1	
	1	1	0	0	1	1	1	1	1	0	0	
	1	1	0	1	1	1	0	1	0	0	1	
	1	1	1	0	0	1	0	1	0	1	0	
	1	1	1	1	1	1	1	1	1	1	1	

- Here is an example of non-linear code.
- Again, we read $x_1 = d_1 \oplus d_2 \oplus d_4$.
- We add the message columns corresponding to d_1, d_2, d_4,
  - We see that the first bit of the 13th codeword does not conform with the structure above.
  - The corresponding message is 1100.
  - We see that $\underline{\mathbf{g}}^{(1)}$ and $\underline{\mathbf{g}}^{(2)}$ are codewords but $\underline{\mathbf{g}}^{(1)} \oplus \underline{\mathbf{g}}^{(2)} =$ 0111100 is not one of the codewords.

Implementation

- Linear block codes are typically implemented with modulo-2 adders tied to the appropriate stages of a shift register.



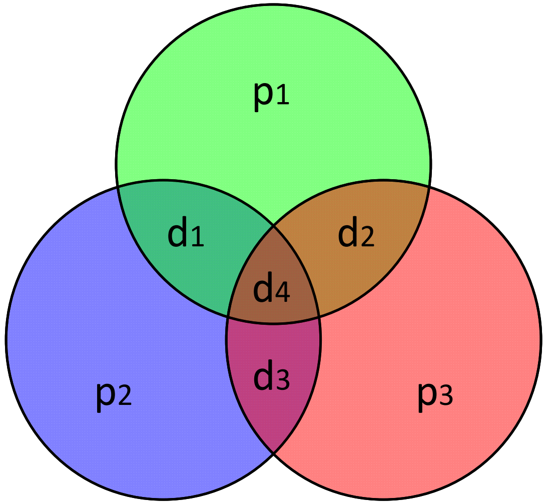
$$\underline{\mathbf{x}} = [p_1 \quad d_1 \quad p_2 \quad d_2 \quad p_3 \quad d_3 \quad d_4]$$

where

$$\begin{aligned} p_1 &= d_1 \oplus d_2 \oplus d_4 \\ p_2 &= d_1 \oplus d_3 \oplus d_4 \\ p_3 &= d_2 \oplus d_3 \oplus d_4 \end{aligned}$$

Back to

Hamming codes: Ex. 1



$$\underline{\mathbf{x}} = \left[\begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ p_1 & d_1 & p_2 & d_2 & p_3 & d_3 & d_4 \end{array}\right]$$

Structure in the codeword:

$$\begin{aligned} p_1 &= d_1 \oplus d_2 \oplus d_4 \\ p_2 &= d_1 \oplus d_3 \oplus d_4 \\ p_3 &= d_2 \oplus d_3 \oplus d_4 \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} p_1 \oplus d_1 \oplus d_2 \oplus d_4 &= 0 \\ p_2 \oplus d_1 \oplus d_3 \oplus d_4 &= 0 \\ p_3 \oplus d_2 \oplus d_3 \oplus d_4 &= 0 \end{aligned}$$

At the receiver, we check whether the received vector $\underline{\mathbf{y}}$ still satisfies these conditions via computing the **syndrome vector**:

$$\underline{\mathbf{s}} = \begin{array}{c} \left[\begin{array}{ccccccc} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \end{array}\right] \\ \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ p_1 & d_1 & p_2 & d_2 & p_3 & d_3 & d_4 \end{array} \end{array} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \underline{0}?$$

Parity Check Matrix: Ex 1

- Intuitively, the **parity check matrix** $\mathbf{H}$, as the name suggests, tells which bits in the observed vector $\underline{\mathbf{y}}$ are used to “check” for validity of $\underline{\mathbf{y}}$.
- The number of rows is the same as the number of conditions to check (which is the same as the number of parity check bits).
- For each row, a one indicates that the bits (including the bits in the parity positions) are used in the validity check calculation.

Structure in the codeword:

$$p_1 \oplus d_1 \oplus d_2 \oplus d_4 = 0$$
$$p_2 \oplus d_1 \oplus d_3 \oplus d_4 = 0$$
$$p_3 \oplus d_2 \oplus d_3 \oplus d_4 = 0$$

⟺

$$\begin{array}{ccccccc} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ p_1 & d_1 & p_2 & d_2 & p_3 & d_3 & d_4 \end{array}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Parity Check Matrix: Ex 1

Relationship between **G** and **H**.

$$\mathbf{G} = \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ p_1 & d_1 & p_2 & d_2 & p_3 & d_3 & d_4 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array} \quad \Longleftrightarrow \quad \mathbf{H} = \begin{array}{ccccccc} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ p_1 & d_1 & p_2 & d_2 & p_3 & d_3 & d_4 \\ \hline 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}$$

Parity Check Matrix

Key property:

$$\boxed{\mathbf{G}\mathbf{H}^T = \mathbf{0}_{k\times(n-k)}}$$

Proof:

- When there is no error $(\underline{\mathbf{e}} = \underline{\mathbf{0}})$, the syndrome vector calculation should give $\underline{\mathbf{s}} = \underline{\mathbf{0}}$.
- By definition,

$$\underline{\mathbf{s}} = \underline{\mathbf{y}}\mathbf{H}^T = (\underline{\mathbf{x}} \oplus \underline{\mathbf{e}})\mathbf{H}^T = \underline{\mathbf{x}}\mathbf{H}^T \oplus \underline{\mathbf{e}}\mathbf{H}^T = \underline{\mathbf{b}}\mathbf{G}\mathbf{H}^T \oplus \underline{\mathbf{e}}\mathbf{H}^T.$$

- Therefore, when $\underline{\mathbf{e}} = \underline{\mathbf{0}}$, we have $\underline{\mathbf{s}} = \underline{\mathbf{b}}\mathbf{G}\mathbf{H}^T$.
- To have $\underline{\mathbf{s}} = \underline{\mathbf{0}}$ for any $\underline{\mathbf{b}}$, we must have $\mathbf{G}\mathbf{H}^T = \underline{\mathbf{0}}$.

Systematic Encoding

- Code constructed with distinct information bits and check bits in each codeword are called **systematic codes**.
  - **Message bits are "visible" in the codeword**.
- Popular forms of **G**:

$$\mathbf{G} = \left[\begin{array}{c|c} \mathbf{P}_{k\times(n-k)} & \mathbf{I}_k \end{array}\right]$$

$$\begin{aligned} \underline{\mathbf{x}} = \underline{\mathbf{b}}\mathbf{G} &= \left[\begin{array}{cccc} b_1 & b_2 & \cdots & b_k \end{array}\right] \left[\begin{array}{c|c} \mathbf{P}_{k\times(n-k)} & \mathbf{I}_k \end{array}\right] \\ &= \left[\begin{array}{cccc|cccc} x_1 & x_2 & \cdots & x_{n-k} & \underset{x_{n-k+1}}{b_1} & \underset{x_{n-k+2}}{b_2} & \cdots & \underset{x_n}{b_k} \end{array}\right] \end{aligned}$$

$$\mathbf{G} = \left[\begin{array}{c|c} \mathbf{I}_k & \mathbf{P}_{k\times(n-k)} \end{array}\right]$$

$$\begin{aligned} \underline{\mathbf{x}} = \underline{\mathbf{b}}\mathbf{G} &= \left[\begin{array}{cccc} b_1 & b_2 & \cdots & b_k \end{array}\right] \left[\begin{array}{c|c} \mathbf{I}_k & \mathbf{P}_{k\times(n-k)} \end{array}\right] \\ &= \left[\begin{array}{cccc|cccc} \underset{x_1}{b_1} & \underset{x_2}{b_2} & \cdots & \underset{x_k}{b_k} & x_{k+1} & x_{k+2} & \cdots & x_n \end{array}\right] \end{aligned}$$

Parity check matrix

- For the generators matrices we discussed in the previous slide, the corresponding **parity check matrix** can be found easily:

$$\mathbf{G} = \left[\mathbf{P}_{k\times(n-k)} \;\vdots\; \mathbf{I}_k\right] \Longrightarrow \mathbf{H} = \left[\mathbf{I}_{n-k} \;\vdots\; -\mathbf{P}^T\right]$$

$$\text{Check: } \mathbf{G}\mathbf{H}^T = \left[\mathbf{P} \;\vdots\; \mathbf{I}\right]\begin{bmatrix}\mathbf{I} \\ \textbf{-P}\end{bmatrix} = \mathbf{P} \oplus (-\mathbf{P}) = \mathbf{0}_{k\times(n-k)}$$

$$\mathbf{G} = \left[\mathbf{I}_k \;\vdots\; \mathbf{P}_{k\times(n-k)}\right] \Longrightarrow \mathbf{H} = \left[-\mathbf{P}^T \;\vdots\; \mathbf{I}_{n-k}\right]$$

Hamming codes: Ex. 2

- Systematic (7,4) Hamming Codes

$$\mathbf{G} = \left[\begin{array}{ccc|cccc} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\mathbf{H} = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}\right]$$

Hamming codes

Now, we will gives a general recipe for constructing Hamming codes.

Parameters:

- $m = n - k =$ number of parity bits
- $n = 2^m - 1 \in \{3,7,15,31,63,127, \dots\}$
- $k = n - m = 2^m - m - 1$

It can be shown that, for Hamming codes,

- $d_{\text{min}} = 3.$
- Error correcting capability: $t = 1$

Construction of Hamming Codes

- Start with m.

1. Parity check matrix **H**:
   - Construct a matrix whose columns consist of *all* nonzero binary m-tuples.
   - The ordering of the columns is arbitrary.
     However, next step is easy when the columns are arranged so that $\mathbf{H} = \left[\mathbf{I}_m \;\vdots\; \mathbf{P}\right]$.
2. Generator matrix **G**:
   - When $\mathbf{H} = \left[\mathbf{I}_m \;\vdots\; \mathbf{P}\right]$, we have $\mathbf{G} = \left[-\mathbf{P}^T \;\vdots\; \mathbf{I}_k\right] = \left[\mathbf{P}^T \;\vdots\; \mathbf{I}_k\right]$.

Hamming codes: Ex. 2

- Systematic (7,4) Hamming Codes

$$\mathbf{H} = \left[\begin{array}{ccc:c} 1 & 0 & 0 & \multirow{3}{*}{$\overbrace{\begin{matrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{matrix}}^{-\mathbf{P}^T}$} \\ 0 & 1 & 0 & \\ 0 & 0 & 1 & \end{array}\right]$$

- Columns are all possible 3-bit vectors
- We arrange the columns so that $\mathbf{I}_3$ is on the left to make the code systematic. (One can also put $\mathbf{I}_3$ on the right.)

$$\mathbf{G} = \left[\begin{array}{c:cccc} \underbrace{\begin{matrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{matrix}}_{\mathbf{P}} & \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \end{array}\right]$$

- Note that the size of the identity matrices in **G** and **H** are not the same.

Minimum Distance Decoding

- At the decoder, suppose we want to use minimum distance decoding, then
  - The decoder needs to have the list of all the possible codewords so that it can compare their distances to the received vector $\underline{\mathbf{y}}$.
  - There are 2^k codewords each having n bits.
    Therefore, saving these takes $2^k \times n$ bits.
  - Also, we will need to perform the comparison 2^k times.
- Alternatively, we can utilize the syndrome vector (which is computed from the parity-check matrix).
  - The syndrome vector is computed from the parity-check matrix $\mathbf{H}$.
  - Therefore, saving $\mathbf{H}$ takes $(n-k) \times n$ bits.

Minimum Distance Decoding

- Observe that

$$d(\underline{\mathbf{x}}, \underline{\mathbf{y}}) = w(\underline{\mathbf{x}} \oplus \underline{\mathbf{y}}) = w(\underline{\mathbf{e}})$$

- Therefore, minimizing the distance is the same as minimizing the weight of the error pattern.
- New goal:
  - find the decoded error pattern $\hat{\underline{\mathbf{e}}}$ with the minimum weight
  - then, the decoded codeword is $\hat{\underline{\mathbf{x}}} = \underline{\mathbf{y}} \oplus \hat{\underline{\mathbf{e}}}$
- Once we know $\hat{\underline{\mathbf{x}}}$ we can directly extract the message part from the decoded codeword if we are using systematic code.
- For example, consider

$$\mathbf{G} = \left[\begin{array}{ccc|cccc} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Suppose $\hat{\underline{\mathbf{x}}} = 1011010$, then we know that the decoded message is $\hat{\underline{\mathbf{b}}} = 1010$.

Properties of Syndrome Vector

- From $\mathbf{G}\mathbf{H}^T = \mathbf{0}$, we have

$$\underline{\mathbf{s}} = \underline{\mathbf{y}}\mathbf{H}^T = \left(\underline{\mathbf{x}} \oplus \underline{\mathbf{e}}\right)\mathbf{H}^T = \left(\underline{\mathbf{b}}\mathbf{G} \oplus \underline{\mathbf{e}}\right)\mathbf{H}^T = \underline{\mathbf{e}}\mathbf{H}^T$$

- Thinking of $\mathbf{H}$ as a matrix with many columns inside,

$$\mathbf{H} = \begin{bmatrix} \underline{\mathbf{h}}_1 \\ \boxed{\quad \underline{\mathbf{h}}_2 \quad} \\ \vdots \\ \underline{\mathbf{h}}_{n-k} \end{bmatrix}_{(n-k)\times n} = \begin{bmatrix} \underline{\mathbf{v}}_1^T & \boxed{\underline{\mathbf{v}}_2^T} & \cdots & \underline{\mathbf{v}}_n^T \end{bmatrix}$$

$$\underline{\mathbf{s}} = \underline{\mathbf{e}}\mathbf{H}^T = \sum_{j=1}^{n} e_j \underline{\mathbf{v}}_j$$

- Therefore, $\underline{\mathbf{s}}$ is a linear combination of the columns of $\mathbf{H}$.

Hamming Codes: Ex. 2

$$\mathbf{H} = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}\right]$$

$$\underline{\mathbf{s}} = \underline{\mathbf{e}}\mathbf{H}^T = \underbrace{\sum_{j=1}^{n} e_j \underline{\mathbf{v}}_j}_{\text{Linear combination of the columns of } \mathbf{H}}$$

Note that for an error pattern with a single one in the j^{th} coordinate position, the syndrome $\underline{\mathbf{s}} = \underline{\mathbf{y}}\mathbf{H}^T$ is the same as the j^{th} column of $\mathbf{H}$.

Error pattern $\underline{\mathbf{e}}$	Syndrome = $\underline{\mathbf{e}}\mathbf{H}^T$
(0,0,0,0,0,0,0)	(0,0,0)
(0,0,0,0,0,0,1)	(1,1,1)
(0,0,0,0,0,1,0)	(1,1,0)
(0,0,0,0,1,0,0)	(1,0,1)
(0,0,0,1,0,0,0)	(0,1,1)
(0,0,1,0,0,0,0)	(0,0,1)
(0,1,0,0,0,0,0)	(0,1,0)
(1,0,0,0,0,0,0)	(1,0,0)

Properties of Syndrome Vector

- We will assume that the columns of $\mathbf{H}$ are nonzero and distinct.
  - This is automatically satisfied for Hamming codes constructed from our recipe.
- Case 1: When $\underline{\mathbf{e}} = \underline{\mathbf{0}}$, we have $\underline{\mathbf{s}} = \underline{\mathbf{0}}$.
  - When $\underline{\mathbf{s}} = \underline{\mathbf{0}}$, we can conclude that $\hat{\underline{\mathbf{e}}} = \underline{\mathbf{0}}$.
    - There can also be $\underline{\mathbf{e}} \neq \underline{\mathbf{0}}$ that gives $\underline{\mathbf{s}} = \underline{\mathbf{0}}$.
      - For example, any nonzero $\tilde{\underline{\mathbf{e}}} \in \mathcal{C}$, will also give $\underline{\mathbf{s}} = \underline{\mathbf{0}}$.
      - However, they have larger weight than $\underline{\mathbf{e}} = \underline{\mathbf{0}}$.
    - The decoded codeword is the same as the received vector.
- Case 2: When, $e_i = \begin{cases} 0, & i = j, \\ 1, & i \neq j, \end{cases}$ (a pattern with a single one in the j^{th} position)

  we have $\underline{\mathbf{s}} = \underline{\mathbf{v}}_j$ = the j^{th} column of $\mathbf{H}$.
  - When $\underline{\mathbf{s}}$ = the j^{th} column of $\mathbf{H}$, we can conclude that $\hat{e}_i = \begin{cases} 0, & i = j, \\ 1, & i \neq j, \end{cases}$
    - There can also be other $\underline{\mathbf{e}}$ that give $\underline{\mathbf{s}} = \underline{\mathbf{v}}_j$. However, their weights
      - can not be 0 (because, if so, we would have $\underline{\mathbf{s}} = \underline{\mathbf{0}}$ but the columns of $\mathbf{H}$ are nonzero)
      - nor 1 (because the columns of $\mathbf{H}$ are distinct).
    - We flip the j^{th} bit of the received vector to get the decoded codeword.

Decoding Algorithm

- Assumption: the columns of **H** are nonzero and distinct.
- Compute the **syndrome** $\underline{\mathbf{s}} = \underline{\mathbf{y}}\mathbf{H}^T$ for the received vector.
- Case 1: If $\underline{\mathbf{s}} = \underline{\mathbf{0}}$, set $\underline{\hat{\mathbf{x}}} = \underline{\mathbf{y}}$.
- Case 2: If $\underline{\mathbf{s}} \neq \underline{\mathbf{0}}$,
  - determine the position j of the column of **H** that is the same as (the transposition) of the syndrome,
  - set $\underline{\hat{\mathbf{x}}} = \underline{\mathbf{y}}$ but with the j^{th} bit complemented.
- For Hamming codes, because the columns are constructed from all possible non-zero m-tuples, the syndrome vectors must fall into one of the two cases considered.
- For general linear block codes, the two cases above may not cover every cases.

Hamming Codes: Ex. 1

- Consider the Hamming code with

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \Longleftrightarrow \mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- Suppose we observe $\underline{\mathbf{y}} = [0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1]$ at the receiver. Find the decoded codeword and the decoded message.

[To be explored in the HW]

Hamming Codes: The original method

- Encoding
  - The bit positions that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits.
  - The rest (3, 5, 6, 7, 9, etc.) are filled up with the *k* data bits.
  - Each check bit forces the parity of some collection of bits, including itself, to be even (or odd).
    - To see which check bits the data bit in position *i* contributes to, rewrite *i* as a sum of powers of 2. A bit is checked by just those check bits occurring in its expansion
- Decoding
  - When a codeword arrives, the receiver initializes a counter to zero. It then examines each check bit at position *i* (*i* = 1, 2, 4, 8, …) to see if it has the correct parity.
  - If not, the receiver adds *i* to the counter. If the counter is zero after all the check bits have been examined (i.e., if they were all correct), the codeword is accepted as valid. If the counter is nonzero, it contains the position of the incorrect bit.

Interleaving

- Conventional error-control methods such as parity checking are designed for errors that are isolated or statistically independent events.
- Some errors occur in bursts that span several successive bits.
  - Errors tend to group together in bursts. Thus, errors are no longer independent
  - Examples
    - impulse noise produced by lightning and switching transients
    - fading or in wireless systems
    - channel with memory
- Such multiple errors wreak havoc on the performance of conventional codes and must be combated by special techniques.
- One solution is to spread out the transmitted codewords.
- We consider a type of interleaving called **block interleaving**.

Interleaving: Example

Consider a sequence of *m* blocks of coded data:

$$\left(x_1^{(1)}x_2^{(1)}\cdots x_n^{(1)}\right)\left(x_1^{(2)}x_2^{(2)}\cdots x_n^{(2)}\right)\cdots\left(x_1^{(\ell)}x_2^{(\ell)}\cdots x_n^{(\ell)}\right)$$

$$\Big\Downarrow$$

$$\begin{array}{cccc} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(\ell)} & x_2^{(\ell)} & \cdots & x_n^{(\ell)} \end{array}$$

- Arrange these blocks as rows of a table.
- Normally, we get the bit sequence simply by reading the table by rows.
- With interleaving (by an interleaver), transmission is accomplished by reading out of this table by columns.
- Here, ℓ blocks each of length *n* are interleaved to form a sequence of length ℓn.

$$\Big\Downarrow$$

$$\left(x_1^{(1)}x_1^{(2)}\cdots x_1^{(\ell)}\right)\left(x_2^{(1)}x_2^{(2)}\cdots x_2^{(\ell)}\right)\cdots\left(x_n^{(1)}x_n^{(2)}\cdots x_n^{(\ell)}\right)$$

The received symbols must be deinterleaved (by a deinterleaver) prior to decoding.

Interleaving: Advantage

- Consider the case of a system that can only correct single errors.
- If an error burst happens to the original bit sequence, the system would be overwhelmed and unable to correct the problem.

$$\text{original bit sequence}\quad \left(x_1^{(1)}x_2^{(1)}\cdots x_n^{(1)}\right)\left(x_1^{(2)}x_2^{(2)}\cdots x_n^{(2)}\right)\cdots\left(x_1^{(\ell)}x_2^{(\ell)}\cdots x_n^{(\ell)}\right)$$

$$\text{interleaved transmission}\quad \left(x_1^{(1)}x_1^{(2)}\cdots x_1^{(\ell)}\right)\left(x_2^{(1)}x_2^{(2)}\cdots x_2^{(\ell)}\right)\cdots\left(x_n^{(1)}x_n^{(2)}\cdots x_n^{(\ell)}\right)$$

- However, in the interleaved transmission,
  - successive bits which come from ***different*** original blocks have been corrupted
  - when received, the bit sequence is reordered to its original form and then the FEC can correct the faulty bits
  - Therefore, single error-correction system is able to fix several errors.

Interleaving: Advantage

- If a burst of errors affects at most ℓ consecutive bits, then each original block will have at most one error.
- If a burst of errors affects at most $r\ell$ consecutive bits (assume $r < n$),
  then each original block will have at most r errors.
- Assume that there are no other errors in the transmitted stream of ℓn bits.
  - A single error-correcting code can be used to correct a single burst spanning upto ℓ symbols.
  - A double error-correcting code can be used to correct a single burst spanning upto 2ℓ symbols.