P2 Digital Electronics

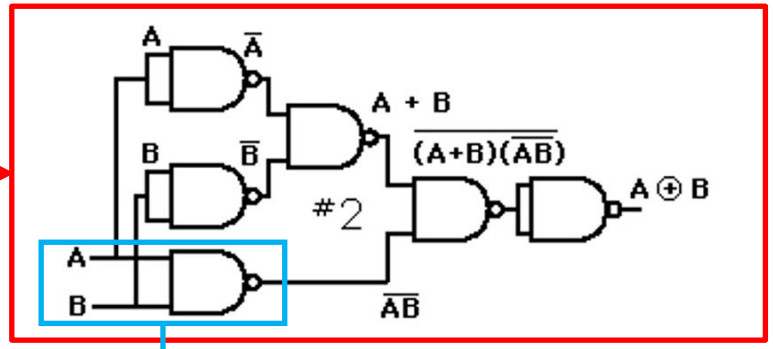
Lecture 3: Binary number representation

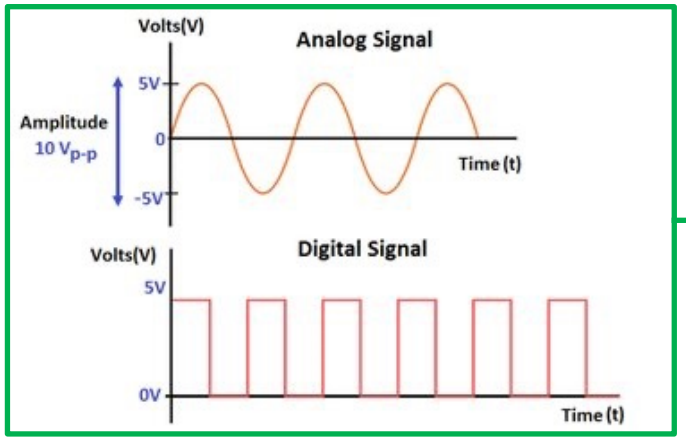
Mark Cannon

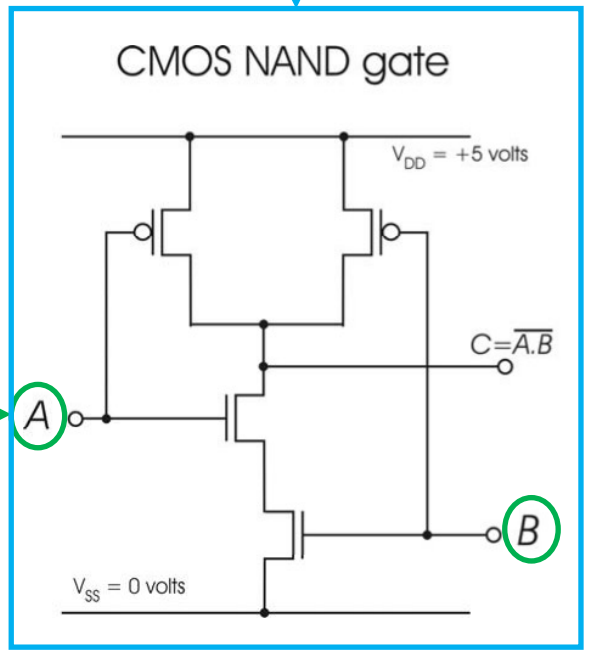
mark.cannon@eng.ox.ac.uk

Trinity Term 2026

Overview of lectures







Overview of lectures

1. Logical functions and logic gates
2. Low level logic design
3. **Binary number representation**
4. Binary arithmetic
5. Integration of digital logic components
6. Memory and sequential circuits
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk

Overview of lecture 3

- Counting in Binary, Octal and Hexadecimal
- Negative numbers
- Binary codes
- Error detection

- What are numbers and how do we represent them?

Binary number representation

There are 10 types of people in the world:

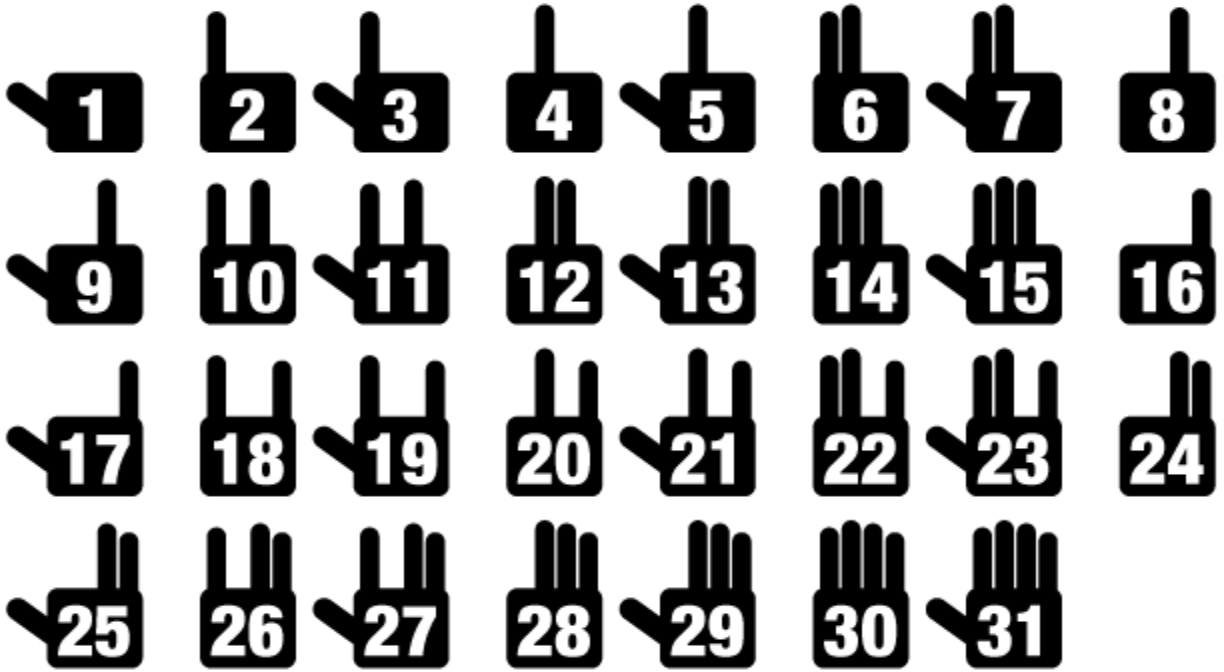
those who understand binary ... and those who don't

Numeral systems

5 digits per hand – two hands - we ended up with base 10

Position isn't important

Can you count higher?



Position is important

Back to basics

Decimal

$$\begin{aligned} 5621_{10} &= 5 \times 10^3 + 6 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 \\ &= 5000 \quad + 600 \quad + 20 \quad + 1 \end{aligned}$$

Nonary

$$\begin{aligned} 5621_9 &= 5 \times 9^3 + 6 \times 9^2 + 2 \times 9^1 + 1 \times 9^0 \\ &= 3645 \quad + 486 \quad + 18 \quad + 1 \\ &= 4150_{10} \end{aligned}$$

Binary

$$\begin{aligned} 101101_2 &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 32 \quad + 0 \quad + 8 \quad + 4 \quad + 0 \quad + 1 \\ &= 45_{10} \end{aligned}$$

Some terminology

Digit — individual numeric symbol in a number

e.g. in 2597_{10} the third digit is 9

Bit — short for "binary digit"

e.g. in 1101_2 the third bit is 0

Least significant bit (LSB) — the bit with the lowest numerical value in a number

e.g. 10110011_2

Most significant bit (MSB) — the bit with the highest numerical value in a number

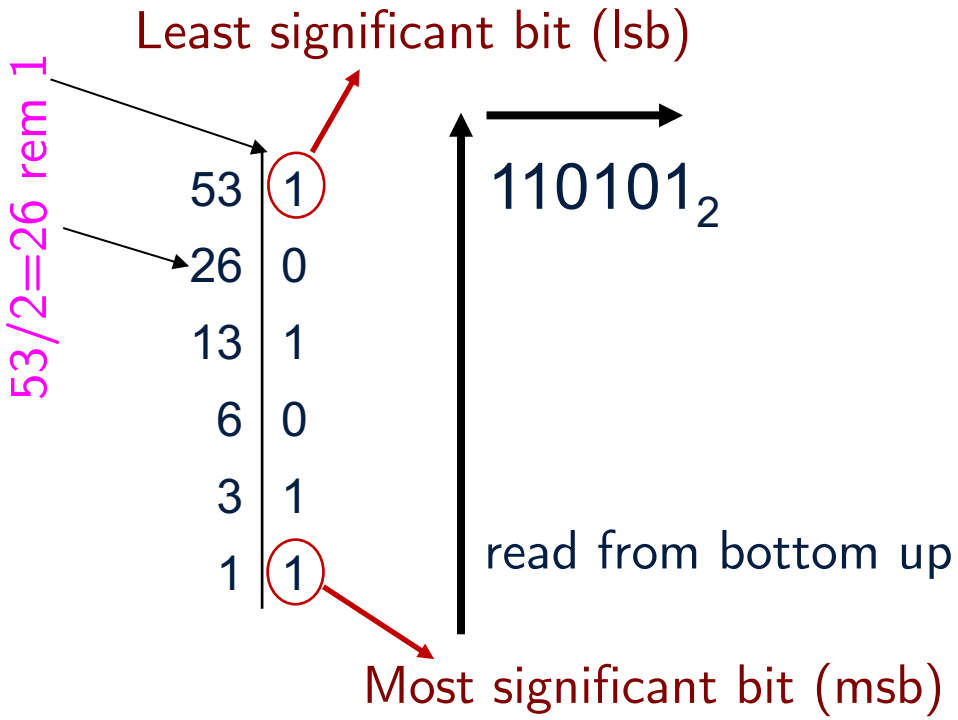
e.g. 10110011_2

How to convert between bases

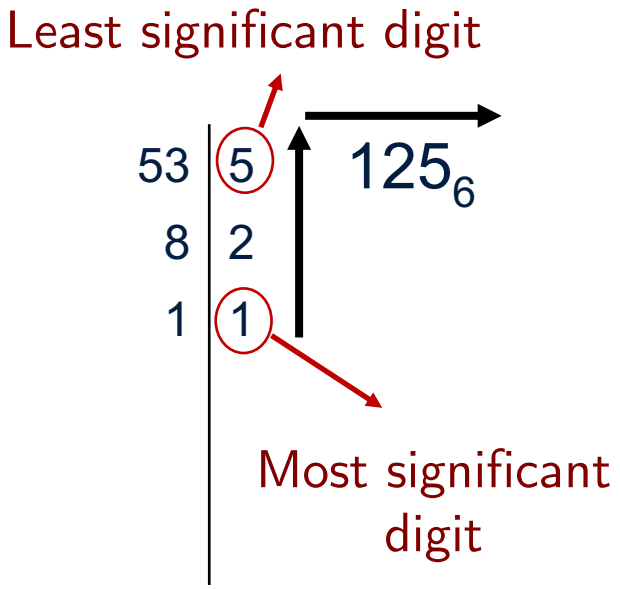
General procedure:

1) Successive division by the number representing the new base
2) The number in the new base is made from remainders

Example: Convert 53_{10} to binary



Example: Convert 53_{10} to senary (base 6)



Hexadecimal

To represent numbers in binary can require long string of digits (eg. 001111010000)

We can use Hexadecimal (base 16) for digital electronics and programming

- group bits for convenience
- one hexadecimal digit: four bits

$$1000_{10} = 001111010000_2$$

$$(512+256+128+64+16)$$

$$1000_{10} = \underbrace{0011}_{3}\ \underbrace{1101}_{D}\ \underbrace{0000_2}_{0_{16}}$$

Notation: hexadecimal numbers can be written as $3D0_{16}$ or $3D0_{HEX}$ or $3D0_{H}$ or 0x3D0

x_{10}	x_{16}	x_2
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

What about negative numbers?

Conventionally, we use a symbol "$-$" before the number, e.g. -123456

How should we represent this in binary?

We could use a bit: e.g. "$+$" $= 0$ and "$-$" $= 1$

Where should this "sign bit" be placed in the number?

- beginning?
- end?
- somewhere else?

} all potentially valid as long as the same convention is used

Other representations are possible

- Offset binary
- 2's complement

Negative numbers: Offset binary

Add a constant negative number to a normally represented number

- $b + c = b_{\text{offset}}$

Most often $c = 2^{n-1}$ (for n-bit number), then:

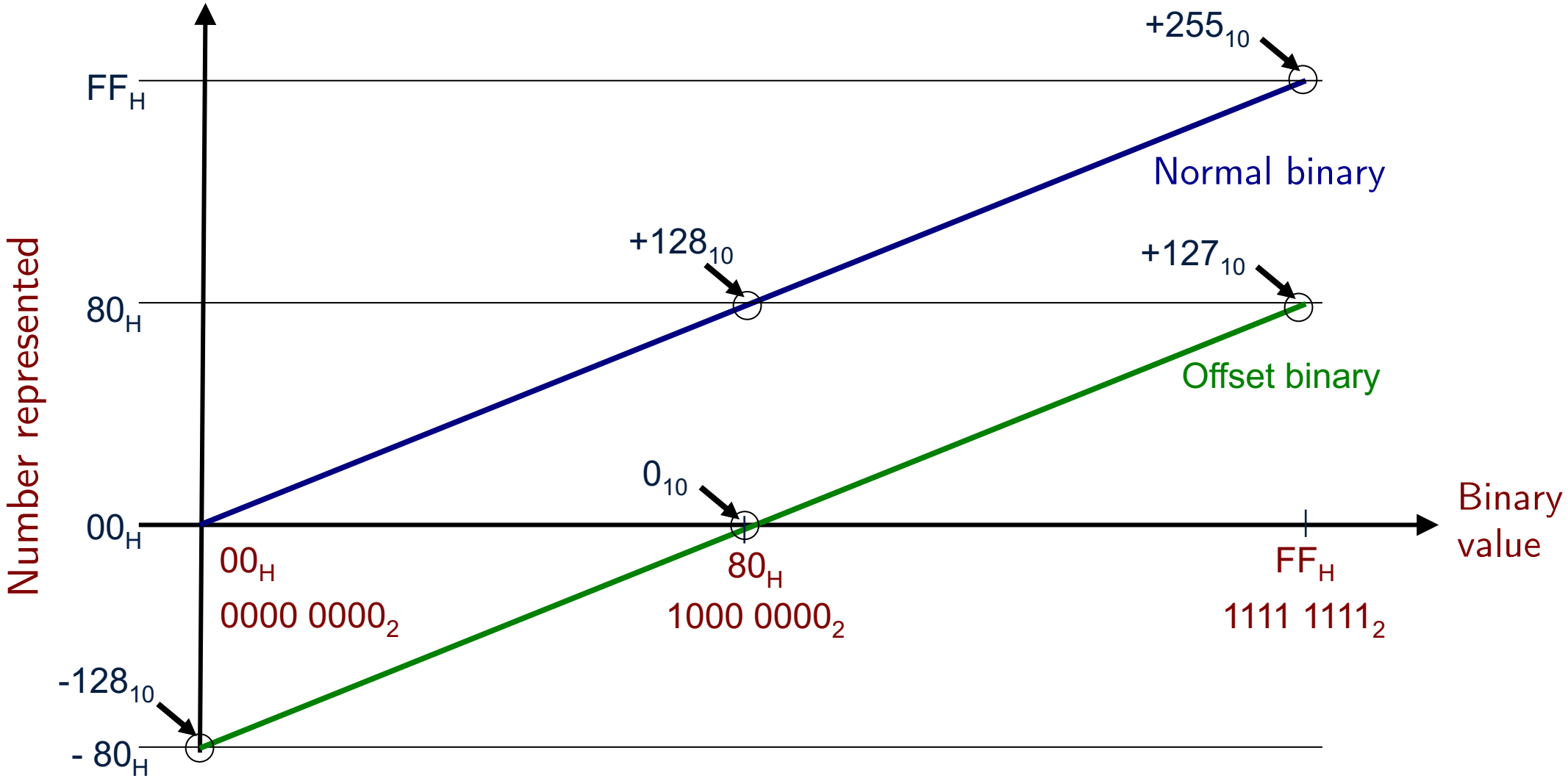
- the minimum negative value is represented by all-zeros
- the max positive value is represented by all-ones

Equivalent to shifting the origin

e.g. for an 8-bit number:

$0000\ 0000_2$	represents	-128_{10}
$0000\ 0001_2$	represents	-127_{10}
$\vdots$		
$1000\ 0000_2$	represents	0_{10}
$\vdots$		
$1111\ 1111_2$	represents	127_{10}

Graphical representation of offset binary

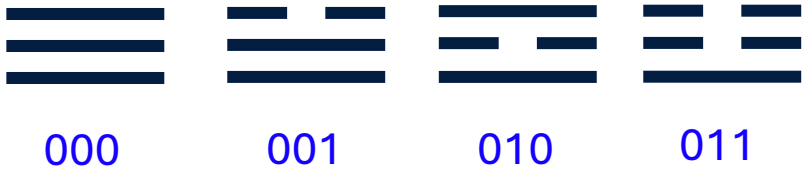


General comments about number representation

- What does a number mean?
  - 10
  - 170
  - 07273612718
- Meaning of the symbols depends on context:
  00000001_2 could mean 1_{10} or -127_{10}
- Digital electronics do not understand the context – just Boolean algebra laws

Origins of binary numbers

- Various forms of binary representation from Egypt, China, India, Africa, ...
- Gottfreid Leibniz (1646–1716) studied binary numbering in 1679
  – inspired by Chinese text I Ching 易經
- Trigrams (sets of three lines) or hexagrams (six) form binary representation



Negative numbers: 2's complement

- 2's complement representation makes it easier to add and subtract positive and negative numbers
- MSB encodes the sign but not as a simple minus symbol...

An m-bit unsigned binary number $d_{m-1} \cdots d_1 d_0$ would be

$$N' = \sum_{i=0}^{m-1} d_i \times 2^i$$

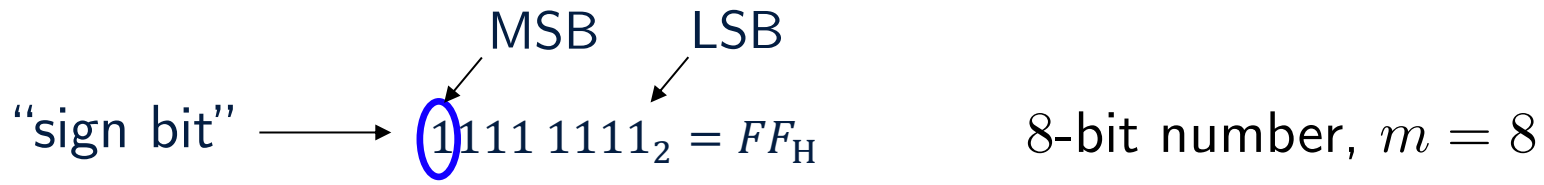
e.g. $m = 4$, $1011_2 = 8 + 2 + 1 = 11_{10}$

An m-bit number $d_{m-1} \cdots d_1 d_0$ in 2's complement is given by

$$N = -1 \times d_{m-1} \times 2^{m-1} + \sum_{i=0}^{m-2} d_i \times 2^i$$

e.g. $m = 4$, $1011_{2'} = -8 + 2 + 1 = -5_{10}$

Negative numbers: 2's complement



$$N = -1 \times d_{m-1} \times 2^{m-1} + \sum_{i=0}^{m-2} d_i \times 2^i$$

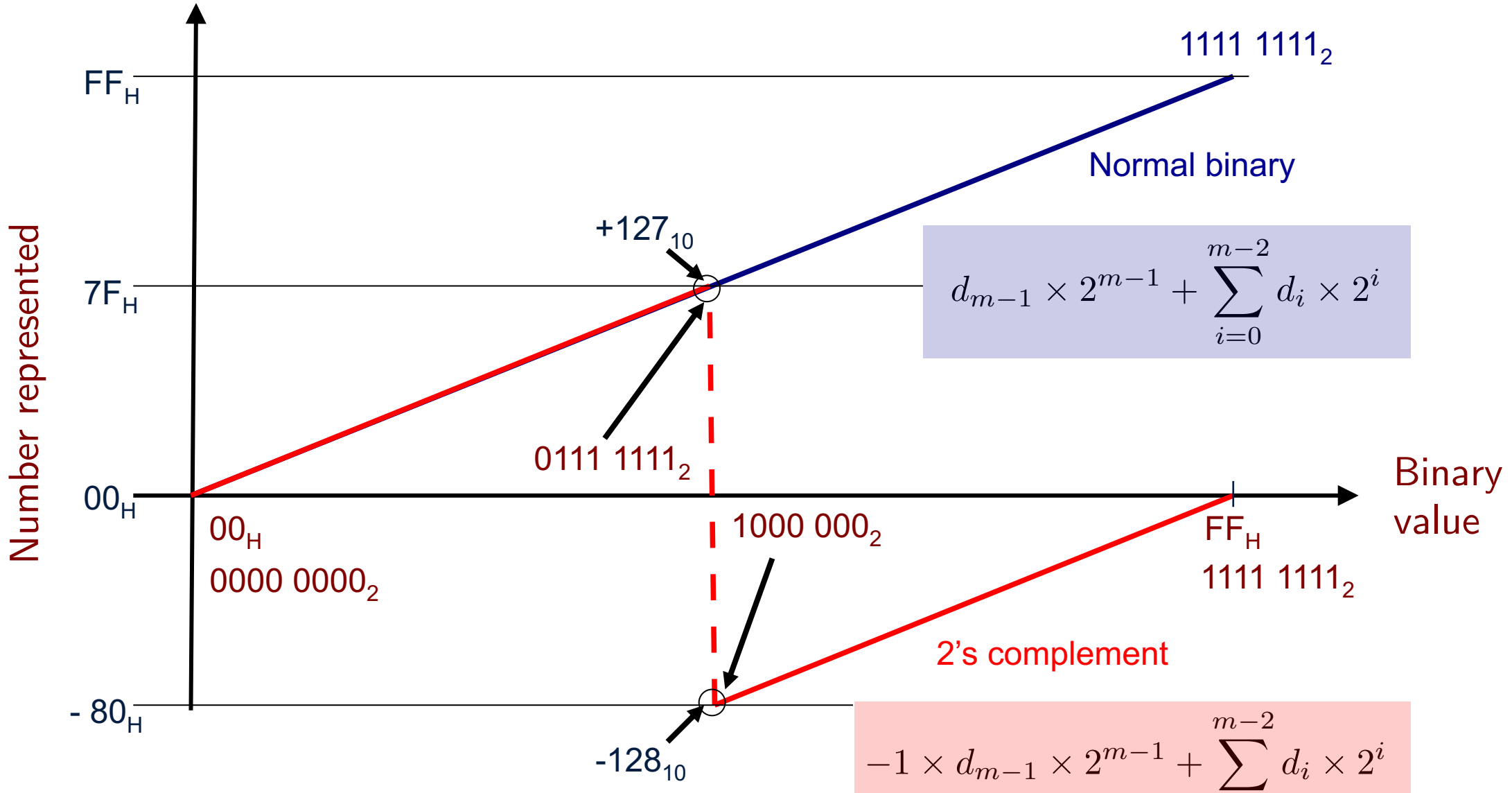
e.g. $01111111_2, = -1\times0\times2^7 + 1111111_2 = 127_{10}$

$11111111_2, = -1\times1\times2^7 + 1111111_2 = -128_{10} + 127_{10} = -1_{10}$

$10000000_2, = -1\times1\times2^7 + 0_2 = -128_{10}$

$10011001_2, = -1\times1\times2^7 + 0011001_2 = -128_{10} + 25_{10} = -103_{10}$

Graphical representation of 2's complement



Obtaining 2's complement representation

1. Write down the binary number for the absolute value $|-86_{10}| = 86_{10} = 01010110_2$
2. Flip all the bits (bitwise NOT) 10101001_2
3. Add 1 $+ \quad 1_2$

$= 10101010_2$

$\Rightarrow \quad -86_{10}$ is represented by 10101010_2

Check: $10101010_2 = -1 \times 1 \times 2^7 + 0101010_2 = -128_{10} + 42_{10} = -86_{10}$

Alternative method (harder for digital electronics): $-b$ is represented as $(2^n - b)_2$, so

$$2^8 - 86_{10} = 256_{10} - 86_{10} = 170_{10} = (128 + 32 + 8 + 2)_{10} = 10101010_2$$

Using 2's complement

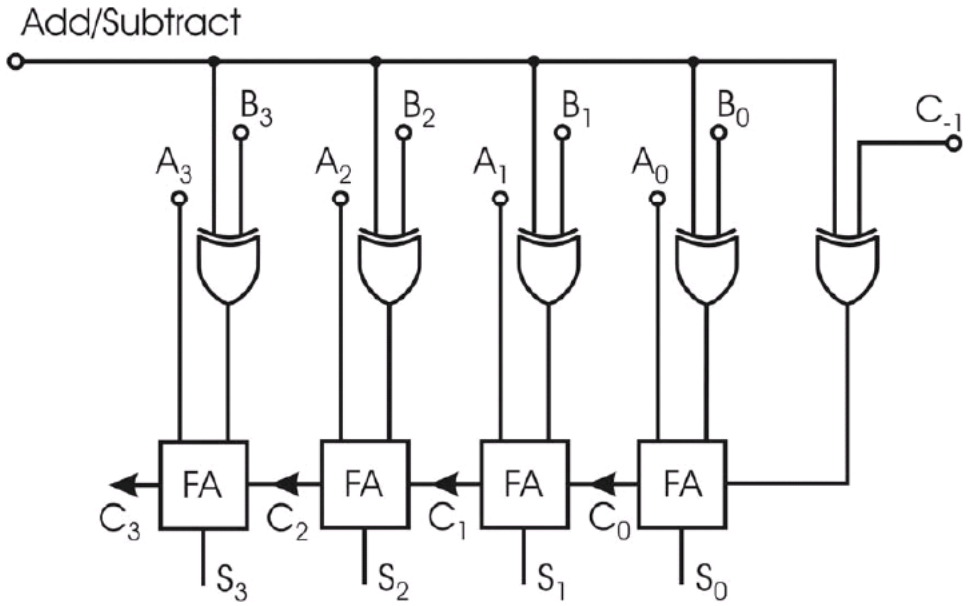
Example : Using 8-bit binary, subtract 86_{10} from 111_{10}

-86_{10} is represented by $1010\ 1010_2$

111_{10} is represented by $0110\ 1111_2$

Add 111_{10} to -86_{10}:

	1010 1010	-86_{10}
+	0110 1111	+ 111_{10}
	~~1~~ 0001 1001	25_{10}



Add/ Subtract	B_i	XOR
0	0	0
0	1	1
1	0	1
1	1	0

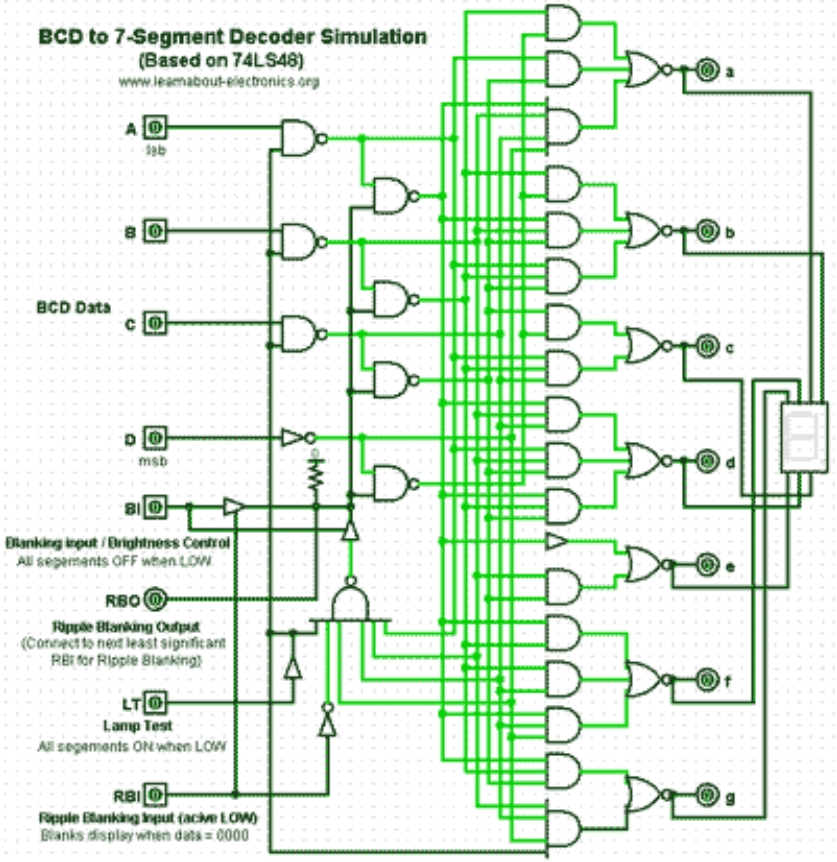
Remember: here we are using only 8-bit numbers so the extra MSB is ignored as it never really exists in the 8-bit system

Binary codes

- Binary code is used more generally to represents characters, instructions, data etc.
- This is useful for transmitting data (it is encoded into a form that can be transmitted) but not only for this...
- BCD = Binary Coded Decimal = digit-by-digit encoding: each digit is replaced by its corresponding 4 bits

$$9257_{10} = \underbrace{1001_2}_{9}\ \underbrace{0010_2}_{2}\ \underbrace{0101_2}_{5}\ \underbrace{0111_2}_{7}$$

Example: Seven segment display
4 input bits → 7 output bits



Binary codes

- Character encoding
- ASCII (American Standard Code for Information Interchange)
  This is a 7-bit code for basic characters, plus some “control” codes

$0010\ 0001_2$
$= 33_{10}$

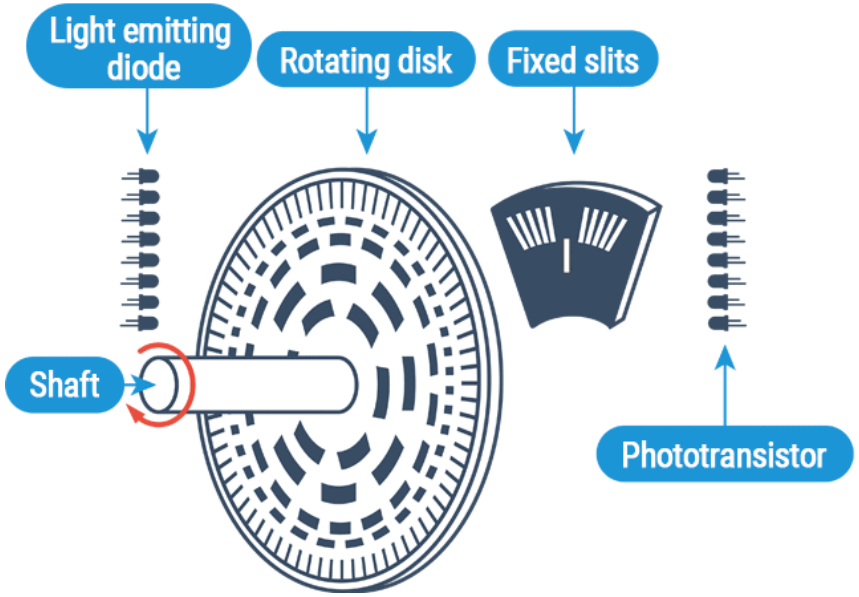
```
! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
```

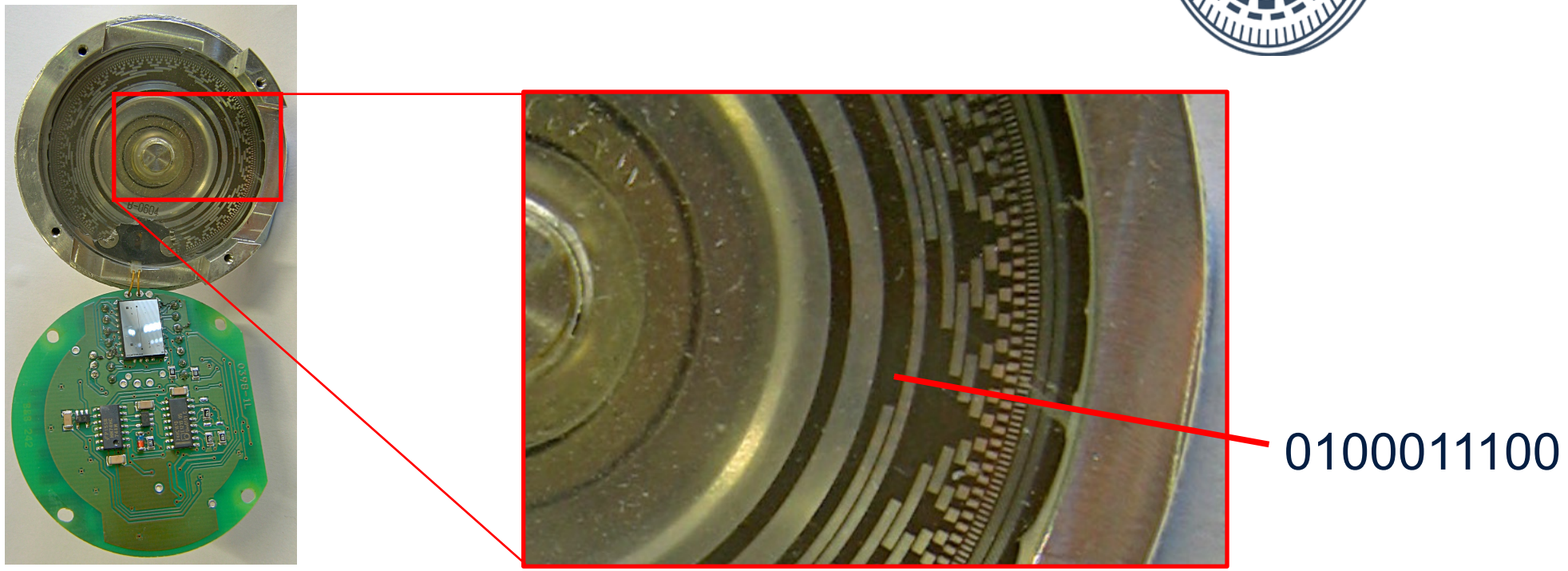
$0111\ 1110_2$
$= 126_{10}$

- Unicode: UTF-8 (8 bit, similar to ASCII), UTF-16, UTF-32, etc.
  These use more bits to encode more symbols

Gray codes

- Arise in e.g. shaft encoder systems used to monitor position of rotating machines
- Typically use a disc with rings of marks in binary number sequence
- Only one bit changes between adjacent numbers

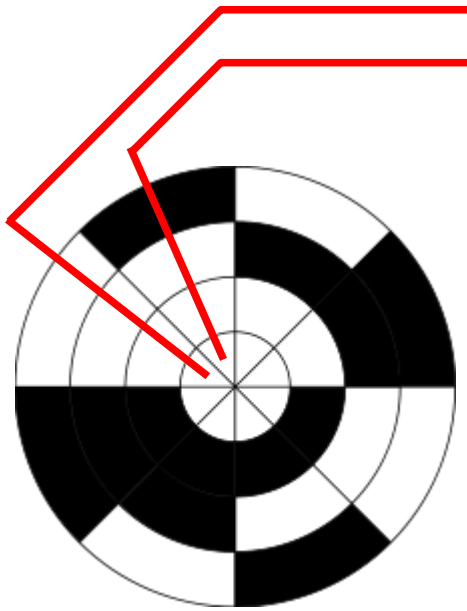




https://en.wikipedia.org/wiki/Gray_code#/media/File:Gray_code_rotary_encoder_13-track_opened.jpg

Gray code advantages

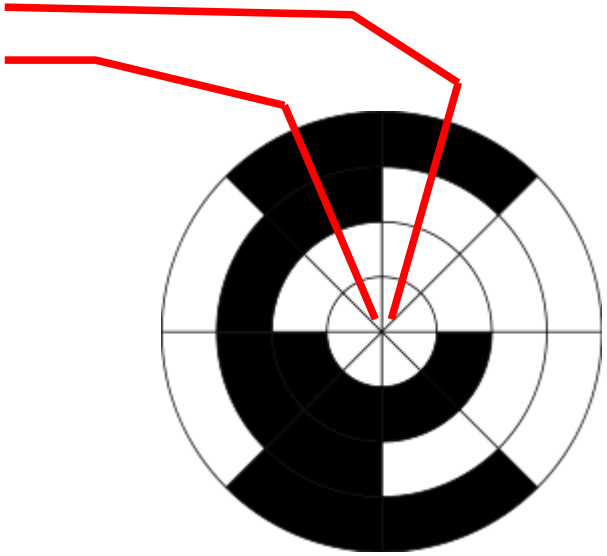
3 bits regular binary
8 positions 45° per line



0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

3 bit Gray code

0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

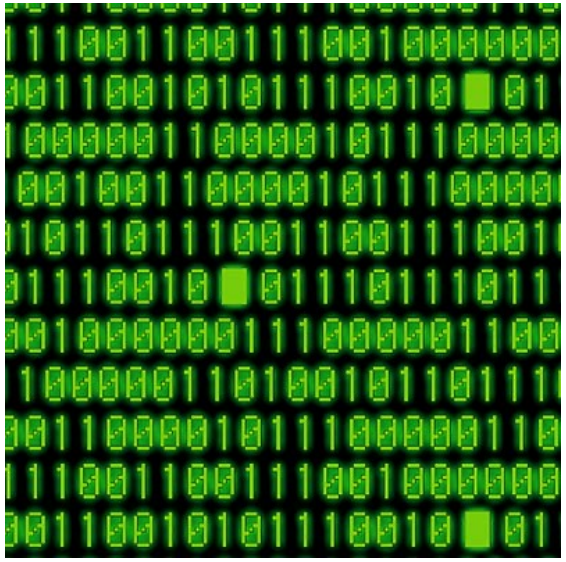


Only one bit changes at each transition – prevention of transient errors

Not only valuable for this mechanical system, also useful in computing where timing is a problem

Error detection

- Digital data encoded as binary numbers
- Data transmission & storage not always 100% reliable
- Error detection and correction techniques are needed to correct single bit errors that might occur
  - Repetition
  - Parity
  - Checksums
  - Cyclic Redundancy



Error detection – Parity

Parity determines if the number we are sending has an even or odd number of non-zero bits

```
111 0010    four 1's - even parity
101 0010    three 1's - odd parity
```

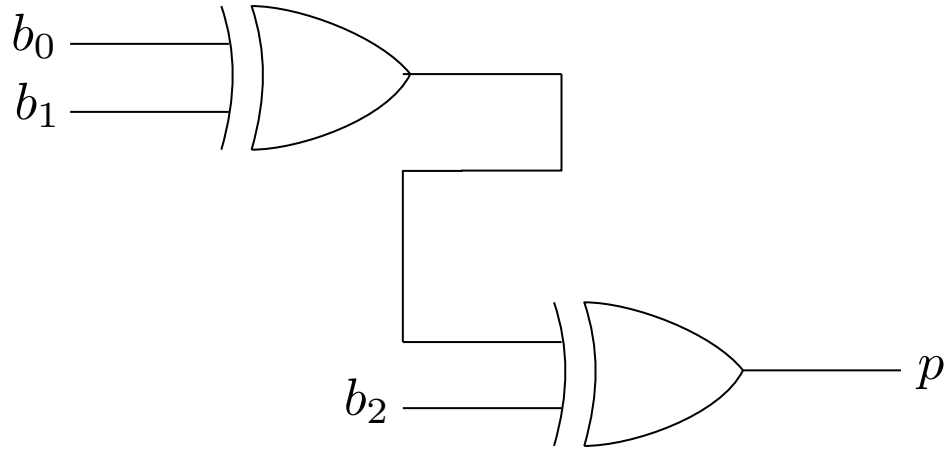
- e.g. ASCII uses 7 bits to encode characters – can use 8th bit to represent parity
- Even parity: even number of bits: 0111 001 0 ← Parity bit
- Odd parity: odd number of bits: 0111 001 1
- Doesn't allow correction of error – no way to know which bit was wrong.
- Limitation in terms of detecting 2-bit errors

Generating a parity bit

Using even parity – i.e. correct values must have an even number of 1s

b_1	b_0	parity
0	0	0
0	1	1
1	0	1
1	1	0

b_2	b_1	b_0	parity
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



b_0 \ b_1b_2	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$p = b_2 \oplus b_1 \oplus b_0$$

Parity: XOR all the bits together and record result

Overview of lectures

1. Logical functions and logic gates
2. Low level logic design
3. **Binary number representation**
4. Binary arithmetic
5. Integration of digital logic components
6. Memory and sequential circuits
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk