COSC2473

Data Error Handling

<u>Intro and Gray Codes</u>

Parity and Hamming Codes

SECDED and Templates

Data Errors

- Digital data comes in a variety of forms:
  - Numerical data
    - eg. Numbers collected from some sampling device
    - Easily read by programs
    - Data errors generally related to instrument uncertainties
  - Categorical data
    - eg Classifications, Groupings, etc
    - Easily read and categorised by programs
    - Wrong-category Error is more problematic (how to correct?)
  - Descriptive / formatted / structured data
    - eg free text, XML, JSON, special file formats
    - Requires syntax (structure) & semantic (meaning) checking

Binary Data Errors

- All the data errors in the previous slide rely on the bits themselves being correctly transmitted.
- What if they are not?
  - Firstly can we ***tolerate*** bit errors?
    - eg Video streams have built in error tolerance and recovery
  - Secondly can we ***detect*** bit errors?
    - This may be sufficient, as we can ask to retransmit the data
  - Thirdly, can we ***correct*** bit errors?
    - If retransmission is not possible or practical (eg Big Data), then can we at least correct some of it?

Data Error Tolerance

- Most binary data is structural
  - Data structures include field lengths, data sizes, formatting rules
  - Having bit errors in structured streaming usually means the downstream system gets confused as to where it is up to and so may need to resynchronise
  - So error tolerance can mean
    - wrong pixel values in images video (only visible to data consumers, not actually a system error.
    - discovery that the system is confused and needs to recover
  - In the case of MPEG video, all stream data headers start with 12 1 bits in a row, which never happens in the data itself.
    - so if a video client gets confused, it simply waits for the next set of 12 1 bits, and restarts from there.
- *In images, salt and pepper noise is where individual pixels get set to white or black instead or their colour.*
  - *In binary data, this is a bit flip.*
- *For purely numerical binary data, we can use Gray Codes for the case where random bits flip.*

Data Error Tolerance – Gray Code

- When numeric data is digitally sampled from an instrument, you cannot be sure if any error is due to but errors in transmission. But you can limit its effects.
  - One way we can limit the effect of such errors is for the device to use Gray codes instead of binary.
- Consider the following:

SourceBin → **0111** → 1-bit error → **1111** → Dest
SourceGray → 0100 → 1-bit error → 1100 → Dest

  - In the first case the 1 bit error has caused the reading to be almost double the correct value, from 7 to 15..
  - In Gray Code, the value for 7 is 0100 and that for 8 is 1100
    - So in the second case, the same value (7) as Gray code suffer the same bit error, but the new value is 8 in Gray Code, and so the damage is reduced.

Data Error Tolerance – Gray Code

- Gray codes are designed so that each value differs from its neighbour by only a single bit, so that single bit errors will only change the value by +/- 1.
- Consider the following truth table (b_2-b_0 are bits of Val)

Val	**Bin**	**Gray**	**b_2**	**b_1**	**b_0**
0	000	000	-	-	-
1	001	001	-	-	-
2	010	011	-	1	flip if b_1=1
3	011	010	-	1	flip if b_1=1
4	100	110	1	flip if b_2=1	flip if b_2=1, flip if b_1=1
5	101	111	1	flip if b_2=1	flip if b_2=1, flip if b_1=1
6	110	101	1	flip	flip if b_1=1
7	111	100	1	flip	flip if b_1=1

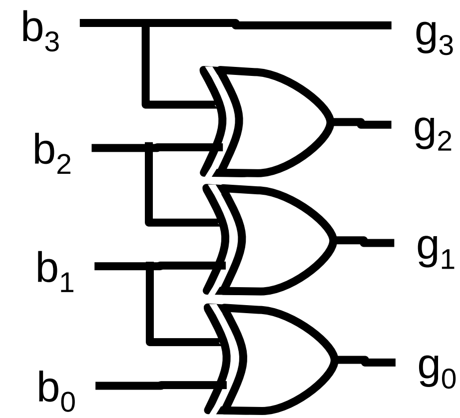
Data Error Tolerance – Gray Code

- Now extend the pattern to b_3 of Val.
  - Does the same pattern hold?

Val	Bin	Gray	Val	Bin	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Data Error Tolerance – Gray Code

- Shown below are the logic gates for conversion from Binary to Gray code. The circuit shows each bit XORed with next higher bit.



<u>Python Code</u>

```
def BinToGray(num):
    return  num ^ (num >> 1)
```

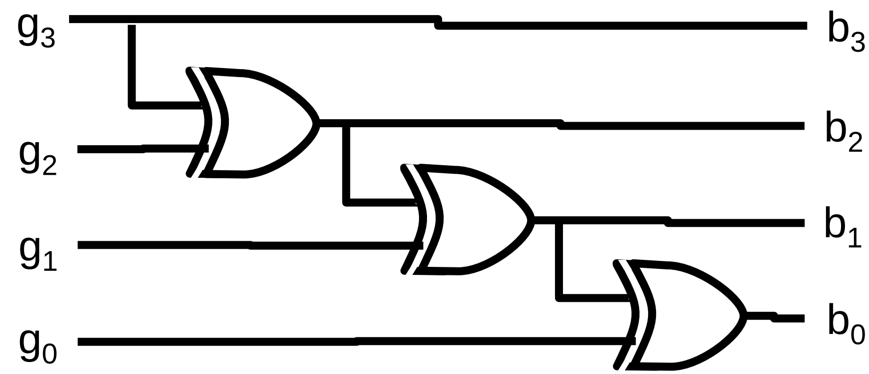
Java Code

```
int function BinToGray(int num) {
    return num ^(num >> 1);
}
```

	Val	b_3	b_2	b_1	b_0
num	6	0	1	1	0
N = num >> 1	3	0	0	1	1
num ^ N	5	**0**	1	0	1

Data Error Tolerance – Gray Code

- Shown below are the logic gates for conversion from Gray Code to Binary. The circuit shows each bit right-propagated with XOR with next lower bit.



Python Code

```
def GrayToBin(num):
    prop = num >> 1
    while prop != 0:
        num ^= prop
        prop >>= 1
        return num
```

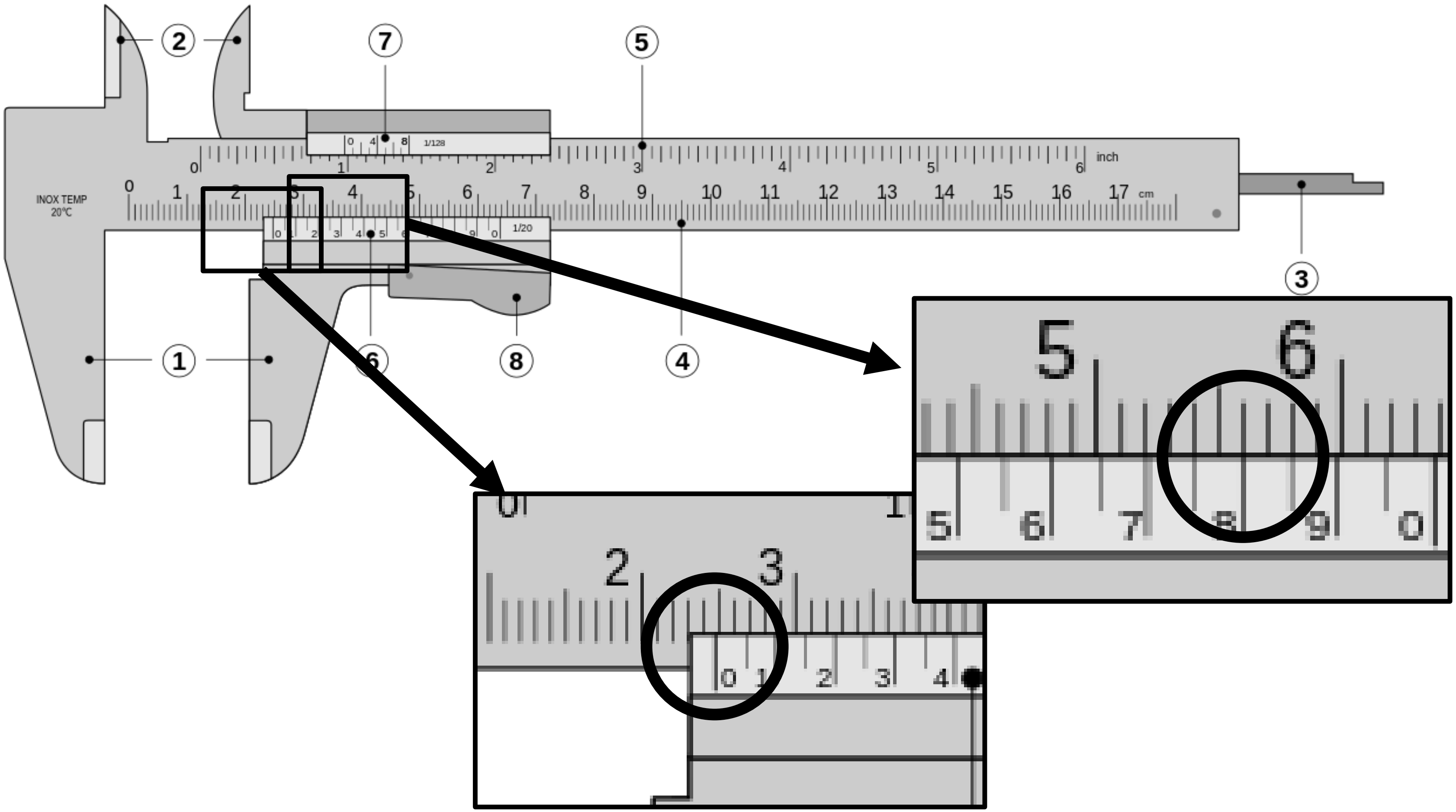
Java Code

```
int function GrayToBin(int num) {
     // Simulate right propagation
    int prop = num >> 1;
    int while prop != 0
        num = num ^ prop;
        prop = prop >> 1;
    }
    return num;
}
```

	Val	b_3	b_2	b_1	b_0
Num. Prop = 2	5	0	1	0	1
num ^= prop, prop = 1	7	0	1	1	1
num ^= prop, prop = 0	6	**0**	1	1	0

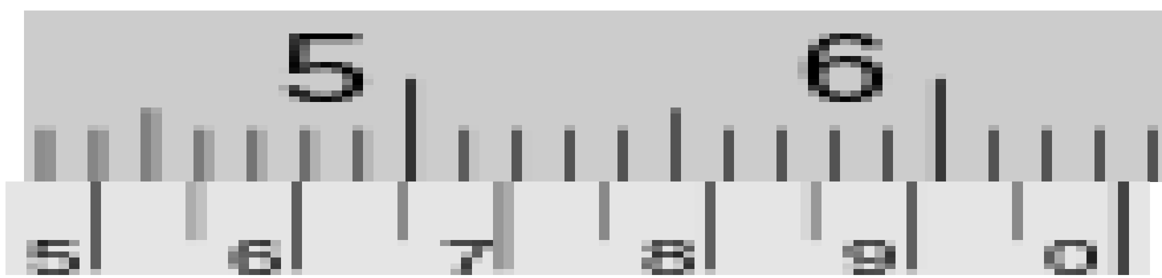
Vernier Caliper

- A vernier caliper is used to measure the width of small objects to 0.1 mm accuracy. ①②③ outside/inside/depth measure. ④⑥ cm measure and vernier. 2.48 cm shown.



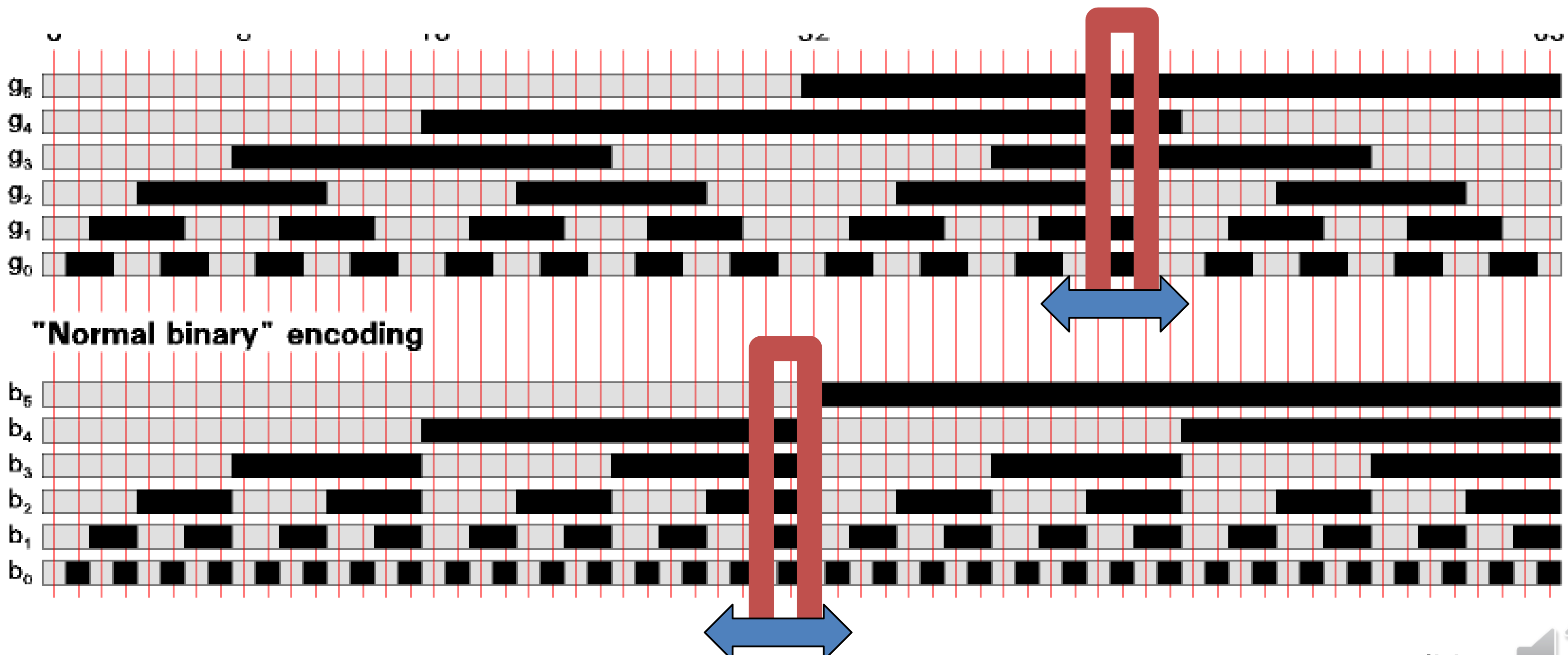
Vernier Scale

- See how the marks line up as the lower part moves.
- This works because the width between the lower mark is 90% of the widths between the upper marks.
- Thus each movement of 10% pf the mark spacing causes a different pair of marks to line up.



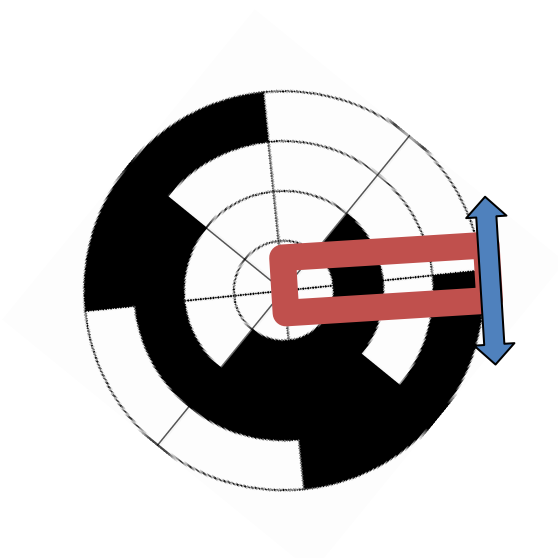
Example of Gray Code for Position

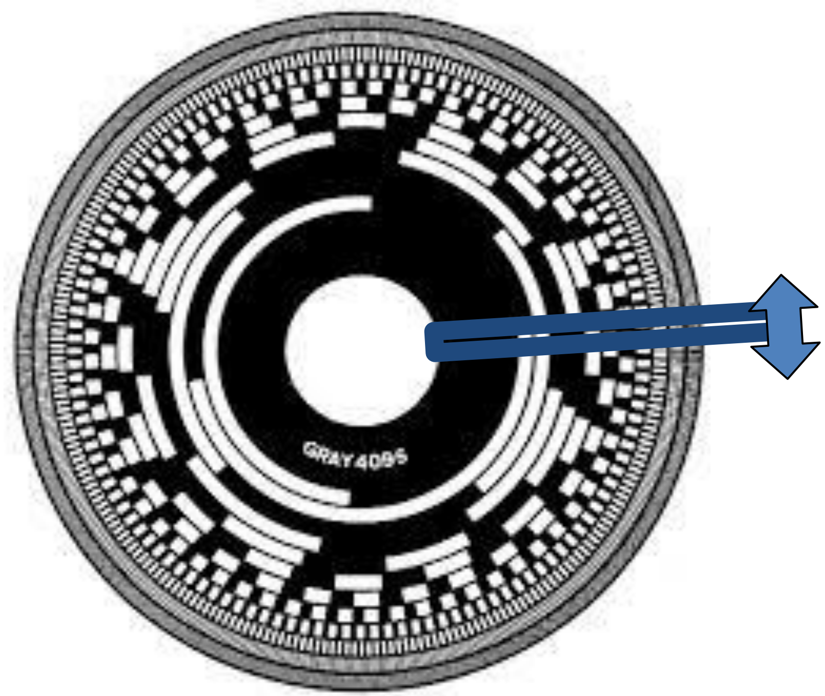
- the lower part of the chart shows a binary counting sequence. You can see how a vertical line traveling from left to right will hit multiple transitions at some places.
- The Gray code is above it and you can see the pattern



Example of Gray Code for Angle

- At left is a 3-bit Gray Angle encoder. As the circle rotates, different bit patterns are seen in the window. They vary by 1 bit from their neighbours, so the accuracy is $45^{o} \pm 45^{o}$
- The Gray code is above it and you can see the pattern





Gray Code – Summary

- Some interesting properties of Gray codes
  - The ***Numerical difference*** between adjacent values is 1
  - The ***Hamming distance*** (see later) between adjacent values is 1
  - A ***permutation*** of the columns of a Gray code is still a Gray code, where neighbours differ by 1 bit :

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

    swap columns 1,2

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

  - You could use Gray codes for bit-error tolerant encryption