Special-purpose codes

There are many different ways (codes) to represent numbers, instructions and characters. Different codes are best suited to particular tasks for which they are designed. We will discuss a few of these codes.

BCD codes

The binary-coded-decimal provides us with an alternative to the natural binary representation. We start with the following chart used to code individual digits:

Binary code	Digit
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	Forbidden
1 0 1 1	Forbidden
1 1 0 0	Forbidden
1 1 0 1	Forbidden
1 1 1 0	Forbidden
1 1 1 1	Forbidden

Each digit is coded as a 4 bit binary sequence.

1523 would be coded to

0001 0101 0010 0011

Addition is done the same way as it is done in base 2

```
  Decimal        BCD
  1523           0001 0101 0010 0011
 +3691           0011 0110 1001 0001
 ------------------------------------
  5214           0101 0010 0001 0100
```

Notice when we added 2+9 , `0010 +1001` we did not get `1011`
We carried 1 and left 1 behind. (1011= 1010 + 0001 , and 1010 represents the 10 or carry one portion)

Advantage:

Very easy to convert decimal to binary form.

Disadvantages:

Arithmetic becomes more complex:

If the sum of the two digits is 9 or less then addition is the same as ordinary 4 digit binary numbers. When the sum is greater than 9 (1001_2) then we carry 1 to the next digit and add 6 (0110_2) to the sum of the two digits.
Consider again the sum of 1523 and 3691 and look at the second digit addition:

Decimal	BCD	
2	0010	
9	1001	
11	1011	Note neither represents a digit

For the BCD we carry one and add 6 (0110_2)

```
  1011
 +0110
  0001
```

Adding digits for 9 and 9 or 9 and 8 , 8 and 8 we have both the ordinary carry we would have for binary addition and the addition of 0110_2.

Storage is inefficient compared to natural binary:

Consider again the number 5214_{10}

$$5214_{10} = 0101001000010100_{BCD}$$

And

$$5214_{10} = 1010001011110_2$$

Often used for applications that require little storage such as calculators and digital watches. Often microprocessors will special instructions to help carry out BCD operations.

Unweighted Codes

Our previous representation of binary numbers was pure binary or natural binary (sometimes called 8421 weighted binary) . The position of each digit was a direct indication of its weight.
Some binary codes do not use the position of a bit to represent weight. Such codes are called unweighted codes.

The **unit distance** codes are examples of unweighted codes. Distance can be measured in a variety of ways. We are intereseted in a particular distance called the **Hamming distance**.

The Hamming distance between two words is the number of bits in which they are different. Thus $d_h(10110, 11010) = 2$

A unit distance code is a code in which any two consecutive words have hamming distance = 1.

Note: 8421 weighted binary is note a unit distance code:

1111= 15 and 0111 = 7 15 and 7 are not consecutive yet $d_h(1111,0111)=1$

The most popular of the unit distance codes are the Gray codes (named after their inventor Frank Gray).

How can we generate an N-bit gray code?

To generate a Gray Code of N bits you can do the following:

1. Write down the Gray code for N-1 bits
2. Copy that same list in reverse order below the original one.
3. Add a 0 bit in front of the codings in the original half of the list and a 1 bit in front of the reversed copy.

1- bit Gray Code

0
1

2 - bit Gray Code

$$\left.\begin{array}{l}0\ 0\\0\ 1\end{array}\right\}\quad \text{Original order 1-bit}$$

$$\left.\begin{array}{l}1\ 1\\1\ 0\end{array}\right\}\quad \text{Reversed order 1-bit}$$

3 -bit Gray Code

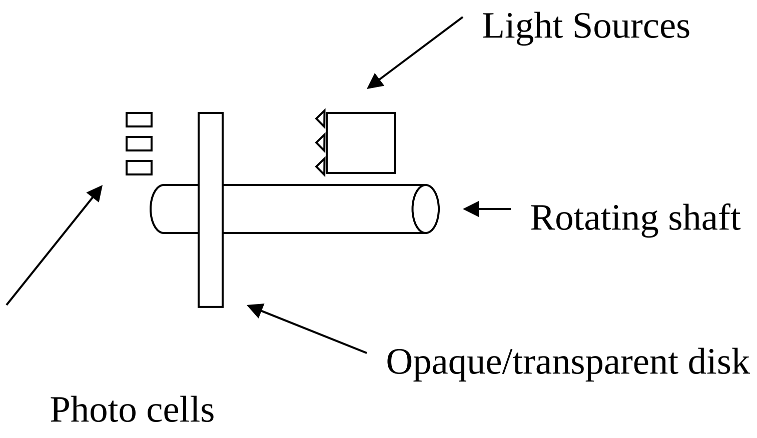
$$\left.\begin{array}{l}0\ 0\ 0\\0\ 0\ 1\\0\ 1\ 1\\0\ 1\ 0\end{array}\right\}\quad\text{Original Order 2-bit}$$

$$\left.\begin{array}{l}1\ 1\ 0\\1\ 1\ 1\\1\ 0\ 1\\1\ 0\ 0\end{array}\right\}\quad\text{Reversed Order 2-bit}$$

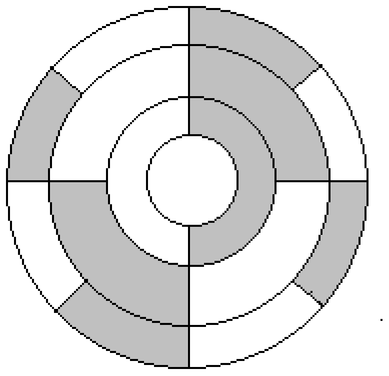
4-bit gray code:

Number$_{10}$	Natural Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

A typical application of a Gray code is that of the optical encoder.
We can visualize this device as a glass or plastic disc with certain regions opaque and certain regions transparent. The disk is attached to a rotating shaft. A tight source is above the disk and photo cells sit below the disk. If light passes through a region of the disk it activates the sensor below.

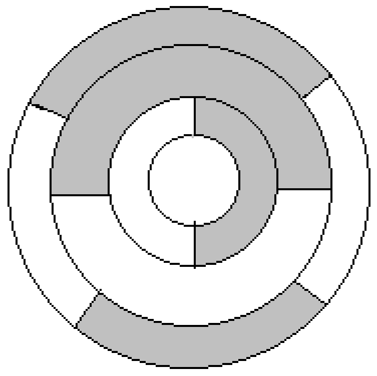


Natural Binary encoding



Notice as we rotate through consecutive 45 degree angles the code changes. But, it does not change by 1-bit per 45 degrees.

Gray encoding



Notice as we rotate through consecutive 45 degree angles the code changes. This time it does change by 1 –bit per 45 degrees.

So what is the problem with the first disk?

Light sources are not perfectly aligned and edges of the sectors are not perfectly straight. As the disk rotates from one sector to the next and two bits change with one changing slightly faster than the other we might encounter the following input from the photo cells : 011 *111* 100 where 111 is the reading caused by the slight misalignment.

Thus the reading went from 3 to 7 to 4

Note the Gray coded disk always changes in only one location and what happened above can't happen.

Once we have the Gray code we can convert it to a coding more natural for arithmetic calculations.

How could we design a circuit to convert from Natural Binary to Gray code?

Consider each bit of the Gray code as a Boolean function. Lets look at the right most bit of the Gray code:

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

If we label the inputs ABCD for the natural binary and the Gray code outputs as EFGH, we can read from the table:

H= $\overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}C\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + A\overline{B}\,\overline{C}D + A\overline{B}C\overline{D} + AB\overline{C}D + ABC\overline{D}$

And the K-map

CD \ AB	00	01	11	10
00	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$	$\overline{A}B\overline{C}\,\overline{D}$	$AB\overline{C}\,\overline{D}$	$A\overline{B}\,\overline{C}\,\overline{D}$
01	$\overline{A}\,\overline{B}\,\overline{C}D$	$\overline{A}B\overline{C}D$	$AB\overline{C}D$	$A\overline{B}\,\overline{C}D$
11	$\overline{A}\,\overline{B}CD$	$\overline{A}BCD$	$ABCD$	$A\overline{B}CD$
10	$\overline{A}\,\overline{B}C\overline{D}$	$\overline{A}BC\overline{D}$	$ABC\overline{D}$	$A\overline{B}C\overline{D}$

And we see H reduces to H = $\overline{C}D + C\overline{D} = C \oplus D$
The exclusive or !

In the same fashion one can show:
G= $B\overline{C} + \overline{B}C = B \oplus C$
F= $A\overline{B} + \overline{A}B = A \oplus B$
E=A

We can thus go to Digital Works and build the following converter:

Natural to binary to gray code converter: