Journal of Universal Computer Science, vol. 13, no. 11 (2007), 1573-1597
submitted: 9/10/07, accepted: 19/11/07, appeared: 28/11/07 © J.UCS

The Gray Code[1]

Robert W. Doran
(Department of Computer Science, The University of Auckland, New Zealand
Email: bob@cs.auckland.ac.nz)

Abstract: Here we summarise the properties and algorithms of the Gray code. Descriptions are given of the Gray code definition, algorithms and circuits for generating the code and for conversion between binary and Gray code, for incrementing, counting, and adding Gray code words. Some interesting applications of the code are also treated. Java implementations of the algorithms in this paper are available at: `http://www.jucs.org/jucs_13_11/the_gray_code/data/DoranGrayPrograms.zip`.
Key Words: Gray code, single-distance code
Category: E.4

1 Introduction

What we now call "Gray code" was invented by Frank Gray. It was described in a patent that was awarded in 1953; however, the work was performed much earlier, the patent being applied for in 1947. Gray was a researcher at Bell Telephone Laboratories; during the 1930s and 1940s he was awarded numerous patents for work related to television.[2] According to Heath [Hea72] the code was first, in fact, used by Baudot for telegraphy in the 1870s, though it is only since the advent of computers that the code has become widely known.

The term "Gray code" is sometimes used to refer to any single-distance code, that is, one in which adjacent code words (perhaps representing integers differing by 1) differ by 1 in one digit position only. Gray introduced what we would now call the canonical binary single-distance code, though he mentioned that other binary single-distance codes could be obtained by permuting the columns and rotating the rows of the code table. The codes of Gray, and natural extensions to bases other than binary, are only a very small subset of all single-distance codes. Here we will use the term "the Gray code" to refer to the code of Gray and "single-distance" to refer to the more general case; we will be concerned mainly with properties of the Gray code.

Much has been discovered and written about the Gray code in the past; it is associated with many elegant algorithms and circuits. However, this wealth of technical material had never been gathered together and treated in a consistent

---

[1] C. S. Calude, G. Stefanescu, and M. Zimand (eds.). *Combinatorics and Related Areas. A Collection of Papers in Honour of the 65th Birthday of Ioan Tomescu.*

[2] Gardener [Gar72] states that Gray "died in 1969. His contributions to modern communications technology were immense. The method now in use for compatible color television broadcasting was developed by Gray in the 1930's".

form, hence, this self-contained survey of the code's properties, algorithms and circuits.

2 Definition of the Gray Code

2.1 Origin of the code

The Gray code arises naturally in many situations. Gray's interest in the code was related to what we would now call analog to digital conversion. The goal was to convert an integer value, represented as a voltage, into a series of pulses representing the same number in digital form. The technique, as described in Gray's patent, was to use the voltage being converted to displace vertically an electron beam that is being swept horizontally across the screen of a cathode ray tube. The screen has a mask etched on it that only allows the passage of the beam in certain places; a current is generated only when the beam passes through the mask. The passage of the beam will then give rise to a series of on/off conditions corresponding to the pattern of mask holes that it passes.

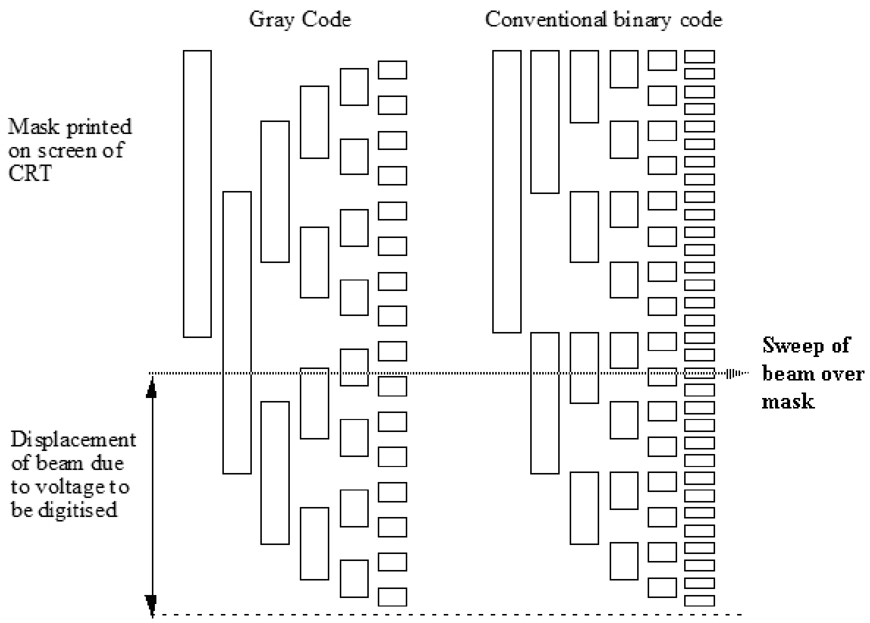

Figure 1: Gray's analogue to digital device

The original scheme was to use a mask representing a standard binary encoding. However, this has the problem that, if the beam is close to the boundary between two values, a slight distortion in the beam can give an output that is neither of the two adjacent values but a combination of each (in the example below, in the transition from 011011 (27) to 011100 (28), the device could produce these two values but also 011111 (31) or 011000 (24) and others. To deal with this problem Gray proposed using a mask corresponding to a code in which adjacent code words differed in one bit position only. In that case, a slight beam displacement would give only a small change to the encoding. Figure 1 is an adaptation of the figure in the patent.

2.2 Gray's definition of the code

Figure 2 is a word-for-word reproduction of the definition given by Gray in the patent [Gra53] - it has never been explained better.

Gray's definition is a procedure for generating, what we now call, the Gray code of width n. As well as discussing the process, he has shown, by construction that:

Property P1: Adjacent words in the Gray code sequence differ in one bit position only.

2.3 Direct application of the code

Because, apart from the leading bit, the second half of the code is the same as the first, but reversed, it follows that the first and last words of the code sequence differ in only the leading bit. In other words:

Property P2: The Gray code is cyclic.

These first two properties underlie the most common practical use found for the code which was for locating the rotational position of a shaft (see, for example, [Fos54])[3]. A pattern representing the Gray code was printed on a shaft, or on a disk, and the pattern sensed by an optical or electrical detector (see Figure 3). Note that the least significant end of the code has fewer transitions than does normal binary so the Gray code has another apparent advantage that the pattern may be printed to another bit of precision with the same printing resolution [Wal70]. Note that the read-out of the shaft's rotational position is a completely parallel operation.[4]

[3] This was Baudot's application as well [Hea72].

[4] Another application of Gray code, where its cyclic and adjacency properties are made use of, is in the labelling of Karnaugh maps for simplifying logic functions.

The manner in which the primary reflected binary number system is built up will now be explained.

First: write down the first two numbers in the 1-digit orthodox number system, thus:

Zero	0
One	1

Note that the symbols differ in only one digit.

Second: below this array write its "reflection" in a transverse axis:

Zero	0
One	1

	1
	0

The symbols still differ in not more than one digit. However, the first is identical with the fourth and the second with the third.

Third: to remove this ambiguity, add a second digit to the left of each symbol, 0 for the first two symbols and 1 for the last two, thus:

Zero	00
One	01
Two	11
Three	10

and identify the last two symbols with the numbers "two" and "three." Each symbol is now unique and differs from those above and below in not more than one digit. The array is a representation of the first four numbers in the primary 2-digit reflected binary number system.

The process is next repeated giving -

First:

Zero	00
One	01
Two	11
Three	10

Second:

Zero	00
One	01
Two	11
Three	10

	10
	11
	01
	00

Third:

Zero	000
One	001
Two	011
Three	010

Four	110
Five	111
Six	101
Seven	100

Figure 2: Gray's definition of his reflected binary code

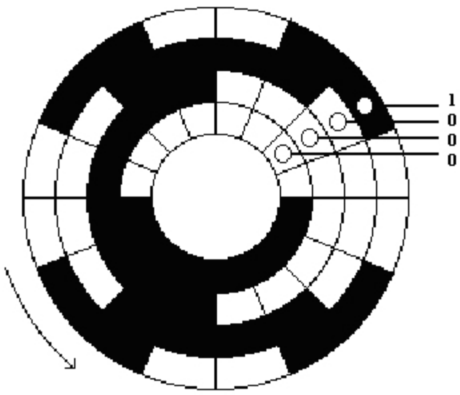

Figure 3: Gray code as used on a shaft encoder for determining angle of rotation

2.4 Generation of the code sequence by means related to its definition

Let us say that going through the Gray code sequence normally, is going *up*, or *ascending* and the opposite direction *down*, or *descending*. Generating a sequence going down is the same as reflecting it, in Gray's sense. The sequence of width n comprises, by definition:

0 preceding each member of the width $n-1$ sequence
1 preceding each member of the width $n-1$ sequence reflected.

To generate going down, this is reflected to give:

1 preceding each member of the width $n-1$ sequence reflected reflected
0 preceding each member of the width $n-1$ sequence reflected.

But reflecting a sequence twice gives back the original sequence, so the width n sequence reflected is:

1 preceding each member of the width $n-1$ sequence
0 preceding each member of the width $n-1$sequence reflected.

This gives us the property:

Property P3: A descending Gray code sequence of width n is the same as an ascending sequence except that the leading bit is inverted.

For example, the width 3 sequence appears in Table 1.

up	down
000	100
001	101
011	111
010	110
110	010
111	011
101	001
100	000

Table 1: Width 3 sequence

Let's use the following notation. The Gray code word $\mathbf{G}$, of width n, is a vector of n bits, $(\mathbf{G}_{n-1}, \mathbf{G}_{n-2}, \ldots, \mathbf{G}_0)$ and represents a number G. Likewise, a number B has the standard representation $(\mathbf{B}_{n-1}, \mathbf{B}_{n-2}, \ldots, \mathbf{B}_0)$. We will most usually be interested in the situation when $B = G$ and there are two different representations $\mathbf{G}$ and $\mathbf{B}$ of the same value.

(In expressing algorithms we will use Java but modified, to make the algorithms clearer, so that we have "classes" Bit and Word, and use mathematical expressions. To test the algorithms, convert to Java by replacing Bit with boolean and Word with boolean [], and constants 0 by false and 1 with true. The "global" variables have to be represented as class variables in a testing class. Note use of Java ! for "not". The algorithms are available converted to Java on request.)

Gray's definition of his code sequence of width n is captured by the following algorithm:

```
Word G;

public void generate (int n, Bit d){ // d: up 0, down 1
    G = new Word(n);
    generate1(n,d);
    }
```

```
public void generate1 (int m, Bit d){
    if (d==0) { // up
        G[m-1] = 0;   generate1(m-1,0);  // up
        G[m-1] = 1;   generate1(m-1,1);  // down
        }
    else { // down
        G[m-1] = 1;   generate1(m-1,0);  // up
        G[m-1] = 0;   generate1(m-1,1);  // down
    }
```

Dealing with the termination of recursion, and simplifying, we end up with the simple algorithm:[5]

```
/* ALGORITHM A1: GENERATE WIDTH N GRAY CODE SEQUENCE */

Word G;

public void generate (int n, Bit d){ // d: up 0, down 1
    G = new Word(n);
    generate1(n,d);
    }

public void generate1 (int m, Bit d){
    if (m == 0) display(G);
    else {
        G[m-1] = d;    generate1(m-1,0);  // up
        G[m-1] = !d;   generate1(m-1,1);  // down
        }
    }
```

3 Relationship between binary code and Gray code

3.1 Generating the Gray code from binary

The above algorithm, with two calls per recursion, has a binary tree of possible method calls. We can label the nodes of the tree, and thus give each Gray code word a binary equivalent, by setting a bit prior to each recursive call:

```
public void generate (int n, Bit d){ // d: up 0, down 1
    B = new Word(n);
    G = new Word(n);
    generate1(n,d);
    }
```

[5] This algorithm is given in [Er84] for a generalised code in a slightly more complex form.

```
public void generate1 (int m, Bit d){
    if (m == 0) display(B, G);
    else {
        G[m-1] = d;   B[m-1] = 0; generate1(m-1,0);
        G[m-1] = !d;  B[m-1] = 1; generate1(m-1,1);
        }
    }
```

The algorithm will now generate the binary integers B along with the associated Gray codes. The inner group of statements is equivalent to:

```
G[m-1] = d;  B[m-1] = 0; generate1(m-1,B[m-1]);
G[m-1] = !d; B[m-1] = 1; generate1(m-1,B[m-1]);
```

i.e. (if we initialize `B[n]` to 0, and reduce generate1 to one parameter):

```
G[m-1] = B[m];   B[m-1] = 0; generate1(m-1);
G[m-1] = !B[m];  B[m-1] = 1; generate1(m-1);
```

i.e., using an exclusive-or operator (^ in Java):

```
B[m-1] = 0;G[m-1] = B[m]^B[m-1]; generate1(m-1);
B[m-1] = 1;G[m-1] = B[m]^B[m-1]; generate1(m-1);
```

Now, as G is not used except in "display", the generation of its elements may be done in any order following the generation of the necessary bits of B—it can thus be generated at "display time". Giving:

```
/* ALGORITHM A2: GENERATE WIDTH N BINARY AND GRAY
CODE */

Word G, B;

public void generate (int n){ // ascending sequence
    B = new Word(n+1); B[n] = 0;
    G = new Word(n);
    generate1(n);
    }

public void generate1 (int m){
    if (m > 0) { // generate binary code
        B[m-1] = 0; generate1(m-1);
        B[m-1] = 1; generate1(m-1);
        }
    else { // generate Gray code and display
        for (int i = G.length-1; i>=0; i--) G[i] = B[i+1]⊕B[i];
        display(B, G);
        }
    }
```

3.2 Conversion from binary to Gray

The above generation algorithm gives us immediately the property (specified by Gray):

Property P4: $(\mathbf{G}_i = \mathbf{B}_{i+1} \oplus \mathbf{B}_i), i = n-1, \ldots, 0$, where $\mathbf{B}_n$ is taken as 0.

This gives a parallel algorithm or circuit for generating **G** from **B**, because the expressions are independent. Alternatively, if a computer has a bitwise exclusive-or between words then we can calculate **G** using a right shift:

$\mathbf{G} = \mathbf{B} \oplus (\mathbf{B}/2)$.

Exclusive-or is the opposite of "equal", so another way of thinking of this is:

Property P5: $\mathbf{G}_i = (\mathbf{B}_{i+1} \neq \mathbf{B}_i), i = n-1, \ldots, 0$ (where $\mathbf{B}_n$ is taken as 0).

The Gray code word is a record of the *transitions* within the corresponding binary word.

Here is an example:

binary word	0011110011001110100110111101101
Gray code word	0010001010101001110101100011011

3.3 Conversion of Gray to binary

Conversion of Gray to binary is not as simple as the other direction. We have from property **P4**:

$\forall i(\mathbf{B}_{i+1} \oplus \mathbf{G}_i = \mathbf{B}_{i+1} \oplus \mathbf{B}_{i+1} \oplus \mathbf{B}_i)$, where $\mathbf{B}_n$ is taken as 0. So, we have:

Property P6: $\mathbf{B}_i = \mathbf{B}_{i+1} \oplus \mathbf{G}_i, i = n-1, \ldots, 0$, where $\mathbf{B}_n$ is taken as 0.

Unfortunately these are not independent and individual equations. They do give rise naturally to a nice sequential algorithm but the parallel version involves a prefix accumulation of exclusive-or:

Property P6': $\mathbf{B}_i = \mathbf{G}_{n-1} \oplus \mathbf{G}_{n-2} \ldots \mathbf{G}_i$.

This can be generated by a parallel prefix circuit as in Figure 4.

Alternatively [Wan66], if a computer has a bitwise xor between words and fast parallel shifts then the binary code may be generated by a succession of xors and shifts that implement the work of figure 4, level by level:

$$\mathbf{B} = \mathbf{G} \oplus (\mathbf{G}/2); \mathbf{B} = \mathbf{B} \oplus (\mathbf{B}/4); \mathbf{B} = \mathbf{B} \oplus (\mathbf{B}/16) \ldots$$

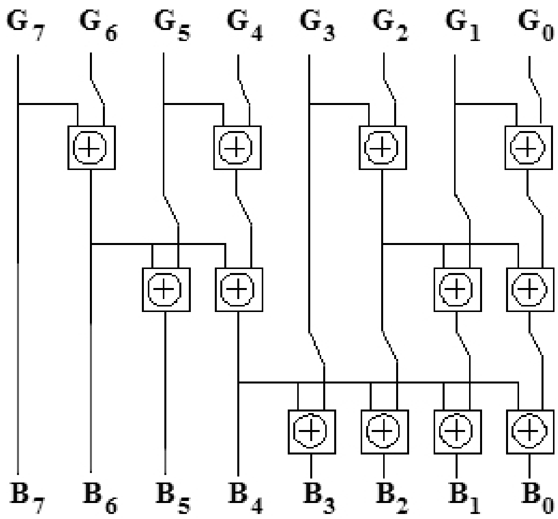

Figure 4: Parallel Gray to Binary Conversion Circuit

However, there are conversion techniques that are more suited to software. Let's concentrate on the bits of the Gray code word that are 1. Define for each $\mathbf{G}$ a vector of integers $\mathbf{I}$ of length z, $\mathbf{I} = (\mathbf{I}_{z-1}, \mathbf{I}_{z-2}, \ldots, \mathbf{I}_0)$ which is the set of subscripts for which the Gray code is not zero. Recalling property P5, that the Gray code word is a record of the transitions within the corresponding binary word, $\mathbf{I}_{z-1}$ is the position of the first 1 in the binary code and $\mathbf{I}_{z-2}$ is the next 0, etc. Now, we have:

$$B = \mathbf{B}_{n-1}2^{n-1} + \mathbf{B}_{n-2}2^{n-2} + \ldots + \mathbf{B}_0 2^0.$$

Listing only the bits of $\mathbf{B}$ that are non-zero:

$$B = \left[2^{\mathbf{I}_{z-1}} + \ldots 2^{\mathbf{I}_{z-2}+1}\right] + \left[2^{\mathbf{I}_{z-3}} + \ldots 2^{\mathbf{I}_{z-4}+1}\right] + \ldots$$

Applying a Booth recoding:

$$B = \left[2^{\mathbf{I}_{z-1}+1} - 2^{\mathbf{I}_{z-2}+1}\right] + \left[2^{\mathbf{I}_{z-3}+1} - 2^{\mathbf{I}_{z-4}+1}\right] + \ldots$$

(-1 if the number of 1 bits in $\mathbf{B}$ is odd)

Let's write $P(\mathbf{X}, a, b)$ for the parity of $(\mathbf{X}_a, \ldots \mathbf{X}_b)$, which can be defined as $\mathbf{X}_a \oplus \cdots \oplus \mathbf{X}_b$, or $(\mathbf{X}_a + \ldots + \mathbf{X}_b) \mod 2$. Also write $P(\mathbf{X}, i)$ for $P(\mathbf{X}, n-1, i)$ and $P(\mathbf{X})$ for $P(\mathbf{X}, n-1, 0)$.

We may write the above:

Property P7:
$B = (-1)^{P(\mathbf{G},\mathbf{I}_{z-1}+1)}.[2^{\mathbf{I}_{z-1}+1}] + \ldots + (-1)^{P(\mathbf{G},\mathbf{I}_0+1)}.[2^{\mathbf{I}_0+1} - 1] - P(\mathbf{G})$.

This property may be used to convert from Gray to binary by adding the shifted bits of the Gray code with appropriate sign.

Here is an example:

binary word	0011100111
Gray code word	0010010100

$$B = 0100000000 - 0000100000 + 0000001000 - 0000000001$$

This property also explains the origin of difficulty with doing arithmetic on Gray code words. In a conventional binary word, if bit i is one it means 2^i, but bit i in a Gray code word could represent $+2^{i+1}$ or -2^{i+1} - the sense can only be resolved if the parity of the leading part of the word up to the bit is determined. In a sense, Gray code is a signed-bit ternary representation [Wal70], where each bit can be 1, 0, or -1 (but with the restriction that non-zero bits must alternate in sign).

Although the property P7 could be used to convert from Gray to Binary, it is not a good approach, because the subtractions involve propagation of carry. A better approach, ([Irs87], also noted by Gray himself), is to replace each power 2^i in the above by $(2^i$-1)+1. We get:

$$\begin{aligned} B =&(-1)^{P(\mathbf{G},\mathbf{I}_{z-1}+1)}.[2^{(\mathbf{I}_{z-1}+1)} - 1] + \ldots + (-1)^{P(\mathbf{G},\mathbf{I}_0+1)}.[2^{\mathbf{I}_0+1} - 1] \\ &+ (-1)^{P(\mathbf{G},\mathbf{I}_{z-1}+1)} + (-1)^{P(\mathbf{G},\mathbf{I}_{z-2}+1)} + \ldots + (-1)^{P(\mathbf{G},\mathbf{I}_0+1)} - P(\mathbf{G}). \end{aligned}$$

The second line is zero. So we have:

Property P8:
$B = (-1)^{P(\mathbf{G},\mathbf{I}_{z-1}+1)}.[2^{(\mathbf{I}_{z-1}+1)} - 1] + \ldots + (-1)^{P(\mathbf{G},\mathbf{I}_0+1)}.[2^{\mathbf{I}_0+1} - 1]$.

The reason that this is useful is that the successive additions and subtractions can now be performed to construct the binary equivalent with no carry being required at any stage (in fact, addition and subtraction may be replaced by exclusive-or).

Here is an example:

binary word	0011100110
Gray code word	0010010101

$$
\begin{aligned}
\mathrm{B} &= 0011111111 - 0000011111 + 0000000111 - 0000000001 \\
&= 0011100000 + 0000000110 \\
&= 0011100110
\end{aligned}
$$

3.4 Parity of Gray code word

Property P8 shows that knowledge of the parity of a Gray code word can useful. We will see other examples of its use later.

Recall that in going up from B to B+1 exactly one bit of **G** changes. It follows that exactly two bits change in going from B to B+2. Thus the number of bits that are 1 remains the same or changes by 2, i.e. the parity remains the same. This gives us:

Property P9: The parity of a Gray code word is 0 if and only if it represents an even number, i.e. $P(\mathbf{G}, n-1, 0) = \mathbf{B}_0$.

One of the drawbacks of the conventional binary representation is that the parity of the result of an arithmetic operation is not easy to predict from the parities of its operands. However, the sum or difference of two numbers is even if, and only if, the inputs are both even or both odd, and the product is even if either operand is even. This allows the parity of Gray-code results to be predicted:

Property P10: If the parities of two Gray code operands are P_a and P_b, then the parity of the Gray code result is:

$$
\begin{array}{ll}
+ & P_a \oplus P_b \\
* & P_a \& P_b.
\end{array}
$$

3.5 Gray codes arising in binary counters

In [Bur70] it was noted that Gray codes arise naturally if one constructs a binary counter from master-slave (i.e. race-free or edge-triggered) toggle flip-flops. In a master-slave flip-flop the "second" flip-flops, represent the value. However, if we concentrate on the "first" flip-flops they are seen to be following a different pattern.

So, as the input and second flip-flops run through the ordinary binary integers, the first flip-flops run through the Gray code. The values of F_i and S_{i+1} are entirely governed by the changes that occur in S_i. Assuming that the counter is

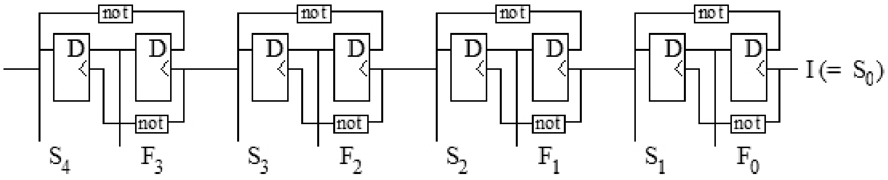

Figure 5: Binary counter with master/slave flip/flops

S_4	F_3	S_3	F_2	S_2	F_1	S_1	F_0	S_0	S	F
0	0	0	0	0	0	0	0	0	00000	0000
0	0	0	0	0	0	0	1	1	00001	0001
0	0	0	0	0	1	1	1	0	00010	0011
0	0	0	0	0	1	1	0	1	00011	0010
0	0	0	1	1	1	0	0	0	00100	0110
0	0	0	1	1	1	0	1	1	00101	0111
0	0	0	1	1	0	1	1	0	00110	0101
0	0	0	1	1	0	1	0	1	00111	0100
0	1	1	1	0	0	0	0	0	01000	1100
0	1	1	1	0	0	0	1	1	01001	1101
0	1	1	1	0	1	1	1	0	01010	1111
0	1	1	1	0	1	1	0	1	01011	1110
0	1	1	0	1	1	0	0	0	01100	1010

Table 2: Binary counter with master/slave flip/flops: and example

initially cleared, the sequence of events detalied in table 2 will be repeated. It can be seen that at all times $F_i = S_{i+1} \oplus S_i$, so that F indeed is the Gray code. Because S_{i+1} is always set to F_i , but delayed, we see another interesting fact:

Property P11: Column i of a listing of the Gray code is the same as column $i+1$ of binary, rotated up by 2^i.

4 Properties related to the transition bit index

4.1 Generation by minimal change

The Algorithms A1 and A2 generate a full Gray Code word at each step. However, because only one bit changes it is possible to generate each word from the previous by changing just that bit. But which bit?

S_{i+1}	F_i	S_i
0	0	0
0	1	1
1	1	0
1	0	1
0	0	0

Follow the execution of a certain level of recursion i in Algorithm A1- it is called from level $i + 1$ and passes control to level $i - 1$. Successive calls to level i will be with direction:

up (d=0); down (d=1); up (d=0); down (d=1); etc.

The action of level i is then:

G[i-1] := 0, call level i-1, G[i-1] := 1, call level i-1; return;

G[i-1] := 1, call level i-1, G[i-1] := 0, call level i-1; return; etc.

Assuming that the Gray code word is initialized to 0, it can be seen that the above sequence is equivalent to:

call level i-1, G[i-1] := 1, call level i-1;

call level i-1, G[i-1] := 0, call level i-1; etc.

That is, level i *switches* bit $i - 1$ between successive calls to level $i - 1$. So we get [Er85]:

```
/* ALGORITHM A3.1: GENERATE WIDTH N GRAY CODE SEQUENCE,
BY SWITCHING */

Word G;

public void generate (int n){ // ascending sequence
    G = new Word(n); // initialized to all 0
    generate1(n);
    }

public void generate1 (int m){
    if (m == 0) display(G);
    else {
        generate1(m-1);
        G[m-1] = !G[m-1];
```

```
        generate1(m-1);
        }
    }
```

4.2 The sequence of transition indices

The algorithm A3.1 identifies the location of each element as it is switched. It is straightforward to modify the algorithm so that it produces the sequence in which indices change[6]

```
/* ALGORITHM A3.2: GENERATE SEQUENCE OF GRAY CODE
TRANSITIONS */

public void generate (int n){
    if (n > 0) {
        generate(n-1);
        display(n-1);
        generate(n-1);
        }
    }
```

We see that the sequence of transitions is an inorder traversal of a binary tree. Its definition is even simpler than the Gray code itself [BER76]:

> sequence for width-n = sequence for width-$(n-1), n-1$, sequence for width-$(n-1)$.

5 Gray Code Incrementers

The task of an incrementer is, given a Gray code word, find the next in ascending order (likewise decrementers and descending). Incrementers are related to counters and to generating algorithms based on incrementing.

There are many papers, disclosures, and patents on this topic [Fis57, Maj71, CoSh69]. They all seem to have as a common concept the condition that is satisfied for a count-up to occur. Consider algorithm A3.1. When the algorithm switches $G[m-1]$ at level $m > 1$, level $m-1$ has been entered an odd number of times and level $m-2$, and below, an even number of times. Thus, when $G[m-1]$ is switched, $G[m-2] = 1$ and $G[m-3]$ and below are all zero. Conversely, when this condition occurs, $G[m-1]$ must be the next to be switched[7]

[6] Gardener [Gar72] shows that the solution to the "Towers of Hanoi" problem with n disks is given by the transition sequence for a width-n Gray code. If the disks are numbered from the top, the transition sequence specifies the disk to be moved next. The algorithm for solving the "Towers" problem is identical in form to algorithm A3.2.

[7] Gardener [Gar72] describes how the "Chinese rings" puzzle has analogous properties so that the Gray code gives the solution to the puzzle.

If $m = 1$ then level 0 does not exist so we need another condition to look at. From the construction of the code we see that every second switch is of $G[0]$. Every switch changes the parity, thus, when counting up, $G[0]$ will be switched next if $P(G)$ is 0. When counting down, $G[0]$ will be switched next if $P(G)$ is 1.

Property P12: When counting an n-bit Gray Code in direction d (=0 for up, =1 for down), the next bit s to be switched is given by:

$P(\mathbf{G}) = d \quad : \quad s = 0$
$P(\mathbf{G}) = 1 - d : \quad s$ is such that $\mathbf{G}_{s-1} = 1$ and $\mathbf{G}_i = 0, i < s - 1$.

This converts readily into a circuit if P($\mathbf{G}$) is known. In making a free-running counter the approach taken seems to be to provide an extra flip/flop that is by driven the clock and is used to select between the two alternatives. So, if flip/flop P is the parity flip-flop then the signals to toggle or switch the counter flip/flops are as in the example in Figure 6.

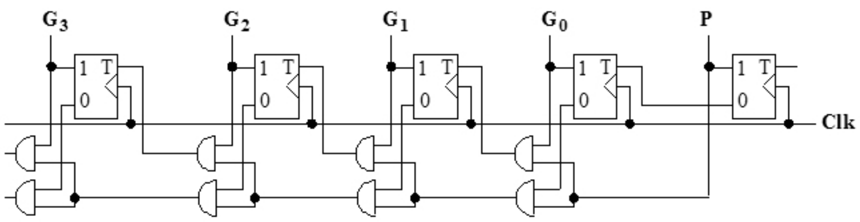

Figure 6: Gray code up counter

In terms of an algorithm for generating the code, Boothroyd [Boo64] calculates the parity and finds the last set bit by a scan from left to right.

```
/* ALGORITHM A4: INCREMENT/DECREMENT A GRAY CODE WORD
G OF WIDTH N */

public void increment (Word G, Bit d){ // d 0 up, 1 down

    Bit p = 0; // parity of word G
    int last1 = G.length; // lowest i such that G[i] == 1
    // find parity and last 1
       for (int i = G.length-1; i>=0; i--)
            if (G[i]==1) {
                p = ! p;
```

```
            last1 = i;
            }

    int switchPos; // index of bit to switch
    // find switchPos
        if (p  ^ !d)             switchPos = 0;
        else
        if (last1 < G.length-1)  switchPos = last1+1;
        else                     switchPos = G.length-1;

    G[switchPos] = ! G[switchPos]; // switch the bit
    }
```

Misra [Mis75] gives a generation algorithm based on the concept of incrementing. However, he keeps track of the parity separately and maintains a stack of indices of bits that are 1, which gives an algorithm that is very fast. [Er85] gives a coding of Misra's algorithm and incorporates some improvements.

6 Serial Addition

We have seen that the sign of the weight assigned to a bit in a Gray code word depends on the parity of the word at that bit, starting at the *high-order* end. However, most serial arithmetic operations must commence with the *low-order* end. If we know the entire parity of the word then it possible to commence serial operation from the low-order bits, because we may compute parities using:

$$P(\mathbf{G}, n-1, k) = P(\mathbf{G}, k-1, 0) \oplus P(\mathbf{G}).$$

We have already seen one example, the Gray code counter, where knowledge of the parity overall is maintained in an auxilliary flip-flop. In [Luc59], Harold Lucal proposed using a modified Gray Code where the parity is maintained as the least significant bit. Lucal showed how serial arithmetic could then be implemented.

It is clear that addition of Gray codes can be performed serially if we commence at the least significant end and know the parity of the two operands. We can work out the high-order parities at each bit as we go using the above formula. From property P10 we can find the parity of the sum and maintain the parity of each bit, and we can propagate a carry. This is straightforward but involves carrying a large amount of information between bits. Lucal, however, showed that addition could be performed by carrying only two bits between each stage as follows:

```
/* ALGORITHM A5: SERIAL ADDITION OF GRAY CODE WORDS */

public void add (int n,       // width of words
             Word A, Word B, // addends, low order bit parity
```

```
            Word S) {       // result, low order bit parity

    Bit E, F;   // running carries
    Bit ET, FT; // temp carries
                // not needed if loop statements parallel
    E = A[0]; F = B[0];
    S[0] = E  ^ F;
    for (int i=1; i<=n; i++) {
        S[i] = (E & F)  ^  A[i]  ^  B[i];
        ET   = (E & (!F))  ^  A[i];
        FT   = ((!E) & F)  ^  B[i];
        E = ET;
        F = FT;
        }
    // at this point, if E and F are not 0, there is an overflow
    // and the value represented by S is the 1s-complement of
    // the sum
    }
```

This surprising algorithm is based on the observation that it is not necessary to know the exact parity of A[i] and B[i] but to know whether they have different parities or the same parity. The interpretation of the bits E and F is:

EF = 00 - parity of A and B the same, no change in binary carry
EF = 01 - parity different and B had the last 1
EF = 10 - parity different and A had the last 1
EF = 11 - parity of A and B the same, change in binary carry.

CE and CF must both be 0 on completion, otherwise there is an overflow. Refer to [Luc59] for details and a proof that this algorithm is correct. Note the expression for the sum bit which represents a change in binary code of the binary sum as occurring when one of the inputs change (indicated by A[i] and by B[i]) or the carry changes (indicated by (E and F)) - this is the same equation as for binary addition.

7 Extensions of Gray codes

7.1 Bases other than binary

The original definition of Gray code applied to binary digits. However, it is very straightforward to extend the concept of a distance-1 transition to numbers of other bases, or even mixed-base numbers. A distance-1 transition is extended to mean a change by 1 in one digit only. The algorithms for generation and conversion are straightforward extensions of those for the binary case.

For example, to generate the code sequence, for each increment at a given digit, the lower order code is generated once, alternately up and down. Suppose, for example, a code is desired for numbers that have a base 5 digit followed by a base 3 digit.

Natural sequence	Gray sequence	Binary code
0,0	0,0	000,00
0,1	0,1	000,01
0,2	0,2	000,11
1,0	1,2	001,11
1,1	1,1	001,01
1,2	1,0	001,00
2,0	2,0	011,00
2,1	2,1	011,01
2,2	2,2	011,11
3,0	3,2	010,11
3,1	3,1	010,01
3,2	3,0	010,00
4,0	4,0	110,00
4,1	4,1	110,01
4,2	4,2	110,11

If the digits are encoded in canonical binary Gray code then the encoding is itself a binary Gray code. Note that the Gray code will not normally be cyclic. The rules for conversion and counting are also natural extensions of the binary case.

There have been many papers exploring the generation of Gray codes in bases other than binary [ER84], [Dar72], [Bar81]. [ThMu93] introduces a parallel algorithm for generation, but using the power of a reconfigurable bus for fast carry propoagation.

7.2 Related concepts

The concept of adjacent symbols differing at one bit position only has been extended in many ways. A shift of concept usually involves refiguring the algorithms that apply to bit strings to apply to the new domain.

Finding the order of selection of + or - in the set of expressions +/- f(+/- f(+/-. . .)) where f is monotone, so that the results are in order, is found to be the Gray code sequence itself [Sal72], [Sal73].

Algorithms have been developed for the single change set partition sequences [Kay76], e.g. (1 2 3); (1 2) (3); (1) (2) (3); (1) (2 3); (1 3) (2). Similarly for the partitions of an integer [Sav89]. P(5,3): 1+1+1+1+1 = !+1+1+2 = 1+2+2 = 1+1+3 = 2+3. Also for compositions, split of n into k parts[Kli82] L(6,3): 2+2+2 = 3 + 2 + 1 = 4 + 1 +1. Others analogous transition sequences are covered in [KoRu93] and [Pro85] .

Another path of generalization remains within weighted number systems but seeks variations to its properties. One direction is to look for uniformly weighted codes (those with the same number of 1 bits) [BER76] and another is for distance-1 codes with the "snake in the box" property that words distance k apart in the counting sequence differ by k bits [Kau70]. There are legions of other codes with similar and related properties studied in the literature.

7.3 Paths on the n-cube

In the binary case, code words of length n can be regarded as the vertices of an n-cube and a complete Gray code sequence represents one of the Hamiltonian paths. This can have an application in hypercube computer networks. If the nodes are assigned binary numbers then the Gray code defines a path that allows a message to be sent to all processors, once only.

As mentioned earlier the Gray codes are just a small subset of the distance-1 codes and Hamiltonian paths. The number of such paths as a function of n is not known, however paths that have additional properties have been enumerated [Gil58].

The n-cube can be generalized to a more-complex graph in the case of bases other than binary. Paths of special interest have also been studied in this case [ShRa78], [Coh63].

8 Applications of Gray codes

Gray codes continually turn out to have new applications. Two of the more-interesting applications are considered here.

8.1 Gray codes and Walsh functions

Yuen [Yeu71] has shown that there is a nice relationship between width n Gray code sequence and the set of Walsh functions of length 2^n. A set of Walsh functions of length 2^n is usually defined as a set of discrete-valued functions in an interval with values that are orthogonal [Bea75]. However, they may also be regarded as set of 2^n binary code words of length that are maximally distant, i.e. each word is distance 2^{n-1} from all others. For example, n=3, a set of 8 length-8 Walsh functions, each distance 4 from all others, is presented in Table 3.

	Gray rank	Gray code	Walsh code
0 000	0	000	00000000
1 001	1	001	00001111
2 010	3	011	00110011
3 011	2	010	00111100
4 100	7	111	01010101
5 101	6	110	01011010
6 110	4	100	01100110
7 111	5	101	01101001

Table 3: Gray and Walsh codes

The contrast with Gray code is striking. Walsh are maximally distant, there is no natural sequence to the code words. Gray words are minimally distant with a well-defined sequence. It is surprising that there is a relationship between the two concepts.

The algorithm to generate each member of a set of Walsh functions is also delightfully simple:

```
/* ALGORITHM A6: GENERATE WIDTH 2**n WALSH FUNCTION */
/* CODE WORD i  (0<=i<=2**n-1) */

public void generateW (int n, int i){
    generateW1(n,i,0);
    }

public void generateW1 (int m, int i, Bit d){
    if (m == 0) display(d);
    else {
        generateW1(m-1, i/2, d);
        generateW1(m-1, i/2, d^(i mod 2));
        }
    }
```

This algorithm is quite similar to algorithm A3.1. As pointed out by Yuen, the number of transitions in the Walsh code word is the rank of the binary pattern of i in the Gray code sequence, and there must be one word for each possible number of transitions. Furthermore, the bits of the corresponding Gray code word may be used to specify the transition points in the Walsh word, in the same sequence as the sequence of index changes when generating the Gray sequence. So, if i is represented as Gray code $\mathbf{G}$ with bits $\mathbf{G}_2$,$\mathbf{G}_1$,$\mathbf{G}_0$ then the sequence of changes in Walsh word number i is 0,$\mathbf{G}_2$,$\mathbf{G}_1$,$\mathbf{G}\ _2$,$\mathbf{G}_0$,$\mathbf{G}_2$,$\mathbf{G}_1$,$\mathbf{G}_2$.

Although there is no natural order as such, one which has reason is where the code words are listed in order of number of transitions. This can be generated by replacing $d \oplus (i \mod 2)$: by $d \oplus (i \mod 2) \oplus ((i/2) \mod 2))$ in the above algorithm, effectively converting from binary to Gray *en passent*.

8.2 Analog to digital conversion

The original application of Gray code was in A to D conversion. It is interesting that even with fully electronic A-D it appears to be somewhat faster and simpler to convert an analog signal v to a Gray code than to convert it to binary. The standard approach, if v is in the range 0 to 2^n-1, is to determine the first bit by subtracting 2^{n-1} - if the result is positive then the first bit is 1, if negative, the first bit is 0. The process continues with the reduced signal in the first case but with the original signal in the second case. There is therefore a decision to be made at each stage that slows the process down.

```
v1 = v;
for (int i = n-1; i>=0; i--){
    v2 = v1- Math.pow(2,i-1);
    B[i] = (v2>0);
    if (v2>0) v1 = v2;
    }
```

However, it is possible to produce the Gray code version of the signal without making decisions, though it requires the determination of the absolute value of a voltage (which is realized as a rectifier).

```
/* ALGORITHM A7: CONVERSION OF ANALOG SIGNAL V TO
GRAY CODE */
public void AtoD(double v, Word G){
    int n = G.length;
    double v2 = v  Math.pow(2,n-1);
    G[n-1] = (v2>0);
    double v1 = |v2|;
    for (int i=n-2; i>=0; i--) {
        v2 = (v1- Math.pow(2,i));
        G[i] = (v2<0);
        v1 = |v2|;
        }
    }
```

The fact that the Gray code is produced can be shown by noting that the V1 in the second algorithm is the same as the first where B[i] = 1 but is the 2^{i-1} complement elsewhere. The second algorithm treats the first bit in the opposite fashion to the others. This algorithm has difficulty in dealing with the integer boundary values. This is finessed, as is the issue of rounding, by incrementing v by 0.5 before commencing the algorithm.

The algorithm was expounded by Yuen in the papers [Yue77], [Yue78]. Lippel [Lip78] pointed out that the idea was in [Smi56] and attributed there to a 1949 thesis by R. P. Sallen. The algorithm above is related to non-restoring division. Yuen [Yue88] showed how it could be extended to division and square rooting.

9 Summary

The simple Gray code offers a dense counting sequence that is not very suited to humans but has the potential of being more "natural" for machines. The properties of Gray code were explored in the early days of computing and the code shown to be suitable for simple serial arithmetic. However, when computer arithmetic became parallel, the Gray code turned out to be slower or more-complex than standard binary.

Be that as it may, the algorithms and circuits involving Gray codes deserve to be remembered because of their elegance and simplicity. Gray code continues to turn up in diverse areas – who knows, it may again find more widespread practical uses in the future.

References

[Bar81] Barr, K. G.: "A decimal Gray code"; Wireless World, Vol. 87, No. 1542, pp. 86-87, March 1981.

[Bea75] Beauchamp, K. G.: "Walsh functions and their applications"; Academic Press, 1975.

[BER76] Bitner, J. R., Ehrlich, G., Reingold, E. M.: "Efficient generation of the binary reflected Gray code and its applications"; CACM, Vol. 19, No. 9, pp. 517-521, September. 76.

[Boo64] Boothroyd, J.: "Algorithm 246 Graycode"; CACM, Vol. 7, No. 12, p. 701, December 1964.

[Bur70] Burkhart, William H.: "Comment on "A Gray code counter""; IEEE Transactions on Computers, pp. 653-654, July 1970.

[Cav75] Cavior, S. R.: "Upper bounds associated with errors in Gray code"; IEEE Trans. Inf. Theory, Vol. IT-21, No. 5, p. 596, September 1975.

[CCC92] Chang, C. C., Chen, H. Y., Chen, C. Y.: "Symbolic Gray code as a data allocation scheme for two-disc systems"; Computer Journal, Vol. 35, No. 3, pp. 299-305, June 1992.

[CLD82] Chang, C. C., Lee, R. C. T., Du, M. W.: "Symbolic Gray code as a perfect multiattribute hashing scheme for partial match queries"; IEEE Trans. Softw. Eng., SE-8, No. 3, pp. 235-249, 1982.

[Coh63] Cohn, M.: "Affine m-ary Gray codes"; Inform. Control. 6, pp. 70-78, 1963.

[CoSi69] Cohn, M., Even, S.: "A Gray code counter"; IEEE Transactions on Computers, pp. 662-664, July 1969.

[Dar72] Darwood, N.: "Using the decimal Gray code"; Electronic Engineering (London), Vol. 44, No. 528, pp. 28-29, February 1972.

[Deu73] Deutsch, E. S.: "On the use of binary and Gray code schemes for continuous-tone picture representation"; Pattern Recognition, Vol. 5, No. 2, pp. 121-132, June 1973.

[DuLe80] Du, H. D., Lee, R. C. T.: "Symbolic Gray code as a multikey hashing function"; IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, pp. 83-90, January 1980.

[Er84] Er, M. C.: "On generating the n-ary reflected Gray codes"; IEEE Trans. Comput., Vol. C-33, No. 8, pp. 739-741, August, 1978.

[Er85.1] Er, M. C.: "Remark on algorithm 246 (Gray code)"; ACM Trans Math Software. Vol. 11, No. 4, pp. 441-3, Dec 1985.

[Fos54] Foss, F. A.: "The use of a reflected code in digital control systems"; IRE Trans. Elec. Comps, pp. 1-6, December 1954.

[Fis57] Fischmann, A. F.: "A Gray code counter"; IRE Trans Elec. Comp., Vol. EC-6, p. 120, June 1957.

[FR80] Flajolet, P., Ramshaw, L.: "A note on Gray code and odd-even merge"; SIAM Journal on Computing, Vol. 9, No. 1, pp. 142-58, February 1980.

[Gan80] Ganesan, S.: "Fast 16 bit Gray code to binary code converter"; Electronic Engineering, p. 17, April 1980.

[Gar72] Gardner, M.: "The curious properties of the Gray code and how it can be used to solve puzzles"; Scientific American, 227, 2, pp. 106-109, August 1972.

[Gil58] Gilbert, E. N.: "Gray codes and paths on the n-cube"; Bell Syst Tech J., Vol. 37, pp. 815-826, May 1958.

[Gra53] Gray, F.: "Pulse Code Communication"; US patent #2,632,058. March 17th, 1953.

[Hea72] Heath, F. G.: "Origins of the Binary Code"; Scientific American, 227, 2, pp. 76-83, August 1972.

[Irs87] Irshid, M. I.: "Gray code weighting system"; IEEE Trans. Inform. Theory, Vol. 33, No. 6, pp. 930-931, 1987.

[Kau70] Kautz, W. H.: "A readily implemented single-error-correcting unit-distance counting code"; IEEE Trans. Computers, Vol. C-19, pp. 972-975, 1970.

[Kay76] Kaye, R.: "A gray code for set partitions"; Information Processing Letters, Vol. 5, No. 6, pp. 171-173, 1976.

[Kli82] Klingsberg, P.: "A Gray code for compositions"; Journal of Algorithms, Vol. 3, No. 1, pp. 41-44, 1982.

[KR93] Koda, Y., Ruskey, F.: "A Gray code for the ideals of a forest poset"; Journal of Algorithms, Vol. 15, No. 2, pp. 324-340, 1993.

[LT87] Larcher, G., Tichy, R. F.: "A note on Gray code and odd-even merge"; Discrete Applied Mathematics, Vol. 18, No. 3, pp. 309-313, 1987.

[Lip78] Lippel, B.: "Comments on 'A fast analog to Gray code converter"'; Proceedings of the IEEE, Vol. 66, No. 7, pp. 797-798, July 1978.

[Luc59] Lucal, Harold M.: "Arithmetic operations for digital computers using a modified reflected binary code"; IEEE Transactions on Computers, pp. 449 - 458, December 1959.

[Lud81.1] Ludman, J. E.: "Gray code generation for MPSK signals"; IEEE Transactions on Communications, Vol. COM-29, No. 10, pp.1519-1522, October 1981.

[Maj71] Majithia, J C.: "Simple design for up/down Gray-code counters"; Electronics Letters, 4th November 1971, Vol. 7, No. 22, pp. 658-659.

[MPS75] Mecklenburg,P., Pehlert, W. K. Jr., Sullivan, D. D.: "Correction of errors in multilevel Gray-coded data"; IEEE Transactions on Information Theory, Vol. IT-19. No. 3, pp. 336-340, May 1973.

[Mis75] Misra, J.: "Remark on algorithm 246: Graycode[Z]"; ACM Trans. Math. Software, Vol. 1, No. 3, p. 285, September 1975.

[PrRu85] Proskurowski, A., Rusky, F.: "Binary tree Gray codes"; Journal of Algorithms, Vo.l 6, pp. 225-238, 1985.

[Ric86] Richards, D.: "Data compression and Gray-code sorting"; Information Processing Letters, Vol. 22, No. 4, pp. 201-5, 17 April 1986.

[Sal73] Salzer, H. E.: "Ordering +or-f(+or-f(+or-f(...+or-f(x)...))) when f(x) is positive momotonic"; Communications of the ACM, Vol. 15, No. 1, p. 45, January 1972.

[Sal73] Salzer, H. E.: "Gray code and the +or-sign sequence when +or-f(+or-f(+or-f(...+or-f(x)...))) is ordered"; Communications of the ACM, Vol. 16, No. 3, p. 180, March 1973.

[Sav89] Savage, C. D.: "Gray code sequences of partitions"; Journal of Algorithms, Vol. 10, No. 4, pp. 577-95, December 1989.

[SK78] Sharma, B. D., Khanna, R. K.: "On m-ary Gray codes"; Information Sciences, Vol. 15, pp. 31-43, 1978.

[TM93] Thangavel, P., Muthuswamy, V. P.: "A parallel Algorithm to generate n-ary reflected Gray codes in a linear array with reconfigurable bus system"; Parallel Processing Letters, Vol. 3, No. 2, pp. 157-164, June 1993.

[TO78] Takizawa, K., Okada, M.: "High-speed Gray-binary and binary-Gray code convertors using electro-optic light modulators"; Electronics Letters, Vol. 14, No. 22, pp. 708-710, October 1978.

[Wal70] Walker, M.: "Decipher the Gray code"; Electron. Des., Vol. 18, No. 4 , pp. 70-74, February, 1970.

[Wan66] Wang, M. C.: "An algorithm for Gray to binary conversion"; IEEE Transactions on Electronic Computers, Vol. EC-5, No. 4, pp. 659-660, August 1966.

[WT94] Worley, R. T., & Tischer, P. E.: "An alternative to Gray coding for bit-plane compression"; ACSC-17, Seventeenth Annual Computer Science Conference, Australian Computer Science Comunications, Vol. 16, No. 1, pp.189-197, January 1994.

[Yue71] Yuen, C.: "Walsh functions and Gray code"; IEEE Trans Electromagnetic Compatibility, Vol. EMC-13, pp. 68-73, August 1971.

[Yue74.1] Yuen, C. K.: "The separability of Gray code"; IEEE Transactions on Information Theory, Vol. IT-20, No. 5, pp. 668, September 1974.

[Yue77] Yuen, C. K.: "A fast analog to Gray code converter"; Proceedings of the IEEE Vol. 65, No. 10, pp. 1510-11, October 1977.

[Yue78] Yuen, C. K.: "Analog-to-Gray code conversion"; IEEE Transactions on Computers Vol. C-27, No. 10, pp. 971-973, October 1978.

[Yue89] Yuen, C. K.: "Binary division and square-rooting using gray code"; Technical report No. TRC8/88, Department of Information Systems and Computer Science, National University of Singapore, 1988. Seventeenth Annual ACM Computer Science Conference, p. 441, CA-DSP '89. 1989. International Symposium on Computer Architecture and Digital Signal Processing, Vol. 1, pp. 217-220, 1989.