Autres Codages Binaires
Gray & BCD

Ce document a pour objectif de vous familiariser avec les autres codages binaires utilisés dans l'industrie : le code Gray et le code BCD.

Tout comme le format binaire classique que vous avez déjà vu, ces codages permettent de représenter des nombres entiers en binaire (avec des 0 et des 1), mais dans des formats différents. Ils sont fréquemment utilisés dans certains contextes tels que l'électronique, ou pour aider à la résolution de problèmes logiques.

Encore une fois : une donnée binaire n'est qu'une donnée, c'est son interprétation qui est importante afin de comprendre l'information qu'elle véhicule ainsi que pour savoir quelles opérations il est possible d'effectuer avec. Par exemple, l'opération d'incrémentation en binaire classique propage une retenue de la droite vers la gauche. Néanmoins, lorsque l'on utilise d'autres formats, incrémenter ou décrémenter ne s'effectue pas de la même manière. Ainsi, il faut choisir les opérateurs adaptés au format de la donnée manipulée, ou éventuellement d'abord la transformer en binaire classique, effectuer l'opération désirée, puis re-transformer le résultat dans le format initial.

1 Code Gray

Le code Gray est un codage utilisé dans certains instruments, en particulier dans des capteurs de positions utilisant des angles, mais aussi dans les tableaux de Karnaugh pour les circuits logiques (qui serons abordés dans un autre cours).

L'idée du codage Gray est de ne modifier qu'un seul bit entre chaque incrémentation (on retrouve ce principe dans le code Baudot anciennement utilisé dans les télégraphes). Par exemple, entre 143 et 144, en représentation classique il faut modifier 5 bits, tandis qu'en code Gray, il n'y a qu'un seul bit qui varie :

Codage binaire classique

143	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
144	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
				↑	↑	↑	↑	↑

Code Gray

143	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**
144	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**
				↑				

Incrémenter en code Gray ne respecte donc absolument pas le principe de propagation de retenue de la droite vers la gauche. Il n'est donc pas possible d'additionner ou soustraire de la manière habituelle des nombres codés dans le format Gray.

Plusieurs méthodes pour « construire » le Gray code sont possibles. La plus connue est la méthode dite du *code binaire réfléchi*, qui impose de construire le code depuis 0. Parmi les autres méthodes, il existe plusieurs formules qui permettent de passer du code binaire classique au code Gray et réciproquement.

1.1 Code Binaire Réfléchi

Le code binaire réfléchi est une technique visuelle pour énumérer toutes les valeurs du code Gray. On écrit chaque nombre sur une ligne à la fois, en commençant tout d'abord par 0 et 1, puis, une fois que l'on a consommé tous les emplacements/bits de position possible, on recopie les lignes déjà écrites en appliquant un miroir tout en ajoutant un 1 dans la position suivante.

0		0
1		1
2	1	1
3	1	0

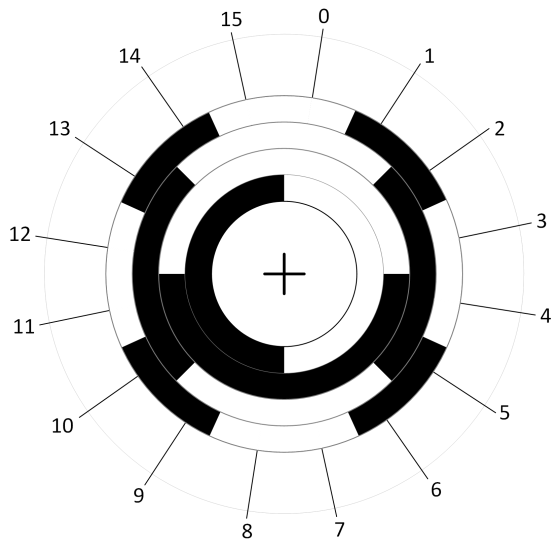
0		0	0
1		0	1
2		1	1
3		1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

0				0
1				1
2			1	1
3			1	0
4		1	1	0
5		1	1	1
6		1	0	1
7		1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

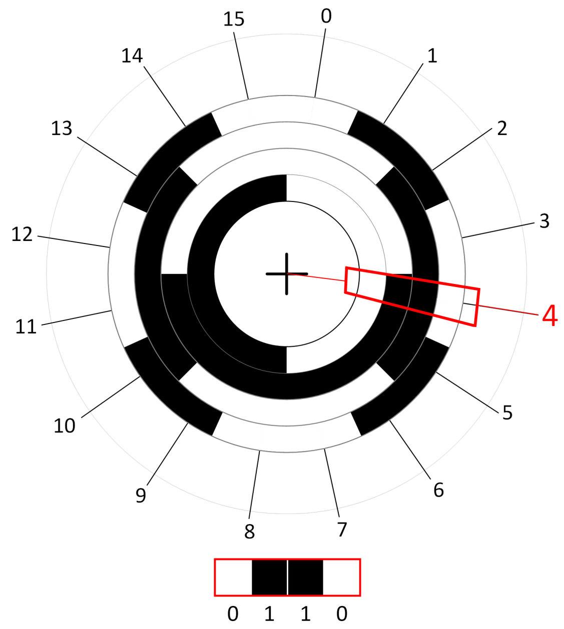
	Code Gray				Code Binaire			
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

On remarque que les plus grandes valeurs avant chaque puissance de 2 (c'est-à-dire 3, 7, 15, ...) ne sont plus constituées exclusivement de 1, mais sont au contraire constituées d'un 1 suivi de 0 exclusivement.

Une des raisons de ce fonctionnement est que le code Gray se transpose parfaitement dans des capteurs d'angles. En effet, les transitions qui ne modifient qu'un seul bit avec l'effet miroir correspondent en réalité à une roue graduée où chaque anneau représente un bit précis :



Roue codeuse Gray sur 4 bits



Codage Gray du nombre « 4 » : %0110

Ainsi, la position d'un objet sur le cadran peut être détectée et comprise au travers d'un nombre au format Gray. On peut également en déduire l'angle depuis la position codant la valeur 0.
Chaque anneau représentant un bit : plus on voudra gagner en précision sur la position, plus il faudra ajouter d'anneaux extérieurs, et donc élargir le format de la représentation binaire.

1.2 Binaire vers Gray : Formule mathématique

Une première formule permet de passer du code binaire classique au code Gray. C'est même la méthode la plus simple pour passer de n'importe quel nombre vers du code Gray lorsque vous maîtrisez le binaire classique.

$$\text{Code Gray}(N) = \frac{N \oplus 2N}{2} = \frac{N \text{ XOR } 2N}{2}$$

La formule est relativement simple :

1. on prend un nombre que l'on convertit en binaire
2. on calcule son double (c'est-à-dire qu'on le décale vers la gauche en ajoutant un 0),
3. on effectue l'opération XOR sur les deux,
4. on divise le résultat par 2 (c'est-à-dire que l'on décale vers la droite en effaçant le nombre après la virgule).

Prenons 143 comme exemple :

1. $143 = \%1000\ 1111$
2. $143 \times 2 = \%1000\ 1111 \times 2 = \%1000\ 1111\ 0 = \%1\ 0001\ 1110 = 286$
3. $143 \oplus 286 = \%1000\ 1111 \oplus \%1\ 0001\ 1110 = \%1\ 1001\ 0001$

$$\begin{array}{cc|cccc|cccc} & & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \oplus & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ \hline & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}$$

4. $\frac{143 \oplus 286}{2} = \%1\ 1001\ 0001 \div 2 = \%1100\ 1000{,}1 \Rightarrow \%\underline{1100\ 1000}$

A	B	**XOR**
0	0	0
0	1	1
1	0	1
1	1	0

1.3 Binaire vers Gray : Formule bit à bit

Une autre formule plus technique existe et consiste à déduire l'état de chaque bit entre le code Gray et son équivalent binaire. Cette technique est l'application directe en électronique de la formule précédente.

Dans la notation qui suit, les B_n correspondent au $N^{\text{ième}}$ bit de la valeur binaire classique, en partant de la puissance 0. La notation G_n correspond au $N^{\text{ième}}$ bit de la valeur en code Gray. Concrètement, 13_{10} équivaut à %1101 en binaire classique, ce qui indique que le bit 2 (donc B_2) est à 0, tandis que les bits 4, 3, et 1 (donc $B_4\,B_3\,B_1$) sont à 1.

Binaire classique

13	B_4	B_3	B_2	B_1
% 1101	1	1	0	1

Code Gray

13	G_4	G_3	G_2	G_1
% 1011	1	0	1	1

On notera également que ce que l'on appelle le *LSB* en anglais (*Least Significant Bit*) correspond en français au *bit de poids faible*, c'est-à-dire le numéro de bit qui contient la puissance de 2 la plus petite, donc le bit codant 2^0.
On peut également trouver dans la littérature la mention du *MSB* en anglais (*Most Significant Bit*) qui correspond en français au *bit de poids fort*, c'est-à-dire le bit contenant la puissance de 2 la plus grande dans le format choisi.
Ainsi, sur 12 bits, le *LSB* sera B_1 qui correspond à 2^0, et le *MSB* sera B_{12} qui correspond à 2^{11}. Sur 8 bits, le *LSB* sera le même (B_1 correspondant à 2^0), et le *MSB* sera B_8 qui correspond à 2^7.

Ces mentions *LSB* et *MSB* permettent de savoir dans quel sens lire les valeurs (de gauche à droite ou de droite à gauche).

On peut déduire un nombre au format Gray à partir de son équivalent au format binaire classique grâce à une formule calculant l'état de chaque bit. Il s'agit en réalité de la formule représentant le circuit d'un composant électronique dédié à la traduction de nombres au format binaire classique vers le format Gray. Ce circuit est l'application réelle de la formule mathématique vu précédemment.

$$G_n = B_n \oplus B_{n+1} = B_n \text{ XOR } B_{n+1}$$

Ainsi, on déduit l'état de G_1 à partir de l'état de B_1 et B_2.

Essayons avec 13 , c'est-à-dire % 1101 en binaire classique :

— $G_1 = B_1 \oplus B_2 = 1 \oplus 0 = 1$
— $G_2 = B_2 \oplus B_3 = 0 \oplus 1 = 1$
— $G_3 = B_3 \oplus B_4 = 1 \oplus 1 = 0$
— $G_4 = B_4 \oplus B_5 = 1 \oplus 0 = 1$

13	B_4	B_3	B_2	B_1
% 1101	1	1	0	1

G_4	G_3	G_2	G_1	13
1	0	1	1	% 1011

$\Rightarrow$ % 1011

On se rend bien évidemment compte que l'on effectue un XOR bit à bit de la valeur binaire initiale à la valeur binaire initiale divisée par 2.

1.4 Gray vers Binaire : Formule bit à bit

Pour retrouver un nombre au format binaire classique depuis un nombre au format Gray, il suffit d'appliquer la formule précédente dans l'autre sens... Ce qui implique cette fois de calculer les chiffres depuis le bit de poids fort jusqu'au bit de poids faible.

$$\begin{aligned}
&B_n = G_n \\
&B_{n-1} = G_n \oplus G_{n-1} \\
&... \\
&B_1 = G_n \oplus G_{n-1} \oplus ... \oplus G_1
\end{aligned}$$

Ainsi, pour un nombre sur 4 bits, on devra appliquer les formules suivantes :

$$\begin{aligned}
&B_4 = G_4 \\
&B_3 = G_4 \oplus G_3 \\
&B_2 = G_4 \oplus G_3 \oplus G_2 \\
&B_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1
\end{aligned}$$

Essayons de nouveau avec 13 au format Gray : % 1011

— $B_4 = G_4 = 1$
— $B_3 = G_4 \oplus G_3 = 1 \oplus 0 = 1$
— $B_2 = G_4 \oplus G_3 \oplus G_2 = 1 \oplus 0 \oplus 1 = 0$
— $B_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$

13	G_4	G_3	G_2	G_1
% 1011	1	0	1	1

B_4	B_3	B_2	B_1	13
1	1	0	1	% 1101

$\Rightarrow$ % 1101 $= 13_{10}$

2 BCD

Le format BCD (*Binary Coded Decimal* en anglais), ou DCB (*Décimal Codé Binaire*), est souvent utilisé pour représenter des nombres dans des IHM (*Interfaces Homme-Machine*), c'est-à-dire des nombres insérés par un clavier dans une machine ou affichés sur un écran par une machine. Beaucoup de composants électroniques et de systèmes simples s'appuient sur le BCD pour représenter les nombres.

La spécificité du BCD repose sur le fait que chaque chiffre d'un nombre décimal est représenté par une série de 4 bits. L'idée est extrêmement proche de ce qui est fait pour transformer le format binaire classique en hexadécimal, mais cela ne permet pas les mêmes opérations. Là où un système lirait un nombre binaire classique en utilisant les puissances de 2 pour chaque bit, ici, il faut prendre des paquets de 4 bits représentants un chiffre à la fois.
On ne peut donc pas ajouter des nombres aussi facilement qu'en binaire classique, mais, la manipulation des nombres décimaux est grandement simplifiée pour des opérations encore plus basiques, ainsi que pour leur affichage.

Il existe plusieurs variantes du code BCD. Ce document présentera le code BCD 8421 (le plus connu et utilisé), ainsi que le 2421 qui corrige certains aspects du 8421. Il faut surtout retenir que le code BCD est à l'origine de nombreux autres codes tels que le BCDIC et l'EBCDIC qui servent à représenter des caractères textes et les échanger entre plusieurs machines interconnectées (d'où l'usage du BCD dans certains afficheurs et claviers pour saisir/afficher des résultats à un humain).

On peut également retenir que des formules très simples permettent de transformer des nombres au format BCD en caractères ASCII ou EBCDIC, donc en caractères directement manipulables par des ordinateurs.
Il suffit de réaliser un OU logique de chaque chiffre avec % 0011 0000 (48_{10}) pour obtenir son équivalent en ASCII. Dans le cas de l'EBCDIC, il faut réaliser un OU logique de chaque chiffre avec % 1111 0000.

L'avantage du système BCD réside dans le fait qu'il n'y a pas de limite pour représenter les nombres et surtout leur précision, étant donné qu'il s'agit de réserver autant de paquets de 4 bits qu'il y a de nombres décimaux.

2.1 BCD / code 8421

Le code BCD classique, ou BCD 8421, est simplement l'usage de 4 bits pour représenter un chiffre :

7	B_4	B_3	B_2	B_1
% 0111	0	1	1	1

Représenter un seul chiffre n'a que peu d'intérêt pour expliciter le BCD. Le nombre 164 contient trois chiffres, donc il faut utiliser trois fois 4 bits pour le représenter en BCD :

164		*8*	*4*	*2*	*1*		*8*	*4*	*2*	*1*		*8*	*4*	*2*	*1*
% 0001 0110 0100 0111		0	0	0	1		0	1	1	0		0	1	0	0

$$\Rightarrow \quad \begin{array}{cccc} \% & 0001 & 0110 & 0100 \\ & 1 & 6 & 4 \end{array}$$

Cette représentation est donc extrêmement simple dès que l'on maîtrise les quatre premières puissances de 2. Chaque bit est associé à une des puissances de 2 : 8, 4, 2, et 1, d'où le nom de « *code 8421* ».
Néanmoins, on se rend compte que ce format présente également un défaut majeur : 4 bits permettent de représenter 16 valeurs, mais seules les 10 premières sont utilisées (depuis %0000, jusqu'à %1001 inclus). Certaines valeurs de la plage couverte par le format sont inutilisées (de 10 à 15), en plus de ne pas utiliser tous les états possibles sur 4 bits.

2.2 code 2421

Le code 2421 est une adaptation du code 8421 qui annule l'effet d'inutilité de certaines valeurs de la plage, mais, la couverture de tous les états possibles donne une redondance partielle (certains nombres peuvent être représentés de plusieurs manières possibles). En effet, avec 4 bits, dont deux représentent la valeur 2, on peut représenter 10 valeurs au plus : de 0 à 9 $(4+2+2+1)$.

2	**4**	**2**	**1**	**Valeur**
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	*2*
1	0	0	1	*3*
1	0	1	0	*4*
1	0	1	1	*5*
1	1	0	0	*6*
1	1	0	1	*7*
1	1	1	0	8
1	1	1	1	9

Le code 2421 tel quel n'est pas utilisé en pratique, mais le code AIKEN, qui n'est pas abordé dans ce document, l'utilise en respectant certaines contraintes afin d'obtenir des effets intéressants (7 par exemple, n'est codé qu'avec % 1101 en AIKEN).

L'objectif de ces représentations est de comprendre que la largeur de bus, donc la quantité de bits sélectionnés, implique de pouvoir représenter une quantité précise d'états, mais ces états ne représentent pas nécessairement des valeurs distinctes ou des valeurs utiles/valides dans le format choisi. De plus, ces états représentent bien une information, mais on ne peut comprendre l'information issue de cette donnée que si l'on choisit la bonne interprétation : le système qui produit la donnée respecte un format, et seules les opérations adaptées à ce format permettent de la manipuler et l'afficher correctement.

Ce document et ses illustrations ont été réalisés par Fabrice BOISSIER en octobre 2023
Son contenu est inspiré de plusieurs cours existants