Sistemas de Codificação

Circuitos Digitais I

Prof. Fernando Passold

Tabela ASCII:

32:	33: !	34: "	35: #	36: $	37: %	38: &	39: '	40: (	41:)
42: *	43: +	44: ,	45: -	46: .	47: /	48: 0	49: 1	50: 2	51: 3
52: 4	53: 5	54: 6	55: 7	56: 8	57: 9	58: :	59: ;	60: <	61: =
62: >	63: ?	64: @	65: A	66: B	67: C	68: D	69: E	70: F	71: G
72: H	73: I	74: J	75: K	76: L	77: M	78: N	79: O	80: P	81: Q
82: R	83: S	84: T	85: U	86: V	87: W	88: X	89: Y	90: Z	91: [
92: \	93:]	94: ^	95: _	96: `	97: a	98: b	99: c	100: d	101: e
102: f	103: g	104: h	105: i	106: j	107: k	108: l	109: m	110: n	111: o
112: p	113: q	114: r	115: s	116: t	117: u	118: v	119: w	120: x	121: y
122: z	123: {	124: |	125: }	126: ~	127: ⌂	128: Ç	129: ü	130: é	131: â
132: ä	133: à	134: å	135: ç	136: ê	137: ë	138: è	139: ï	140: î	141: ì
142: Ä	143: Å	144: É	145: æ	146: Æ	147: ô	148: ö	149: ò	150: û	151: ù
152: ÿ	153: Ö	154: Ü	155: ø	156: £	157: Ø	158: ×	159: ƒ	160: á	161: í
162: ó	163: ú	164: ñ	165: Ñ	166: ª	167: º	168: ¿	169: ®	170: ¬	171: ½
172: ¼	173: ¡	174: «	175: »	176: ░	177: ▒	178: ▓	179: │	180: ┤	181: Á
182: Â	183: À	184: ©	185: ╣	186: ║	187: ╗	188: ╝	189: ¢	190: ¥	191: ┐
192: └	193: ┴	194: ┬	195: ├	196: ─	197: ┼	198: ã	199: Ã	200: ╚	201: ╔
202: ╩	203: ╦	204: ╠	205: ═	206: ╬	207: ¤	208: ð	209: Ð	210: Ê	211: Ë
212: È	213: ı	214: Í	215: Î	216: Ï	217: ┘	218: ┌	219: █	220: ▄	221: ¦
222: Ì	223: ▀	224: Ó	225: ß	226: Ô	227: Ò	228: õ	229: Õ	230: µ	231: þ
232: Þ	233: Ú	234: Û	235: Ù	236: ý	237: Ý	238: ¯	239: ´	240: -	241: ±
242: ‗	243: ¾	244: ¶	245: §	246: ÷	247: ¸	248: °	249: ¨	250: ·	251: ¹
252: ³	253: ²	254: ■	255:						

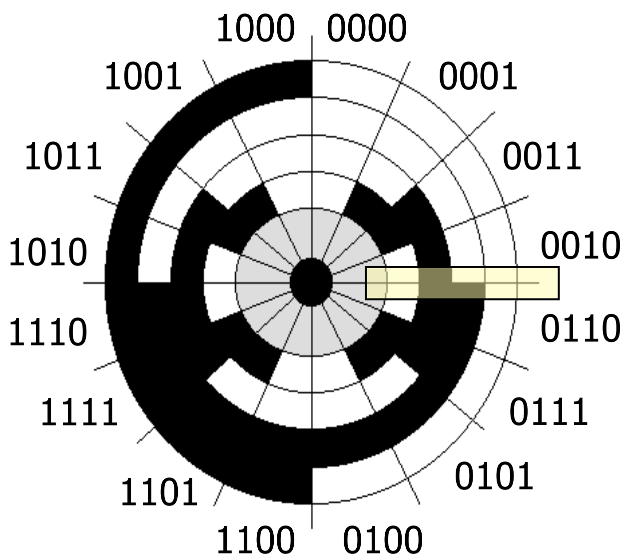
Ascii.cpp (gera tabela ASCII):

```
// Tabela ASCII
// Fernando Passsold, 05/09/2001

#include <stdio.h>
// #include <stdlib.h>
void main() {
   int codigo = 32, coluna=1, linha=1, key, aux;

   for (codigo=32; codigo<256; codigo++) {
      printf("%3d: ",codigo);
      fputchar(codigo);
      coluna++;
      if (coluna<11){
            printf("  ");
      }
      else {
        printf("\n");
            coluna=1;
            linha++;
            if (linha>24){
              // espera uma tecla ser apertada
              while ( (key = getchar()) != '\n' )
                 printf("%c",key);
            linha=1;
        }
      }
   }
}
```

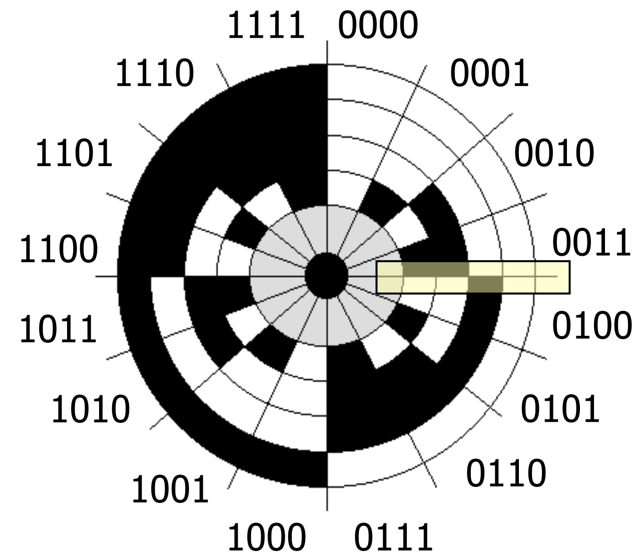
Disco Gray:

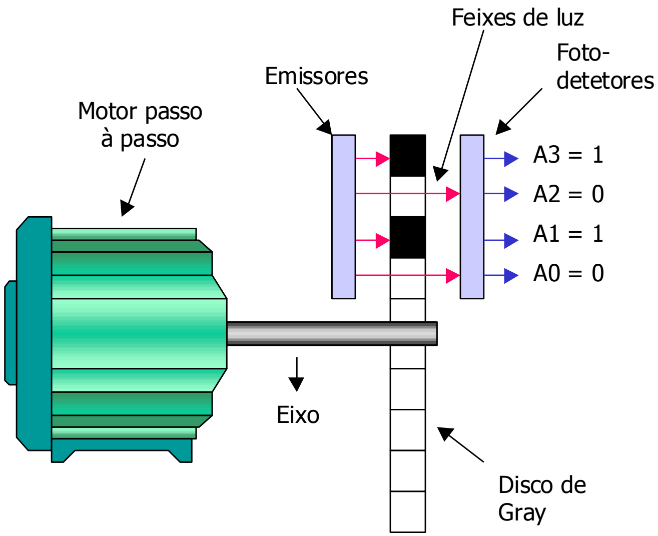


Repare a seqüência:

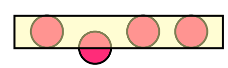
Ref	Gray	Dec	Bin
0	0000	0	0000
1	0001	1	0001
2	0011	3	0010
3	0010	2	0011
4	0110	6	0100
5	0111	7	0101
6	0101	5	0110
7	0100	6	0111
8	1100	12	1000
9	1101	13	1001
10	1111	15	1010
11	1110	14	1011
12	1010	10	1100
13	1011	11	1101
14	1001	9	1110
15	1000	8	1111

Disco Binário:





Note que no código Gray: entre uma variação e outra de código, somente um bit se altera. Evita erros de leitura se alguns dos foto-detetores se encontra desalinhado frente à seus "companheiros":



Encoders Relativos

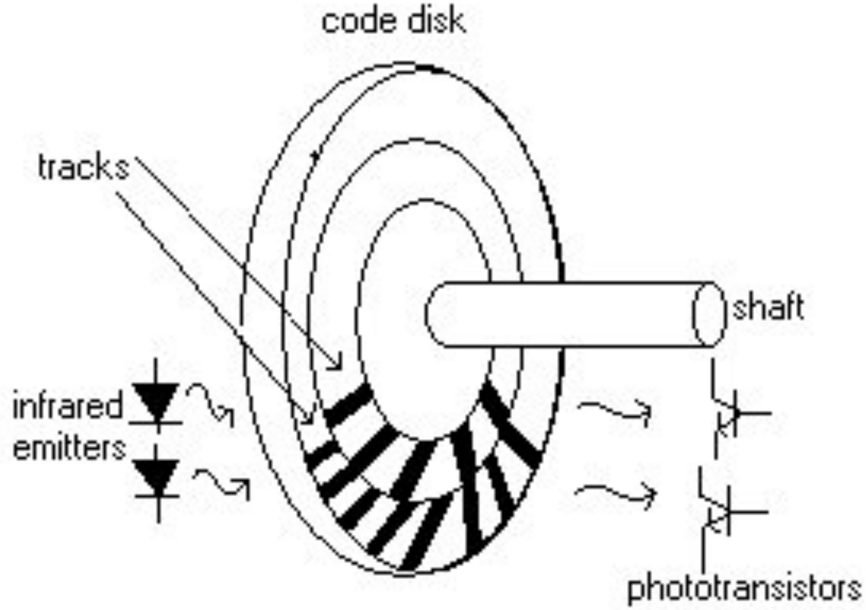


Fig 1. A rotary optical encoder

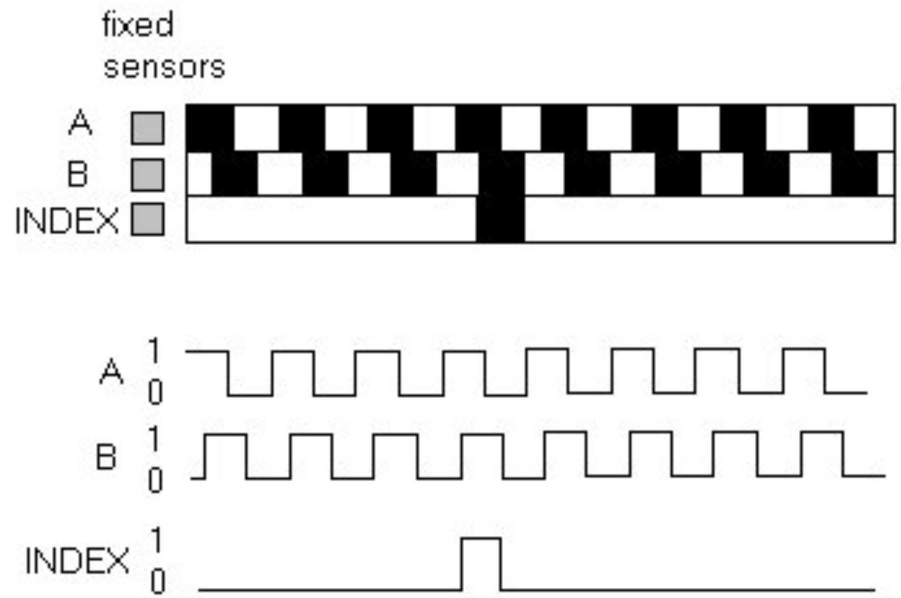


Fig 5. Incremental encoder disk track patterns

Encoders Relativos
(com detecção em quadratura)

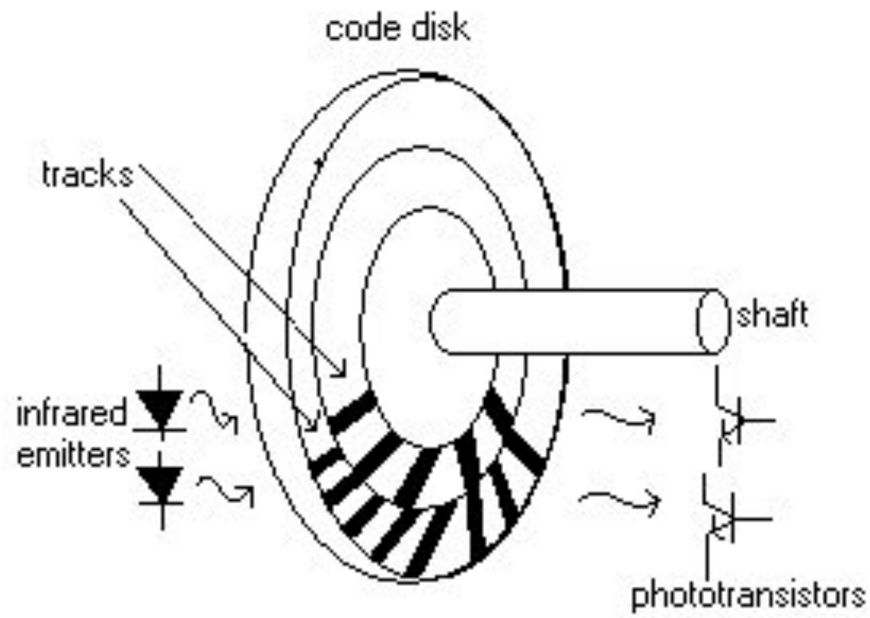


Fig 1. A rotary optical encoder

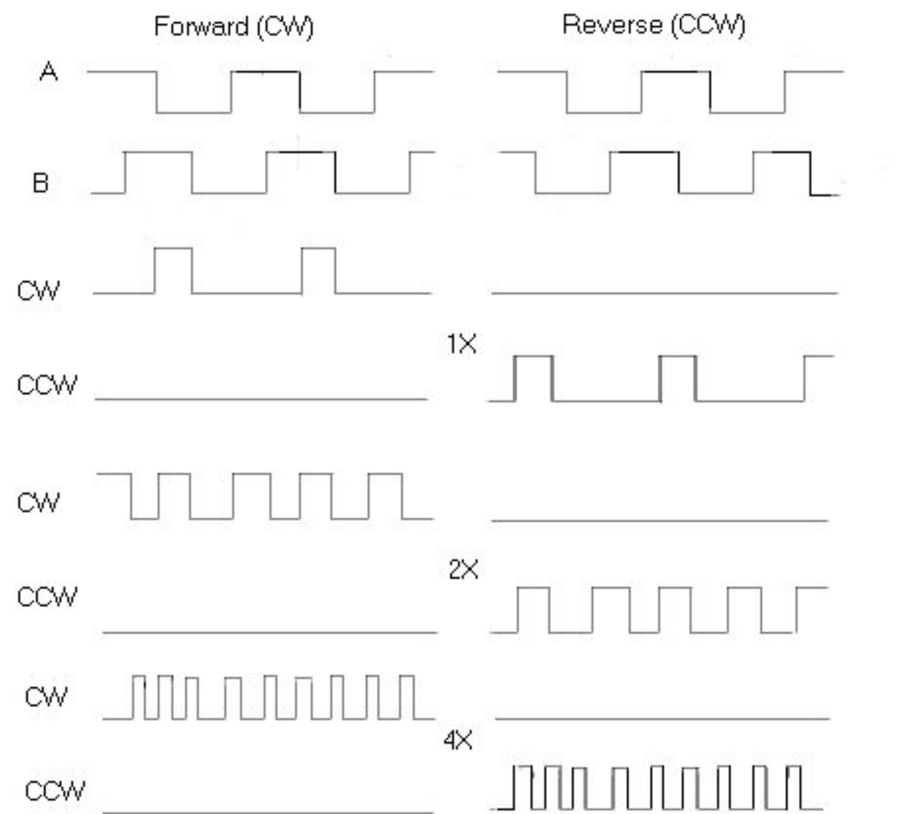
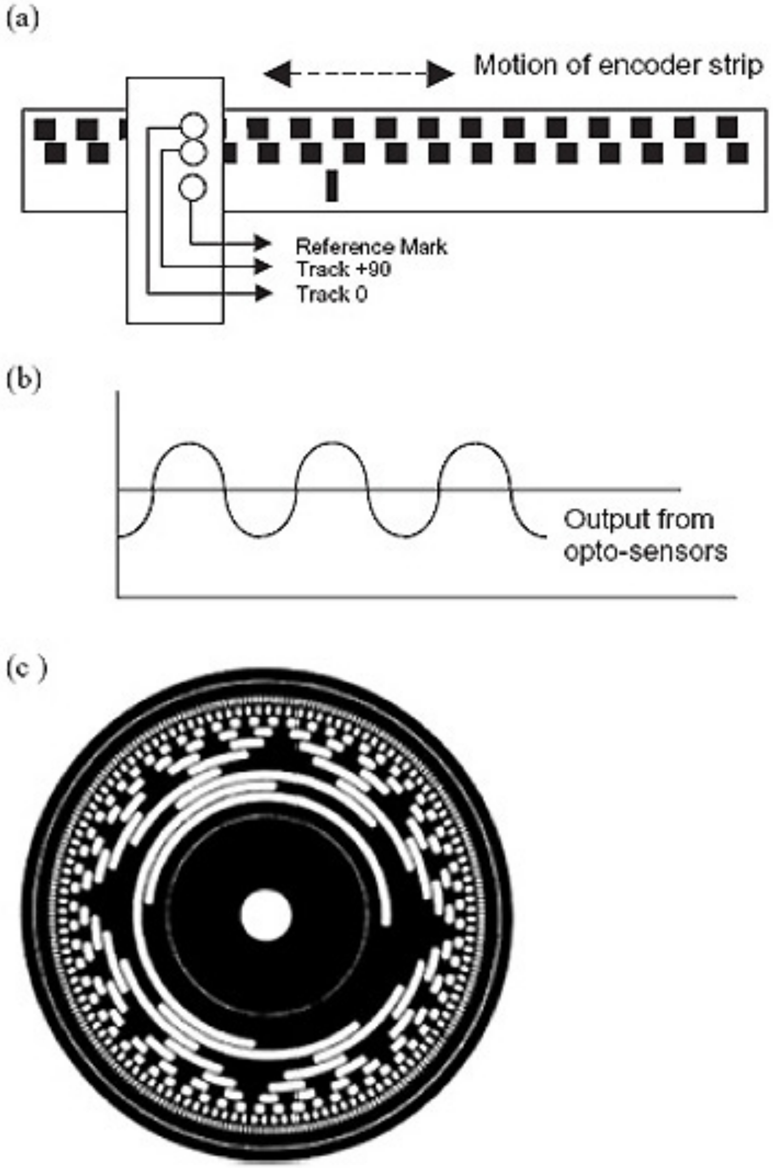


Fig 6. Quadrature direction sensing and resolution enhancement. (CW = clockwise, CCW= counter-clockwise)

Código Gray:: Encoders Absolutos



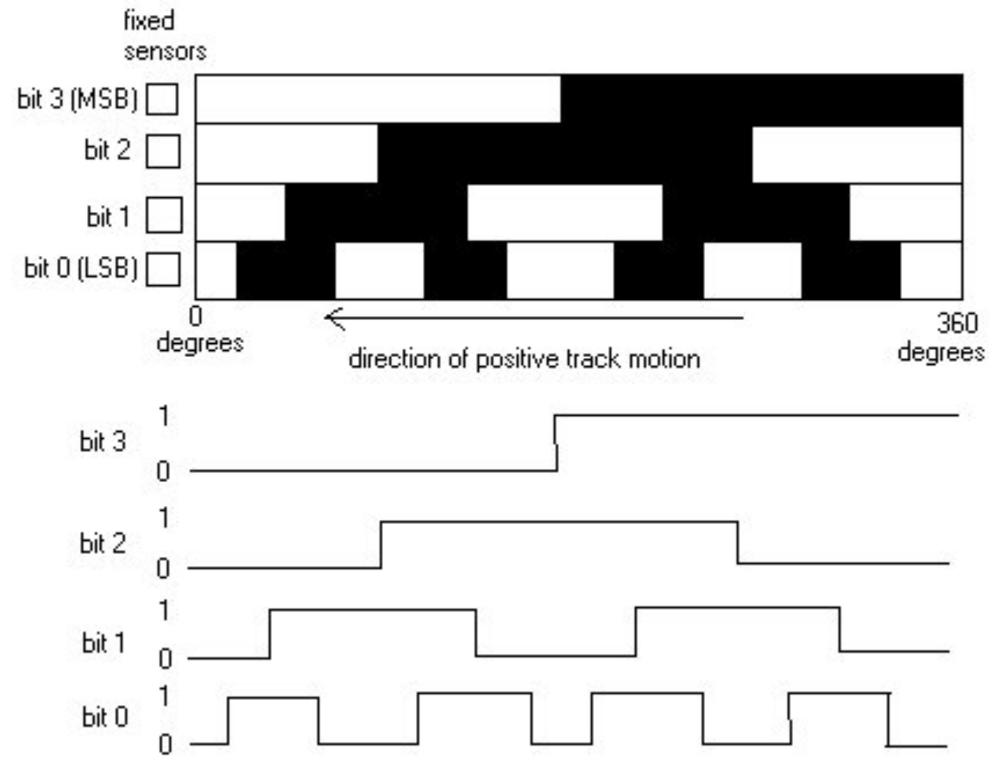
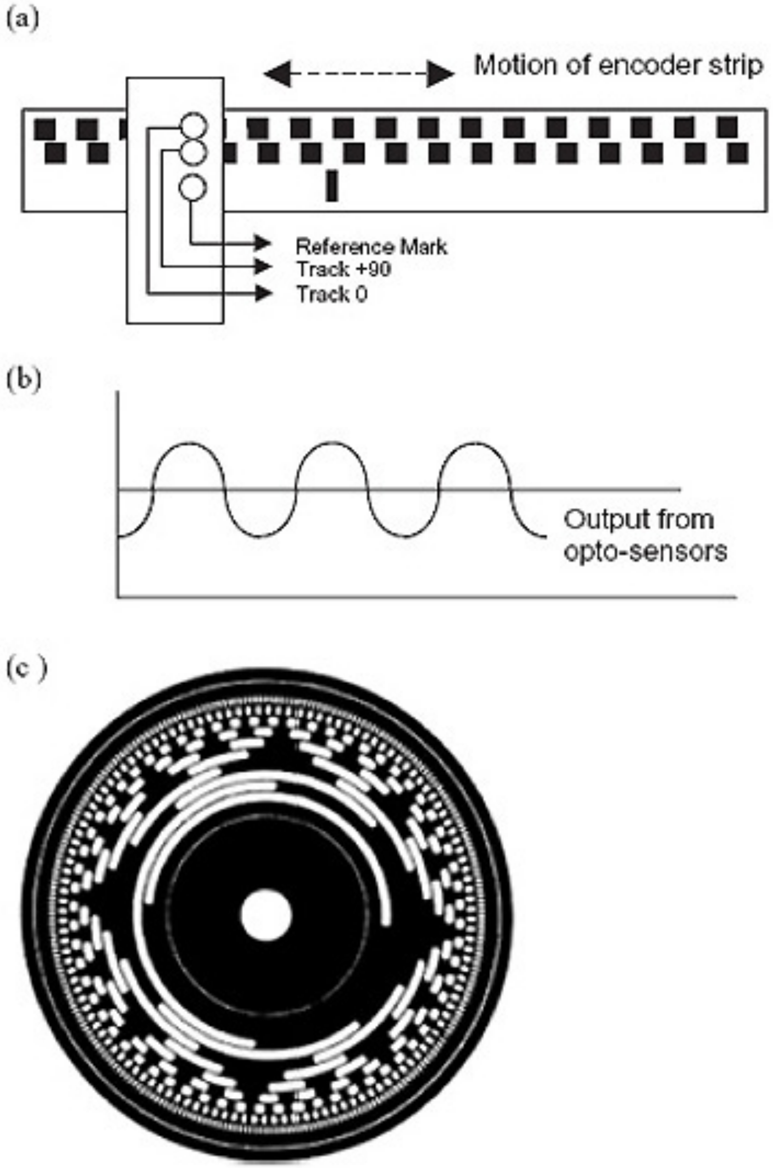
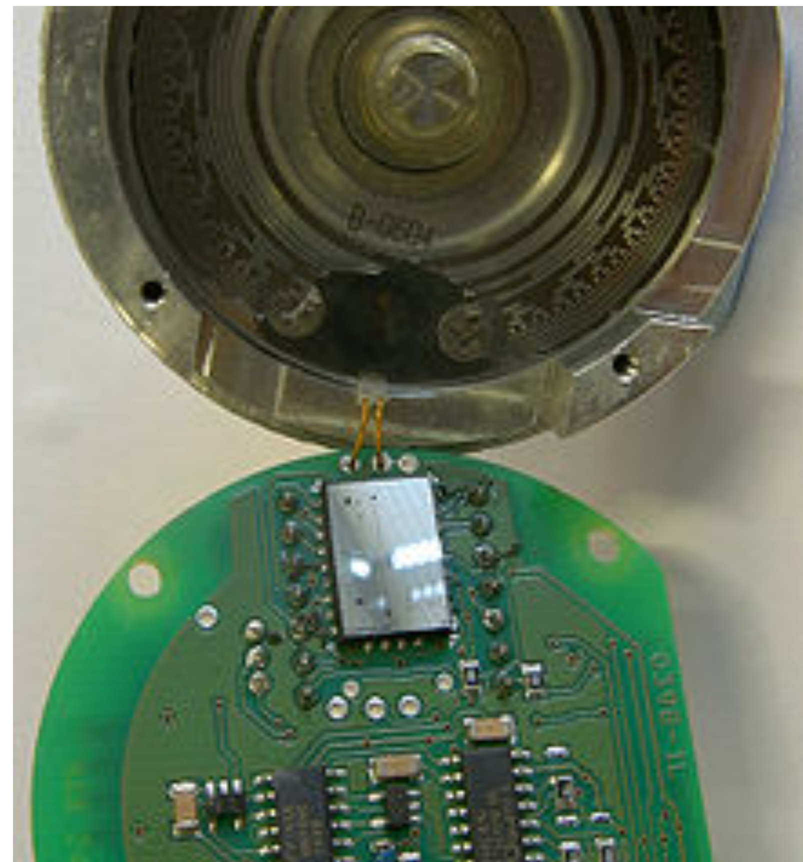


Fig 2. 4-Bit gray code absolute encoder disk track patterns

Código Gray:: Encoders Absolutos





Código Binário:: Não é Encoder Absoluto!

Seq. **Gray** (4-bits) X Seq. **Binária** (4-bits)

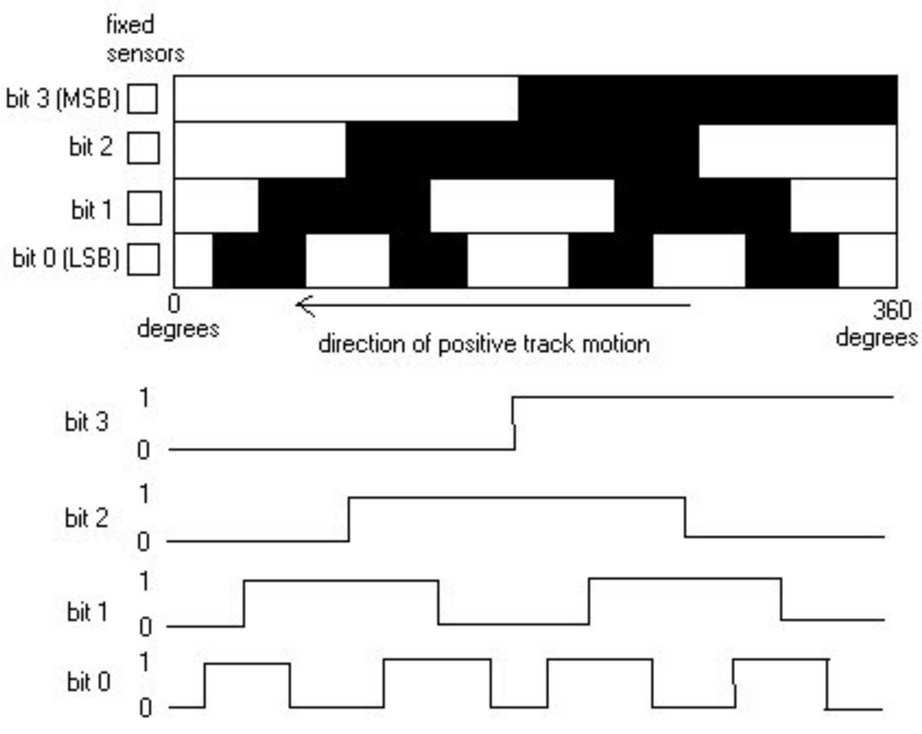


Fig 2. 4-Bit gray code absolute encoder disk track patterns

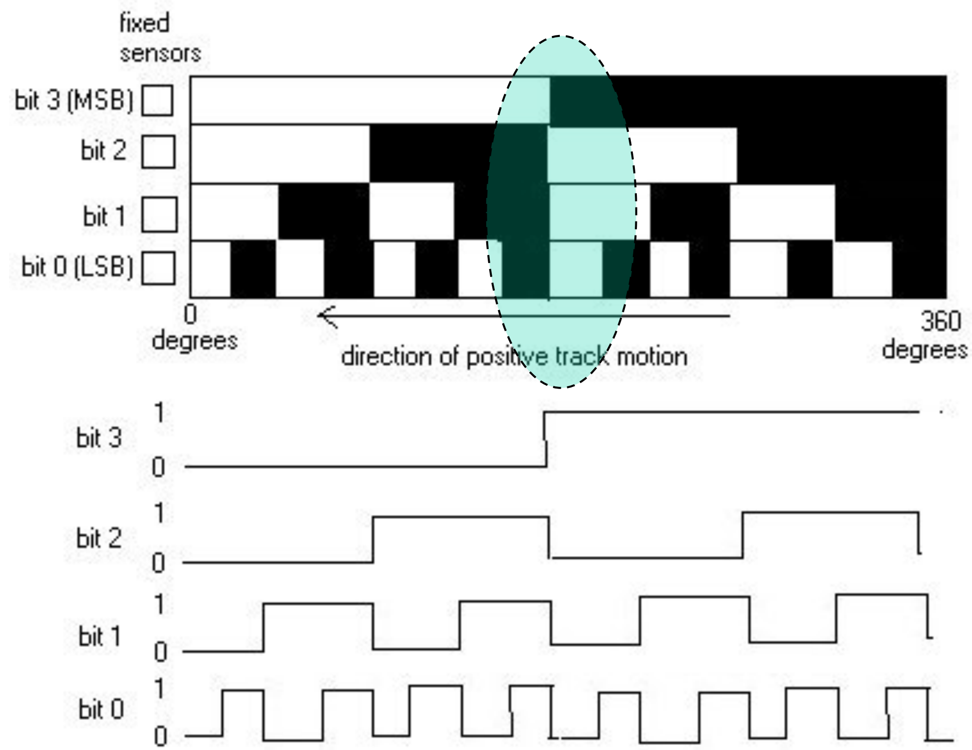


Fig 3 4-Bit binary code absolute encoder disk track patterns

Menos erros
A cada passo: 1 bit apenas varia de estado!

“Saltos grandes” => + erros

Código Binário:: Não é Encoder Absoluto!

Seq. **Gray** (4-bits) X Seq. **Binária** (4-bits)

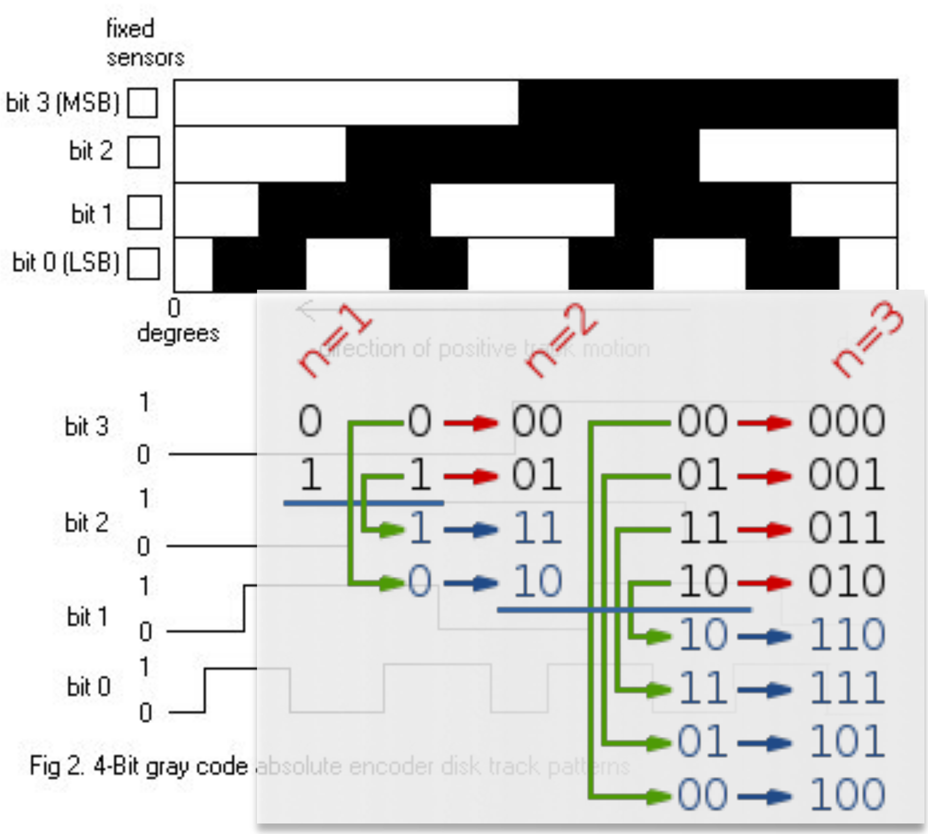


Fig 2. 4-Bit gray code absolute encoder disk track patterns

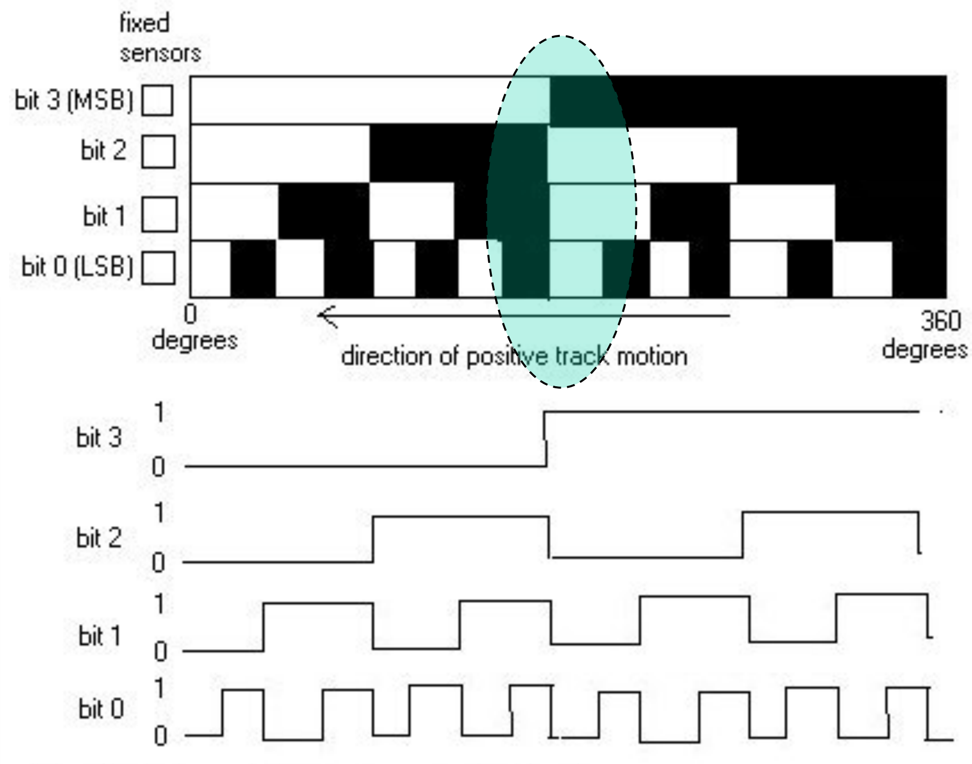
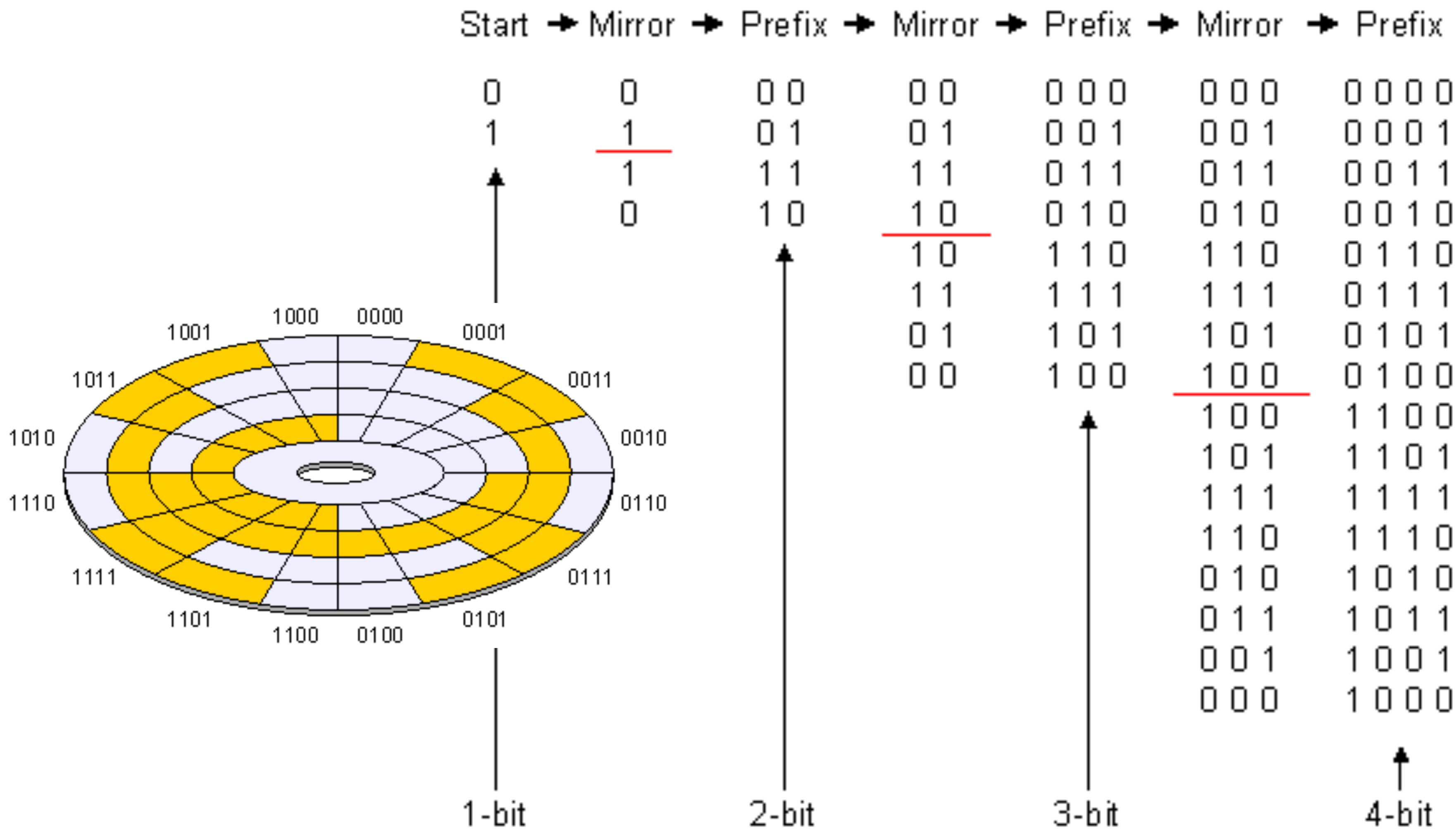


Fig 3 4-Bit binary code absolute encoder disk track patterns

Menos erros
A cada passo: 1 bit apenas varia de estado!

"Saltos grandes" => + erros

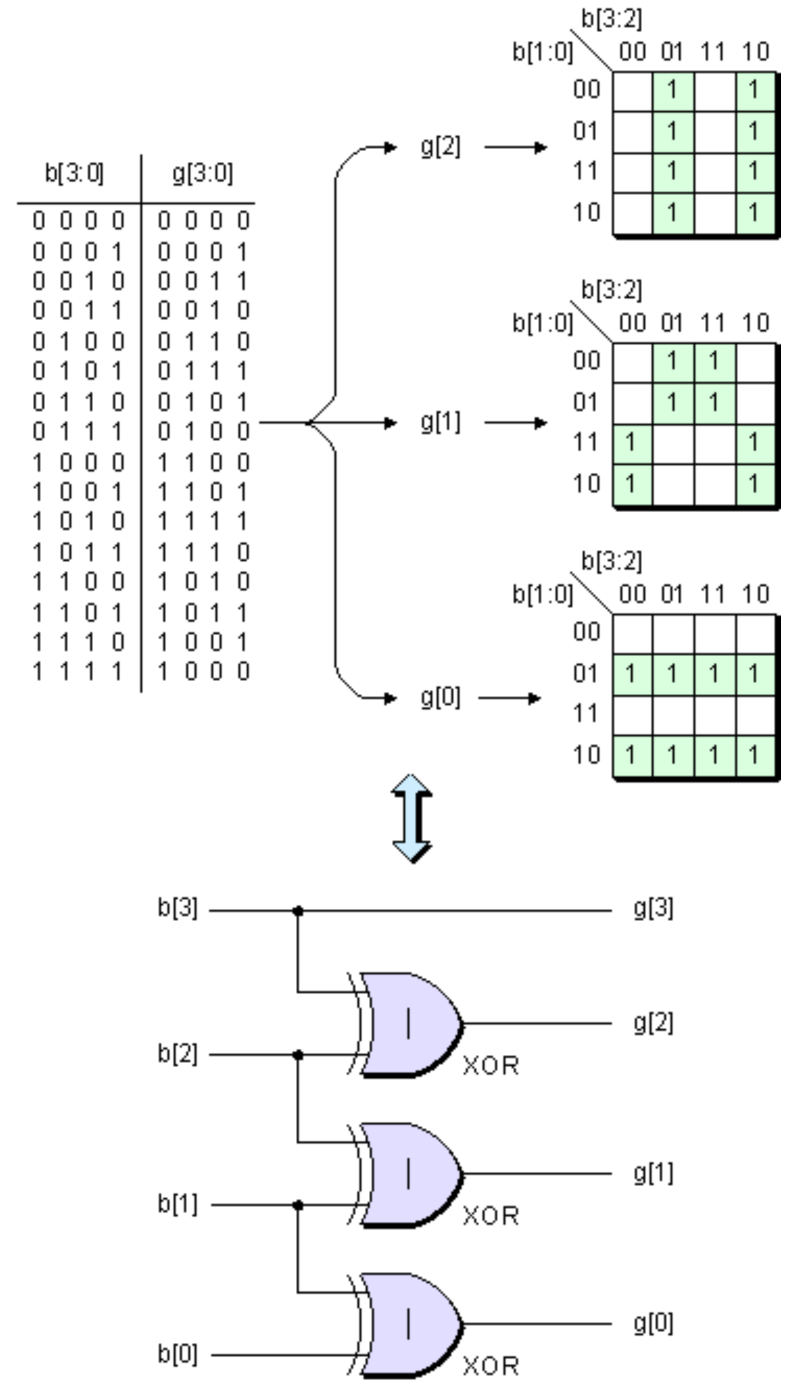
Sequência Gray



Conversão Binário → Gray

Binary Code → | b[3:0] | ← Gray Code g[3:0]

b[3:0]	g[3:0]
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 0
0 1 0 0	0 1 1 0
0 1 0 1	0 1 1 1
0 1 1 0	0 1 0 1
0 1 1 1	0 1 0 0
1 0 0 0	1 1 0 0
1 0 0 1	1 1 0 1
1 0 1 0	1 1 1 1
1 0 1 1	1 1 1 0
1 1 0 0	1 0 1 0
1 1 0 1	1 0 1 1
1 1 1 0	1 0 0 1
1 1 1 1	1 0 0 0

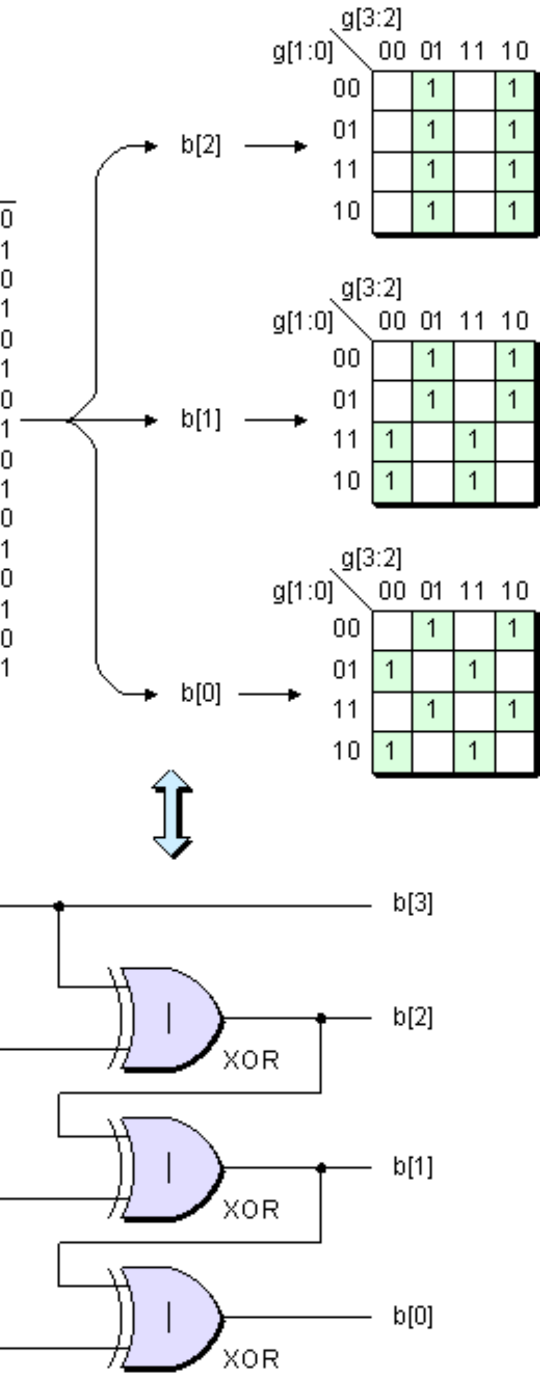


Conversão Gray → Binário

Binary Code → | ← Gray Code

b[3:0]	g[3:0]
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 0
0 1 0 0	0 1 1 0
0 1 0 1	0 1 1 1
0 1 1 0	0 1 0 1
0 1 1 1	0 1 0 0
1 0 0 0	1 1 0 0
1 0 0 1	1 1 0 1
1 0 1 0	1 1 1 1
1 0 1 1	1 1 1 0
1 1 0 0	1 0 1 0
1 1 0 1	1 0 1 1
1 1 1 0	1 0 0 1
1 1 1 1	1 0 0 0

g[3:0]	b[3:0]
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 1	0 0 1 0
0 0 1 0	0 0 1 1
0 1 1 0	0 1 0 0
0 1 1 1	0 1 0 1
0 1 0 1	0 1 1 0
0 1 0 0	0 1 1 1
1 1 0 0	1 0 0 0
1 1 0 1	1 0 0 1
1 1 1 1	1 0 1 0
1 1 1 0	1 0 1 1
1 0 1 0	1 1 0 0
1 0 1 1	1 1 0 1
1 0 0 1	1 1 1 0
1 0 0 0	1 1 1 1



Demonstração Código Gray:

http://demonstrations.wolfram.com/GrayCodesErrorReductionWithEncoders/