Apuntes de Clases

Sistemas de Representación Numéricos.

Realizado por Sergio Noriega

Introducción a los Sistemas Lógicos y Digitales
Departamento de Electrotécnia
Facultad de Ingeniería
Universidad Nacional de La Plata
2003

INDICE

1 - Introducción:

En el mundo moderno en que vivimos la electrónica juega un papel importante en todo lo que se relacione con la transmisión información, el control del medio ambiente, salud, etc.

La tendencia desde la década del ’80 es la de desarrollar sistemas electrónicos a base de técnicas digitales no sólo para realizar cálculos matemáticos sino para el control de procesos industriales que implica la adquisición de señales y su posterior procesamiento y control.

Años antes, estos procesos se realizaban con técnicas analógicas, es decir, por ejemplo, el control de temperatura de un horno se podía hacer utilizando una termocupla como elemento sensor, cuya señal se compensaba, amplificaba y se comparaba con otra de referencia.

Posteriormente la señal resultante era amplificada, derivada y/o integrada (control PID) para excitar a la etapa de potencia (por ejemplo una resistencia eléctrica).

Hoy es muy común el empleo de técnicas digitales, donde manteniendo a los elementos sensores y excitadores, a través de conversiones analógicas – digitales y digitales – analógicas, es posible convertir por ejemplo esa señal de tensión de una termocupla en un número, procesarlo convenientemente y luego volver a convertirlo en una señal de tensión que sirva para excitar a la etapa de potencia.

Con respecto a los sistemas informáticos, como por ejemplo las populares computadoras personales, todo su funcionamiento se basa en un dispositivo digital denominado microprocesador, el cual, bajo el control de un programa (sucesión de instrucciones almacenadas en una memoria), puede realizar entre otras cosas operaciones matemáticas, operaciones lógicas, comunicación con otros dispositivos complejos como impresoras, disco rígido, etc..

Este programa, el cual viene a ser como el “repertorio” que debe seguir el microprocesador para que todo funcione correctamente, está escrito en un lenguaje a base de números.

Estos son sólo ejemplos, de lo que un sistema digital puede realizar.

La base del entendimiento de los sistemas digitales pasa en principio por la comprensión de cómo se define y opera con las nuevas reglas de juego de este lenguaje numérico.

Así como en las tareas cotidianas es común que nosotros realicemos operaciones matemáticas de suma, resta, multiplicación, división, etc. con los llamados números decimales, en los sistemas digitales se emplea otro tipo de numeración, otro código, que se denomina código binario, el cual fue elegido por su sencillez de implementación con la electrónica que disponemos hoy en día.

2 - Sistemas de numeración:

Un sistema numérico se podría definir como aquel conjunto de reglas creadas para representar una cantidad dada.

Cantidad: Propiedad de agrupar cosas.

Número: Representación de una cantidad. Se compone de uno ó mas dígitos.

Dígito: Elemento componente de un número para representar una cantidad.

Símbolo: Elemento componente de un dígito.

Base: Cantidad de símbolos diferentes que puede representar a un dígito.

Rango: Cantidad de cantidades que se pueden expresar con un dado número de dígitos.

Podemos decir que hay infinitos sistemas a emplear, para representar una cantidad.

Los mas difundidos son el decimal, binario, octal y hexadecimal.
La diferencia entre ellos es la base empleada, es decir, cuantos símbolos diferentes se permiten para formar un dígito.

En el sistema decimal, hay 10 símbolos diferentes: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9.
En el sistema binario, hay 2: 0 y 1.
En el octal, hay 8: 0, 1, 2, 3, 4, 5, 6, y 7.
En el hexadecimal hay 16: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, y F.

En forma general, podemos escribir un número de la siguiente manera, para representar números enteros:

$$N = \sum_{i=0}^{n-1} Si \cdot b^{i}$$

Donde ***Si*** es un dígito en la posición i, representado por alguno de los ***b*** símbolos existentes en dicho sistema.

La forma definitiva de representar a N, es como la sucesión de dígitos ***Si***, es decir:

$$N = S_{n-1} S_{n-2} \cdots S_2 S_1 S_0$$

El cual sale de realizar la sumatoria anterior que es:

$$N = S_{n-1} \cdot b^{n-1} + S_{n-2} \cdot b^{n-2} + \cdots + S_2 \cdot b^{2} + S_1 \cdot b^{1} + S_0 \cdot b^{0}$$

Para representar un número donde puede tener parte entera y/o decimal (dígitos a la derecha de la coma) se puede emplear la misma expresión anterior:

$$N = \sum_{i=-m}^{n-1} Si \cdot b^{i}$$

$$N = S_{n-1} \cdot b^{n-1} + S_{n-2} \cdot b^{n-2} + \cdots + S_2 \cdot b^{2} + S_1 \cdot b^{1} + S_0 \cdot b^{0} + S_{-1} \cdot b^{-1} + S_{-2} \cdot b^{-2} + \ldots + S_{-m} \cdot b^{-m}$$

Sistema de representación decimal:

Para representar números enteros en este sistema de representación, se debe usar la base b = 10.

En tal caso la expresión de N quedará:

$$N = \sum_{i=0}^{n-1} Si \cdot 10^{i}$$

Cuyo desarrollo será:

$$N = S_{n-1} \cdot 10^{n-1} + S_{n-2} \cdot 10^{n-2} + \cdots + S_2 \cdot 10^{2} + S_1 \cdot 10^{1} + S_0 \cdot 10^{0}$$

En este caso, cada dígito ***Si*** podrá tomar cualquiera de los 10 símbolos posibles del 0 al 9.

Se observa que el número N está representado por ***n*** dígitos, donde cada uno de ellos tiene un "peso" diferente, siendo el de mayor valor el que está mas a la izquierda y el de menor el que está mas sobre la derecha.
Cada uno tiene un factor de 10 elevado a un índice que está directamente relacionado con su posición en la representación de dicho número.

El rango de un número depende de la cantidad de dígitos que lo represente. Cuanto mayor sea este número, mayor cantidad de números diferentes se podrán representar.

Por ejemplo un número entero que se quiera representar con 4 dígitos, podrá ir de 0000 a 9999, es decir, que se pueden representar 10000 números diferentes (incluído el 0), siendo el rango de 10000 y el número máximo, 9999.

Matemáticamente podemos expresar esto como:

Rango = 10^{n}

Número máximo = $10^{n1} - 1$

Número mínimo = **0**

Sistema númerico binario:

En dicho sistema, la base es igual a ***b = 2***. Esto quiere decir que cada dígito que representa a un número en este sistema podrá tener uno de dos símbolos diferentes, el 0 o el 1.

Este sistema es la base de la lógica digital para poder realizar funciones matemáticas ó lógicas (se verá esto mas adelante, en otro capítulo).

Al igual que en el caso anterior, podemos expresar un dado número como:

$$N = \sum_{i=0}^{n-1} Si \cdot 2^{i}$$

Siendo su desarrollo:

$$N = S_{n-1} \cdot 2^{n-1} + S_{n-2} \cdot 2^{n-2} + \cdots + S_2 \cdot 2^2 + S_1 \cdot 2^1 + S_0 \cdot 2^0$$

Se denomina ***bit*** a cualquiera de los ***n*** dígitos que forman al número.
Se dice que si el número tiene 5 dígitos es de 5 bits y viceversa.

Aquí cada dígito ***Si*** tiene un peso diferente según su posición, siendo el MSB (Most Significative Bit) ó bit mas significativo, el de mayor peso, es decir, el primero de la izquierda y el LSB (Least Significative Bit) ó bit menos significativo, el de menor peso, el primero desde la derecha.

Por ejemplo, el número binario **1**010010**0** de 7 bits, tiene un “1”como MSB y un “0” como LSB.

El rango de un número binario, al igual que en el caso anterior, depende de la cantidad de dígitos binarios que lo represente.

Por ejemplo un número que se quiera representar con 4 bits podrá ir de 0000 a 1111, es decir, que se pueden representar 16 números diferentes (incluído el 0), siendo el rango de 16 y el número máximo, 15.

Matemáticamente podemos expresar esto como:

Rango = 2^n

Número máximo = $2^n - 1$

Número mínimo = **0**

Representación de números binarios con coma:

$$N = S_{n-1} \cdot 2^{n-1} + S_{n-2} \cdot 2^{n-2} + \cdots + S_2 \cdot 2^2 + S_1 \cdot 2^1 + S_0 \cdot 2^0 + S_{-1} \cdot 2^{-1} + S_{-2} \cdot 2^{-2}$$
$$+ \ldots + S_{-m} \cdot 2^{-m}$$

Como se observa en esta representación un número con parte entera y decimal está formado por el desarrollo de dígitos cada uno con un peso diferente (potencias de 2 para los dígitos enteros y potencias de –2 para los dígitos decimales).

Ejemplo: 1011100001,0010001, donde 1011100001 es la parte entera y 0010001 la parte decimal.

Sistema numérico Octal:

Este sistema numérico de representación usa base 8, es decir que cada dígito consta de uno de entre 8 símbolos del 0 al 7.

La siguiente ecuación expresa un número entero de n dígitos en octal

$$N = \sum_{i=0}^{n-1} Si \cdot 8^i$$

Siendo su desarrollo:

$$N = S_{n-1} \cdot 8^{n-1} + S_{n-2} \cdot 8^{n-2} + \cdots + S_2 \cdot 8^2 + S_1 \cdot 8^1 + S_0 \cdot 8^0$$

Ejemplos de números en octal son: 123, 567, 776, 3, etc.

El mínimo número entero a representar con ***n*** dígitos es el 0.
El máximo número entero es el 7....7 (todos siete) con ***n*** dígitos.

Matemáticamente esto es:

Rango = 8^n

Número máximo = $8^n - 1$

Número mínimo = **0**

De igual manera, un número natural en octal se representa con parte entera y decimal.

Por ejemplo: 33,7652, 0,000023, etc.

Sistema númerico Hexadecimal:

En este caso la base es b = 16.

Dado que se requiere por dígito un solo símbolo de entre 16, como en este caso, y siendo que la máxima cantidad de símbolos numéricos son 10 (del 0 al 9), es necesario agregar algún otro tipo de símbolo para completar los 6 faltantes.

Es por ello que se emplean letras, donde:

A representa el 10 decimal.
B representa el 11 decimal.
C representa el 12 decimal.
D representa el 13 decimal.
E representa el 14 decimal.
F representa el 15 decimal.

Su expresión para números enteros es:

$$N = \sum_{i=0}^{n-1} Si \cdot 16^{i}$$

Y su desarrollo:

$$N = S_{n-1} \cdot 16^{n-1} + S_{n-2} \cdot 16^{n-2} + \cdots + S_2 \cdot 16^{2} + S_1 \cdot 16^{1} + S_0 \cdot 16^{0}$$

Ejemplos de números hexadecimales enteros:

ABC4, 712, FFFF, DD10, etc.

Matemáticamente podemos expresar el rango, número mínimo y máximo a representar, como:

Rango = 16^{n}

Número máximo = $16^{n} - 1$

Número mínimo = **0**

De igual manera que con los enteros, los números reales tendrán la forma:

$$N = S_{n-1} \cdot 16^{n-1} + S_{n-2} \cdot 16^{n-2} + \cdots + S_2 \cdot 16^{2} + S_1 \cdot 16^{1} + S_0 \cdot 16^{0} + S_{-1} \cdot 16^{-1} + S_{-2} \cdot 16^{-2} + \ldots + S_{-m} \cdot 16^{-m}$$

Ejemplos: ABD,FF34, 0,00A, 87,1, etc.

Sistema númerico BCD (Binary Coded Decimal):

Con este sistema de núméración se expresa un número decimal (base 10), por medio de dígitos, los cuales cada uno está formado por un paquete de cuatro números binarios.

Los dígitos binarios posibles son:

0000 = "0"
0001 = "1"
0010 = "2"
0011 = "3"
0100 = "4"
0101 = "5"
0110 = "6"
0111 = "7"
1000 = "8"
1001 = "9"

De allí el nombre de BCD (Binary Coded Decimal ó Decimal Codificado en Binario).

Ejemplos: El número decimal 779 se expresa en BCD como : 0111 0111 1001

y el número 6523,4201 como: 0110 0101 0010 0011 , 0100 0010 0000 0001

Como ejemplo podemos decir que este sistema de representación es ampliamente utilizado en circuitos digitales para la visualización de datos numéricos en displays de calculadoras, etc.

3 - Conversión de un sistema numérico a otro

Conversión de binario a decimal:

Dado un número real en binario, su conversión a base 10 es automática, empleando la expresión anteriormente vista:

$$N = S_{n-1}\cdot 2^{n-1} + S_{n-2}\cdot 2^{n-2} + \cdots + S_2\cdot 2^2 + S_1\cdot 2^1 + S_0\cdot 2^0 + S_{-1}\cdot 2^{-1} + S_{-2}\cdot 2^{-2} + \ldots + S_{-m}\cdot 2^{-m}$$

Basta con realizar las operaciones de suma y producto y se tiene el número decimal.

Ejemplo: El número binario 1011100,001 se puede expresar como:

1x2^6 + 0x2^5 + 1x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 0x2^0 + 0x2^(-1) + 0x2^(-2) + 1x2^(-3) = 64 + 0 +16 + 8 + 4 + 0 + 0 + 0 + 0 + 0,125 = 92,125.

El resultado es: 92,125 en base 10.

Una forma rápida de poder hallar el número decimal que le corresponde a otro en binario es confeccionando la siguiente tabla, donde debajo de los "pesos" que implica cada posición de un bit se pone el dígito que forma el número binario.

Debajo del mismo se marca el "peso" respectivo cuando hay un "1" en esa posición, haciendo luego la suma total de los "pesos" que dará el valor definitivo del número en base 10.

Peso	1024	512	256	128	64	32	16	8	4	2	1	Decimal
Número binario	1	0	0	1	1	1	0	0	0	0	1	
Valor parcial	1024	0	0	128	64	32	0	0	0	0	1	1249

Este es un ejemplo para números binarios enteros de hasta 11 bits. Su extensón a un rango mayor solo requiere seguir agregando términos a la izquierda:
(2048, 4096, etc.) .

Lo mismo para los númeroa binarios que tienen parte decimal donde se debe agregar hacia la derecha después del "1", los términos 0,5, 0,25, 0,125, 0,0625, etc.

Conversión de octal a decimal:

En forma similar al caso anterior, planteando el desarrollo de la expresión de N en función de los dígitos Si en octal, es posible convertir un número en formato octal a decimal, directamente:

Ejemplo: El número 773 en octal desarrollado será: 7 x 8^2 + 7 x 8^1 + 3 x 8^0 = 507 en decimal, donde aquí los "pesos"son 1 para el dígito menos significativo, 8 para el siguiente, luego 8^2 = 64, etc.

Como se vió para números binarios, es posible confeccionar una tabla de conversión a fin de convertir rápidamente un número de octal a decimal.
Lo único que cambia son los valores de los "pesos" en cada posición.

Conversión de hexadecimal a decimal:

Empleando la expresión correspondiente al desarrollo de N en función de los dígitos Si hexadecimales, se puede convertir de un formato al otro:

El análisis es el mismo que con las otras dos conversiones anteriores.

Ejemplo: El número AB45,E = 10x16^3 + 11x16^2 + 4x16^1 + 5x16^0 + 14x16^(-1)
AB45,E = 10x4096 + 11x256 + 4x16 + 5 + 14x0,0625
AB45,E = 40960 + 2816 + 64 + 5 + 0,875
AB45,E = 43845,875

Conversión de decimal a binario:

Se ha visto que un número binario se puede representar como:

$$N = S_{n-1}\cdot 2^{n-1} + S_{n-2}\cdot 2^{n-2} + \cdots + S_2\cdot 2^2 + S_1\cdot 2^1 + S_0\cdot 2^0 + S_{-1}\cdot 2^{-1} + S_{-2}\cdot 2^{-2} + \ldots + S_{-m}\cdot 2^{-m}$$

El número decimal N está expresado como el desarrollo de potencias de 2, donde cada posición de un dígito binario tiene diferente peso.

El proceso para convertir un número en el sistema decimal en uno en formato binario, se puede dividir en tres partes:

1) Convertir la parte entera por un lado.

2) Convertir la parte decimal por el otro.

3) Complementar ambos resultados.

Para explicar esto tomaremos como ejemplo la conversión del número 135,138 decimal a binario.

1) Tomando solo la parte entera del número N (135,138), la que llamaremos N1(135), lo que se quiere hacer es determinar el valor (0 ó 1) de cada uno de los dígitos binarios que lo componen.

$$N1 = S_{n-1}\cdot 2^{n-1} + S_{n-2}\cdot 2^{n-2} + \cdots + S_2\cdot 2^2 + S_1\cdot 2^1 + S_0\cdot 2^0$$

Si dividimos a N1 por la base binaria 2, tendremos que la expresión queda:

$$N1/2 = S_{n-1}\cdot 2^{n-2} + S_{n-2}\cdot 2^{n-3} + \cdots + S_2\cdot 2^1 + S_1\cdot 2^0 + S_0\cdot 2^{-1}$$

Como en una división cualquiera, tenemos que en el término S0x2^(-1), S0 es el resto de la división, el 2 es el divisor, mientras que el resto de los términos corresponde al cociente de la división.

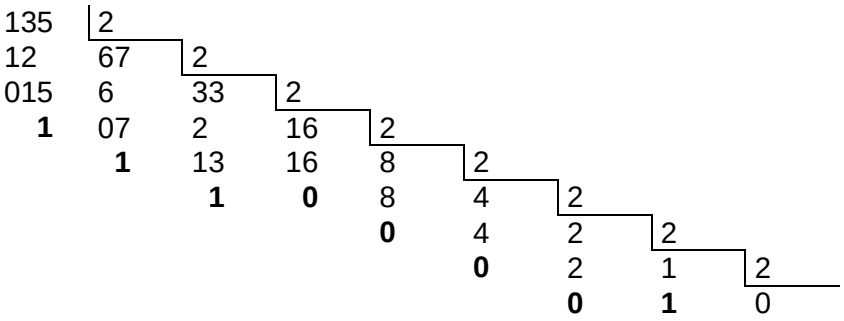
S0, entonces ya se ha determinado y es el bit menos significativo del número binario.

Para determinar S1, se procede igual que en el paso anterior, donde ahora hay que tomar ese cociente y volver a dividirlo por 2.
Se obtendrá así que el resto de esa división corresponderá a S1 (segundo bit menos significativo).

Se procede así hasta determinar el último dígito binario, el cual corresponde al resto de la división cuyo cociente sea nulo.

Del ejemplo:

Realizando las sucesivas divisiones, obtendremos:



Se indican en "**negritas**" los dígitos binarios.
El número binario será entonces: 10000111 que corresponde al 135 decimal.

2) Tomando la parte decimal, llamaremos a este número: N2 (0,138), el cual representa a la parte fraccionaria del número N, es decir no tiene parte entera.

Su representación es:

$$N2 = S_{-1} \cdot 2^{-1} + S_{-2} \cdot 2^{-2} + ... + S_{-m} \cdot 2^{-m}$$

Como para el caso del número entero, lo que hay que determinar aquí son los dígitos binarios que lo representan.

Analizando la expresión de N2, vemos que si lo multiplicamos por 2, nos quedará S_1, como parte entera, fácil de identificar y el resto como fracción.

Si luego volvemos a quedarnos con esa fracción y la multiplicamos por 2, tendremos que S_2, quedará como parte entera y así sucesivamente.

Cabe aclarar que salvo que el número fraccionario en base 10 sea múltiplo de potencias de -2, el mismo puede estar representado por un número muy grande ó hasta infinitos dígitos binarios, donde obviamente habrá que redondear con n cifras significativas.

Del ejemplo, realizamos las siguientes operaciones:

2 x 0,138 = 0,276 → S_1 = 0
2 x 0,276 = 0,552 → S_2 = 0
2 x 0,552 = 1,104 → S_3 = 1
2 x 0,104 = 0,208 → S_4 = 0
2 x 0,208 = 0,416 → S_5 = 0
2 x 0,416 = 0,832 → S_6 = 0
2 x 0,832 = 1,664 → S_7 = 1
2 x 0,664 = 1,328 → S_8 = 1
2 x 0,328 = 0,656 → S_9 = 0
2 x 0,656 = 1,312 → S_10 = 0

y así siguiendo, quizás indefinidamente (aquí se cortó ex profeso en la décima cifra significativa, pero el número tiene mas cifras).

El número 0,138 ≅ 0,0010001101 = 0,137695312 el cual es aproximado a 0,138 pero no igual dado el truncamiento realizado (el error relativo en este caso al haber cortado aquí es del -0,22%).

Por lo tanto el número 135,138 será aproximadamente

135,138 ≅ 10000111, 0010001101

Conversión de octal a binario:

Los sistemas de representación binario, octal y hexadecimal tienen una particularidad, y es que sus bases son potencia de 2, (binario: b = 2 = 2^1, octal: b = 8 = 2^3, hexadecimal: b = 16 = 2^4).

Esto es una ventaja ya que resulta muy fácil realizar conversiones entre estos sistemas.

Dada la expresión que representa el desarrollo en potencias de 2 de un número entero binario, si agrupamos desde el bit menos significativo hacia la izquierda de a tres términos, obtenemos lo siguiente:

Inicialmente, tenemos para el caso de 9 bits:

$$N = S_8 \cdot 2^8 + S_7 \cdot 2^7 + S_6 \cdot 2^6 + S_5 \cdot 2^5 + S_4 \cdot 2^4 + S_3 \cdot 2^3 + S_2 \cdot 2^2 + S_1 \cdot 2^1 + S_0 \cdot 2^0$$

Al agrupar:

$$N = (S_8 \cdot 2^2 + S_7 \cdot 2^1 + S_6) \cdot 2^6 + (S_5 \cdot 2^2 S_4 \cdot 2^1 + S_3) 2^3 + S_2 \cdot 2^2 + S_1 \cdot 2^1 + S_0 \cdot 2^0$$

$$N = (S_8 \cdot 2^2 + S_7 \cdot 2^1 + S_6) \cdot 8^2 + (S_5 \cdot 2^2 S_4 \cdot 2^1 + S_3) 8^1 + (S_2 \cdot 2^2 + S_1 \cdot 2^1 + S_0) \cdot 8^0$$

Vemos que ahora aparecen términos cada uno con un "peso" igual a una potencia de 8, desde 1 para el menos significativo, luego 2^1 = 8, 2^8 = 64, etc.

Cada término está formado por tres dígitos binarios que pueden variar desde 000 a 111.

De aquí se puede concluir que dado un número binario **entero**, tomando grupos de a tres bits de derecha a izquierda, cada grupo pasado al número decimal que representa nos dará los dígitos en octal.

Ejemplo: Dado el número 1110011100010101110001 en binario, pasarlo a octal.

1) Agrupamos de derecha a izquierda

0 **0** 1	1 1 0	0 1 1	1 0 0	0 1 0	1 0 1	1 1 0	0 0 1
1	6	3	4	2	5	6	1

Se observa en este caso que debido a que el número de bits no es múltiplo de tres se deben agregar "**ceros**" a la izquierda hasta completar un dígito octal con tres bits.

En este caso el número 1110011100010101110001 es igual a: 16342561 en octal.

De igual manera si el número tuviera parte fraccionaria, se deben agrupar los bits desde la posición de la coma hacia la derecha de tres en tres.

Por ejemplo un número natural binario 1101100,0101111 sería el 154,274 en octal.
El último dígito en octal es el 4 ya que hubo que completar con dos ceros al número binario en la parte decimal.

Conversión de hexadecimal a binario:

El análisis es similar al caso anterior en octal, con la diferencia que para lograr obtener dígitos en hexa se deben agrupar de a cuatro bits para convertir un formato binario en base 16.

Si faltan bits en la parte entera se agregan ceros adelante y si faltan bits en la parte fraccional se agregan ceros al final.

Ejemplo: Convertir el número binario del ejemplo anterior a hexadecimal:
El número es (1101100,0101111) y agrupando tendremos:

0 1 1 0	1 1 0 0	,0 1 0 1	1 1 1 **0**
6	C	5	E

6C,5E en hexadecimal representa al número binario 1101100,0101111.

4 - Representación de números binarios con signo.

En el sistema decimal, un número con signo se representa con un símbolo delante del mismo el cual puede ser el "+" para representar un número positivo ó el "-" para representar a uno negativo.

Esta práctica es común en nuestra escritura cotidiana y la empleamos habitualmente para realizar operaciones matemáticas.

En el sistema binario, en cambio, la idea es diferente.

Se vió que la intención de representar números en una base de sólo dos símbolos por dígito era para poder implementar artificialmente un sistema que pudiera tener la habilidad de realizar cálculos.

Por tal motivo habiendo solo bits ó digitos binarios, con la condición "0" ó "1", se defiine el signo como un bit el cual con dos estados puede representar la condición de "negativo"ó "positivo".

De aquí aparecen tres métodos de representación de números binarios con signo:

Signo y Módulo.

Complemento a la base disminuída ó Complemento a 1 (Ca1).

Complemento a la base ó Complemento a 2 (Ca2).

Cada uno de estos métodos tienen la misma forma de representación si el número es **positivo**, es decir, si tenemos un número que sabemos que es positivo, su módulo vale lo mismo sea cual fuere el sistema de representación (Signo y Módulo, Ca1 ó Ca2) con un bit más igual a "0" en la posición mas significativa, es decir, a la izquierda.

Ejemplo: 0111 corresponde al número +7, 01110 al +14, etc..

Signo y Módulo:

Por convención un número positivo se distingue por tener su bit mas significativo (MSB), en "0".

Un número negativo se puede identificar por tener un "1" en su bit mas significativo.

Esto es válido para los tres sistemas de representación con signo.

Dados un número de n bits, entonces se tiene que un bit se reserva para el signo, mientras que los (n - 1) restantes representarán el módulo de dicho número.
Por lo tanto mientras un número sin signo puede valer como máximo 2^n -1, aquí tenemos que al perder un bit en el signo, el máximo módulo será: 2^(n-1) - 1.

Resumiendo:

Máximo número positivo a representar con n bits: +2^(n - 1) - 1.

Máximo número negativo a representar con n bits: -2^(n - 1) - 1.

Ejemplo: Con 5 bits un número sin signo puede ir de 0 a 31, mientras que con signo el máximo módulo es 15, ya que de los 32 números posibles se usan 15 para representar números negativos , 15 para representar números positivos y los otros dos para expresar el número 0, que puede ser +0 ó - 0.

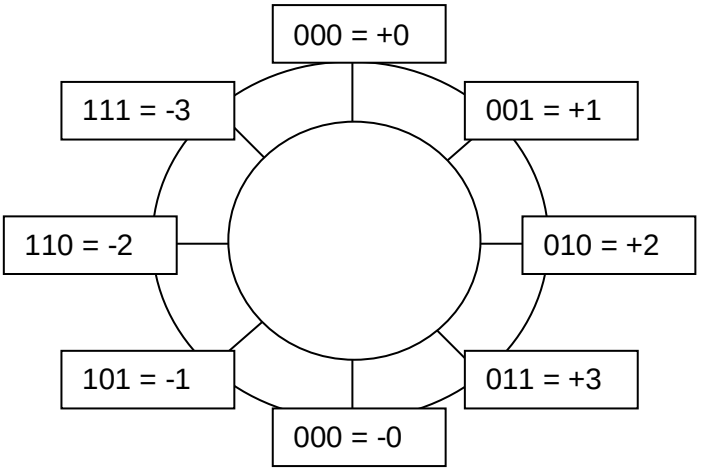
Con 5 bits tendremos entonces:

11111 = - 15
11110 = - 14
11101 = - 13
.....
10001 = - 1
10000 = - 0
00000= + 0
00001 = + 1
.....
01111 = +15

En este sistema un número positivo ó negativo del mismo módulo se diferencian sólo en el bit mas significativo.

Por ejemplo los números 100110 y 000110 corresponden a los números -6 y +6, respectivamente expresados con 6 bits.

En la siguiente figura vemos una representación gráfica de los posibles números que se pueden representar con 3 bits.



Se ve como números del mismo módulo pero diferente signo se encuentran en extremos opuestos de una misma línea que pasa por el centro del disco.

Como desventajas de este método de representación tenemos:

Doble representación del cero.

Dificultad para implementar operaciones de suma y resta.

Complemento a la base disminuída ó complemento a 1(Ca1):

Este sistema de representación, tiene la misma regla que el de signo y módulo para los números positivos.

En cambio para los negativos usa la siguiente definición:

Dado un número N (sea positivo ó negativo) se llama complemento N*1 a aquél número de igual módulo que N pero signo opuesto.

Para hallarlo se debe emplear la siguiente expresión:

N*1 = 2^n - 1 - N

Ejemplo 1: Dado el número +14 en decimal, expresar su complemento con 6 bits.

+14 = 001110 con 6 bits.

2^n = 2^6 = 1000000

2^n - 1 = 2^6 - 1 = 1000000 - 1 = 111111

N*1 = 2^6 - 1 - 14 = 111111 - 001110

1	1	1	1	1	1
0	0	1	1	1	0
1	1	0	0	0	1

Ejemplo 2: Dado el número -12 en decimal, expresar su complemento con 6 bits.

-12 = 110011 con 6 bits.

2^n = 2^6 = 1000000

2^n - 1 = 2^6 - 1 = 1000000 - 1 = 111111

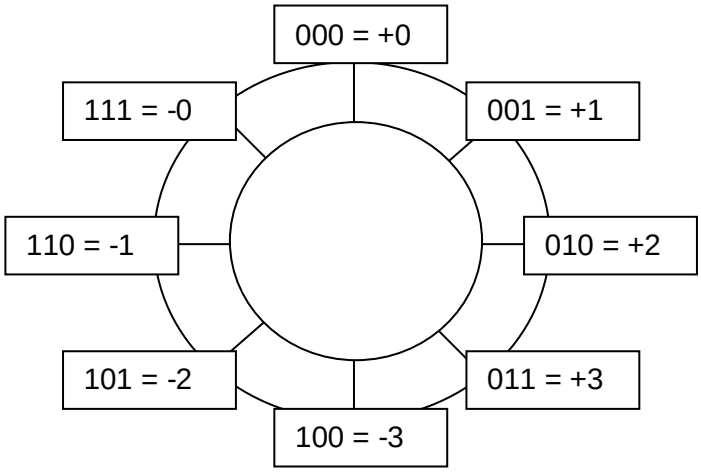
N*1 = 2^6 - 1 - (- 12) = 111111 - 110011

1	1	1	1	1	1
1	1	0	0	1	1
0	0	1	1	0	0

En este ejemplo partimos del conocimiento de como se representa un número negativo.

En la realidad para hallarlo debemos primero partir del número positivo y luego complementarlo.

La siguiente representación gráfica muestra la ubicación relativa de los números que se pueden representar para el caso de tres bits.



En él se observa la doble representación del cero (000 y 111), además de una propiedad particular por la disposición de los números que es la siguiente:

Dado un número positivo o negativo de n bits, su complemento tiene los n bits todos invertidos.

En el ejemplo de n = 3, +1 = 001 y - 1 = 110, + 2 = 010 y - 2 = 101, etc.

Por lo tanto aquí obtenemos una regla simple de conversión de un número binario representado por el método de complemento a 1 de un dado signo a otro de igual módulo pero signo opuesto:

Dado el número N, su complemento N*1 tiene todos sus bits invertidos.

Como en el caso anterior el máximo número positivo y negativo a representar con n bits viene dado por:

Máximo número positivo a representar con n bits: +2^(n - 1) - 1.

Máximo número negativo a representar con n bits: -2^(n - 1) - 1.

Este sistema de representación tiene como ventaja que es mas fácil de implementar operaciones de suma y resta que con el de signo y módulo, aunque sigue teniendo doble representación del cero.

El pasaje de un número a otro de igual módulo y distinto signo es muy fácil de implementar por una unidad aritmético lógica UAL de la cual dispone cualquier microprocesador.

Complemento a la base ó complemento a 2 (Ca2):

En este método como en los anteriores, los números positivos se representan de la misma manera.

En cambio los negativos tienen una convención diferente.

Dado un número N, su complemento, N*2, será:

$$N*2 = 2^n - N$$

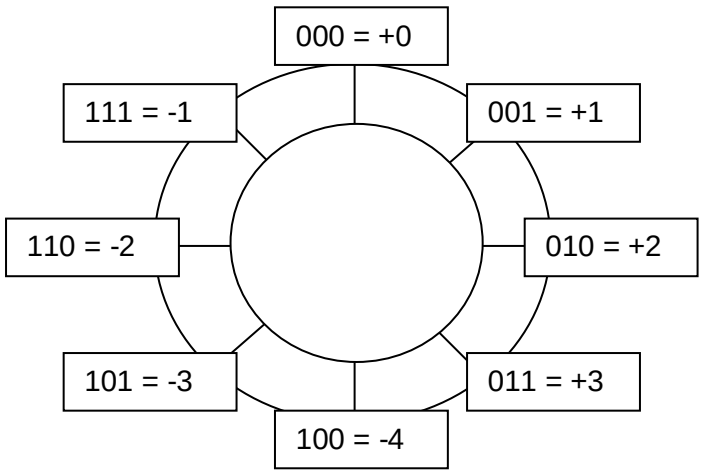
Se nota que la diferencia entre Ca1 y Ca2 es el 1 que en este último caso no se resta a N.

Al hacer esta omisión se logran los siguientes beneficios:

Sólo hay una sola representación del cero.

Se representa un número negativo mas al no tener el (- 0).

En la siguiente gráfica podemos ver esto en el caso de 3 bits.



En lugar de - 0 ahora podemos representar el - 4.

Por lo tanto los rangos de representación quedarán:

Máximo número positivo a representar con n bits: +2^(n - 1) - 1.

Máximo número negativo a representar con n bits: -2^(n - 1).

Ejemplo 1: Dado +22 hallar - 22 con 8 bits en CA2.

N = +22 = 00010110

2^n = 2^10 = 100000000

N*1 = 2^n - N = 2^8 - 00010110

1	0	0	0	0	0	0	0	0
	0	0	0	1	0	1	1	0
	1	1	1	0	1	0	1	0

Como regla nemotécnica se puede establecer que:

Dado un número N su complemento a 2 se puede hallar de la siguiente forma:
Se parte de derecha a izquierda. Se copian los bits como están hasta que aparezca el primer "1" el cual también se copia igual. A partir de allí, se invierten todos los bits.

El pasaje de un número negativo de Ca1 a Ca2 es inmediato, ya que las definiciones tenemos que:

$$N*2 = N*1 + 1$$

5 – Código de Gray

El código de Gray, es un código formado por dígitos binarios que sigue una regla particular para la representación de números decimales

Como ejemplo, tenemos la siguiente tabla para códigos Gray de 1, 2 y 3 bits respectivamente:

Valor decimal	Código Gray de 1 bit	Código Gray de 2 bits	Código Gray de 3 bits
0	0	0**0**	0**00**
1	1	0**1**	0**01**
2	-	11	0**11**
3	-	10	0**10**
4	-	-	110
5	-	-	111
6	-	-	101
7	-	-	100

Se puede notar que tienen una forma particular para generarlos.
El mas elemental es el de un solo bit, donde para el número decimal “0” le corresponde el “0” en código Gray y el “1” decimal al “1” en el mismo código.

De allí en mas, para generar un código Gray de mayor cantidad de bits, se hace un proceso, que es el que se explicará a continuación:

Para generar un código de 2 bits, se copian los dígitos “0” y “1” de la tabla para el código de un bit en forma vertical (ver tabla anterior los bits en “negritas”) y luego se vuelven a copiar esos mismos bits hacia abajo pero de tal manera que se rebaten los mismos, es decir, se vuelven a copiar como un espejo (ver bits en rojo). Para este caso, eso implica que se pone un “1” y luego un “0”.
Quedan entonces cuatro números de un bit que se arriba hacia abajo son 0110, es decir,

0
1
1
0

El último paso es completar una segunda columna (a la izquierda) de estos números, llenando la primera mitad con “cero” y la segunda mitad con “uno”, obteniendo:

00
01
11
10

De la misma manera, en base a este código de 2 bits , se puede generar el de 3 bits, copiando en principio los 4 números de 2 bits cada uno, luego escribir otros 4 pero rebatidos como si fuera un espejo.

00
01
11
10
10
11
01
00

Al tener los 8 números, colocar otra columna a la izquierda, en este caso primero con 4 “ceros” y luego otros 4 “unos”

000
001
011
010
110
111
101
100

Si se quisiera hacer uno de 4 bits, se parte desde uno de 3 bits, realizando el mismo procedimiento.

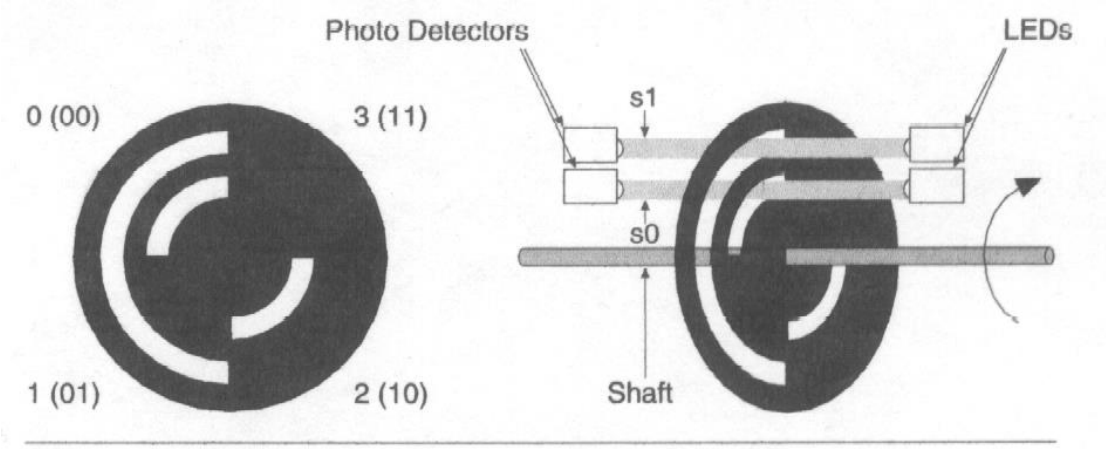
Ejemplo de aplicación del Código de Gray

El caso clásico de utilización del código de Gray, es para la decodificación de posición angular de un shaft encoder.

Un shaft encoder es un dispositivo que generalmente está asociado al eje de giro de algún aparato, y que traduce una determinada posición angular en un número.

Por ejemplo, si el shaft encoder es de 8 bits, podrá darnos un número con 256 valores diferentes en una vuelta completa del eje.

En la siguiente figura, tenemos un dibujo que esquematiza un shaft encoder, pero en principio con códificación binaria de la posición angular, en este caso, de sólo dos bits, a fin de explicar como funciona.

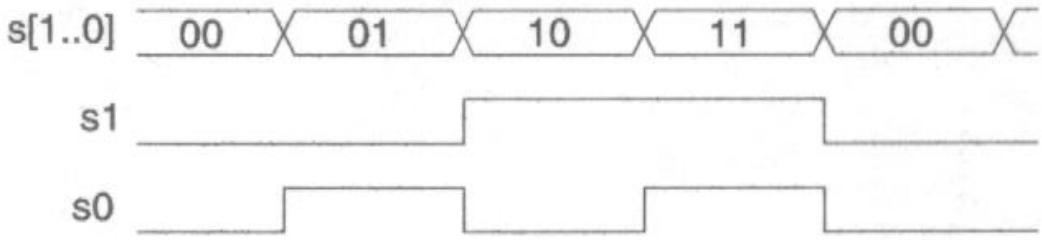


Está constituído por un disco ranurado, solidario a un eje, y dos pares de emisor – detector ópticos, de tal manera que cada juego está alineado con el eje, pero a un radio diferente uno del otro.

Como se observa en la figura, el disco está convenientemente ranurado, en dos sectores radiales, a fin de que cada un cuarto de vuelta, se tenga una de cuatro combinaciones posibles, las cuales están asociadas con las salidas de ambos detectores, S1 y S0..

Se muestra el caso donde justo ambos detectores reciben luz de sus correspondientes emisores, lo que se traduce aquí como “00”.

Si gitamos el dico a una velocidad constante, obtendríamos las siguientes formas de onda en los senosres S0 y S1:

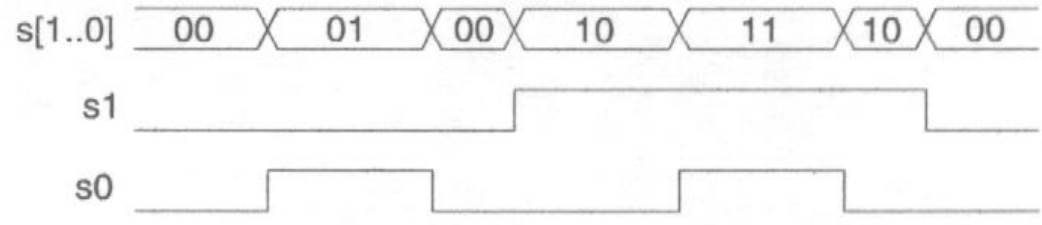


Hasta aquí parecería que un código binario sería suficiente para lograr la codificación de posición angular a binario.

Pero el problema surge cuando por ejemplo ocurre alguna desalineación en uno o ambos detectores, lo cual puede traducirse en errores al interpretar las señales recibidas.

esto significa que una desalineación puede generar que ambos sensores no cambien en el mismo instante cuando el disco gira ya que uno puede estar mas adelantado o atrasado que el otro. Esto entonces genera temporariamente, códigos erróneos que pueden generar problemas serios de control, etc..

En la siguiente figura vemos el caso en que normalmente deberíamos recibir las señales:
S1 S0 => 00 01 10 11 00
para una vuelta completa del disco y sin embargo recibimos la secuencia:
S1 S0 => 00 01 00 10 11 10 00.



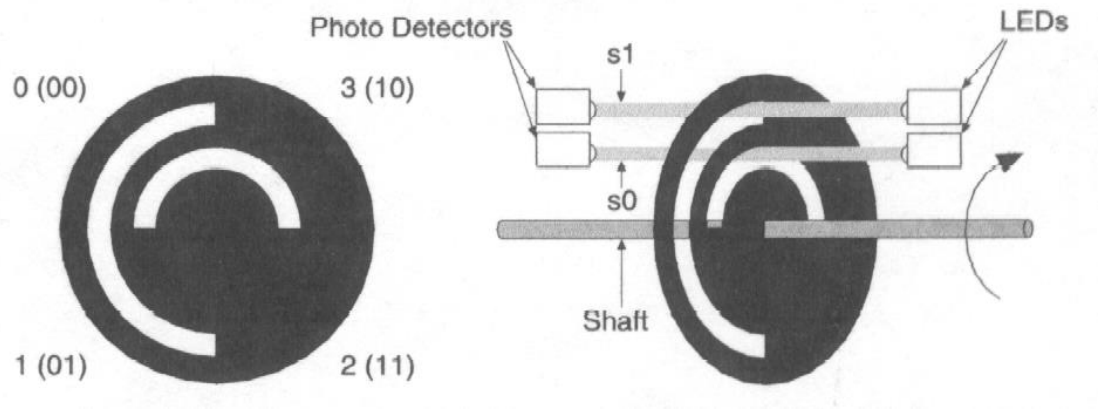
Vemos que aparecen códigos que no debieran aparecer y esto es debido a que el sensor S0 está muy adelantado en el disco y cambia primero que S1, por eso es que en vez de pasar desde S1 S0 = 01 directamente a 10, pasamos antes por 00, ya que S0 recibió el nuevo bit “0” pero S1 todavía sigue viendo el “0” anterior.

Lo mismo ocurre al querer pasar de 11 a 00, donde hay un estado adicional, el 10, dado que S0 responde primero.

Este problema parecería fácil solucionarlo mecánicamente ya que se trata de sólo 2 bits. En realidad los shaft encoder son de 8 o mas bits, es decir, 8 o mas juegos de emisor-detector, haciedo que el problema de desalineación se haga mas complejo de resolver.

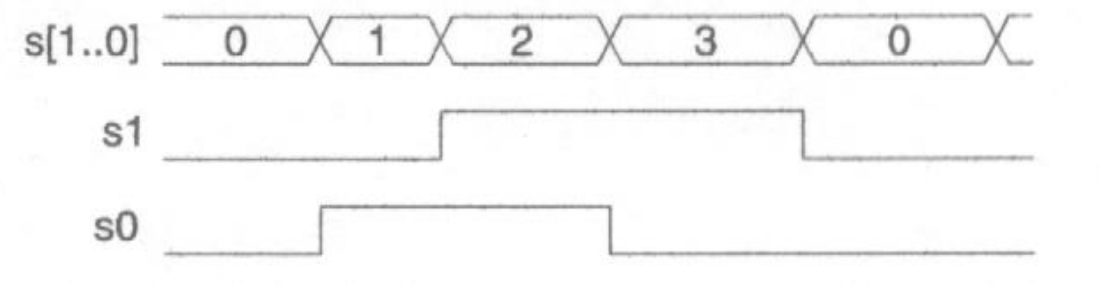
Una solución a esto, es emplear el código de Gray en vez del binario.

En la siguiente figura, vemos lo mismo que antes pero implementado con dicho código.



Aquí, una vuelta completa del disco está codificada como: 00 01 11 10.

La ventaja de sto es que aún para el caso de desalineación como el presentado anteriormente, las salidas siguen indicando correctamente la posición



Esto se debe a que simplemente no existen cambios simultáneos de estado lógico en las salidas (ver esquema del disco ranurado).

Cualquier desalineación en uno de los sensores sólo haría cambiar un poco antes o un poco después el código, pero nada mas que eso.

6 - Bibliografía.

1 Circuitos Digitales y Microprocesadores. Herbet Taub.
Editorial Mc Graw Hill.

2 Sistemas Digitales: Principios y aplicaciones. Ronald Tocci.
Editorial Prentice - Hall.

3 Electrónica Digital. James W. Bignell.
Editorial CECSA.

4 Nota de aplicación de Altera Corporation, A – AN – 083 -01