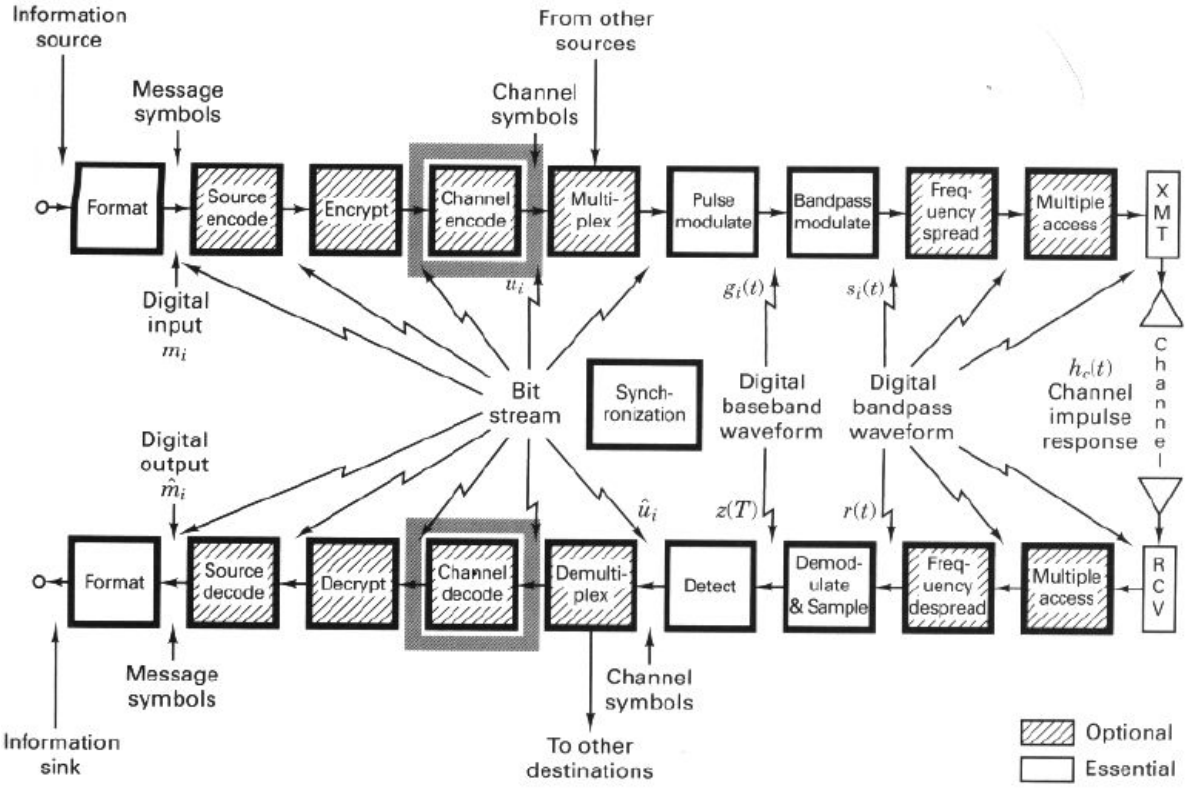
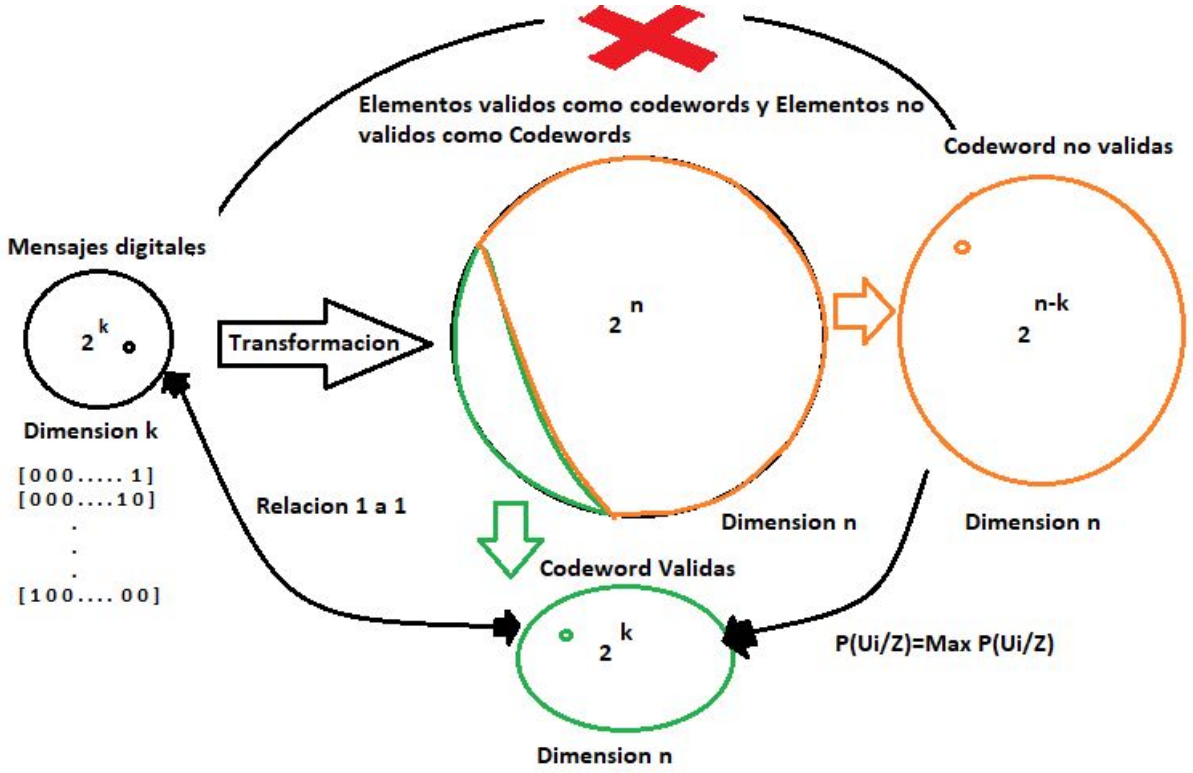
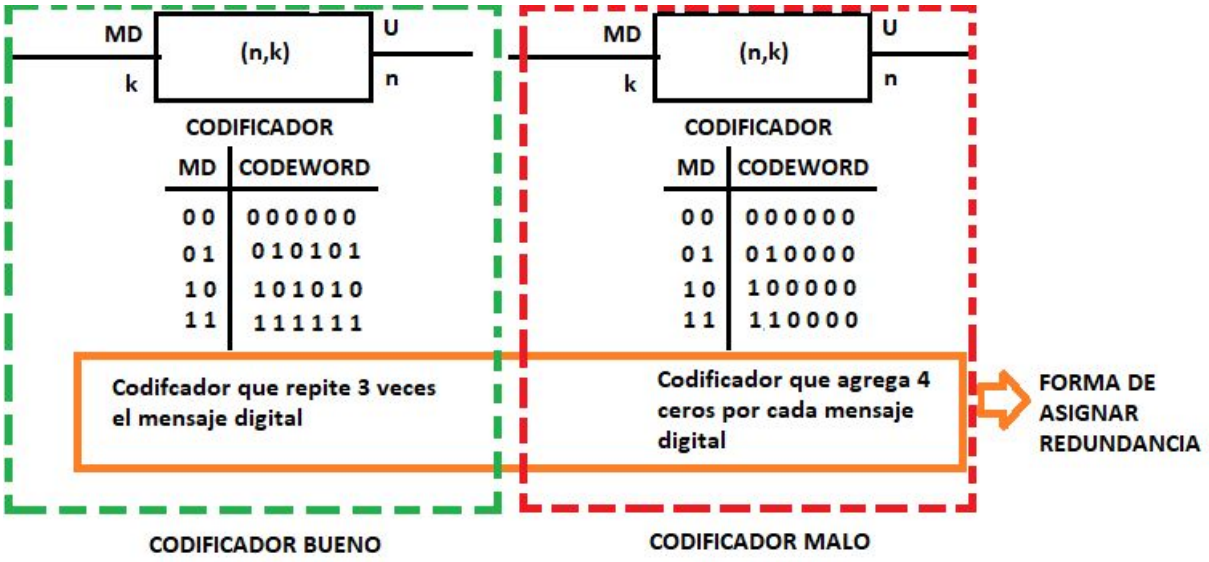
Codificación de Canal



¿Qué es la codificación de canal? y ¿Para qué sirve? La codificación es un proceso que consta del mapeo uno a uno de elementos de un espacio de dimensión menor (mensajes digitales) en un subespacio (codeword) dentro de un espacio de dimensión mayor. Esto significa la creación de una nueva entidad denominada “codeword” a partir de los mensajes digitales, la cual tiene mejor comportamiento frente al ruido gracias a la implementación de más bits, los cuales se conocen como redundancia.



El concepto fuerte en codificación de canal está asociado a la respuesta de las siguiente pregunta ¿Si un proceso introduce redundancia en los mensajes digitales para conformar codewords, es definido siempre como codificación? **NO**. Hay que conocer qué técnicas se deben elegir para que la conversión del mensaje digital a codeword sea lo más eficiente posible y realmente sirva. ¿Qué significa esto? **Que las codeword entre ellas deben ser los más distintas posibles**. De esta manera, los procesos de codificación transforman los mensajes digitales en codeword que poseen redundancia para que cumplan con lo establecido anteriormente. Esto permite mejorar la performance del sistema.

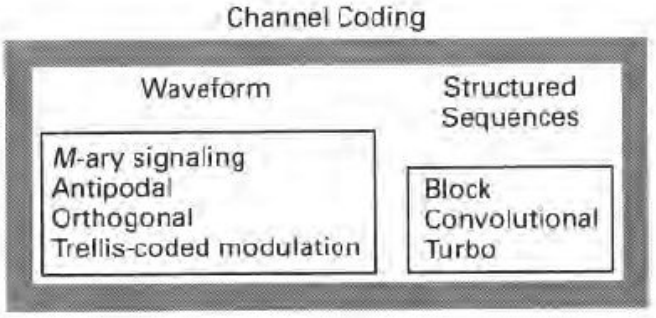


CONCLUSION

NO TODA TRANSFORMACION DE UNA DIMENSION MENOR A UNA DIMENSION MAYOR PUEDE SER CONSIDERADA CODIFICACION.

Pero esta implementación como se puede esperar tienen un costo, y si padre, el agregado de estos bits de redundancia en búsqueda de **aumentar la performance del sistema, se paga en tiempo o en ancho de banda**. Esto se debe a que si aumentamos la cantidad de bits a transmitir, a la misma tasa que antes de codificar, deberemos pagar el costo de esperar más tiempo para poder tomar decisiones. Otra de las alternativas, es pagar con ancho de banda, ya que para transmitir el mensaje más su codificación en el mismo tiempo que antes, necesitaremos si o si aumentar la tasa de bits del codificador para cumplir con los tiempos deseados.

La implementación de codificación de canal se dividirá en 2 grandes grupos, el primero a desarrollar será el correspondiente a la de *forma de onda* o diseño de señales, esta contempla a la señalización M-aria, antipodal, ortogonal y modulación por codificación de Trellis. El otro, es el de *Secuencia Estructurada* o redundancia estructurada, que contempla al Código de Bloque tanto Lineal como Cíclico, al Codificador Convolucional y por último el Turbo Código.



Codificación por forma de onda

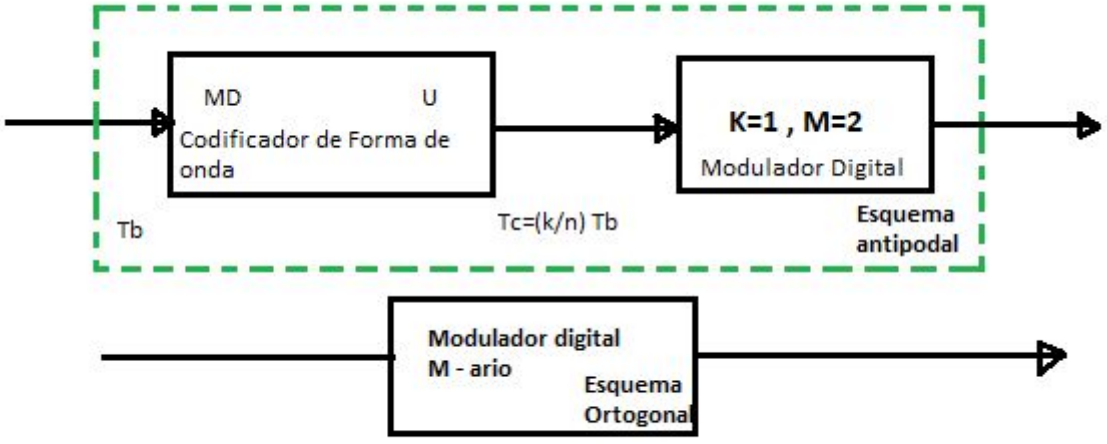
Este tipo de codificador propone la transformación de la forma de onda en una " forma de onda mejor" para hacer el proceso de detección menos sujeto a errores. ¿Forma de onda mejor? Significa que cada una de las waveform sea lo más diferente una de la otra. Con

señalización M-aria, los “k” bits de entrada correspondientes a los mensajes digitales le dan instrucción al modulador para producir uno de los $M = 2^k$ formas de ondas. En el caso binario (k=1), para moduladores en banda de paso, en Esquemas PSK se eligen señales antipodales y FSK señales ortogonales. Para Moduladores en banda base se prefieren aquellos esquemas que contengan señales antipodales. Es decir, cada esquema busca que la correlación cruzada entre las waveform del set de señales sea lo más pequeño posible.

$$Z_{ij} = \frac{1}{E} \int_0^{Ts} S_i(t)\, S_j(t)\, dt$$

En ella se puede ver las formas de ondas $S_i(t)$ y $S_j(t)$, donde i,j=1,...,M y M es el tamaño del set de formas de ondas. El valor más pequeño de los coeficientes de correlación cruzada ocurre cuando las señales son anticorrelacionadas (z_{ij}=-1), sin embargo, esto solo puede ser logrado cuando el número de símbolos en el set es dos (M=2) y los símbolos son antipodales. En general, es posible hacer todos los coeficientes de correlación cruzada iguales a cero cuando M es mayor que dos. Esto es lo que propone la codificación de forma de onda.

En conclusión, la codificación de forma de onda es un proceso que tiene como propósito la transformación de los mensajes digitales en codewords cuyas características en conjunto con las de un modulador digital permitan una correlación cruzada igual a 0 entre cada una de las waveform. Es un proceso que lleva todo esquema de modulacion y demodulacion a una señalización ortogonal. La mayoría de las veces suele confundirse este tipo de codificación con el modulador pero se puede disociar dicha relación si planteamos un análisis distinto.



Cada forma de onda en el set $\{s_i(t)\}$ surge de una codeword que puede pensarse como una secuencia de pulsos. Donde cada pulso es designado con un nivel ± 1, que a su vez representa al 1 o 0 binario.

Cuando el set es representado de esta manera, la ecuación de la correlación cruzada puede ser simplificada por expresar que $\{s_i(t)\}$ constituye un set ortogonal si y solo sí:

$$z_{ij} = \frac{numero \cdot de \cdot digitos \cdot coincidentes - numero \cdot de \cdot digitos \cdot no \cdot coincidentes}{numero \cdot total \cdot de \cdot digitos \cdot en \cdot la \cdot \text{sec}uencia} = \begin{cases} 1 & \text{Para i=j} \\ 0 & \text{Caso contrario} \end{cases}$$

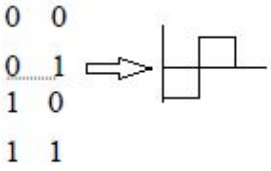
Códigos Ortogonales

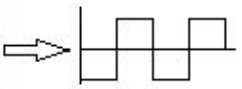
Un set de datos de un bit, que puede ser del tipo antipodal y mediante la utilización de una única base, puede ser transformado usando codewords ortogonal de dos dígitos cada una, descrito por las filas de la matriz conocida como H_1.

set de datos	set de codewords ortogonal
0	$H_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$
1	

Para la codificación de un set de dos bits de datos y verificando el cumplimiento de $Z_{ij}=1$ para i=j, en caso contrario $Z_{ij}=0$. Se extiende el set H_1 vertical y horizontalmente, en donde el cuadrante de abajo a la derecha irá el complemento de H_1, de la siguiente manera:

set de datos	set de codewords ortogonal
0 0 0 1 1 0 1 1	$H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & \overline{H_1} \end{bmatrix} = \left[\begin{array}{cc:cc} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right]$





Como se observa el set de datos consistía en 2 bits pero el set de codewords, el cual es el que se enviará, consta de 4 bits por cada dato, o sea el doble de bits, como se puede ver la comparación para el dato “01” y su correspondiente codeword “0101”. El beneficio que trae este aumento es que cada una de las codewords tiene una diferencia de 2 bits entre quienes componen al set, haciéndola más diferente con el resto, lo cual será un beneficio a la hora de la obtención de los datos en el receptor pero traerá como costo un aumento en el ancho de banda o en el tiempo de decisión según el costo por el cual se opte por afrontar.
Para la construcción de un set ortogonal para un set de de datos de 3 bits, tendremos:

set de datos	set de codewords ortogonal
0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	$H_3 = \begin{bmatrix} H_2 & H_2 \\ H_2 & \overline{H_2} \end{bmatrix} = \left[\begin{array}{cccc:cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$

Como se observa, a medida que aumentemos los k bits que componen a un dato, generamos en el set de codewords ortogonales lo que se conoce como *matriz Hadamard* una matriz H_k con dimensiones 2^k x 2^k, que se encuentra compuesta internamente por 4 H_{k-1}

$$\mathrm{H}_k = \begin{bmatrix} \mathrm{H}_{k-1} & \mathrm{H}_{k-1} \\ \mathrm{H}_{k-1} & \overline{\mathrm{H}_{k-1}} \end{bmatrix}$$

Se debe observar en la Matriz de Hadamard que entre 1 par cual sea de filas o codewords tienen igual cantidad de dígitos coincidentes como no coincidentes. Dado el ejemplo de H_3 el cual tiene una matriz con 8 componentes por fila, hay 4 bits coincidentes entre 2 filas y 4 bits no coincidentes. De esta manera se cumple que Z_{ij}=0 para $i \neq j$ y además de que cada uno de los set es ortogonal.
La codificación de la forma de onda con un set de señales construidas ortogonalmente, en combinación con detección coherente, produce exactamente la misma mejora en performance de P_B que la señalización M-aria con un formato de modulación ortogonal.
Para señales ortogonales de igual energía igualmente probable, la probabilidad del error de la codeword (símbolo), P_E, puede ser limitada superiormente como:

$$P_E(M) \leq (M-1)Q\left(\sqrt{\frac{E_s}{N_0}}\right)$$

En donde el set de codeword M (2^k), k es el número de bit por codeword y considerando que E_s=k*E_b es la energía por codeword para un M fijo. Utilizando la relación entre P_E y P_B, que era:

$$\frac{P_B(k)}{P_E(k)} = \frac{2^{k-1}}{2^k - 1} \qquad \text{o} \qquad \frac{P_B(M)}{P_E(M)} = \frac{M/2}{(M-1)}$$

Combinando esta ecuación con la de la probabilidad del error de codeword, se puede obtener la siguiente probabilidad de error del bit:

$$P_B(k) \leq \left(2^{k-1}\right)Q\left(\sqrt{\frac{kE_b}{N_0}}\right) \qquad \text{o} \qquad p_B(M) \leq \frac{M}{2}Q\left(\sqrt{\frac{E_s}{N_0}}\right)$$

Códigos biortogonales

Un set de señales biortogonales de un total de M señales o codewords puede ser obtenido a partir de un set ortogonal de M/2 señales sumado al negado de cada una de las codeword.

$$\mathbf{B}_k = \begin{bmatrix} \mathbf{H}_{k-1} \\ \overline{\mathbf{H}_{k-1}} \end{bmatrix}$$

Por ejemplo, un set de datos de tres bit pueden ser transformados dentro de un set de codeword biortogonales como sigue:

set de datos			set de codewords biortogonal			
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	0	1	1
0	1	1	0	1	1	0
1	0	0	1	1	1	1
1	0	1	1	0	1	0
1	1	0	1	1	0	0
1	1	1	1	0	0	1

$$B_3 = \begin{bmatrix} H_2 \\ \overline{H_2} \end{bmatrix}$$

Ortogonal

Antipodal

La ventaja de este tipo de código es que consiste en una combinación de codewords ortogonales y antipodales. Con respecto z_{ij}, los códigos biortogonales pueden ser caracterizados como:

$$z_{ij} = \begin{cases} 1 & para \cdot i = j \\ -1 & para \cdot i = j, |i-j| = \dfrac{M}{2} \\ 0 & para \cdot i \neq j, |i-j| \neq \dfrac{M}{2} \end{cases}$$

El set biortogonal consiste en realidad de dos set ortogonales tal que cada codeword en uno de los set tiene su codeword antipodal en el otro set.

En definitiva ¿Por qué surgen los códigos biortogonales? En primer lugar requieren la mitad de bits que los códigos ortogonales, entonces los requerimientos de ancho de banda se reducen a la mitad y además, el distanciamiento entre codewords en algunos casos es mayor, lo que permite mejorar aún más la performance del sistema frente a los códigos ortogonales. **En conclusión, son mejores los códigos biortogonales en todo sentido.**

Para señales biortogonales de igual energía igualmente probables, la probabilidad del error de símbolo producto de haber aplicado codificación puede ser limitada superiormente como sigue:

$$P_E(M) \leq (M-2)Q\left(\sqrt{\frac{E_s}{N_0}}\right) + Q\left(\sqrt{\frac{2E_s}{N_0}}\right)$$

La función de la probabilidad de error de bit es una función complicada de la probabilidad de error de símbolo, por lo tanto podemos aproximarla mediante la relación detallada para señalización ortogonal cuando el valor de M tendía a infinito, lo cual nos decía que la probabilidad de error de bit era la mitad de la probabilidad de error de símbolo.

$$P_B(M) \approx \frac{P_E(M)}{2}$$

Esta aproximación es bastante buena para M>8, resultando:

$$P_B(M) \leq \frac{1}{2}\left[(M-2)Q\left(\sqrt{\frac{E_s}{N_0}}\right) + Q\left(\sqrt{\frac{2E_s}{N_0}}\right)\right]$$

Códigos Transorthogonal (Simplex)

Partiendo desde un código ortogonal, si a este se le elimina el primer dígito de cada codeword o lo que sería equivalente a la primer columna de todo el set, obtendremos lo que se denomina Código Transorthogonal o Simplex.

$$z_{ij} = \begin{cases} 1 & para \cdot i = j \\ \dfrac{-1}{M-1} & para \cdot i \neq j \end{cases}$$

Se puede observar que para Z_{ij} con $i \neq j$ se obtiene -1/(M-1), el -1 en el denominador se debe a que al eliminar la primer columna del código ortogonal original sacamos un "número coincidente" para el cálculo de Z_{ij}. Esto hace que siempre la cantidad de números no coincidentes sea mayor en 1 por lo cual Z_{ij} da negativo.

Esta diferencia de códigos simplex con respecto al resto va a ser significante siempre y cuando el M sea de un valor chico haciendo que este tipo de código es el que requiere la mínima E_S/N_o para una razón de error de símbolo especificada.

Implementación de sistemas de comunicación digital con codificación de forma de onda

La codificación de forma de onda consiste en un proceso de transformación de una secuencia de bits (mensajes digitales) en una nueva secuencia (codeword) de mayor longitud y con mejores características de distancia. Cada mensaje digital de "k" bits, se convierte en alguna de las posibles codeword establecidas por la matriz Hadamard en el caso de códigos ortogonales ó simplemente en algunos de los códigos de la matriz asociada a biortogonal dependiendo del método de codificación elegido. En el primer caso, la cantidad de bits que conforman a las codeword es M y en el segundo caso es M/2.

De las codeword (M códigos de longitud M) que reemplazan a los k bits de mensaje, la secuencia escogida modula una portadora que por lo general (existe también la posibilidad de utilizar una modulación digital banda base PCM) usa PSK binario, tal que la fase $\left(\phi_i = 0 \text{ ó } \pi\right)$ de la portadora cambia durante cada tiempo de bit codificado $0 \leq t \leq T_c$.

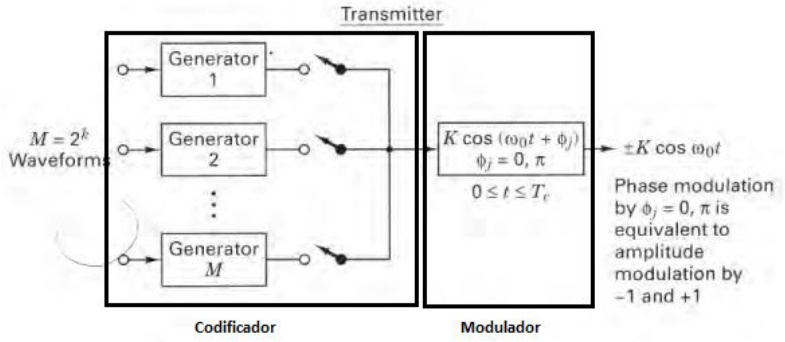
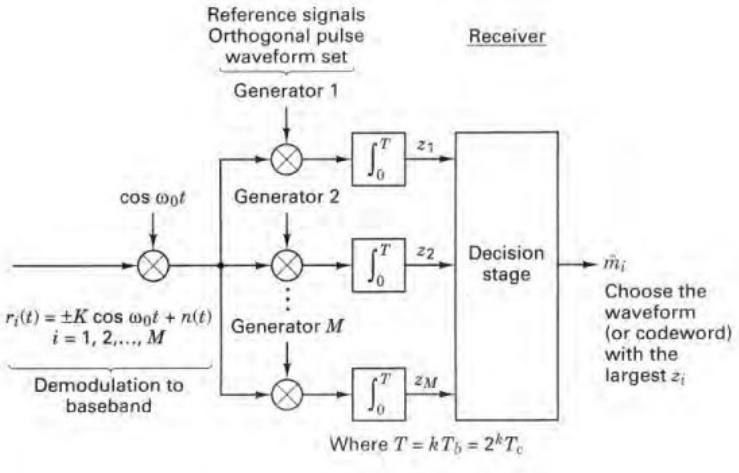
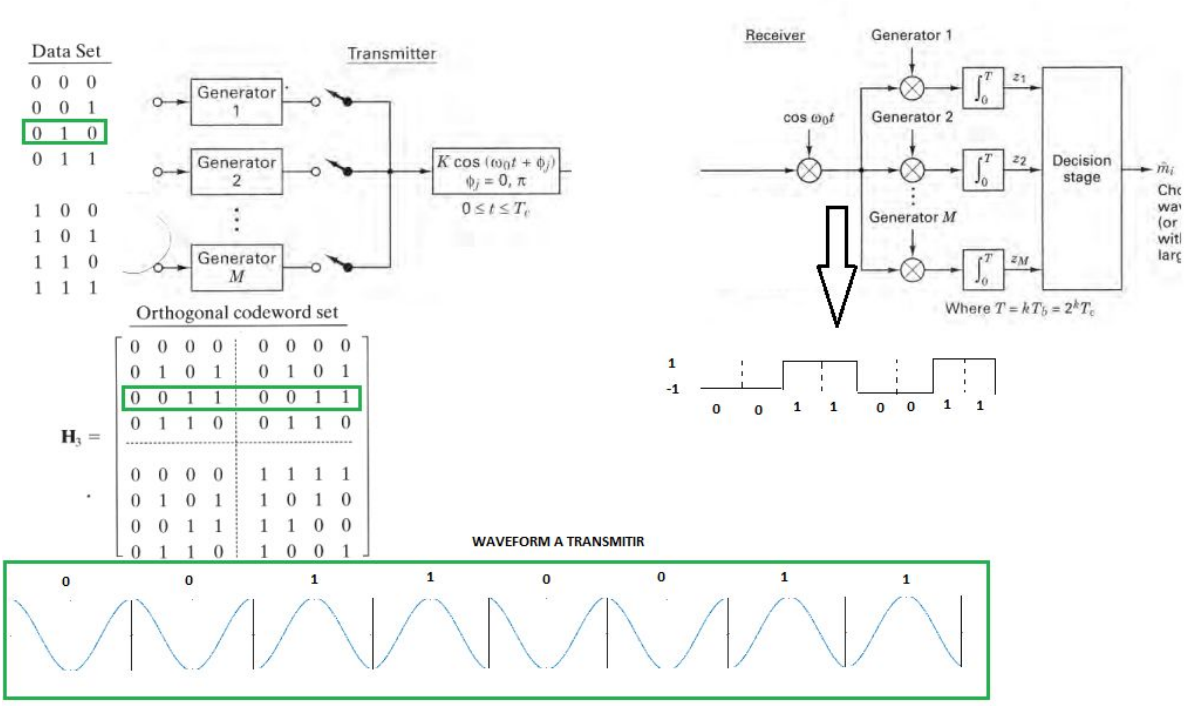


Figure 6.4 Waveform-encoded system (transmitter).

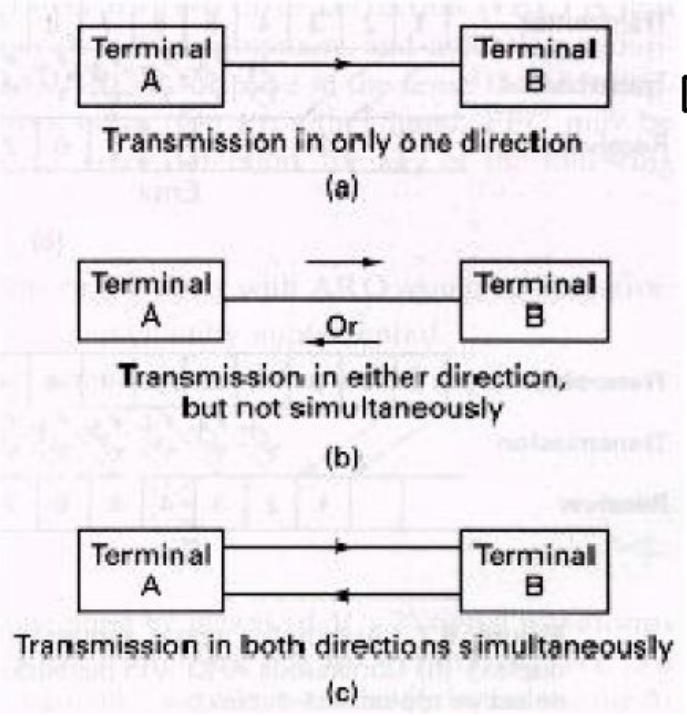
En el receptor la señal es demodulada en banda base y aplicada a M correladores (o filtros acoplados). La correlación es realizada durante la duración de la codeword, lo cual puede expresarse como M.Tc = k.Tb. Esto se debe a que en los sistemas de tiempo real los mensajes digitales no pueden ser retrasados, entonces el tiempo de la codeword debe ser igual al tiempo de duración del mensaje digital.
En otras palabras, los bits de código o la codificación de pulsos (que es modulada en PSK) deben moverse a una razón M/k mas rápido que los bits de mensaje. Para tal codificación ortogonal de formas de onda y canal AWGN, el valor esperado a la salida de cada correlador, a un tiempo T, es cero, excepto para el correlador correspondiente a la codeword transmitida.





Tipos de control de error

Existen dos formas básicas de utilizar a la redundancia como controlador de errores. La primer forma es conocida como detección de errores y retransmisión, utilizando bits de paridad adheridos a los bits de datos con el fin de detectar que se está produciendo un error. En la etapa de recepción al percibir un error vía este bit de paridad, el receptor no trata de corregirlo sino que esta pide que se retransmita el dato. La otra forma y que necesita solo una conexión y en una única dirección es conocida como Corrección de Error Avanzado (FEC).
Con el modo de conexión por parte del tipo de control de error denominado FEC, se debe recordar los distintos tipos de conectividades que pueden existir o necesitar entre terminales de comunicación.



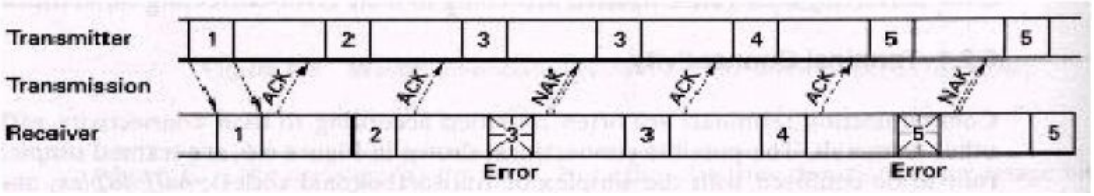
FEC

Detección de errores y retransmisión

En la figura se pueden ver 3 tipos distintos, el primer caso que cuenta con una única conexión y en un único sentido, el cual se conoce como Simplex y es el modo implementado para los FEC. Luego está la conexión de una única vía pero en ambos sentidos, conocido como Half Duplex y por ultimo esta el Full Duplex el cual cuenta con 2 vías de comunicación pero cada una tiene un único sentido. De esta manera el Half Duplex y Full Duplex pueden ser utilizado para el control de error que consta de detección de errores y retransmisión.

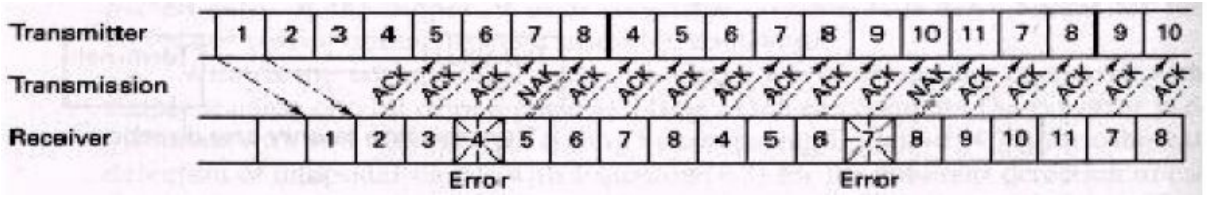
Requerimiento de Repeticion Automatica (Automatic Repeat Request ó ARQ: Automatic Request Query)

Cuando el control de error consiste solamente en detección del error, se necesita como se mencionó anteriormente poder detectar y pedir una retransmisión por lo que se utilizaran las conectividades, para un procedimiento, Half Duplex y en otros dos, Full Duplex. Estos 3 casos serán más o menos útiles dependiendo de la aplicación y/o lugar a implementar.
El procedimiento implementado en Half Duplex es conocido como ARQ Stop and Wait, al utilizar este tipo de conectividad se espera que sea el más lento (pero más sencillo en cantidad de componentes del sistema), ya que el transmisor espera por un ACK o confirmación para realizar la próxima transmisión, en caso de no recibirlo (NAK) y cumplirse el tiempo de espera, se envía el dato que no acusó recepción. Esto se podría observar comenzando desde la izquierda, de la siguiente manera:

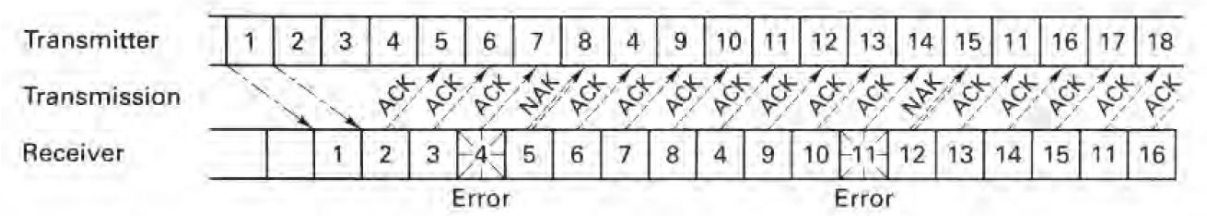


El segundo procedimiento es el ARQ Pullback, el cual utiliza Full Duplex, dando la posibilidad de por una vía hacer transmisión de datos y por la otra, desde el receptor,

acusar recepción o no al transmisor. De esta manera se puede observar que la transmisión ideal sera mas rapida que en el caso de ARQ Stop and Wait. En ARQ Pullback, en caso de un error, el receptor da aviso mediante un NAK, en donde el transmisor comenzará nuevamente a enviar los mensajes desde el último que no fue recibido, por lo cual los datos posteriores se enviaran otra vez sin importar que hayan sido bien recibidos. Este detalle es muy importante, ya que si la conexión entre terminales no es buena, se haría reenvío de datos en exceso perjudicando los tiempos de comunicación.



El último procedimiento es ARQ con repetición selectiva, nuevamente se implementa Full Duplex pero en este caso se soluciona el problema a la hora de reenvío de mensajes. En este caso si el receptor acusa un NAK el transmisor envía únicamente el mensaje asociado al NAK y luego continúa desde el último mensaje que había enviado agilizando al sistema de comunicación. Este tipo de procedimiento es el que se implementa para las redes de datos TCP.



Retomando las dos formas básicas que utilizan a la redundancia como controlador de errores, tenemos que los procedimientos ARQ tienen la ventaja sobre FEC de que la detección de error requiere un equipamiento de codificación más simple y mucho menos redundante que haga la corrección de errores.

Codificación por secuencia Estructurada

La codificación por secuencia estructurada es un proceso que tiene como propósito transformar la secuencia de datos en " secuencias mejores " insertando redundancia dentro de la fuente de datos, así la presencia de errores pueden ser detectados o corregidos. *En definitiva es como si se tratara de una receta para agregar redundancia.*
Para aplicar dicha técnica de codificación el canal debe cumplir con ciertas características:

- Independencia de errores (*Canal discreto sin memoria*).
- Probabilidad completa del evento es la suma de ambas probabilidades, la de equivocarme y la de acertar, es decir, la probabilidad de error es: P(Error) = 1- P(No Error) (*Canal Simétrico Binario*)
- Canal Gaussiano

→ <u>Canal discreto sin memoria:</u> Un canal de este tipo es caracterizado por una entrada alfanumérica discreta, una salida alfanumérica discreta y un conjunto de probabilidades condicionales P(j/i). Donde "i" son los símbolos que entrega el modulador al canal y "j" corresponden a los símbolos contaminados por ruido recibidos por el demodulador, de

manera tal que P(j/i) representa la probabilidad de recibir j dado que se transmitió i. Cada símbolo de salida depende solamente de la entrada correspondiente.
Lo importante de esto es que no existe probabilidades conjuntas, solo tenemos el producto de las probabilidades, no aparece el término correspondiente a la suma.

$$P(\mathbf{Z}|\mathbf{U}) = \prod_{m=1}^{N} P(z_m|u_m)$$

→ Canal binario Simétrico: Caso particular de canal discreto sin memoria. En este caso, la entrada y salida alfanumérica consiste de los elementos binarios (0 y 1). Las probabilidades condicionales son simétricas:

$$P(0|1) = P(1|0) = p$$

$$P(1|1) = P(0|0) = 1 - p$$

Las relaciones mostradas anteriormente reflejan lo que se conoce como probabilidad de transición. Esto es, dado que un símbolo es transmitido en el canal, cual es la probabilidad de que sea recibido erróneamente “p” y la probabilidad de que sea recibido correctamente es “(1-p)”.
Como la salida del demodulador consiste en elementos discretos 0 y 1, el demodulador es programado para tomar una decisión “Hard” sobre cada símbolo. Un sistema de código comúnmente usado consiste en datos modulados en BPSK y demodulacion por decisión “hard”.
¿Que significa Hard decision y Soft decision? En una hard, el codificador alimenta al sistema mediante bits de decisión del tipo hard. Esto le permite al demodulador, decidir si el símbolo transmitido por el canal fue un “1” o “0” binario codificado. Entonces, a la entrada del decodificador siempre observaremos ceros y unos binarios y mediante un arreglo de mapeo (Tabla de asignación de codeword a mensajes digitales) se obtiene el mensaje digital correspondiente. La existencia de un demapeador supone una decisión por parte del demodulador, una decisión realizada por un bloque "apartado" de las tareas de decodificación. En cambio, en una soft en lugar de hacer una construcción por bloques, se construye todo junto. Ya no se tiene una codificación y a continuación una modulación, o, más bien, una demodulación y luego una decodificación. Se trata de la "modulación codificada". La idea principal es que el decodificador interno, el continuo, no trabaje con símbolos, sino con números reales. Ya no hace falta que el demodulador, mediante el demapeador, decida la palabra de n símbolos que corresponde al punto de la constelación recibido, y basta con que entregue al decodificador un valor multi-escalado del bit recibido del canal (ejemplo: un valor del 0 al 7). Esta información provee al decodificador de una métrica que le permite tomar confianza para decidir sobre los símbolos recibidos.
Conclusión: Decisión Hard, es el demodulador que decide a partir de los símbolos recibidos cuales fueron los bits transmitidos y el decodificador detecta y corrige errores. En cambio, en decisión Soft, decide directamente el decodificador de acuerdo a una métrica entregada por el demodulador.

→Canal Gaussiano: Consideramos que el ruido presente en nuestro sistema de comunicación es AWGN. Es decir, una variable aleatoria con distribución gaussiana de media cero y varianza σ^2. La función de distribución de probabilidad (pdf) sobre la variable aleatoria recibida z, condicionada por sobre el símbolo U_k puede ser escrita como:

$$p(z|u_k) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(z-u_k)^2}{2\sigma^2}\right]$$

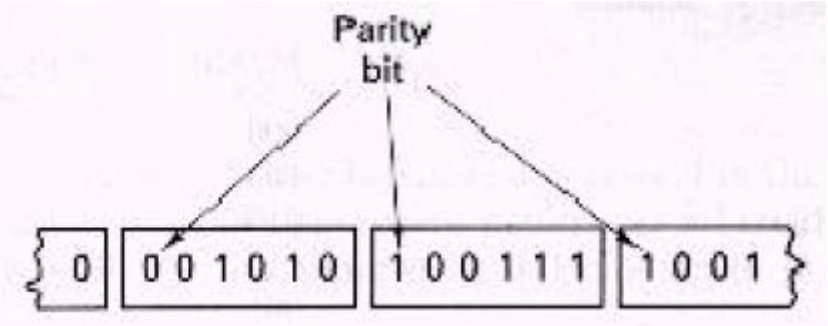
Paremos la pelota dos segundos. ¿Por que nos mudamos al mundo de la codificación estructurada y abandonamos las codificaciones de forma de onda? Resulta que esta última, es ineficiente en el uso del ancho de banda. Para un código seteado ortogonalmente de M=2^k formas de onda, el ancho de banda requerido para la transmisión es M/k veces del que se necesita para sistemas sin codificar. Además, convengamos que las codewords que surgen de la técnica aplicada no llevan el mensaje digital en su interior de forma explícita (el decodificador requiere un procedimiento más tedioso para obtener el mensaje digital) y si bien aportan mejoras en sus características que permiten distinguirlas mucho más entre ellas no es la mejor manera de realizarlo. Ante la presencia de errores no se plantea una técnica explícita para corregirlos. Por estas razones y muchas más es que abandonamos las características ortogonales de las codewords y nos centramos en la forma de agregar redundancia al mensaje digital, es decir, a la fuente de dato. Ahora las codewords van a tener el mensaje digital de k elementos y n-k bits de redundancia, insertados inteligentemente, con algún criterio, para permitir detectar y corregir errores.

Mensaje digital	Codeword de un set ortogonal
0 1 0	0 0 1 1 0 0 1 1
0 1 1	0 1 1 0 0 1 1 0

mensaje digital	Codeword de una secuencia estructurada	
0 1 0	Uso de la paridad par	1 0 1 0
0 1 1		0 0 1 1

Códigos de Chequeo de Paridad

A la hora de hablar de chequeos de paridad encontraremos dos maneras de implementarlos, estos se conocen con el nombre de *código de chequeo de paridad simple* y el otro a analizar conocido como *código rectangular.*
El *código de chequeo de paridad simple* consiste en la adición de un bit de paridad del lado izquierdo al bloque de bits de datos. Este bit puede asignarse un valor de 0 o 1 según se implemente el modo de paridad par o impar, siendo el modo de implementación, por ejemplo para el caso de paridad par como se verá en la figura, debemos obtener que la sumatoria de 1 junto con el bit de paridad debe dar como resultado un valor par. Para hacer esta verificación se implementa suma aritmética de módulo 2.



El procedimiento de decodificación en el receptor consiste en testear que la suma de módulo 2 de los bits de la codeword para el caso de paridad par resulte cero, si así no fuera, se sabe que la codeword tiene errores. Respecto al tiempo de bit de código se expresa como k/(k+1) en donde el "+1" del denominador se debe al haber agregado el bit de paridad pero que no es de información.
Asumiendo que todos los bits erróneos tienen igual probabilidad y ocurren independientemente, nosotros podemos escribir la probabilidad de "j" errores ocurriendo en un bloque de "n" símbolos como:

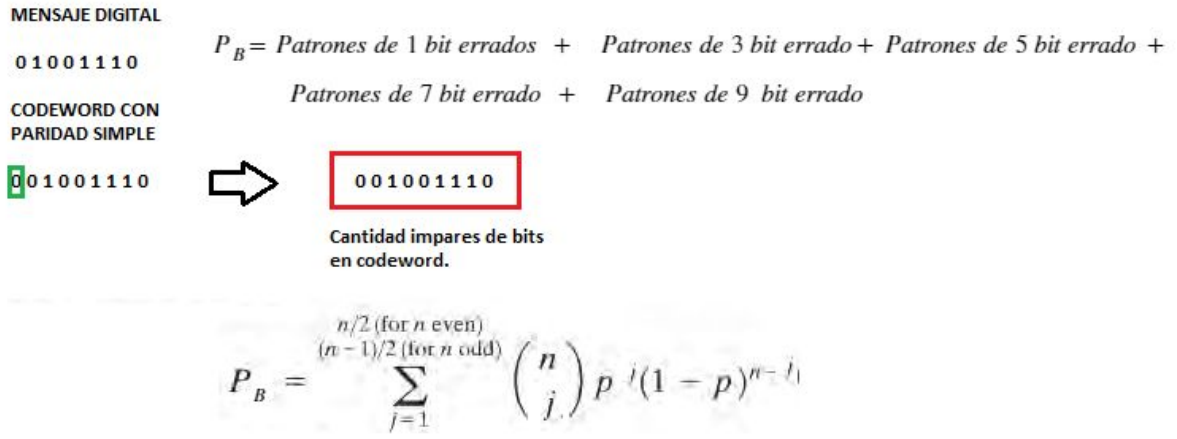
$$P(j,n) = \binom{n}{j} p^j (1-p)^{n-1}, \qquad \binom{n}{j} = n!/\,j!(n-j)!$$

En donde *p* es la probabilidad que un "channel symbol" sea recibida con error y lo que se encuentra dentro del óvalo rojo es el número de maneras en que j bits de salida de "n" pueden tener errores. De esta manera, la probabilidad de un error no detectable *Pnd* con un bloque de *n* bits es computada como:

$$P_{nd} = \sum_{j=1}^{n/2\,paranpar \;\; n-1/2\,nimpar} \binom{n}{2j} p^{2j} (1-p)^{n-2j}$$

¿A qué se debe dicha formulanga? El chequeo de paridad simple permite la detección siempre de una cantidad impar de bits erróneos, es decir, permite detectar patrones de error

de 1 bit, 3 bit, 5 bit, y así sucesivamente. Pero si los errores son pares, por ejemplo, dos bits cambiados simultáneamente, el bit designado a paridad no acusara errores. **No logramos detectar en esta situación.**



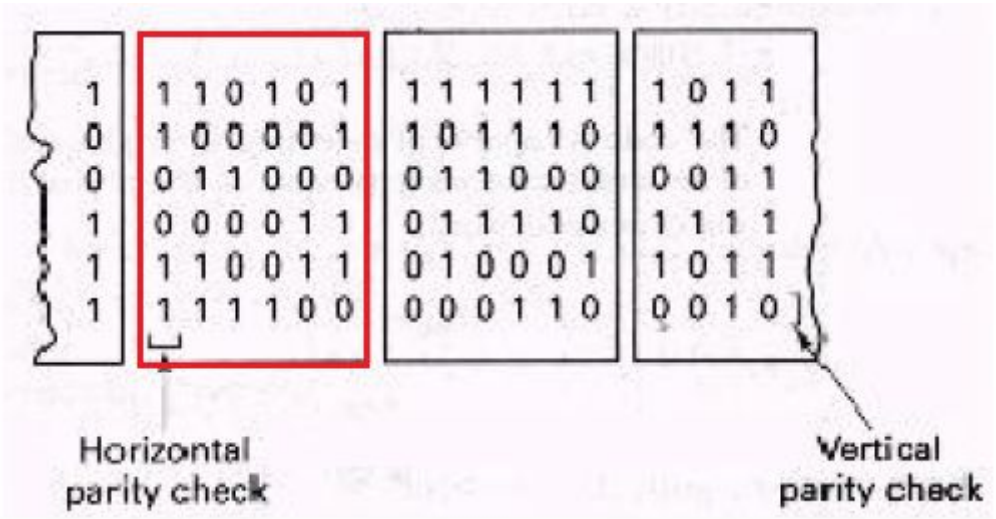
$$P_B = \text{Patrones de 1 bit errados} + \text{Patrones de 3 bit errado} + \text{Patrones de 5 bit errado} + \text{Patrones de 7 bit errado} + \text{Patrones de 9 bit errado}$$

$$P_B = \sum_{j=1}^{\substack{n/2 \text{ (for } n \text{ even)} \\ (n-1)/2 \text{ (for } n \text{ odd)}}} \binom{n}{j} p^j (1-p)^{n-j}$$

Código Rectangular

Para la implementación de este código primeros debemos pensar en una matriz de M x N a la cual se le adicionará una fila y una columna para el chequeo de paridad tanto de manera horizontal como vertical, obteniendo una matriz de (M+1) x (N+1). Esta implementación traerá la ventaja de que en caso de un error de un solo bit, poder saber su ubicación.
Para este caso, el tiempo de bit de código se expresa como

k/n=MN/(M+1)*(N+1)



La principal diferencia entre el código rectangular y el código de simple paridad es que este último tiene la capacidad únicamente de detectar errores pero el código rectangular cuenta con la capacidad de detectar y además poder corregir errores simples ya que cada error es únicamente localizado en la intersección del error detectado en la fila y el error detectado en la columna. Para el rectángulo rojo mostrado en la figura en donde la matriz es de 6 x 6 con 5 bits de datos y 1 para paridad, se puede corregir un error SIMPLE localizado en algún lugar de los 36 bits de la matriz.

Para el cálculo de probabilidades se debe reutilizar la probabilidad vista en código simple con la diferencia que ahora consideraremos todas las formas en el que un mensaje erróneo puede ser hecho:

$$P_M = \sum_{j=t+1}^{n} \binom{n}{j} p^j (1-p)^{n-j}$$

Se debe observar que el comienzo de la sumatoria es desde j = t+1, en donde t se refiere a los patrones de errores que pueden ser corregidos, en este caso t=1 por lo que j=2 y el valor máximo de la sumatoria para el ejemplo antes citado es de 36, quedando:

$$P_M = \sum_{j=2}^{36} \binom{36}{j} p^j (1-p)^{n-j}$$

Dado el caso de que la probabilidad de que un “channel symbol” sea recibido con error (*p)* sea de un valor pequeño, el primer término de la sumatoria será el dominante. De esta forma llegamos a:

$$P_M = \binom{36}{2} p^2 (1-p)^{34}$$

¿Porque siempre es preferible aquellos códigos que permiten aplicar corrección de errores? Es necesario analizar la performance del sistema con y sin codificación. En la siguiente figura se muestra como se comporta la probabilidad de error de bit en función de la Eb/No tanto para un sistema como para el otro.

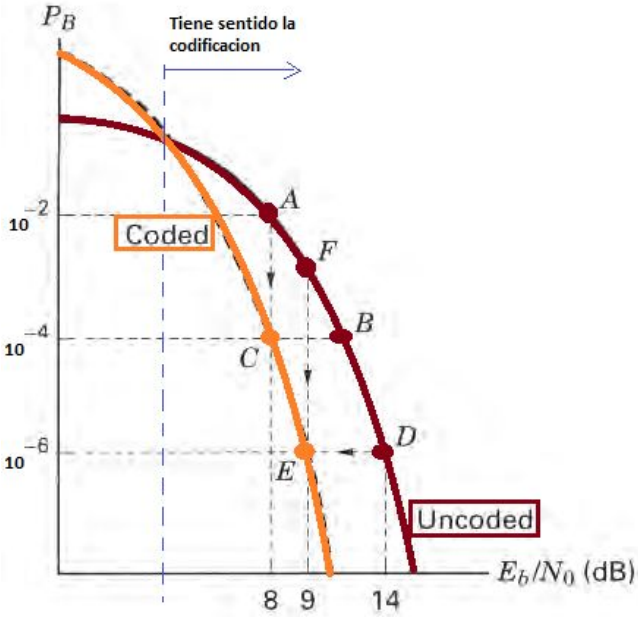


Figure 6.9 Comparison of typical coded versus uncoded error performance.

Hagamos un análisis detallado. Para ello vamos a realizar lo que se denomina “*Trade-off ”. Esta técnica de análisis consiste en dejar fijas algunas variables de análisis y estudiar el comportamiento de aquellas que permitimos que varíen.*

Trade- off 1: Performance de error Versus Ancho de banda.

Nuestro sistema de comunicaciones se encuentra operando sobre el punto A, el cual posee una probabilidad de error de bit de 10^{-2} con una Eb/No de 8 dB pero necesitamos alcanzar una probabilidad de error de bit de 10^{-4} para que nuestro negocio sea rentable y no perder al cliente por no aportar la calidad suficiente al sistema. ¿Qué podemos hacer? Cuando estudiamos cada una de las posibilidades encontramos que todas ellas nos llevan a aumentar la Eb/No de recepción requerida. Es decir, movernos del punto A al punto B sobre la curva de performance del sistema. Esto puede implicar cambiar antenas para obtener mayor ganancia, acercar los equipos, aumentar potencia en el transmisor si tenemos la posibilidad y demás. Pero si aplicamos codificación con corrección de errores logramos desplazarnos del punto A al punto C, cambiando abruptamente de curva de performance de error sobre la cual trabaja nuestro sistema. Manteniendo todas las variables del sistema intactas y agregando solamente codificación logramos que con la misma Eb/No reducir la probabilidad de error de bit.

Todo mas que lindo pero esta mejora tiene un costo. La variable que se nos está escapando y cambiando abruptamente es el ancho de banda. **Insertar codificación requiere un aumento de ancho de banda o tener paciencia.**

Trade-off 2: Potencia versus ancho de banda

Consideremos un segundo caso de análisis. Tenemos un sistema de comunicaciones que se encuentra operando en el punto D, donde P_b es 10^{-6} y Eb/No es 14 dB. En esta situación, aunque el cliente no perciba errores, el sistema está funcionado mal producto de la potencia de 14 dB, esto genera un error permanente en el equipo. La situación mejoraría si fuera posible disminuir la potencia manteniendo la probabilidad de error de bit constante.

Aca medio como que estamos hasta los huevos, porque si nos movemos por la misma curva de performance del sistema para el esquema de modulación/demodulación actual no hay chance de cumplir con los requerimientos establecidos. Debemos cambiar de curva, fuera FSK podríamos aumentar M lo cual implica traer otro esquema de modulación y demodulación, de tal manera que lográramos reducir la Eb/No requerida para la misma probabilidad de error de bit. ¿Me putearán si pido algunos dólares a la empresa para poner un sistema nuevo y solucionar el problema? **SI**, seguro. Lo que queres hacer es tirar todo lo que tenias armado funcionando, para darle el toquecito ingenieril. A ver padre, está más que claro que no podes hacer eso, aplicá codificación ¿Por qué? La codificación tiene un efecto muy similar a FSK cuando se incrementa M, solo que es mejor, porque además corrige errores y no permite que la probabilidad de error de símbolo crezca. En sistemas PSK también tenemos los mismos beneficios. **Eso si, pagas con ancho de banda o con paciencia.**

Trade-off 3: Velocidad del dato Versus Ancho de banda

En este análisis, el planteo es diferente. Consideramos que estamos trabajando con un sistema sin codificación y en el punto D nuevamente. Asumiendo que no hay problema con la calidad del dato y que no es de particular necesidad reducir la potencia. Si quisiéramos incrementar la velocidad de la fuente de información “Rb” ¿Que ocurriría?

$$\frac{E_b}{N_0} = \frac{P_r}{N_0}\left(\frac{1}{R}\right)$$

Al ser constante la potencia recibida e incrementar la tasa de bits vemos una reducción en la energía de bit recibida respecto al ruido. Esto genera que nuestro sistema pase de operar en el punto D al punto F donde la probabilidad de error de bit es mayor. La cagamos muchachos. Apliquemos codificación ¿Cómo nos salva ahora el San Martín de las comunicaciones digitales avanzadas? Resulta que al aplicar estas técnicas para Eb/No constante, tenemos una probabilidad de error de bit mucho mejor de lo que teníamos sin codificar, entonces en esta condición, aumentamos la tasa de bits de la fuente de informacion. La calidad del sistema se degrada, como siempre, pero como ya estábamos en un punto mucho mejor sobre la curva de performance de codificación, esta degradación nos deja en el punto de operación E, justo con la misma probabilidad de error de bit y una tasa de bit más alta. ¿Con qué pagamos? con mucho más ancho de banda producto de codificar a una fuente de información de mayor tasa.

Ganancia de código o de codificación

Esta ganancia nos permite observar o comparar la reducción que se puede obtener respecto a E_b/N_0 relacionándola entre una curva codificada y una sin codificación pero con la condición de mantener la P_B fija. Esto se cumplira siempre y cuando el analisis se haga posterior al cruce entre curvas en donde tienen igual E_b/N_0. La manera de expresar esta ganancia es:

$$G(dB) = \left(\frac{Eb}{No}\right)_u (dB) - \left(\frac{Eb}{No}\right)_c (dB)$$

Donde: $\left(\frac{Eb}{No}\right)_u$ representa la relación sin código y $\left(\frac{Eb}{No}\right)_c$ con código.

Performance del código a bajos valores de E_b/N_o

La codificación es una herramienta que tiene limitaciones. No puede ser aplicada en cualquier caso. Se requiere valores mínimos de E_b/N_o para que produzca una mejora en la performance del sistema. ¿Esto qué significa? Que en los gráficos de probabilidad de error versus Eb/No requerido tanto para sistemas codificados como no codificados existe un cruce entre las curvas (usualmente en valores bajos de E_b/N_o).
La razón por la que se cruzan es que siempre los sistemas con codificación tendrán capacidad de corrección de errores fija.
Imaginemos ahora que la E_b/N_o es reducida continuamente ¿Qué sucede a la salida del demodulador? Esto producirá más y más errores. Hasta ahora, un decrecimiento continuo en E_b/N_o permite eventualmente generar algún umbral donde el decodificador se las arregla

con el error. Pero si dicho umbral es alcanzado y superado, el decodificador es abrumado por el error y codificar empeora aún más las cosas. Una imagen dice mas que mil palabras:

Código de Bloque Lineal

Ahora describiremos lo que se conoce como *Secuencia Estructurada* comenzando por desarrollar el *Código de Bloque Lineal* el cual son una clase de códigos de chequeos de paridad que pueden ser caracterizados por (n,k) notación antes descripta. En donde el codificador transforma un bloque de k dígitos de mensajes (vector de mensaje) dentro de un bloque de un largo de n dígitos de codeword (vector de código) el cual se conforma por el mensaje y la paridad. La cantidad de elementos distintos en el mensaje seguirán siendo solo 2, el 0 y el 1, por lo que el código sigue siendo binario.

Los mensajes de k-bit de 2^K secuencias de mensajes distintos referido como k-tuples (secuencia de k dígitos). El bloque de n-bit puede formar 2^n secuencias distintas, referido a n-tuples.
El procedimiento de codificado consiste en la asignación de cada uno de los 2^K mensajes de k-tuples en uno de los 2^n de los n-tuples. Por lo que un bloque de codificación representa una asignación uno a uno según el cual los 2^k mensajes k-tuples son mapeados unívocamente dentro de un nuevo conjunto de 2^K codeword n-tuples.

TABLA 6.1 Asignación de codewords a mensajes

Vector Mensaje	Codeword
0 0 0	0 0 0 0 0 0
1 0 0	1 1 0 1 0 0
0 1 0	0 1 1 0 1 0
1 1 0	1 0 1 1 1 0
0 0 1	1 0 1 0 0 1
1 0 1	0 1 1 1 0 1
0 1 1	1 1 0 0 1 1
1 1 1	0 0 0 1 1 1

La creación de un subespacio dentro de un espacio de dimensión *n* implica el cumplimiento de ciertas características especiales del espacio vectorial elegido.

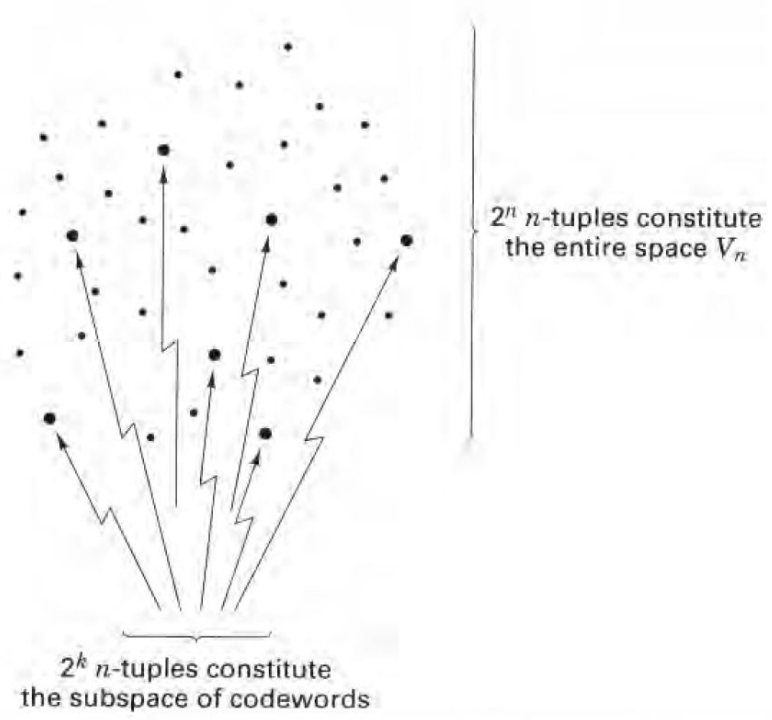
Espacios Vectoriales

Para el espacio antes mencionado como *n*, el cual se compone de n-tuples binarios, Vn, es llamado espacio vectorial sobre un campo binario de dos elementos (0 y 1). En el campo binario habrá dos operaciones, suma que se hará por aritmética de módulo 2 y multiplicación.

Suma	multiplicación
$0 \oplus 0 = 0$	$0 * 0 = 0$
$0 \oplus 1 = 1$	$0 * 1 = 0$
$1 \oplus 0 = 1$	$1 * 0 = 0$
$1 \oplus 1 = 0$	$1 * 1 = 1$

Dentro de dicho espacio, vive un subconjunto de dimensión n pero que contiene k elementos donde cada uno representa las codewords válidas para los mensajes digitales. Este subespacio para que sea definido como tal, debe cumplir con la propiedad de cierre e incluir como codeword válida al vector de todos ceros.
Un codificador de bloques lineal, puede ser definido como tal, si todas las codewords pertenecientes al set cumplen la propiedad de cierre y se incluye el vector de todos ceros como una codeword válida.
La propiedad de cierre establece que la suma de dos codewords válidas cualesquiera dentro del set deben generar una tercera codeword tambien valida. Esta propiedad es fundamental para la característica algebraica del bloque de código lineal. Suponiendo que dos elementos del espacio vectorial V_n, es decir, V_i y V_j considerados como codewords o vectores código en un bloque de codificación (n,k). La misma será definida como Lineal si y sólo si ($V_i \oplus V_j$) es también un vector código. Siendo lo importante de dicha propiedad es

que cualquier vector fuera del subespacio no puede ser obtenido mediante la suma de dos vectores miembros del subespacio.



En definitiva, un mensaje es codificado dentro de los 2^k vectores códigos permisibles y luego es transmitido. A causa del ruido del canal, se produce una versión distorsionada de las codeword (se recibe uno de los 2^n restantes vectores del espacio de n - tuples). Si el vector recibido (perturbado por ruido) no es muy diferente (no muy distante) de la verdadera codeword, el decodificador puede decodificar correctamente el mensaje.
Esto refleja la verdadera meta de la codificación por bloques lineal. Lograr que los códigos válidos estén apartados unos de otros tanto como sea posible siempre pensando en utilizar la menor cantidad de bits de redundancia.

→ Ejemplo de bloque lineal: dado un código (6,3) en donde hay $2^k=2^3=8$ vectores de mensaje (y por lo tanto 8 codewords) y que existen $2^n=2^6=64$ o sea 6-tuples en el espacio vectorial V_6 se puede comprobar, por la siguiente figura, que las 8 codewords mostradas a la derecha, forman parte del subespacio de V_6 ya que se cumple con las 2 condiciones claves de que se contempla la codeword de todos ceros y que la suma entre 2 codewords cuales sean, dan como resultado otra codeword considerada. Por lo que se puede concluir en que esas codewords representan un bloque de codificación lineal y como condición de tal, llevan explicito el vector mensaje contenido dentro de la codeword.

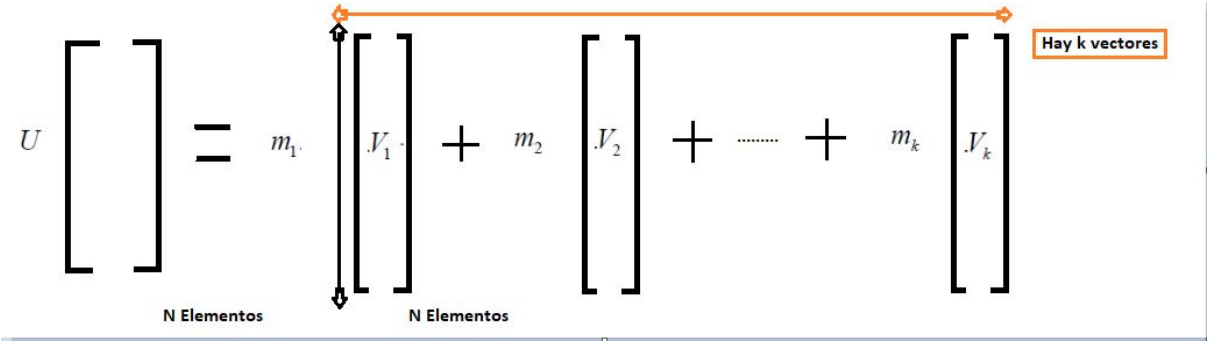
TABLA 6.1 Asignación de codewords a mensajes

Vector Mensaje	Codeword
0 0 0	0 0 0 0 0 0
1 0 0	1 1 0 1 0 0
0 1 0	0 1 1 0 1 0
1 1 0	1 0 1 1 1 0
0 0 1	1 0 1 0 0 1
1 0 1	0 1 1 1 0 1
0 1 1	1 1 0 0 1 1
1 1 1	0 0 0 1 1 1

Matriz generadora

Si ahora dejamos de considerar valores pequeños de k, los cuales nos permitian realizar el procedimiento de asignación de codewords de manera medianamente rápida y metódica y consideramos valores grandes de k, lo cual trae de la mano un *n* mayor y una cantidad de codeword muy grande resulta prácticamente imposible una construcción de bloque de código de forma manual y además se requeriría una enorme cantidad de memoria para almacenar en una tabla todos los valores posibles de los mensajes digitales mapeados uno a uno con sus codewords correspondientes.
Es por esto que se introduce el concepto de matriz generadora, ella permitirá la solución del inconveniente antes comentado, permitiendo que a medida que se generen los mensajes digitales se vaya calculando en el proceso ahorrando significativamente en la memoria o requerimientos que se destinan a nuestros sistemas.
Para la implementación de la matriz generadora se debe tener en cuenta que cualquier conjunto base de k linealmente independientes n-tuples como lo es $V_1,V_{2,..},V_k$ pueden ser usados para generar los vectores de código de bloque lineales requeridos. Es decir requiere el conocimiento de álgebra. Entonces cada vector de código será una combinación lineal de $V_1,V_{2,..},V_k$. Esto es que cada conjunto de 2^k codewords {U} pueden ser descritos por:

$$U = m_1.V_1 + m_2.V_2 + ... + m_k.V_k$$



$$\begin{bmatrix} [\quad V_1 \quad] \\ [\quad V_2 \quad] \\ [\quad \vdots \quad] \\ [\quad V_k \quad] \end{bmatrix}$$

en donde m_i= (0 o 1) son los dígitos del mensaje e i=1,...,K. Por lo que la matriz generadora se compone:

$$G = \begin{bmatrix} V_1 \\ V_2 \\ . \\ . \\ V_K \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & . & . & V_{1n} \\ V_{21} & V_{22} & . & . & V_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ V_{K1} & V_{K2} & . & . & V_{KN} \end{bmatrix}$$

Los vectores código, por conveniencia, son usualmente designados como una fila de vectores. De esta manera el mensaje m, y una secuencia de k bits de mensajes, se muestran abajo como una fila de vectores (matriz de 1 x k teniendo una fila y k columnas):

$$m = m_1, m_2, \ldots\ldots, m_k$$

La generación de las codeword U es escrita en notación de matriz como el producto de m y G, como:

$$\mathrm{U} = \mathrm{mG}$$

donde, en general, la multiplicación de matrices C=A.B es realizada de manera usual por el uso de la regla:

$$C_{ij} = \sum_{k}^{n} a_{ik}.b_{kj} \qquad i = 1.....l \qquad j = 1.....m$$

donde A es una matriz de l x **n** y B de **n** x m y el resultado C es una matriz de l x m. Por el ejemplo introducido en la sección precedida, podemos confeccionar un generador de matriz como:

$$G = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Donde V_1, V_2 y V_3 son tres vectores linealmente independientes (un subconjunto de los 8 vectores código) que pueden generar todos los vectores códigos. Nótese que la suma de dos cualquiera de los vectores generadores no produce cualquier otro vector generador dado que entre ellos son independientes (la regla de cierre no se cumple). Generamos las

codeword U_4 con el cuarto vector mensaje 110 en la tabla 6.1 (al principio de Codigo de Bloque Lineal), usando la matriz generadora de la figura anterior:

$$U_4 = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = 1.V_1 + 1.V_2 + 0.V_3$$

$$= 1\,1\,0\,1\,0\,0 + 0\,1\,1\,0\,1\,0 + 0\,0\,0\,0\,0\,0$$
$$= 1\,0\,1\,1\,1\,0 \quad \text{(codeword para el vector mensaje 1 1 0)}$$

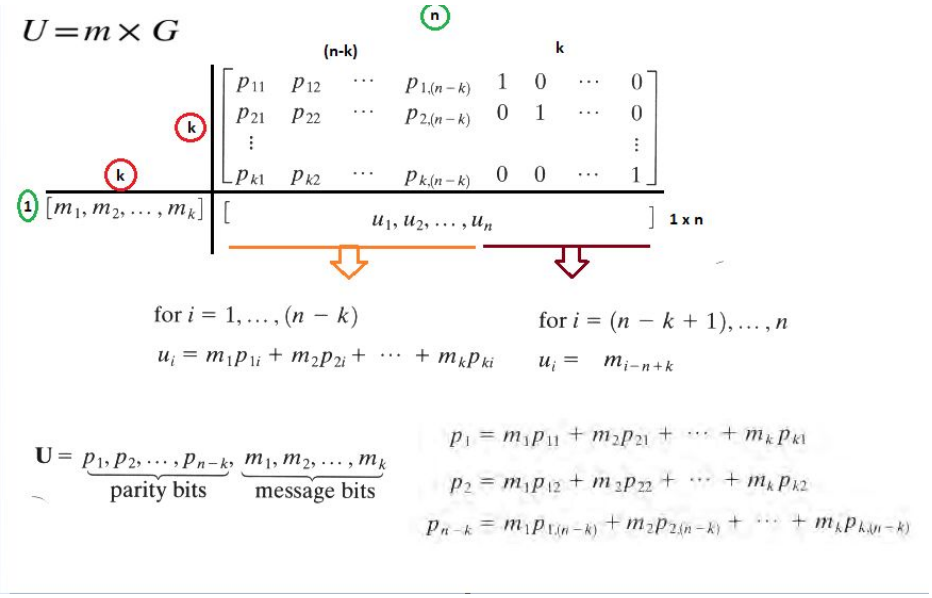
De esta manera los vectores códigos o codeword correspondientes a un vector mensaje (110) es una combinación lineal de las filas de G.
Haciendo que el código es definido totalmente por G y el codificador sólo debe cargar k filas de G en vez del total de 2^k vectores códigos. Haciendo que con la implementación de Matriz Generadora se necesite una matriz de 3x6 reemplazando al caso inicial que era de 8x6(codewords x longitud) haciendo una reducción en la complejidad del sistema. A todo este mundo de flores y colores gracias a la implementación de la matriz generadora, nunca se especifica paso a paso cómo construir el subespacio de codeword válidas dentro del espacio de dimensión n. La matriz generadora tiene sentido si ya tenemos todas las codeword asignadas, es decir, nuestro bloque de codificación lineal listo para implementar.
Pregunta **¿Cómo encontramos rápidamente la matriz generadora a partir de todas las codeword posible?** La respuesta consiste en inspeccionar la matriz generadora presentada de ejemplo.

$$G = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Es característico de este tipo de codificación que los k vectores linealmente independientes elegidos para conformar la matriz generadora sean aquellas codeword cuyos mensajes digitales dentro de la matriz generan la identidad de K por K. El resto de coeficientes de cada vector linealmente independiente, aportan al cálculo de la redundancia para cada codeword.

$$\mathbf{G} = \left[\mathbf{P} \;\vdots\; \mathbf{I}_k \right]$$

$$= \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1,(n-k)} & 1 & 0 & \cdots & 0 \\ p_{21} & p_{22} & \cdots & p_{2,(n-k)} & 0 & 1 & \cdots & 0 \\ \vdots & & & & & & \vdots & \\ p_{k1} & p_{k2} & \cdots & p_{k,(n-k)} & 0 & 0 & \cdots & 1 \end{bmatrix}$$



Matriz de Control de Paridad

Esta matriz representada como H, tiene la tarea de hacer la decodificación respecto a los vectores recibidos con la **única finalidad de controlar que ninguno de los bits de la codeword hayan sido alterados** mediante la verificación del cálculo de la paridad. El decodificador reconstruye los cálculos que hizo el codificador pero a partir de la codeword recibida. Si en el canal se produjeron errores, en lugar de obtener un cero al hacer la verificación de la ecuación, se obtiene un 1.

$$p_1 = m_1 p_{11} + m_2 p_{21} + \cdots + m_k p_{k1}$$

$$p_2 = m_1 p_{12} + m_2 p_{22} + \cdots + m_k p_{k2}$$

$$p_{n-k} = m_1 p_{1,(n-k)} + m_2 p_{2,(n-k)} + \cdots + m_k p_{k,(n-k)}$$

A partir de esto se podría pensar en las dimensiones de H^T como sigue:

- "U" es una codeword válida de dimensiones (1 x n)
- r X H^T tiene como finalidad verificar los valores de "P" entonces tenemos un vector que arroja valores de 0 si las ecuaciones no se alteraron y 1 si hay modificaciones. Este vector de verificación, cuya dimensión es (1 x (n-k)) se denomina **Síndrome** o comunmente "S"**.**
- Por lo tanto H^T tiene dimensiones de (n x (n-k))
- Verificamos S(1x(n-k))= U (1 x n) X H^T (n x (n-k))

En definitiva, se debe de tener en cuenta que para cada G de (k x n) codificadora existirá una matriz H de ((n-k) x n), tal que las filas de G sean ortogonales a las filas de H haciendo que $G_{(k \times n)} * H^T_{(n \times (n-k))} = 0_{(k \times (n-k))}$ en donde los subíndices son las dimensiones de las matrices.

Para cumplir con las condiciones de ortogonalidad para un código sistemático, escribimos los componentes de H y H^T :

$$\mathbf{H}=[\mathrm{I}_{n-k}\ \mathrm{P}^T] \Rightarrow H^T = \begin{bmatrix} I_{n-k} \\ \cdots\cdots\cdots \\ P \end{bmatrix} = \begin{bmatrix} 1 & 0 & . & . & 0 \\ 0 & 1 & . & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & 0 & . & . & 1 \\ p_{11} & p_{12} & . & . & p_{1(n-k)} \\ p_{21} & p_{22} & . & . & p_{2(n-k)} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{k1} & p_{k2} & . & . & p_{k(n-k)} \end{bmatrix}$$

$$H^T = \begin{bmatrix} I_{n-k} \\ \cdots\cdots\cdots \\ P \end{bmatrix} = \begin{bmatrix} 1 & 0 & . & . & 0 \\ 0 & 1 & . & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & 0 & . & . & 1 \\ p_{11} & p_{12} & . & . & p_{1(n-k)} \\ p_{21} & p_{22} & . & . & p_{2(n-k)} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{k1} & p_{k2} & . & . & p_{k(n-k)} \end{bmatrix} \begin{matrix} p_1 \\ \\ \\ \\ \\ m_1 p_{11} \\ m_2 p_{21} \\ \\ \\ m_k p_{k1} \end{matrix} \quad \begin{matrix} p_2 \\ \\ \\ \\ \\ m_1 p_{12} \\ m_2 p_{22} \\ \\ \\ m_k p_{k2} \end{matrix} \quad \begin{matrix} p_{n-k} \\ \\ \\ \\ \\ m_1 p_{1(n-k)} \\ m_2 p_{2(n-k)} \\ \\ \\ m_k p_{k(n-k)} \end{matrix}$$

Ec 1 **Ec 2** **Ec k**

Cuando llega la codeword verificacion de los pesos pk calculados por el codifcador

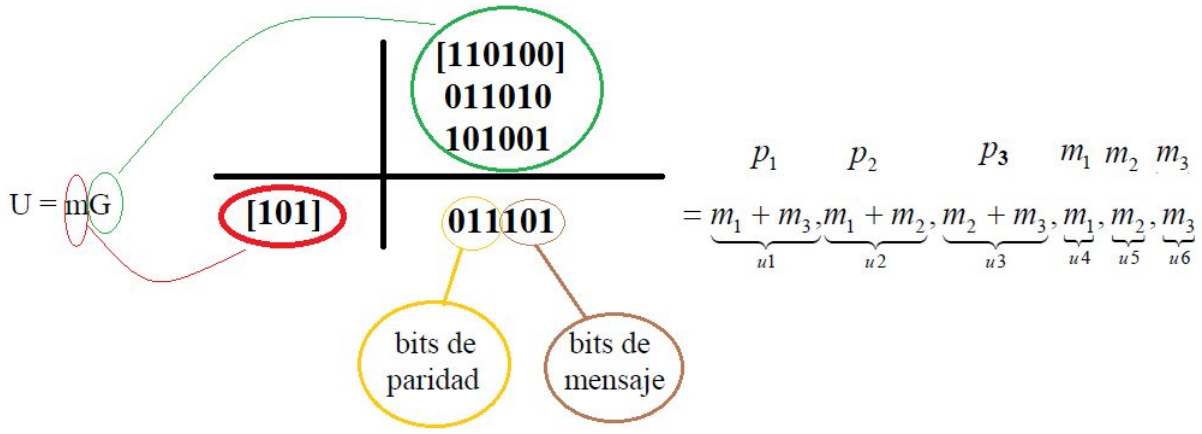
Se puede verificar que el producto de $U*H^T$ de cada codeword U ha sido generado por la matriz G. Esta multiplicación dará como resultado los bits de paridad p.

$$UH^T = p_1 + p_1, p_2 + p_2, \ldots\ldots\ldots\ldots p_{n-k} + p_{n-k} = 0$$

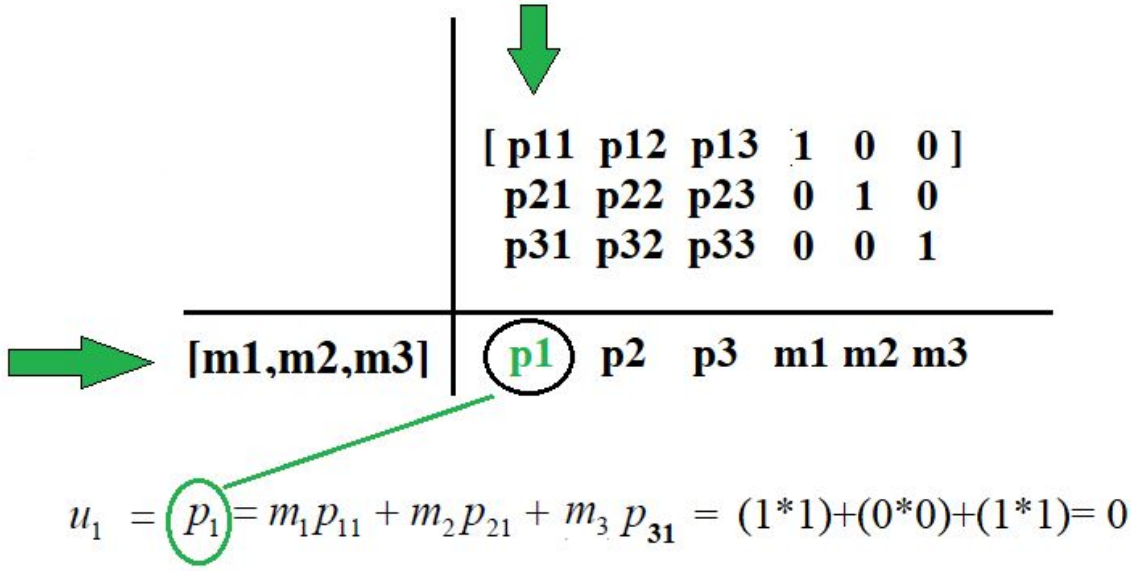
Por lo que una vez que la matriz de control de paridad H es construida y cumple con los requerimientos de ortogonalidad, se puede comprobar mediante la recepción de que un vector es una parte válida del conjunto de codeword, esto quiere decir que si realmente U es una codeword generada por la matriz G, debe de cumplir con que $U*H^T$ sea igual a cero. Para que todo este procedimiento, desde el transmisor hasta el receptor, quede un poco más claro vamos a plantear un ejemplo con los siguientes datos:

m=[m1,m2,m3]=[101] **G=[110100]**
011010
101001

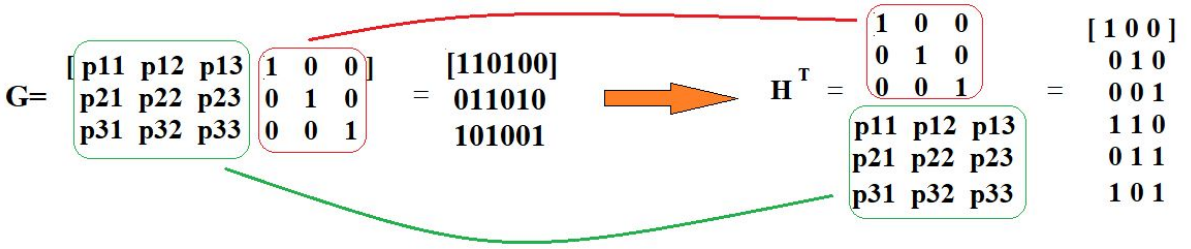
Recordando que G se compone de los bits de redundancia(p_{xx}) y la matriz identidad, el mensaje elegido para enviar es 101 y que U el código a transmitir se obtiene como:



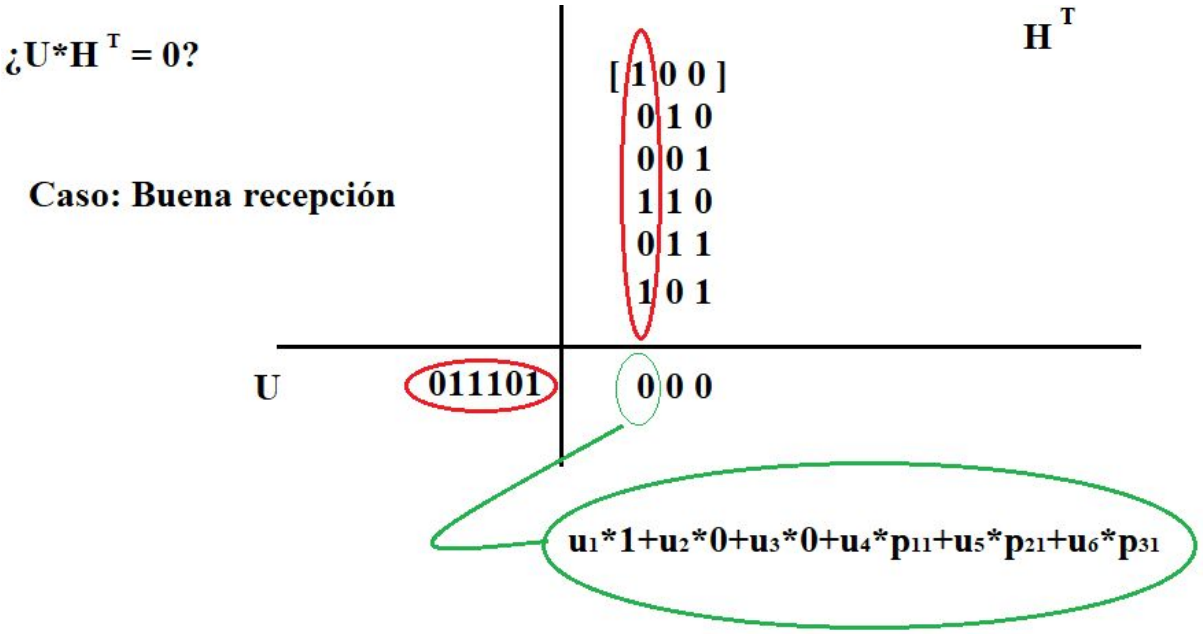
Como se observa, U está compuesta de un vector de 6 componentes culpa de (m*G) en donde se puede distinguir los bit de paridad y los bits de mensaje pero por si no quedo claro, veamos cómo se combina notación con resultados para un cierto u_x, por ejemplo el u_1:

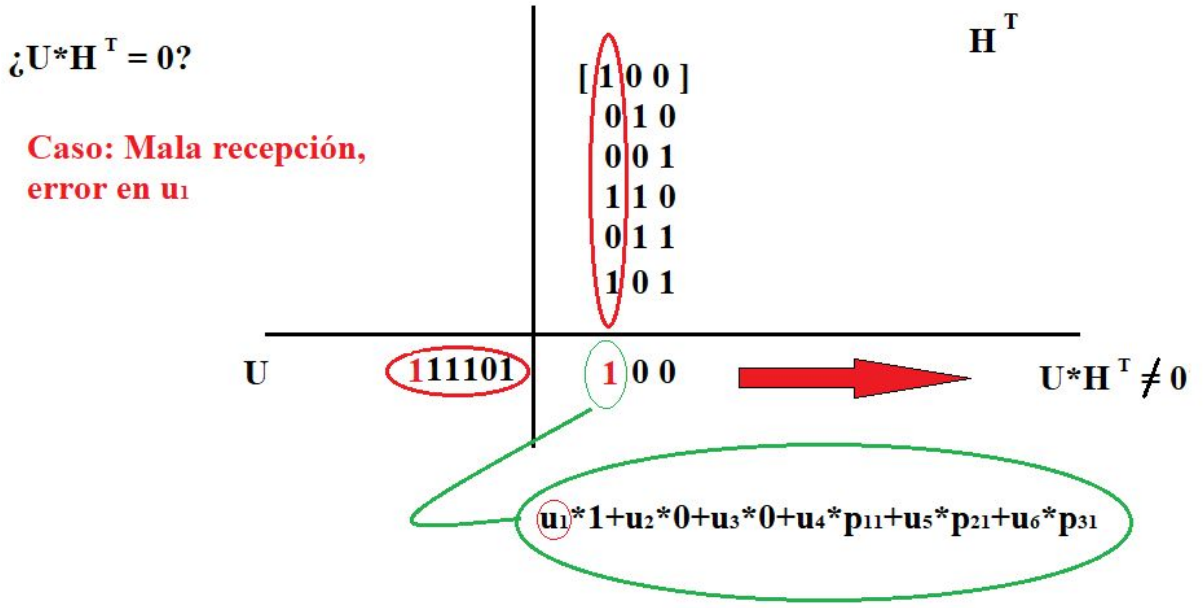


Una vez que tenemos la U compuesta de U_1,..,U_6 se enviará. Luego situados en el receptor haremos el **control de paridad**, si, únicamente controlar. Para esto debemos de recordar el H^T que se obtenía desde la matriz generadora G:



Finalmente para comprobar de que los bits de paridad mantienen sus valores correctos y por lo tanto asumir de que el mensaje a llegado bien, se debe hacer $U*H^T$, lo cual debe de dar cero si no hay errores. Para esto vamos a tomar 2 casos, uno con el U que ha llegado de manera correcta y otro caso en el que la U se ve modificada a la hora de la recepción:





Uso del Síndrome:

En un sistema de comunicaciones digital, el receptor recibe “r” producto que se transmitió la codeword “U” y el canal adicionó ruido produciendo un error “e”. Dicho error puede ser expresado como un vector de (2^{n-1}) valores posibles porque puede afectar a cualquiera de los bits de la codeword que se propaga por el canal de comunicaciones. Entonces “r” queda definido de la siguiente forma: r = U + e

En función de la codeword recibida “r” , el síndrome se define como:

$$S = r.H^T \qquad S = (U + e)\,H^T$$

El síndrome no es ni más ni menos que el resultado del chequeo de paridad sobre “r” para determinar si es un miembro válido del set de codeword. Si en efecto “r” es un miembro válido, el síndrome tiene valor cero. Si “r” tiene errores corregibles, el síndrome tiene valores diferentes de cero que se pueden enmarcar en un patrón de error particular. El decoder, dependiendo de cómo están implementados los FEC y ARQ, en un caso tomará acciones para localizar los errores y corregirlos (FEC) ó ,de otra forma, pedirá la retransmisión (ARQ). Combinando ecuaciones obtenemos:

$$S = (U + e) H^T$$

$$S = UH^T + eH^T$$

Sin embargo, como UH^T =0 para todos los miembros del set de codeword. Tenemos:

$$S = e.H^T$$

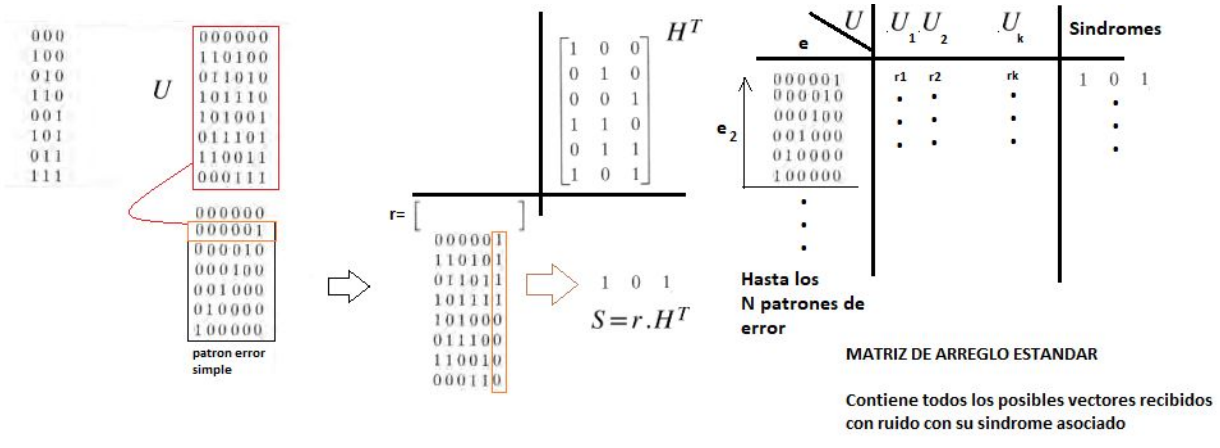
Esta expresión muestra que si el test de sindrome se hace sobre cualquier vector de código corrupto o sobre el patrón de error que lo causa, se produce el mismo síndrome. Una propiedad importante de los bloques de código lineal es que el mapeo entre el **patrón de error corregible** y el síndrome es uno a uno.

Es importante mencionar, que el síndrome es una herramienta muy útil en la decodificación de bloques lineales si se respetan ciertas características en la matriz de chequeo de paridad:

- 1. Las columnas de H no pueden ser nulas (Vector de ceros)
- 2.Todas las columnas deben ser únicas para poder distinguir qué posición tiene un error.

Cuando se dice que un bloque de código lineal tiene una capacidad de corrección de un bit, se está especificando que tiene capacidad de corregir todos los errores de un bit (patrón de error de un bit). Para este caso, donde la cantidad de combinaciones que generan errores de un bit son $\binom{n}{j}$ con “n” como cantidad de bits de codeword y “j” cantidad de bits erróneos de dicha codeword, existe una relación uno a uno entre cada combinación del patrón de error con los valores de síndrome. Para el resto de patrones de error, es decir, todas las combinaciones que generan errores de dos bits, tres bits y así sucesivamente, los síndromes ya no tienen una relación uno a uno con cada combinación, se repiten.

Esta observación puede visualizarse mediante el siguiente ejemplo:



La matriz de arreglo estándar para este ejemplo es la que se puede observar a continuación:

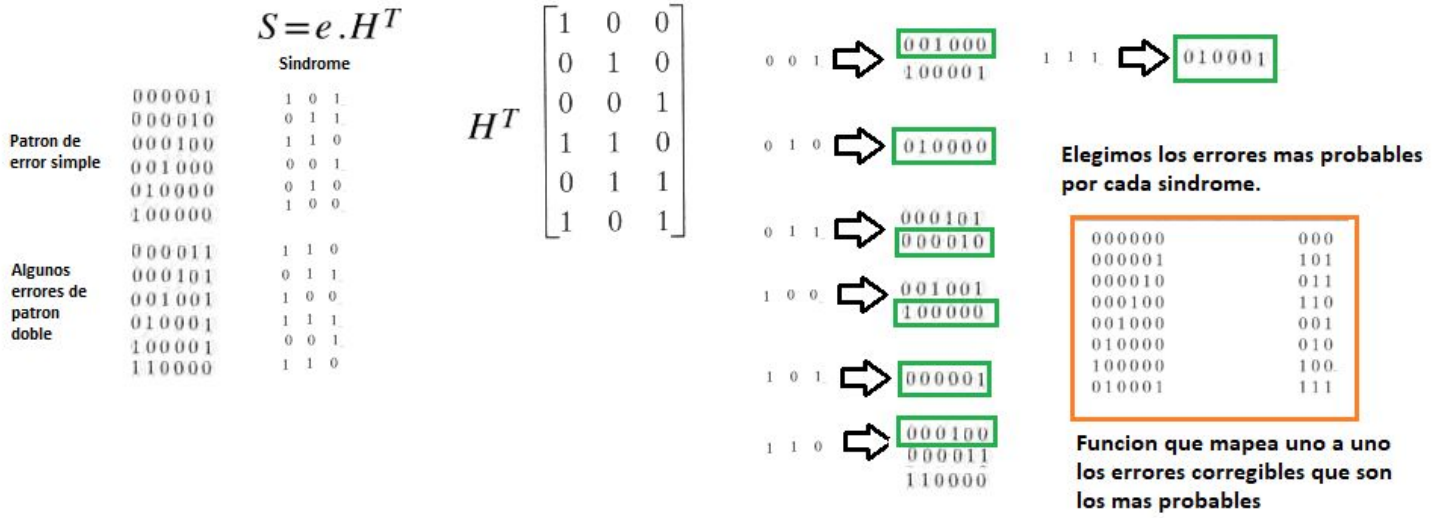
000000	110100	011010	101110	101001	011101	110011	000111
000001	110101	011011	101111	101000	011100	110010	000110
000010	110110	011000	101100	101011	011111	110001	000101
000100	110000	011110	101010	101101	011001	110111	000011
001000	111100	010010	100110	100001	010101	111011	001111
010000	100100	001010	111110	111001	001101	100011	010111
100000	010100	111010	001110	001001	111101	010011	100111
010001	100101	001011	111111	111000	001100	100010	010110
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

La matriz de arreglo estándar es formalmente definida como un arreglo donde cada elemento representa una codeword recibida que se encuentra perjudicada por el ruido.
Dicha matriz contiene en su primera fila todas las codeword válidas comenzando por la de todos ceros y en la primera columna todos los errores posibles. Estos últimos, se agrupan en lo que se conoce como patrones de error. Cada patrón de error de “j” bits contiene todas aquellas combinaciones que de “n” bits solo “j” son diferentes de cero. Ejemplo: Un código de bloque lineal (6,3) puede tener un patrón de error doble con una cantidad de $\binom{6}{2}$ codeword donde 2 bits son distintos de cero (o sea 1) para los 6 bits posibles.

$$
\begin{array}{lllll}
\mathbf{U}_1 & \mathbf{U}_2 & \cdots & \mathbf{U}_i & \cdots \quad \mathbf{U}_{2^k} \\
\mathbf{e}_2 & \mathbf{U}_2 + \mathbf{e}_2 & \cdots & \mathbf{U}_i + \mathbf{e}_2 & \cdots \quad \mathbf{U}_{2^k} + \mathbf{e}_2 \\
\mathbf{e}_3 & \mathbf{U}_2 + \mathbf{e}_3 & \cdots & \mathbf{U}_i + \mathbf{e}_3 & \cdots \quad \mathbf{U}_{2^k} + \mathbf{e}_3 \\
\vdots & \vdots & & \vdots & \\
\mathbf{e}_j & \mathbf{U}_2 + \mathbf{e}_j & \cdots & \mathbf{U}_i + \mathbf{e}_j & \cdots \quad \mathbf{U}_{2^k} + \mathbf{e}_j \\
\vdots & \vdots & & \vdots & \\
\mathbf{e}_{2^{n-k}} & \mathbf{U}_2 + \mathbf{e}_{2^{n-k}} & \cdots & \mathbf{U}_i + \mathbf{e}_{2^{n-k}} & \cdots \quad \mathbf{U}_{2^k} + \mathbf{e}_{2^{n-k}}
\end{array}
$$

¿Por qué (n-k) patrones de error diferentes si las codeword tienen dimensión n? Porque existen 2^k patrones de error que coinciden exactamente con las codeword asignadas como válidas. Si estos patrones de error ocurren, se modificaría la codeword transmitida por otra válida dentro del set de las posibles a transmitir, entonces, el receptor no percibe que dicha codeword recibida es errónea produciendo un **error** pero **no detectable.** *La matriz de arreglo estándar sólo contiene todos los patrones de errores detectables, es decir, $2^{(n-k)}$ codeword no válidas.*
Entonces ¿cómo podemos corregir mediante la herramienta Síndrome si sus valores se repiten un montón de veces dentro del arreglo estándar? No se podría saber que error particular ocurrió ya que existe más de una alternativa posible para asignar al error. Para que se entienda, existen al menos dos filas en la matriz que tienen asociado el mismo síndrome. ¿A cuál culpamos de generar error? Siempre pero siempre que haya dos o más

posibilidades y no sabemos qué hacer ¿qué herramienta conocemos para el desempate? probabilidades de ocurrencia.



Pasos para la decodificación y corrección de errores

El proceso para la decodificación y corrección de error es el siguiente:

1. Calcular el síndrome del vector recibido “r” usando $S=rH^T$
2. Localizar el coset líder (patrón de error)
3. Este patrón de error se asume como el error causado por el canal
4. El vector recibido corregido, o codeword, se identifica como U=r+e. Se recupera la codeword valida mediante la suma del vector recibido con el error identificado. En realidad debería ser una resta pero en aritmética de módulo 2 es lo mismo suma que resta.

Implementación de Decoder

Cuando se tiene un código corto, como puede ser el caso de uno (6,3), se puede realizar una decodificación mediante una receta paso a paso de 4 etapas que implica la utilización de compuertas lógicas.

Antes debemos tener presente, cómo se obtenía el síndrome:

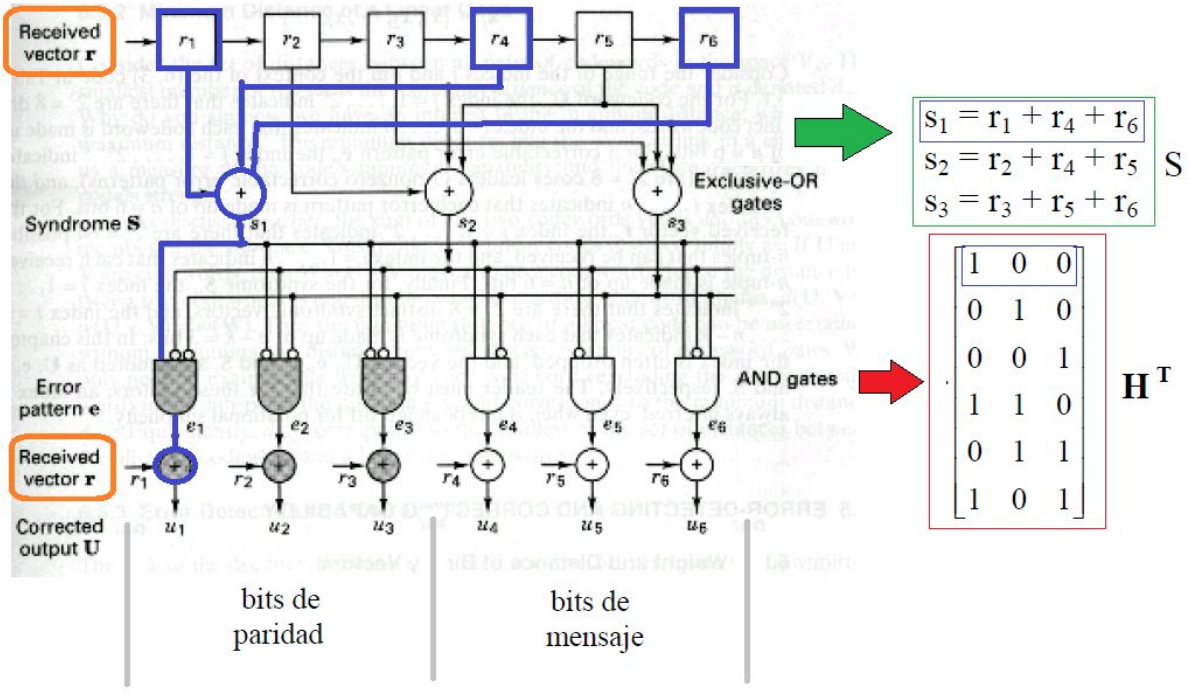
$$S = rH^T$$

$$S = [r_1 \quad r_2 \quad r_3 \quad r_4 \quad r_5 \quad r_6] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$s_1 = r_1 + r_4 + r_6$$
$$s_2 = r_2 + r_4 + r_5$$
$$s_3 = r_3 + r_5 + r_6$$

Considerando la obtención del síndrome, vemos que necesitaremos de los r_x que pueden ir cambiando respecto a lo que se transmita y la adición de ruido del canal pero en el caso de H^T siempre será estática porque ya ha sido predefinida, esto nos da un indicio de que se puede representar por compuertas lógicas. Además sabemos que los distintos síndromes también ya están definidos, generando otro tipo de compuertas, a lo que en un principio definimos como receta a realizar y como última etapa, sabemos que los patrones de error también están definidos, adicionando una etapa más de compuertas lógicas. Finalmente esto se ve expresado de la siguiente manera:

$$S = [r_1 \quad r_2 \quad r_3 \quad r_4 \quad r_5 \quad r_6]$$



Como se observa en la figura el vector recibido, entrara por 2 lados distintos, los cuales se pueden ubicar por los rectángulos naranja. En la parte superior, los r_x serán utilizados para calcular los síndromes, como se ve con S_1. En la parte inferior, se lo usará para corrección de error. Una vez que se ha obtenido el síndrome tras haber sumado los r correspondientes, se irá a la etapa del patrón de error, conformado por compuertas AND, en las que sus entradas son equivalentes a la matriz de la derecha yendo desde la primer fila representada en la AND de la izquierda y avanzando con la matriz hacia abajo en filas y hacia la derecha en compuertas. Para comprender esto, se puede observar que la compuerta AND de más a la izquierda tiene 2 entrada negadas, que se corresponden con el “100” de la matriz.

$$S = \mathbf{r}\mathbf{H}^{\mathbf{T}} = [r_1 \; r_2 \; r_3 \; r_4 \; r_5 \; r_6] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} H^T$$

$$S = (U + e)\, H^T$$

$$S = UH^T + eH^T$$

$$S = e.H^T$$ Operando Algebraicamente

$$SH = e.H^T H \;(1) \Rightarrow e = SH \Rightarrow S\,[S_1 S_2 S_3] \Big| e_1 e_2 e_3 e_4 e_5 e_6$$

$$H = \begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{array}$$

Se debe de observar que en el caso planteado se contemplan 6 compuertas ya que la cantidad de bits de la codeword es 6, pero sabemos que para 3 bits, se pueden considerar 8 tipo de errores distintos por lo que los 2 síndromes restantes que no son de patrón de error de un bit podrían ser considerados para el caso de no error y error de 2 bit. En los errores de 2 bits se debería realizar un diseño más complejo.
Para la segunda etapa del vector recibido o de la parte de abajo del circuito, ya sabemos de antes que r=U+e y que por modulo 2, $x \oplus x = 0$, entonces lo que se hace en el último sumador es dado el caso de que haya un valor a la salida de la AND, se le suma al vector recibido, de esta manera habremos sacado el error, siempre y cuando esté dentro de los patrones de error considerado y finalmente tendremos a la salida la U correcta.
Un detalle a tener en cuenta respecto a la figura es que todas las etapas sombreadas en la práctica no se implementan, ya que estas tienen la tarea de corregir los bits de redundancia o paridad y a nosotros solo nos interesa recuperar de manera debida los bits que llevan la información como lo son las 3 columnas de compuertas de la derecha.

Detección de error y Capacidad de corrección

Hasta el momento desarrollamos los conceptos de codificación de bloque lineal como un método de secuencia estructurada, eficiente en el uso del ancho de banda a comparación de la codificación de forma de onda y además con la capacidad de corregir ciertos errores que se producen en el canal. Especificamos las ventajas de utilizar una matriz generadora para reemplazar la tabla de asignación de mensaje digitales a codeword e introducimos el concepto de síndrome, como un vector de k elementos que acusa la presencia de errores en los vectores recibidos. Pero en ningún momento se especificó que tan eficientes son dichas herramientas. Se tomó un ejemplo para todo el desarrollo del tema, sin mirar si resultaba ser la mejor elección. Resulta que el uso de la matriz generadora, la matriz de chequeo de paridad y el concepto de síndrome quedan sujetos a la elección del subespacio vectorial válido a ser utilizado como codeword dentro del espacio vectorial de dimensión n que contiene todos los vectores codigo posibles.
Las únicas restricciones hasta el momento para el subespacio eran:

- El subespacio vectorial debe tener k elementos diferentes.
- Se debe incluir la codeword de todos ceros.
- La suma de dos codeword dentro del subespacio vectorial debe devolver una codeword que también se encuentre dentro.
- Las codeword válidas deben contener el mensaje digital.

Hasta el momento y con las consideraciones mencionadas, podemos encontrar varios grupos dentro del espacio n que cumplan con esas características, pero no se ha dicho que grupo puede funcionar mejor. Por lo que ahora comenzaremos a tener en cuenta ciertas condiciones adicionales, que nos permitirán optar por el mejor grupo para la codificación y decodificación. Comenzando con las restricciones por el concepto de pesos y distancias de vectores binarios denominadas como **distancias de Hamming**.
La distancia de Hamming entre dos codeword se define como el número de elementos en el cual difieren. Dada las siguientes codeword, se verifica que la distancia de hamming es:

$$\mathbf{U} = 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1$$
$$\mathbf{V} = 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0$$
$$d(\mathbf{U}, \mathbf{V}) = 6$$

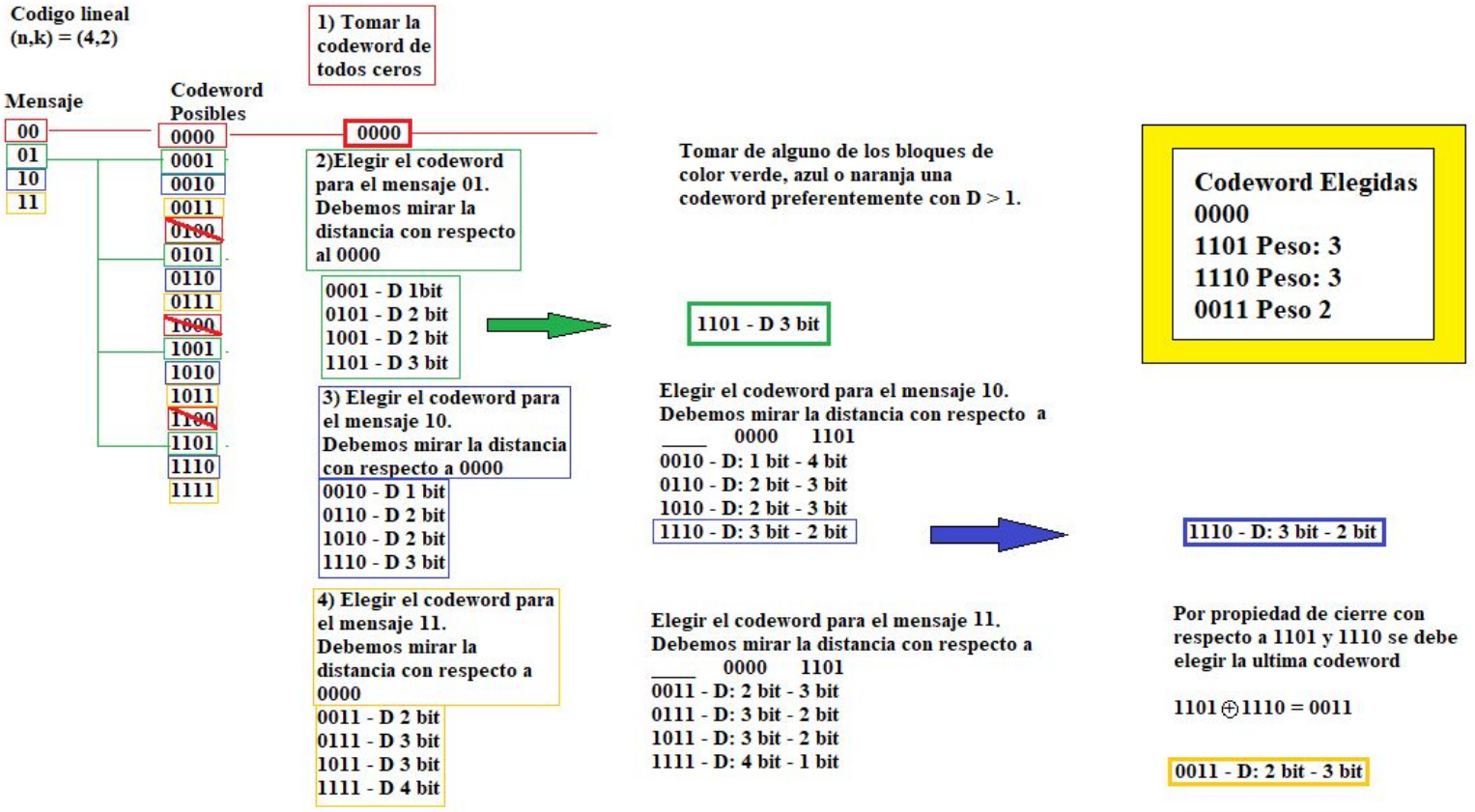
Esta operación de determinar la cantidad de posiciones en las cuales difieren los bits de ambas codeword puede ser pensada de otra forma. Para ello, debe realizarse en primer lugar, la suma de módulo dos de ambas codeword y luego contar la cantidad de unos presentes en la misma. ¿A qué se debe esto? A la propiedad de la suma módulo 2.
Al sumar dos vectores binarios, se genera otro vector, cuyos unos binarios están localizados en las posiciones donde ambos vectores difieren. Para determinar la distancia de Hamming es necesario solo contar la cantidad de unos binarios presentes en este último vector.
De esta forma, se puede introducir otro concepto más, los **pesos de hamming.** El peso de hamming de una codeword es el número de elementos distintos de cero en la codeword. Para un vector binario es equivalente al número de unos en el vector.
Queda claro ahora cual es la definición y relación para ambos conceptos:

- *Distancia de hamming* → Cantidad de posiciones en las cuales difieren los elementos de dos codeword.
- *Pesos de hamming* → Cantidad de unos presentes en una codeword determinada. Puede pensarse como la distancia de hamming entre una codeword con la codeword de todos ceros.

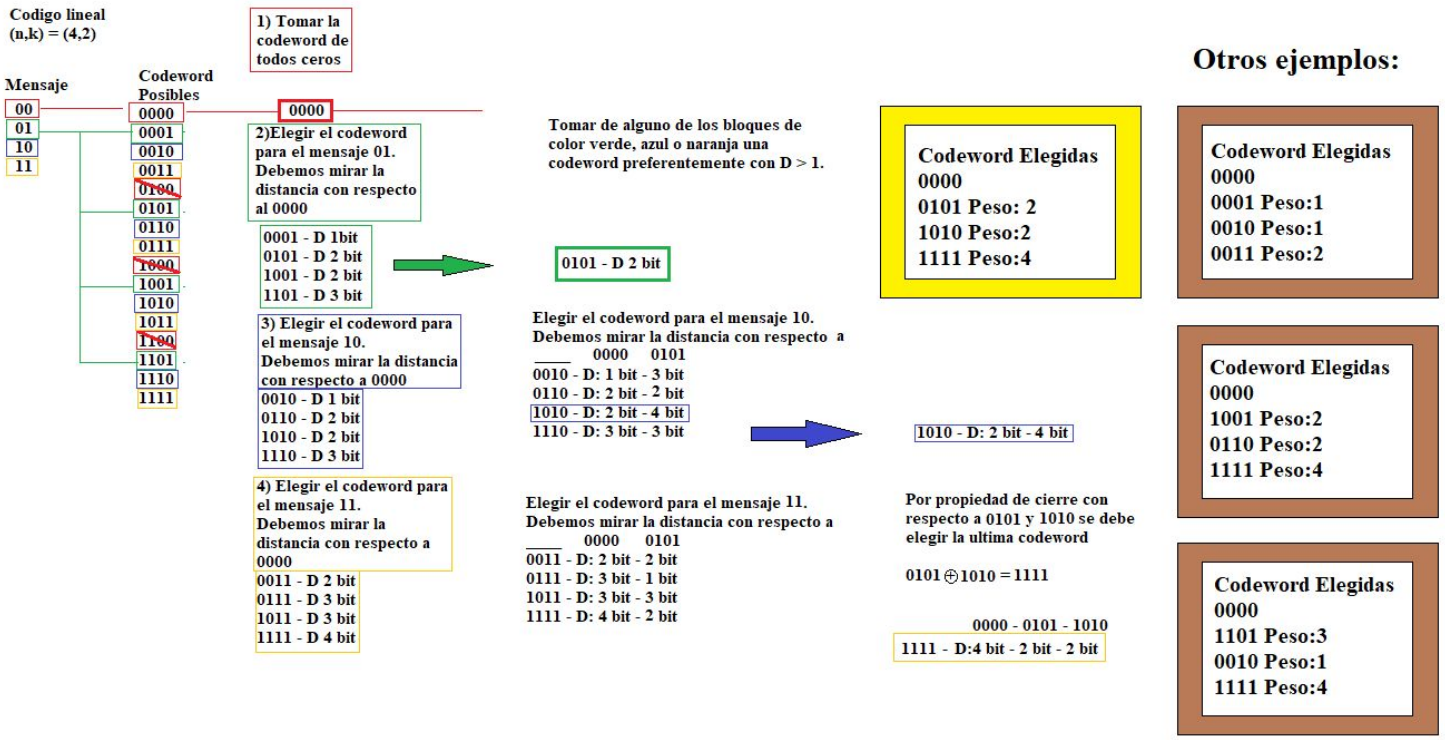
La distancia de hamming entre las codewords “U” y “V” puede calcularse a partir del concepto de pesos de hamming, si se utiliza la suma de los dos codeword d(U,V)=w(U,V).

¿Cómo aplicamos este concepto a Codificación por bloques lineales? Debemos calcular la distancia de hamming entre todas las codewords presentes en el subespacio vectorial elegido y *tomaremos como dato la distancia obtenida más pequeña*. A dicho valor, le denominaremos d_{min}. ¿Por que nos enfocamos en dicho valor y no en la distancia máxima? Porque la distancia mínima es el eslabón más débil de todo el set, es decir, es la combinación de codewords que resultan ser más parecidas y como consecuencia donde hay más probabilidad de error.

Entonces d_{min} nos da una medida de la capacidad mínima del código y por consiguiente caracteriza la fuerza del código.
¿Estas diciendo que si tenemos 2^k codewords diferentes de n elementos cada una, se debe calcular la distancia de hamming para todas las combinaciones posibles, es decir $\binom{2^k}{2}$? SI. Usando el concepto de distancias de hamming, pero si se utilizan como referencia los pesos de hamming solo es necesario calcular dicho parámetro para cada codeword. De todos los resultados obtenidos, el que tenga menor peso de hamming podría ser asociado a la distancia de hamming mínima.



Como se puede observar en la imagen, existen un montón de posibilidades para conformar el subespacio valido de 2^k codewords dentro del espacio de dimensión n que utiliza el codificador, algunos ejemplos se muestran a continuación.



Deteccion y correcion de errores

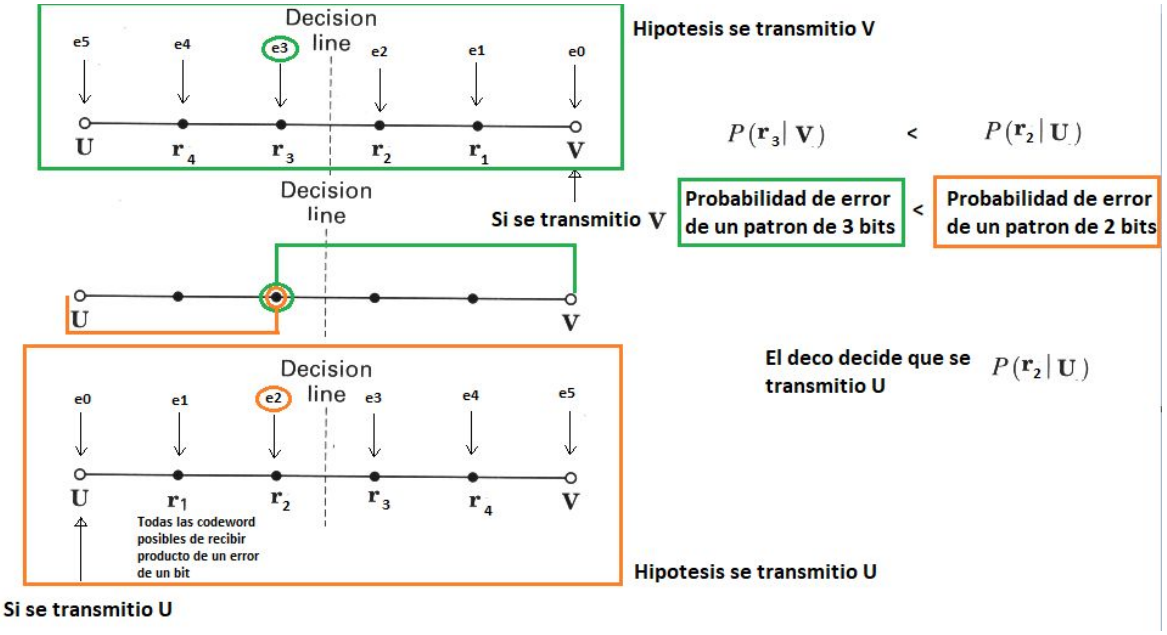
La tarea del decoder, teniendo el vector “r” (recibido), es estimar el codeword transmitido U_i. La estrategia óptima del decoder se puede expresar en términos del algoritmo de “máxima probabilidad”. Esto significa que cuando ocurren errores por culpa del canal, el decodificador, toma la codeword recibida y la compara con cada una de las codeword válidas del subespacio vectorial. El decoder, va a elegir aquella codeword del set que al compararla con la codeword recibida entregue el valor más alto. Esto se debe a que la probabilidad de ocurrencia de patrones de errores más pequeños es más alta que la probabilidad de ocurrencia de los patrones de errores más grandes. En otras palabras, es esperable que la codeword recibida esté contaminada con pequeños errores.
En resumen, este procedimiento no es más ni menos que la comparación de la probabilidad de ocurrencia de las diversas codewords distorsionadas dado que se transmitió determinada codeword.

$$(P(r \mid U_i) = \max P(r \mid U_j) \quad \text{sobre todo } U_j$$

Además, como el canal es simétrico binario (BSC), la probabilidad de U_i con respecto a “r” es inversamente proporcional a la distancia entre r y U_i. De esta forma, se concluye que:

$$d(r,U_i) = \min d(r,U_j) \quad \text{sobre todo } U_j$$

En otras palabras, el decoder determina la distancia entre r y cada uno de los posibles codeword transmitidos U_j y selecciona como más probable la U_i cuya distancia sea la menor al vector r recibido.



Ojo, si la distancia entre dos o más codeword es la misma en relación con "r", el decodificador toma una decisión arbitraria. Este esquema permite observar que el decodificador es capaz de realizar corrección de errores hasta la línea de decisión, que resulta la mitad de la distancia de hamming mínima entre codeword. En general, la capacidad de corrección de error "t" de un código se define como el máximo número de errores corregibles garantizados por codeword, y se escribe como:

$$t = \left(\frac{d_{min} - 1}{2} \right)$$

Siempre dicho número es el entero más grande necesario para no exceder "t". Si da t=2.5 se elige como t=2. Se debe tener en cuenta que la distancia mínima para contar con capacidad de corregir errores debe tener un valor de d_{min}>2, si diera el caso que d_{min}=2, obtendremos t=0.5 y por lo tanto se tomará a t=0, dejándonos sin capacidad de corrección.

Si un bloque de códigos lineal, es utilizado para corregir "t" errores sobre un canal simétrico binario (BSC) con probabilidad de transición "p", la probabilidad de mensaje erróneo, P_M, lo cual significa que el decodificador haga una decodificación errónea se puede calcular mediante la siguiente ecuación:

$$P_M \leq \sum_{j=t+1}^{n} \binom{n}{j} p^j (1-p)^{n-j}$$

La probabilidad de error de bit decodificado P_B, depende de un código y un decoder particular. Esto se puede expresar mediante la siguiente ecuación:

$$P_B \approx \frac{1}{n} \sum_{j=t+1}^{n} j \binom{n}{j} p^j (1-p)^{n-j}$$

Todo bloque de código lineal necesita detectar los errores antes de corregirlos. O, puede usarse para detección de errores solamente. La capacidad de detectar errores se mide en función de d_{min}.

$$e = d_{min} - 1$$

Un código de bloque lineal con distancia d_{min} garantiza que todos los patrones de errores de d_{min}-1 o menos errores pueden ser detectados. Semejante codigo tambien es capaz de detectar una larga fracción de patrones de error con d_{min} o más errores. De hecho, un código (n,k) es capaz de detectar 2^n-2^k patrones de error de largo n. ¿Por qué? Hay una cantidad total de 2^n-1 posibles patrones de error distintos de cero en el espacio de 2^n. Incluso el patrón de bit de un codeword valido representa un patrón potencial de error de bit. De esta manera, hay 2^k-1 patrones de error que son idénticos a los 2^k-1 codeword distintos de cero. Si alguno de esos 2^k-1 patrones de error ocurre, se altera la codeword transmitida de U_i a U_j, la cual también es una codeword válida dentro del subespacio vectorial elegido para la codificación. El decoder acepta a U_j como el codeword transmitido y se equivoca en su decisión. Por lo tanto hay 2^k-1 patrones de error indetectables. Si el patrón de error no es idéntico a uno de los 2^k codewords válidas el error es detectado. Concluimos entonces que existen 2^n-2^k patrones de error detectables.

Detección y corrección de error simultáneo

Un codigo cuenta con la capacidad de hacer corrección, asociado a α y detección asociado a β en donde estos parámetros serán siempre enteros y $\beta \geq \alpha$ debido a que siempre se debe detectar para poder realizar una corrección. Ambos están asociados con la negociación de "t" que era la capacidad máxima de corrección. Podemos definir a la distancia mínima con respecto a estos parámetros como:

$$d_{min} \geq \beta + \alpha + 1$$

Mientras ocurran t o menos errores, el código tendrá la capacidad de detectar y corregirlos pero si sucedieran más errores, todos no podrían ser salvados. Por lo que también definiremos a la capacidad de detección de error ("e") en relación con d_{min}:

$$e = d_{min} - 1$$

Cuando ocurre una cantidad de errores mayor a t ($(d_{min}-1)/2$) pero menor a e+1 (d_{min}) el código es capaz de detectar los errores pero cuenta con la capacidad de corregir solo alguno de ellos. Por ejemplo, si suponemos un código con d_{min}=7, se lo puede utilizar para detectar y corregir simultáneamente en cualquiera de las siguientes formas

Detectado(β)	Corregido(α)
3	3
4	2
5	1
6	0

$$d_{min} \geq \alpha + \beta + 1$$

$$\mathbf{7 >= 2 + 4 + 1}$$

$$e = d_{min} - 1$$

$$\mathbf{6 = 7 - 1}$$

$$t = \left[\frac{d_{min} - 1}{2} \right]$$

$$\mathbf{3 = [(7-1)/2]}$$

Corrección de Codeword difusas

A un receptor se lo prepara para reconocer 2 tipos de símbolos, asociados al 0 o 1, dado el caso de que no pueda ser distinguido se lo determina como difuso. Esto puede ocurrir culpa de una recepción ambigua, presencia de interferencia o malfunción transitoria. Si sucediera alguno de estos 3 casos, se sacará un símbolo llamado erasure (borradura) o difuso γ. Para corregir este símbolo, en el caso binario, sólo se necesita saber la localización como dato brindado al decodificador ya que los valores posibles son únicamente 2.
Un código de control de error puede ser usado para corregir símbolos difusos γ o para corregir errores y símbolos difusos simultáneamente, siendo este último caso más complejo. Si el código cuenta con una d_{min}, cualquier modelo con ρ o menos símbolos difusos puede ser corregido si:

$$d_{min} \geq \rho + 1$$

Si un código no cuenta con errores, sino que únicamente con símbolos difusos, se puede llegar a reconstruir hasta d_{min}- 1. Si tuviéramos el caso de errores y símbolos difusos, se puede llegar a corregir simultáneamente hasta:

$$d_{min} \geq 2\alpha + \gamma + 1$$

La implementación de esta corrección consiste en asignar en un primer caso ceros en donde hay símbolos difusos y corregir los errores, mientras que el segundo caso el procedimiento es idéntico solo que se asignan 1 en los difusos. Después se compara a cual se le tuvo que corregir menos errores con respecto a las codewords válidas y se opta por la que menos correcciones tuvo sin considerar los símbolos difusos.

Utilidad del arreglo Estándar

Tanto para códigos pequeños como para codigos grandes, refiriéndose a los valores de (n,k) de un codificador de bloques lineal, la matriz de arreglo estándar permite visualizar beneficios importantes en la performance, el posible trade- off entre la corrección de errores y la detección y el límite de capacidad de corrección de error.
Uno de los límites de corrección de error es conocido como límite de Hamming:

Números de bits de paridad: $$n-k \geq \log_2\left[1+\binom{n}{1}+\binom{n}{2}+.....+\binom{n}{t}\right]$$

o

Número de cosets $$2^{n-k} \geq \left[1+\binom{n}{1}+\binom{n}{2}+.....+\binom{n}{t}\right]$$

Estas ecuaciones vinculan de forma indirecta todas las herramientas que hemos detallado para codificación de bloques lineal. Si se desea que el código tenga una capacidad "β" de detección de errores y una capacidad "α" para corregir, la distancia mínima como se sugirió anteriormente debe ser $d_{min} \geq \beta + \alpha + 1$. Con dicha distancia mínima, la capacidad de

corrección definida por “t” es : ((d_{min}-1)/2). Entonces bajo esas condiciones se podría usar las ecuaciones de límite de hamming y obtener la relación “n-k” necesaria para construir el codificador.
Notar que las ecuaciones asociadas al límite de hamming proporcionan el mínimo número de filas necesarias en el arreglo estándar para corregir todas las combinaciones de errores completas de “t” bits. La igualdad da un límite inferior de “n-k”, es decir, el número de bits de paridad como una función de la capacidad de corrección de error de t-bits.
Para decir que un código de bloque lineal tiene capacidad de corrección de error de t bits necesita cumplir si o si con el límite de hamming.

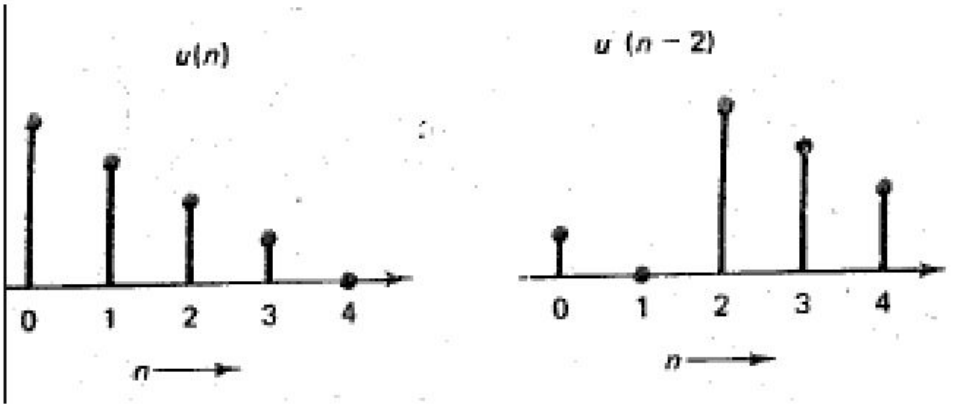
¿Con esto ya esta, solo se necesita cumplir con el límite de hamming? **NO**. Existe otro límite importante que los códigos de bloque lineal tienen que satisfacer si desean corregir todos los errores posibles asociados a un patrón de error de t. Dicho límite es conocido como Límite de Plotking:

$$d_{\min} \leq \frac{n \times 2^{k-1}}{2^k - 1}$$

Código Cíclico.
Este tipo de código es una importante subclase del bloque de códigos lineales. Como su nombre lo induce, se utiliza una realimentación de un registro cambiante. *La estructura fundamental de este código es que tiene un método de decodificación eficiente.*
Para generar o hablar de un código cíclico debemos cumplir con estas características:

→ Si la n-tuple U=(u_0,u_1,...,u_{n-1}) es una codeword en el subespacio S, luego $U^{(1)}$=(u_{n-1},u_0,u_1,...,u_{n-2}), obtenida por un cambio al rotar en el final del vector (clave de lo cíclico), es también una codeword en S.

Un ejemplo de $U^{(2)}$, dada una codeword U:



Generalizando, $U^{(i)}$=(u_{n-i},u_{n-i-1},u_{n-i-1},..,$u_{0,}u_{1,}$..,u_{n-i+1}) obtenida por el i-ésimo cambio cíclico es también una codeword en S.

Las componentes de una codeword U, pueden ser tratados como los coeficientes de un polinomio U(X) de la forma:

$$\mathbf{U}(X)=u_0 + u_1 X + u_2 X^2 + \ldots + u_{n-1} X^{n-1}$$

En donde la presencia o ausencia de cada término en el polinomio indica la presencia de 1 o 0 binario. Si la componente u_{n-1} del vector U es distinta de cero, entonces el polinomio es de grado n-1. Esta descripción polinómica es útil para entender la estructura algebraica de los códigos cíclicos. Traeme la parte más fea del capitulo, no, no tanto.

Estructura algebraica de códigos Cíclicos:
Más allá de la matemática y álgebra que veremos en esta sección, ¿Qué estamos buscando? *Expresar la forma en la que se generan las codeword en este tipo de codificación, es decir, expresar matemáticamente el ciclo natural de las codeword. Como se obtiene matemáticamente $U^{(i)}(X)$.*

→ Dado U(X) como una codeword expresada mediante un polinomio de grado n-1, entonces podemos expresar a $U^{(i)}(X)$ de la siguiente forma:

$$\frac{X^i \mathbf{U}(X)}{X^n+1} = \mathbf{q}(X) + \frac{\mathbf{U}^{(i)}(X)}{X^n+1}$$

¿Qué significa esto? Que $U^{(i)}(X)$ forma parte de una expresión particular. El resultado de una división para ser más preciso. Dicha operación, involucra el cociente entre dos polinomios. Uno de ellos es $X^{(i)}$. U(X) y el otro X^n+1. El resultado es expresado como la suma del cociente propio de la división q(X) y otro término, que se expresa como otro cociente (Resto y divisor). Dentro del último término, tenemos el dato buscado $U^{(i)}(X)$, precisamente en el resto.

$$\frac{\text{Dividendo}}{\text{Divisor}} = \text{Cociente} + \frac{\text{Resto}}{\text{Divisor}}$$

$$\frac{X^i \mathbf{U}(X)}{X^n+1} = \mathbf{q}(X) + \frac{\mathbf{U}^{(i)}(X)}{X^n+1}$$

Al operar matemáticamente y pasar multiplicando el divisor, encontramos:

$$X^i \mathbf{U}(X) = \mathbf{q}(X)\,(X^n+1) + \underbrace{\mathbf{U}^{(i)}(X)}_{\text{Resto}}$$

Esto significa que $U^{(i)}(X)$ puede ser encontrado en el resto de la división entre $X^{(i)}$. U(X) y X^n+1. Generalmente se expresa de otra forma. Mediante módulo aritmético dicha relación puede expresarse de la siguiente forma:

$$\mathbf{U}^{(i)}(X) = X^i\,\mathbf{U}(X) \text{ modulo } (X^n+1)$$

Se demostrará mediante un ejemplo para el caso de i=1. En primer lugar, se expresará a U(X) de forma genérica como un polinomio de n componentes donde el orden máximo del polinomio es n-1. Luego, se confeccionará la expresión asociada al polinomio resultante de operar $X^{(i)}$ U(X) para el caso buscado, que es i=1.

→ Para U(X)

$$\mathbf{U}(X) = u_0 + u_1X + u_2X^2 + \cdots + u_{n-2}X^{n-2} + u_{n-1}X^{n-1}$$

→ Para X.U(X)

$$\mathbf{U}(X) = u_0 + u_1X + u_2X^2 + \cdots + u_{n-2}X^{n-2} + u_{n-1}X^{n-1}$$

$$X\mathbf{U}(X) = u_0 \,(X) + u_1X \,(X) + u_2X^2 \,(X) + \cdots + u_{n-2}X^{n-2}\,(X) + u_{n-1}X^{n-1}\,(X)$$

$$X\mathbf{U}(X) = u_0X + u_1X^2 + u_2X^3 + \cdots + u_{n-2}X^{n-1} + u_{n-1}X^n$$

¿Qué hacemos con esta expresión final? Sumamos y restamos simultáneamente sobre X.U(X) el término u_{n-1} . Recordemos que por aritmética módulo 2, las restas son sumas.

$$X\mathbf{U}(X) = \underbrace{u_{n-1} + u_0X + u_1X^2 + u_2X^3 + \cdots + u_{n-2}X^{n-1}}_{\mathbf{U}^{(1)}(X)} + u_{n-1}X^n + u_{n-1}$$

Luego de sumar y restar, se observa que si se agrupan los términos como en la imagen mostrada anteriormente, resulta $U^{(1)}(X)$ porque es la expresión polinómica de U(X) mediante una simple rotación cíclica, el último pasa a ser el coeficiente independiente del polinomio y el término u_{n-1} queda afectado por el coeficiente más alto del polinomio de U(X), es decir, n-1. ¿El otro término en que resulta?

$$u_{n-1}X^n + u_{n-1} \Rightarrow u_{n-1}(X^n + 1)$$

Factor Comun

Reescribiendo la expresión completa:

$$X\mathbf{U}(X) = \mathbf{U}^{(1)}(X) + u_{n-1}(X^n + 1)$$

Al mirar esta expresión, vemos que $U^{(1)}(X)$ es de grado n-1, por lo tanto no puede ser dividida por (X^n+1), entonces no nos queda otra que pensar que dicha expresión surge de otra división. Surge de la división de X U(X) porque tiene el mismo grado que (X^n+1). MATEMATICA PURA… NO ENROSQUEIS. Se llega a la misma expresión que antes

$$\mathbf{U}^{(1)}(X) = X\mathbf{U}(X) \text{ modulo } (X^n + 1)$$

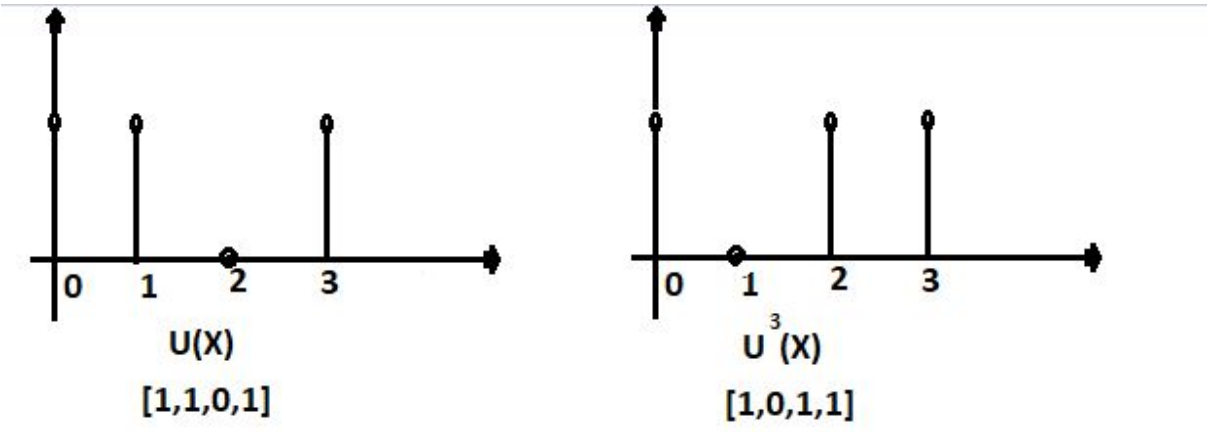
Extendiendo el resultado:

$$\mathbf{U}^{(i)}(X) = X^i\mathbf{U}(X) \text{ modulo } (X^n + 1)$$

Resumen: Hemos demostrado que para generar diferentes codeword mediante codificación cíclica es necesario generar una rotación cíclica sobre la codeword. Ese es el concepto, pero matemáticamente hablando, esto significa multiplicar por X^i al polinomio asociado a la codeword válida y luego dividir dicha expresión por (X^n+1) para obtener el resto, que es $U^{(i)}(X)$. El polinomio con la rotación cíclica. **Hay que quedarse con el concepto.**

Veamos un ejemplo aplicando el concepto y luego la matemática.
Ejemplo: Dado U=[1,1,0,1], para n=4. Hay que lograr expresar la codeword de forma polinómica y resolver un tercer cambio cíclico de la codeword.

Conceptualmente debemos generar:



Mediante la matemática podemos resolver:

$$\mathbf{U}(X) = 1 + X + X^3$$

$$X^i\mathbf{U}(X) = X^3 + X^4 + X^6,$$

$$\mathbf{U}^{(i)}(X) = X^i\mathbf{U}(X) \text{ modulo } (X^n + 1)$$

$$X^6 + X^4 + X^3 \quad \big| \quad X^4 + 1$$

$$-\ X^6 + X^4 + X^3 \quad \big| \quad X^4 + 1$$

$$X^6 \quad + X^2 \qquad\qquad X^2 + 1$$

$$X^4 + X^3 + X^2$$

$$X^4 \quad + 1$$

$$X^3 + X^2 + 1 \qquad \text{remainder } \mathbf{U}^{(3)}(X)$$

Dividimos X^6 con X^4 ⇨ X^2

$X^2\ [X^4 + 1]\quad X^6 + X^2$

Dividimos X^4 con X^4 ⇨ 1

$1\ [X^4 + 1]\quad X^4 + 1$

Reescribiendo el polinomio que tenemos en el resto de la división de menor a mayor tenemos: $1 + X^2 + X^3$, lo cual muestra que es $U^3(X)$= [1,0,1,1]. Conclusión, la matemática nos llevó operacionalmente al mismo lugar que el análisis conceptual. No nos peleemos con los polinomios.

Para utilizar este tipo de codificación se podría pensar en un polinomio generador en lugar de una matriz generadora. No olvidemos que este tipo de codificación no deja de ser una codificación de bloques lineales pero más eficiente en la decodificación.
El polinomio generador g(X) para un código cíclico (n,k) es único y de la forma:

$$\mathbf{g}(X) = g_0 + g_1X + g_2X^2 + \cdots + g_pX^p$$

Donde necesariamente g_o y g_p deben ser diferentes de cero para ser considerado polinomio generador. Esta condición es re importante, porque inspeccionando el polinomio en cuestión a simple vista podemos deducir si es un fiel candidato o no.
Toda codeword dentro del subespacio vectorial válido en el espacio de dimensión n puede expresarse como:

$$\mathbf{U}(X) = \mathbf{m}(X)\mathbf{g}(X)$$

Tengamos en cuenta que estamos trabajando con polinomios, entonces hay que tener cuidado al definir el orden de m(X). ¿Que orden tendrá m(X)? para entender la cantidad de bits que puede tener el mensaje digital para un polinomio generador de orden p debemos inspeccionar el orden de las codewords. Se había especificado anteriormente que las codeword tenían un orden de n-1 porque la cantidad de bits de las codeword era n. Entonces con un polinomio de orden p como polinomio generador, no queda otra que el orden de los mensajes digitales sea n-p-1. ¿WTF?! ¿como te das cuenta? Mediante la ecuación de U(X) que expresa el producto entre m(X) y g(X). Se debe realizar una distributiva. En dicha distributiva, se generan un montón de términos con diferentes órdenes. Sin embargo, hay que considerar para el peor caso, aquellos términos que resultan del producto de los elementos de mayor orden de m(X) con los elementos de g(X). Encontramos que en dicho producto cuando se realiza $g_p.X^p$ con $m_?X^?$ se tiene que cumplir que sea igual a $u_{n-1}.X^{n-1}$. En otras palabras, es lo que se observa en la imagen:

$$\mathbf{U}(X) = \mathbf{m}(X)\mathbf{g}(X)$$

$$\mathbf{g}(X) \quad g_0 + g_1X + g_2X^2 + \cdots + g_pX^p$$

$$\mathbf{m}(X) = m_0 + m_1X + m_2X^2 + \cdots + m_? X^?$$

$$\mathbf{U}(X) = u_0 + u_1X + u_2X^2 + \cdots + u_{n-1}X^{n-1}$$

Caso critico en el producto.. Se debe cumplir que dicho producto genere X^(n-1)

$$X^p \, X^? = X^{n-1}$$

$$p + ? = n-1$$

$$? = n-p-1$$

Entonces podemos expresar a m(X) como:

$$\mathbf{m}(X) = m_0 + m_1X + m_2X^2 + \cdots + m_{n-p-1}X^{n-p-1}$$

Como el orden del polinomio de mensaje digital es (n-p), hay 2^{n-p} , es decir, 2^k polinomios codeword dentro de un bloque de codificación cíclica (n,k). Entonces podemos expresar:

$$n - p = k$$

despejando p

$$p = n - k$$

Por lo tanto, g(X) debe ser de orden (n-k). Haber, haber , haber ¿que tanta ciencia? Lo único que estamos diciendo, es que para ser consistentes en el laburo matemático en un sistema de codificado cíclico (n,k), para generar codewords de n elementos o dígitos binarios, es necesario mensajes digitales de k elementos como siempre, pero el polinomio generador debe tener una cantidad de elementos o dígitos binarios (n-k).

Codificación en forma sistemática

Dado un sistema de codificación cíclica de la forma (n,k), se puede expresar el mensaje polinomial de la siguiente forma:

$$\mathbf{m}(X) = m_0 = m_1X + m_2X^2 + \cdots + m_{k-1}X^{k-1}$$

En la forma sistemática, el mensaje digital se utiliza como parte de la codeword. En códigos de bloque lineal, la codeword contenía en sus primeros bits, en particular en sus (n-k) primeros bits, los bits de paridad y en los últimos k bits se encontraba el mensaje digital. De la misma forma, para el caso de codificación cíclica se podría pensar en desplazar al mensaje digital (n-k) dígitos hacia la derecha para agregar redundancia al comienzo de la codeword.

$$X^{n-k}\,\mathbf{m}(X) = m_0\,X^{n-k} + m_1\,X^{n-k+1} + \ldots + m_{k-1}\,X^{n-1}$$

Si se divide la ultima ecuacion por g(X), el resultado puede expresarse como:

$$\mathbf{p}(X) = p_0 + p_1 X + p_2 X^2 + \ldots + p_{n-k-1} X^{n-k-1}$$

$$X^{n-k}\ \mathbf{m}(X) = \mathbf{q}(X)\ \mathbf{g}(X) + \mathbf{p}(X)$$

Entonces:

$$\mathbf{p}(X) = X^{n-k}\ \mathbf{m}(X) \text{ módulo } \mathbf{g}(X)$$

Sin embargo, resulta de interés trabajar con la ecuación anterior y sumarle a ambos lados p(X). De esta forma:

Suma de aritmetica de modulo 2

$$\mathbf{p}(X) + X^{n-k}\mathbf{m}(X) = \mathbf{q}(X)\mathbf{g}(X) + \mathbf{p}(X) + \mathbf{p}(X)$$

0

$$\mathbf{p}(X) + X^{n-k}\mathbf{m}(X) = \mathbf{q}(X)\mathbf{g}(X)$$

$$\mathbf{q}(X)\mathbf{g}(X) = \mathbf{U}(X)$$

$$\mathbf{p}(X) + X^{n-k}\mathbf{m}(X) = \mathbf{U}(X)$$

La última expresión se reconoce como una codeword polinómica válida, ya que esto es un polinomio de grado n-1 o menor, y cuando dividimos por g(X) hay resto cero. Esta codeword puede ser expandida en términos polinómicos de la siguiente forma:

$$\mathbf{p}(X) + X^{n-k}\ \mathbf{m}(X) = p_0 + p_1 X + \ldots + p_{n-k-1} X^{n-k-1} + m_0 X^{n-k} + m_1 X^{n-k-1} + \ldots + m_{k-1} X^{n-1}$$

La codeword polinómica corresponde a un vector codificado con las siguientes características:

$$\mathbf{U} = (p_0, p_1, \ldots\ldots\ldots, p_{n-k-1}, m_0, m_1, \ldots\ldots, m_{k-1})$$

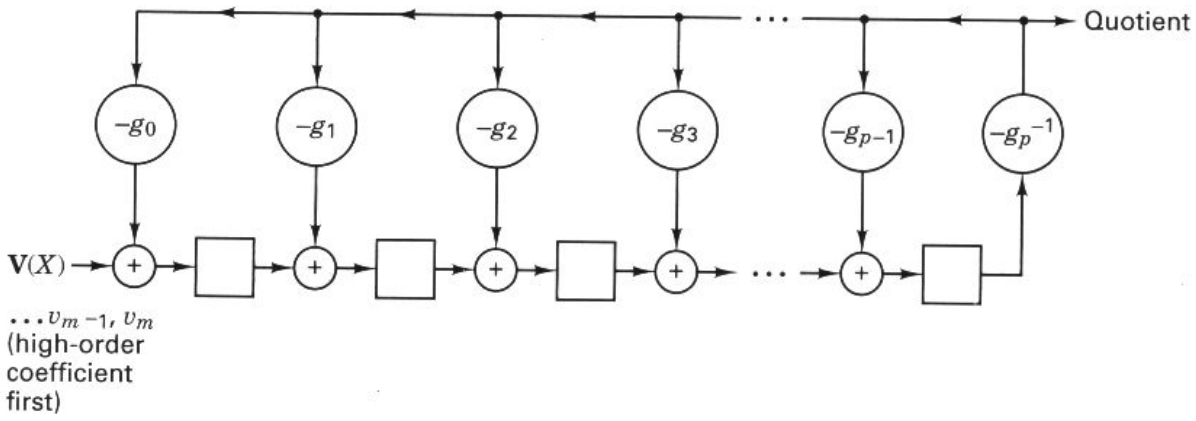
n-k bits de paridad k bits del mensaje

La implementación de este tipo de codificadores consiste en un circuito divisor digital. Ya que los cambios cíclicos de una codeword polinomial y la codificación de un mensaje polinomial implican la división de un polinomio por otro. Dados los polinomios V(X) y g(X), donde:

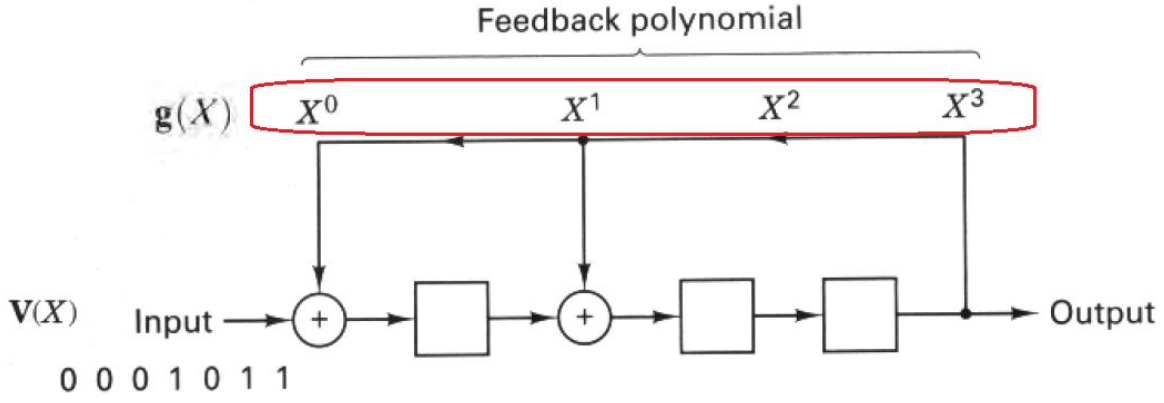
$$\mathbf{V}(X) = v_0 + v_1 X + v_2 X^2 + \cdots + v_m X^m$$

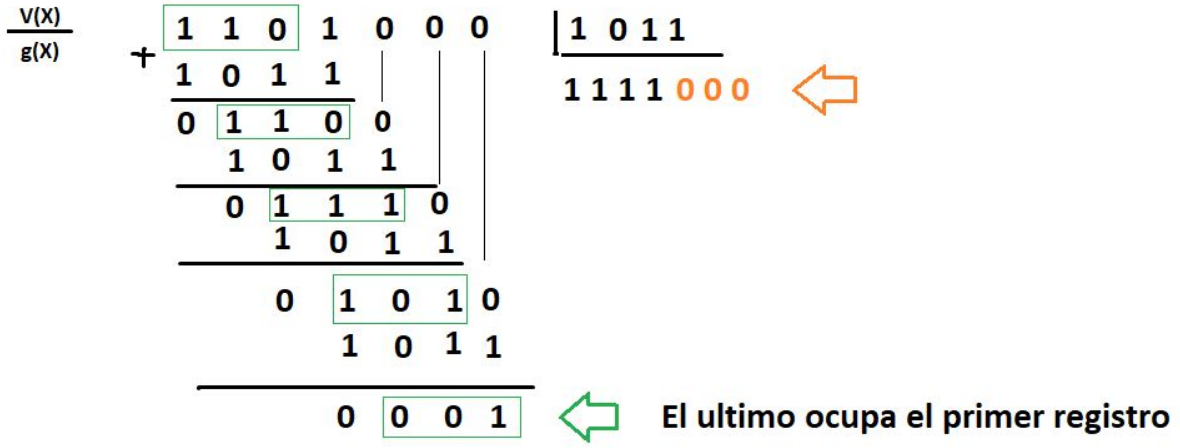
$$\mathbf{g}(X) = g_0 + g_1 X + g_2 X^2 + \cdots + g_p X^p$$

El circuito implementado consiste:



Un ejemplo mostrará mejor el comportamiento del circuito divisor. En donde tenemos un $g(x)=(1*x^0+1*x^1+0*x^2+1*x^3)= (1+x+x^3)$ por lo que contaremos con 3 registros pero 2 sumadores, ya que X^2 es cero. El $v(x) = (0*x^0+0*x^1+0*x^2+1*x^3+0*x^4+1*x^5+1*x^6)= (x^3+x^5+x^6)$.
Como el máximo orden de g(x) (x^3) y por lo tanto 3 registros, hasta que no haya 3 desplazamientos, al menos, la salida será siempre cero. Como se verá en el ejemplo en color naranja.
Una vez que se llega al 4to desplazamiento, se dividen los coeficientes de g(x) con respecto a v(x) comenzando por los de mayor orden. Antes de realizar esta división, tenemos en los registros el valor “011”, leído de derecha a izquierda, correspondiente al rectángulo verde de la primer fila en el ejemplo. Al realizar la división, tendremos a la salida un 1 y como nuevo contenido de los registros “011” correspondiente a la 3er fila del ejemplo. Este procedimiento se irá haciendo sucesivamente hasta llegar al coeficiente de menor orden.





En el decodificador se puede aplicar los conceptos de síndrome para detectar si una codeword recibida es correcta o no.
Se sabe que :

$$\mathbf{U}(X) = \mathbf{m}(X)\mathbf{g}(X)$$

Donde z(X) es la representación de forma polinómica del vector recibido:

$$\mathbf{Z}(X) = \mathbf{U}(X) + \mathbf{e}(X)$$

Donde e(X) es el patrón de error expresado de forma polinómica. El decodificador chequea si z(X) es una codeword polinómica, esto es, si es divisible por g(X), con un resto cero. Esto es en esencia el cálculo del síndrome del polinomio recibido, el cual implica directamente la división entre z(X) y g(X).

$$\frac{\mathbf{Z}(X)}{\mathbf{g}(X)} = \mathbf{q}(X) + \frac{\mathbf{S}(X)}{\mathbf{g}(X)}$$

$$\mathbf{Z}(X) = \mathbf{q}(X)\mathbf{g}(X) + \mathbf{S}(X)$$

donde S(X) es un polinomio de orden n-k-1 o menos. Así, de esta forma, el síndrome resulta ser un vector de n-k elementos, permitiendo detectar y corregir 2^{n-k} errores diferentes.
El cálculo del síndrome es determinado por un circuito divisor, casi idéntico al circuito de codificación usado en el transmisor.

Código de hamming

Los códigos de hamming son una clase particular de códigos cuya estructura es:

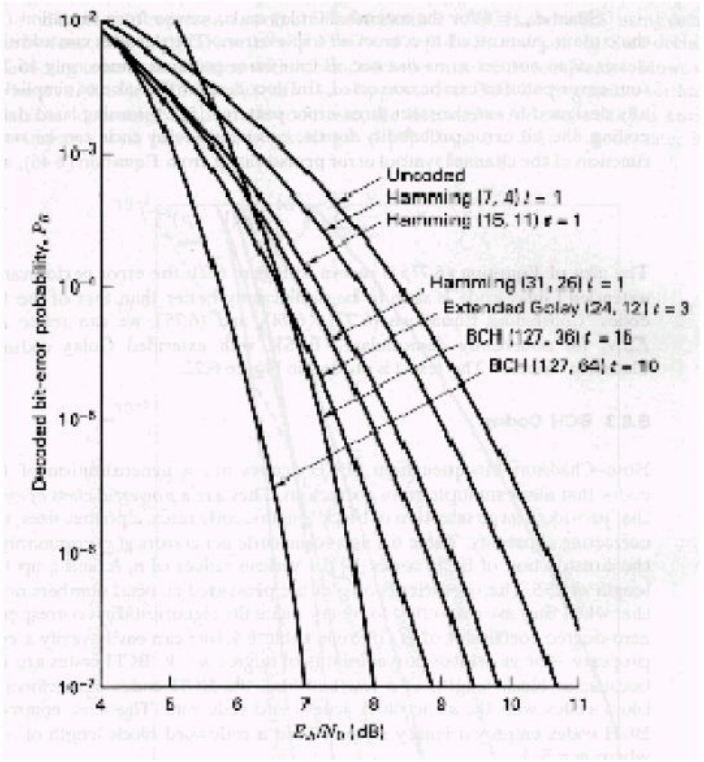
$$(n, k) = (2^m - 1, 2^m - 1 - m) \qquad m = 2, 3, \ldots .$$

Estos códigos tienen una distancia mínima de 3 bits y por las ecuaciones asociadas a la capacidad de corrección de errores “t” y la cantidad de patrones de errores entre codeword “e” = dmin-1 se puede deducir que son capaces de corregir todos los errores simples o detectar todas las combinaciones de dos o menos errores dentro de un bloque. Esto último está asociado a que toda codificación tiene su distancia mínima vinculada a $"\alpha"\ y\ "\beta"$ de la siguiente forma $d_{min} \geq \beta + \alpha + 1$.
La particularidad de este tipo de bloque de codificación es que son ***códigos perfectos.*** Esto significa que la matriz de arreglo estándar contiene todos los patrones de errores de t y menores errores y no otros de los coset líderes.
Asumiendo una decodificación por decisión hard, la probabilidad de error de bit puede expresarse como:

$$P_B \approx \frac{1}{n} \sum_{j=2}^{n} j\binom{n}{j} p^j (1-p)^{n-j}$$

Lo cual significa que el código no puede resolver y decodificar todos aquellos patrones de errores por encima de “t”.

Planteando una relación entre P_B y E_B/N_0 en una demodulación BPSK sobre un canal Gaussiano para 3 casos de Hamming, con m= 3,4 y 5. Se dan los casos de Hamming(7,4) Hamming(15,11) y Hamming(31,26) respectivamente en donde todos tienen t=1.



Cuando se utiliza estos esquemas de codificación debe expresarse E_c/N_0. Dicha expresión se encuentra mediante las siguientes ecuaciones:

$$\frac{E_c}{N_0} = \left(\frac{k}{n}\right)\frac{E_b}{N_0} \qquad \frac{E_c}{N_0} = \frac{2^m - 1 - m}{2^m - 1}\frac{E_b}{N_0}$$

Código Golay extendido

El código golay extendido se define como un bloque de codificación de la forma (24,12), el cual está formado por el agregado de un bit de paridad completo al código perfecto (23,12) conocido como código de Golay. Este bit de paridad agregado incrementa la distancia mínima d_{min} desde 7 a 8 y produce un código de razón ½. lo cual es más fácil de implementar que la razón original de Golay de 23/12.
Utilizando el límite de hamming para el código golay encontramos que se pueden corregir todos los patrones de error de 3 bits. Pero con el código golay extendido como dmin=8 puede corregir todos los patrones de error de tres bits y algunos pero no todos de los errores de 4 bits.
Se puede calcular la probabilidad de error de bit para codificación por golay extendido de la siguiente forma:

$$P_B \approx \frac{1}{24}\sum_{j=4}^{24} j\binom{24}{j}p^j(1-p)^{24-j}$$

Códigos BCH

Estos tipos de código son una generalización de los códigos de hamming ya que permiten la corrección de errores múltiples. Son una clase particular de códigos cíclicos que provee una gran selección de bloques largos, razones de código, tamaños alfabéticos y corrección de múltiples errores.
Los códigos BCH más usados tienen en sus codeword una dimensión de $n=2^m-1$ donde m=3,4,5 … y sus características se pueden resumir mediante la siguiente tabla:

Tamaño del bloque	$n = 2^m - 1$	bits
Bits de paridad	$n-k$	bits
Tamaño del mensaje	$K \geq n - mt$	bits
Distancia mínima	$d_{min} \geq 2t+1$	bits
Errores a corregir	t	bits

Suponiendo un código BCH (127,64) y con respecto a la tabla anterior obtendremos los distintos valores de sus atributos. Teniendo en cuenta que los datos conocidos son n=127 K=64.
Mediante el cálculo de tamaño del bloque, obtenemos ‘m’ de la siguiente manera:

$$m=\log_2(127+1)$$

m=7

La cantidad de bits de paridad, mediante los datos de n y k, son:

Bits de paridad = 127-64

Bits de paridad= 63

La capacidad de corrección de errores ‘t’ del código BCH implementado es de:

$$K \geq n-m*t$$

$$t= \frac{K-n}{-m} = \frac{64-127}{-7}$$

t=9

Por último, la distancia mínima entre codewords válidas es:

$$d_{min} \geq 2*9+1$$

d_{min}= 19