



UNIVERSIDAD DISTRITAL  
FRANCISCO JOSE DE CALDAS

Maestría en Ciencias de la Información y las  
Comunicaciones.

## TEORÍA DE LA INFORMACIÓN

Codificación  
Convolutional

Decodificación  
de Viterbi

&

Turbo  
Código

OSCAR JAVIER JIMÉNEZ

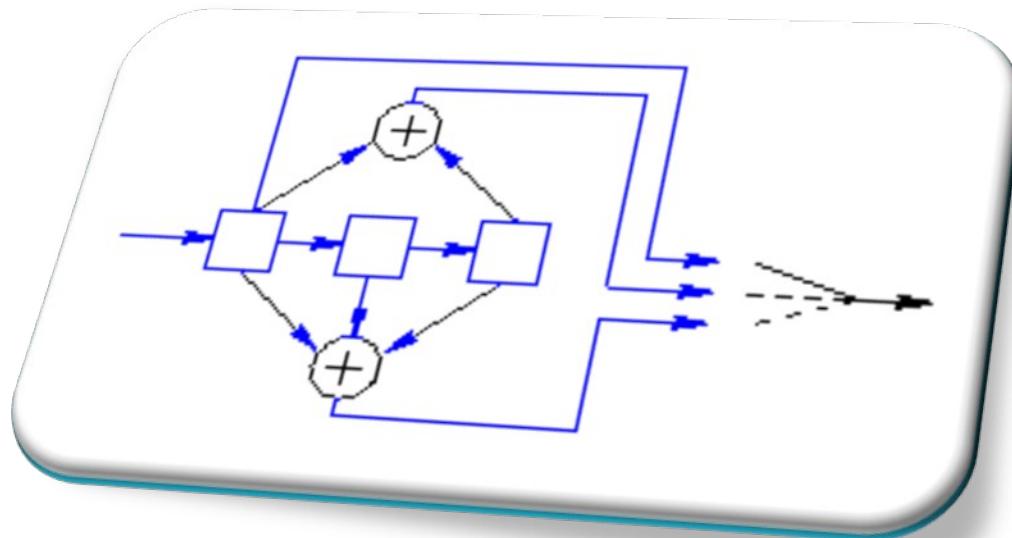
# INTRODUCCIÓN CODIFICACIÓN CONVOLUCIONAL

Existen dos estrategias posibles para recibir de forma fiable y libre de errores la información transmitida desde una fuente:

- **ARQ** (Automatic Repeat Request), basada en la detección de errores, pero sin la posibilidad de corrección, solicitando al transmisor la repetición del mensaje en caso de error.
- **FEC** (Forward Error Correction), basada en la detección y corrección en el extremo receptor de los posibles errores.

En ambos casos, es necesario añadir cierta redundancia al mensaje a transmitir para detectar o corregir estos errores, este proceso se denomina *codificación de canal*.

La codificación convolucional con la decodificación de Viterbi es de las técnicas FEC más adecuadas en canales en los que la señal transmitida se ve corrompida principalmente por ruido gausiano blanco y aditivo (AWGN). Además, han sido las técnicas predominantes usadas en las comunicaciones espaciales, particularmente en las redes de comunicaciones de satélites geoestacionarios.



# CODIFICACIÓN CONVOLUCIONAL

Los Codificadores de Bloque , si bien pueden detectar errores, no pueden corregir el dato de llegada cuando hay más de un bit erróneo

En la Codificación Convolutinal es posible desarrollar un decodificador que corrija errores múltiples en los datos de llegada y se tiene la certeza de determinar los datos originales que fueron afectados con error.

Hay aplicaciones en las que los bits del mensaje vienen en forma serial y no en bloques grandes, en cuyo caso el uso de un búfer puede ser indeseable. En este tipo de situaciones, es probable que el empleo de la codificación convolucional sea el método preferido.

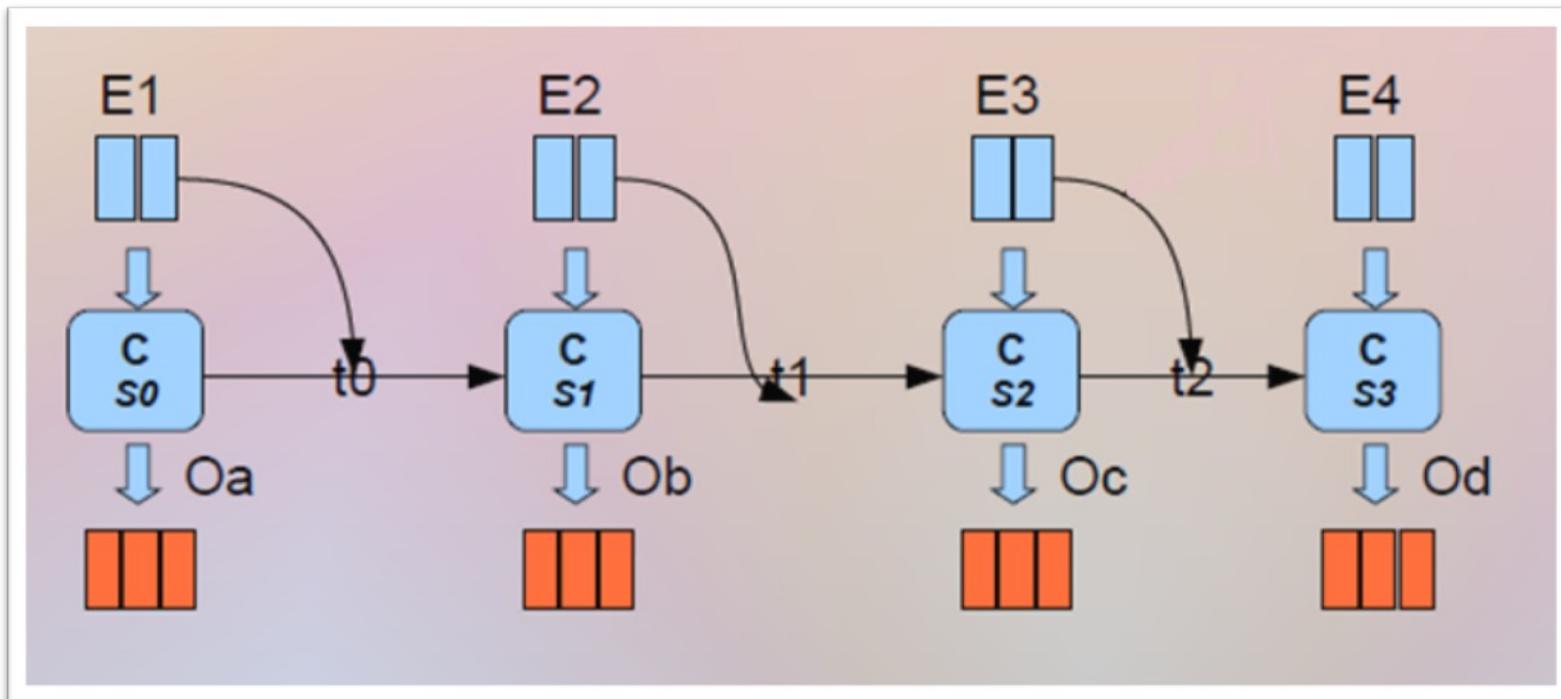
Los códigos convolucionales se describen a partir de ciertos elementos como son la tasa del código, la longitud del código y la memoria del codificador.

- **k** número de bits que entran al codificador.
- **n** número de bits que salen del codificador.
- **R** tasa del código  $k/n$ .
- **K** longitud del código
- **m** memoria del codificador (# de flip-flops del codificador).

La longitud del código, K, denota en cuántos ciclos de codificación tiene influencia un bit que tengamos a la entrada del mismo a partir de un instante dado, ya que este bit que tenemos a la entrada del codificador en un instante dado irá recorriendo la cadena de flip-flops que forman el registro de desplazamiento.

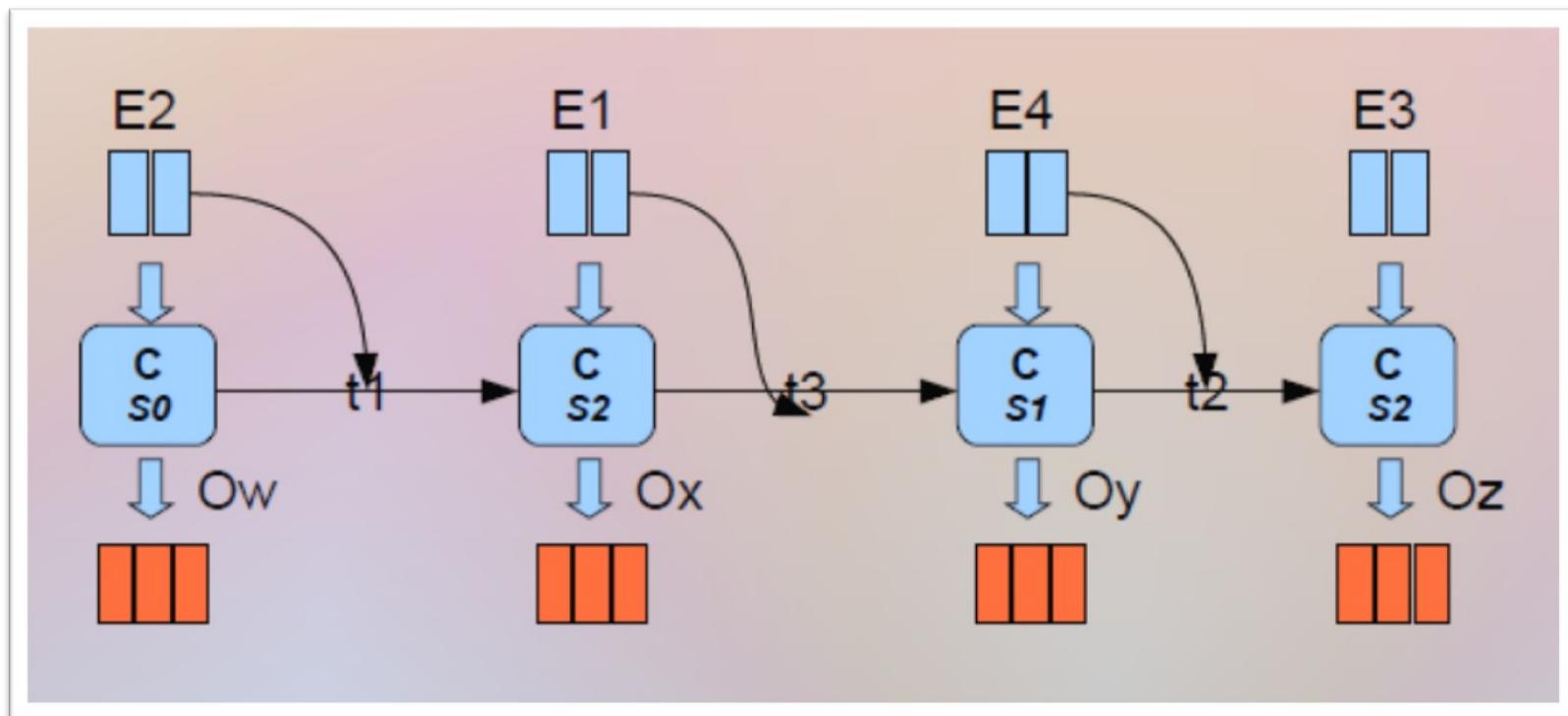
# CODIFICADORES CONVOLUCIONALES

El codificador convolucionador **C**, maneja estados **s<sub>0</sub>, s<sub>1</sub>, s<sub>2</sub> ... s<sub>n</sub>**. Este codificador cambia su estado interno a través de transiciones **t<sub>0</sub>, t<sub>1</sub>...t<sub>n</sub>**.



# CODIFICADORES CONVOLUCIONALES

A diferencia del codificador Hamming, Si los bloques de mensaje se enviaran en otro orden, el resultado sería distinto



# CARACTERÍSTICAS DE LA CODIFICACIÓN CONVOLUCIONAL

- Los codificadores convolucionales se basan en crear dependencia temporal y secuencial entre los bloques codificados.
- Estos condicionan la probabilidad de error de acuerdo al orden de llegada de cada bit del mensaje.
- Las operaciones de codificación no solo dependen de los valores de cada bit del bloque a procesar, sino también del orden de llegada de cada bloque.
- Gracias a la memoria, el codificador maneja estados que están determinados de acuerdo a la configuración de valores de la memoria.

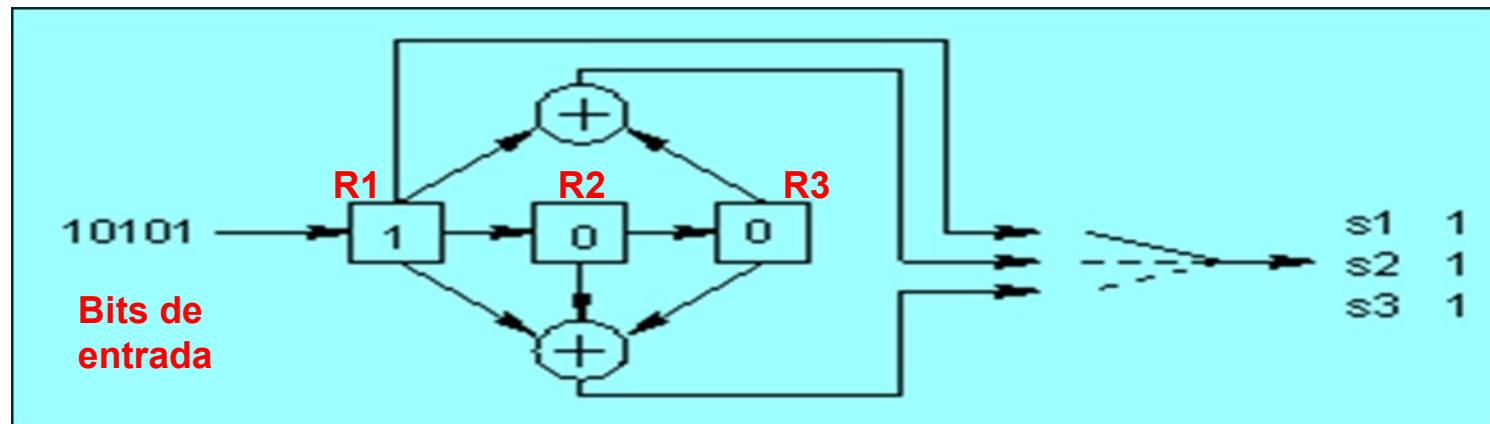
# **CARACTERÍSTICAS DE LA CODIFICACIÓN CONVOLUCIONAL**

- De acuerdo al estado en el que se encuentre, procesa los datos de entrada de forma particular.
- Debido a la dependencia secuencial del procesamiento de cada bloque, se reduce la probabilidad de error porque genera eventos dependientes a través del tiempo.
- Esta característica se conoce como un Proceso Estocástico de Markov, el cual asegura que:

**“La probabilidad de eventos futuros no solo están condicionados al estado presente sino también al resultado del estado anterior.”**

# ARQUITECTURA DE UN CODIFICADOR CONVOLUCIONAL

En este ejemplo se muestra un codificador básico con 3 registros, de los cuales R1 es un registro de transito, y R2 y R3 son registros de memoria. Después de procesado el bit, este se mueve hasta el registro siguiente.



Según la configuración de este diagrama, el circuito está configurado con  $k=1$  entradas, restricción de palabras  $K=3$  (número de registros), y salidas  $n=3$ , con  $m=2$  memorias.

# ANÁLISIS DE UN CODIFICADOR CONVOLUCIONAL BÁSICO

- Las operaciones del circuito anterior para cada salida son:

$$S_1 = R_1$$

$$S_2 = R_1 \text{ xor } R_3$$

$$S_3 = R_1 \text{ xor } R_2 \text{ xor } R_3$$

- La tabla de resultados es la siguiente:

R1	R2	R3	S1	S2	S3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	0	1

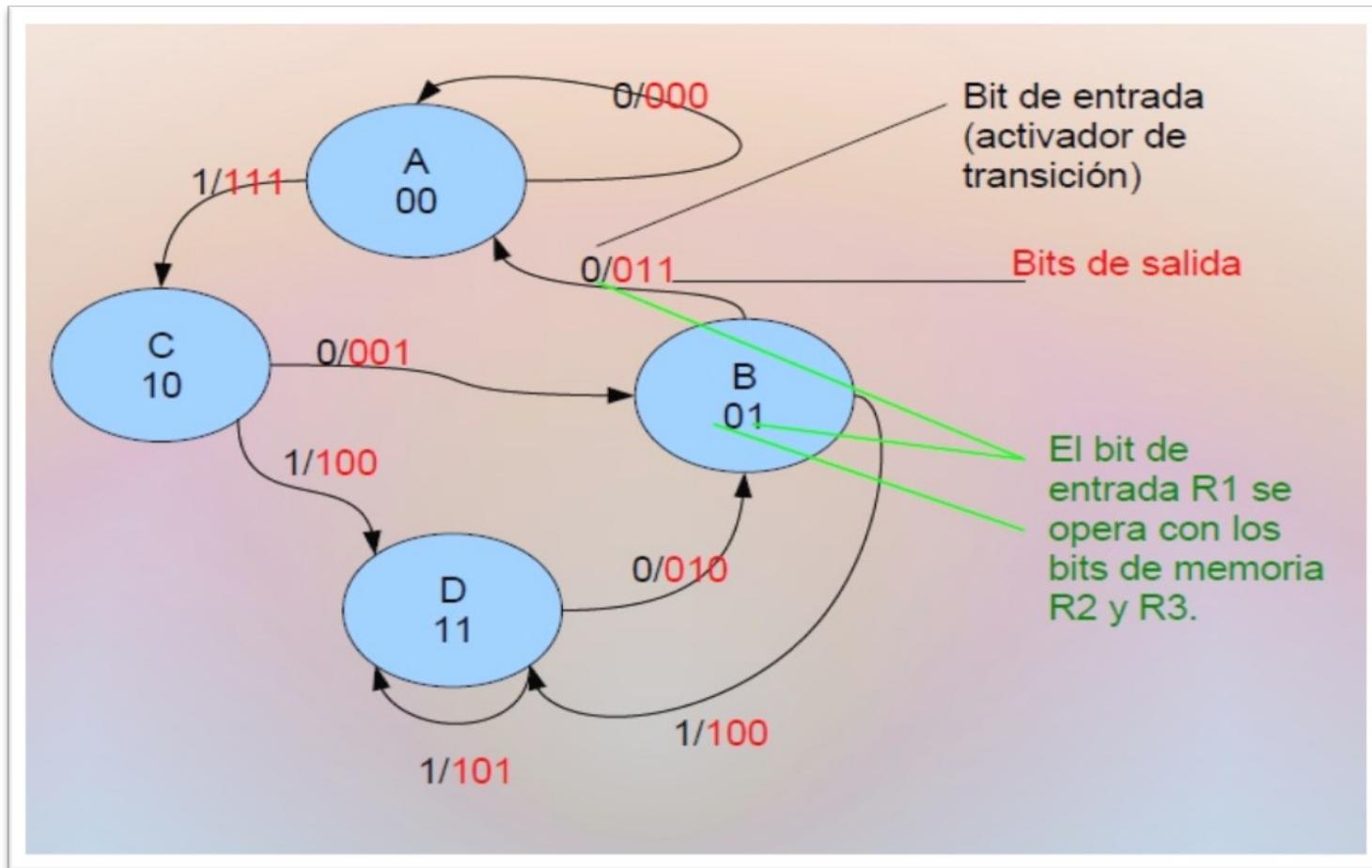
# ANÁLISIS DE UN CODIFICADOR CONVOLUCIONAL BÁSICO

- Debido a que tiene 2 registros de memoria, los posibles estados son 4 (las configuraciones de 2 bits):

R2	R3	ESTADO
0	0	A
0	1	B
1	0	C
1	1	D

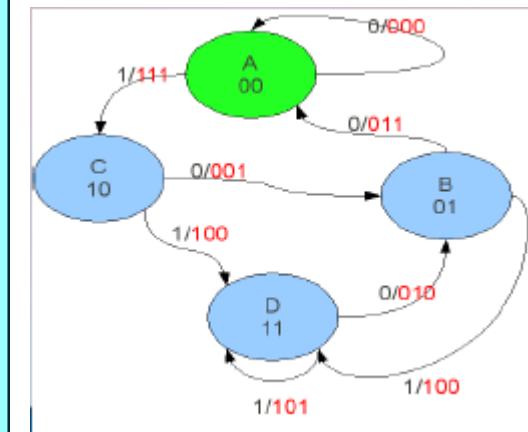
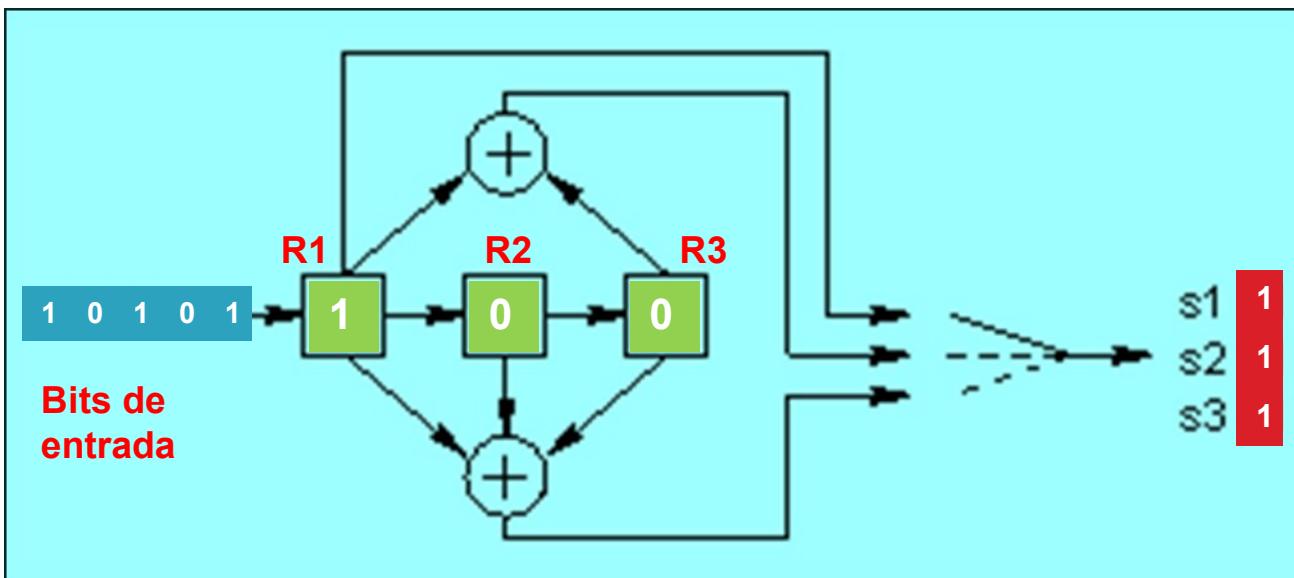
R1	R2	R3	S1	S2	S3	ESTADO
0	0	0	0	0	0	A
0	0	1	0	1	1	B
0	1	0	0	0	1	C
0	1	1	0	1	0	D
1	0	0	1	1	1	A
1	0	1	1	0	0	B
1	1	0	1	1	0	C
1	1	1	1	0	1	D

# DIAGRAMA DE ESTADOS



# EJEMPLO DE CODIFICACIÓN CONVOLUCIONAL

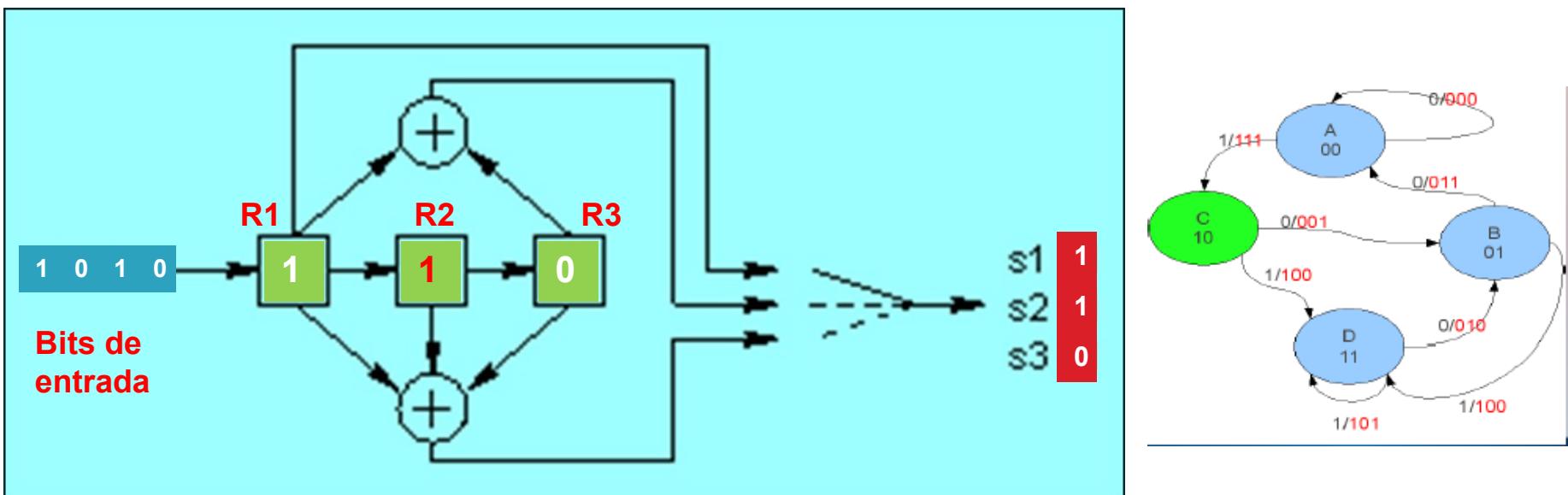
Supongamos que se quiere enviar la secuencia de bits **101011** (donde los bits más a la derecha son los más antiguos). El proceso de codificación es el siguiente:



La salida del bit 1 es **111**

# EJEMPLO DE CODIFICACIÓN CONVOLUCIONAL

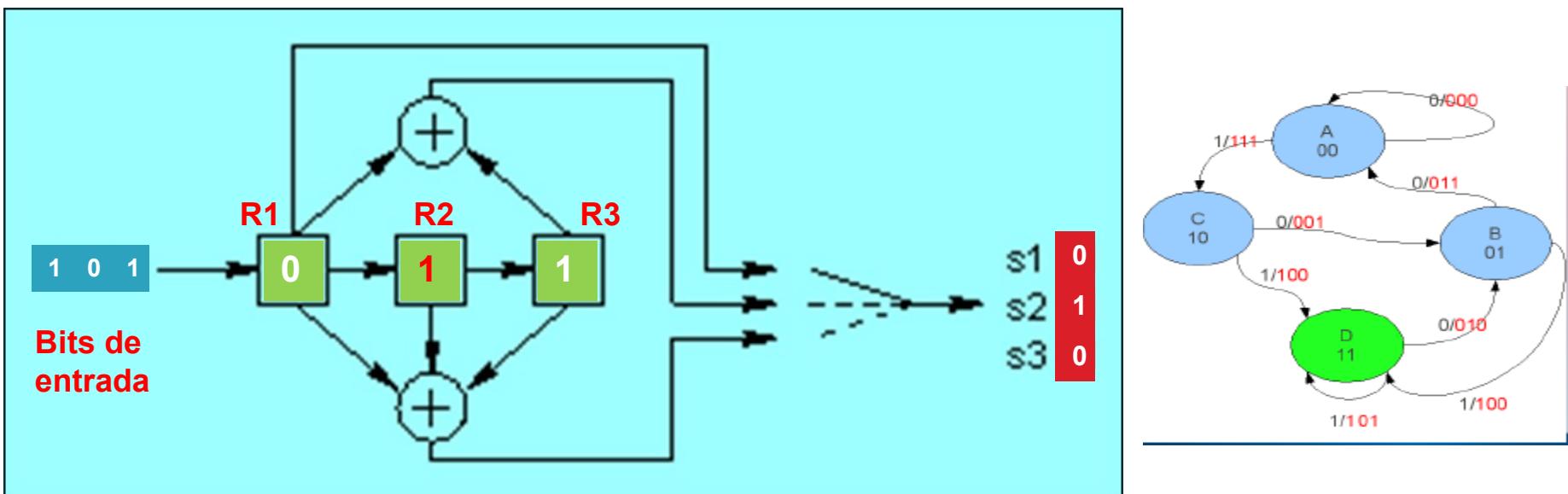
Cada bit que entra en el circuito se corre a la derecha mediante la operación Shift (corrimiento).



**La salida del bit 2 es 110**

# EJEMPLO DE CODIFICACIÓN CONVOLUCIONAL

Los bit desplazados pasan a la memoria condicionando la forma en la que se procesan los bits consecuentes:



Al final del proceso de codificación obtenemos que la secuencia codificada es **111 110 010 100 001 100**

D D

I E

A

G Á

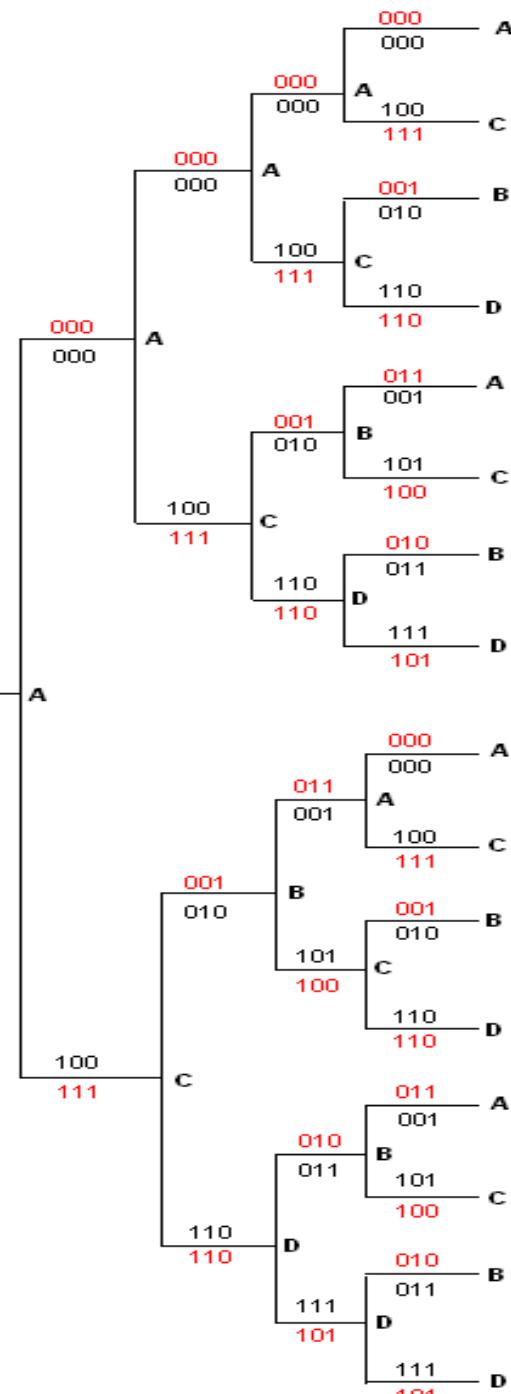
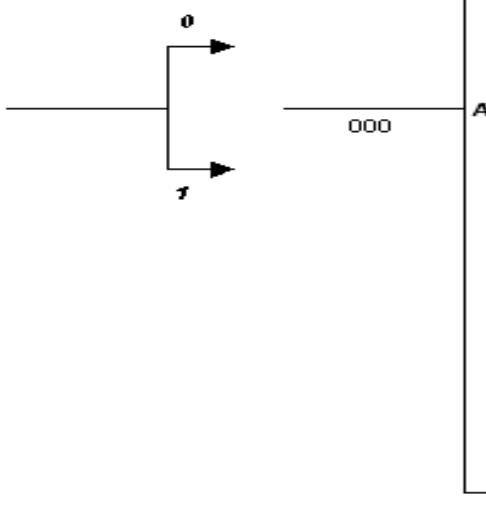
R R

A B

M O

A L

Para el ejemplo  
del codificador  
anteriormente  
especificado  
tenemos el  
siguiente Árbol  
del Código:



# CARACTERÍSTICAS DEL DIAGRAMA DE ÁRBOL

La profundidad del árbol es  $2 \cdot (m-1)$ , y el número de estados es  $2 \cdot (m-1) \cdot k$ . La interpretación del árbol del código es la siguiente:

- Hay dos ramas en cada nodo.
- La rama superior corresponde a una entrada de un 0.
- La rama inferior corresponde a la entrada de un 1.
- En la parte exterior de cada rama se muestra el valor de salida.
- El número de ramas se va multiplicando por dos con cada nueva entrada.

A partir del segundo nivel el árbol se vuelve repetitivo. En realidad, solo hay cuatro tipos de nodos: **A,B,C,D**. Estos tipos de nodos en realidad son estados del codificador. A partir de estos nodos, se producen los mismos bits de salida y el mismo estado. Por ejemplo, de cualquier nodo etiquetado como C se producen el mismo par de ramas de salida: *Salida 001, estado B y salida 110 y estado D*.

# Diagramas Códigos Convolucionales

- ***Diagrama de árbol o árbol de código:*** representación mediante un árbol binario de las distintas posibilidades.
- ***Diagrama de estados:*** es la forma menos utilizada
- ***Diagrama de Trellis o enrejado:*** es la forma mas utilizada porque es la que permite realizar la decodificación de la forma mas sencilla.

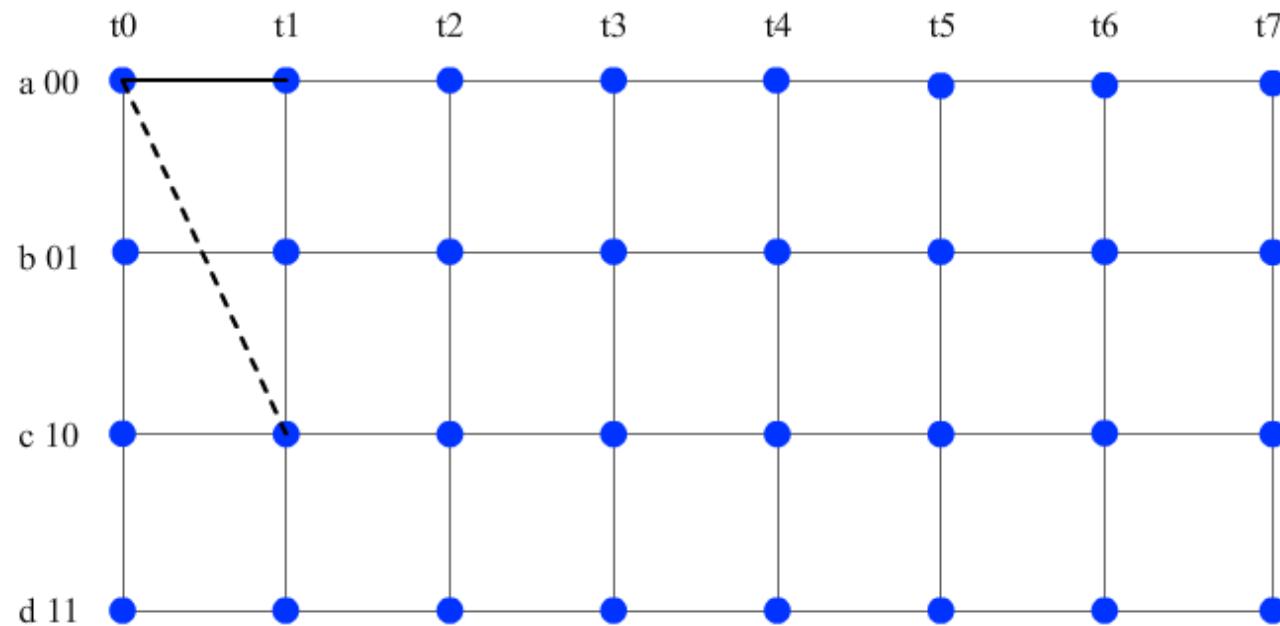
# Diagrama de Trellis

El *árbol de Trellis* se basa en los estados del codificador y en sus posibles caminos. De cada nodo parten ramas hacia los nodos siguientes, permitiendo representar de forma lineal la secuencia de los eventos, el único inconveniente es que luce algo desordenado.

# Elaboración del Diagrama de Trellis

- Se tienen 4 estados, los mismos del diagrama de estado  
 $a=00$ ,  $b=01$ ,  $c=10$  y  $d=11$ .
- Si al codificador de Trellis entra un 1 se pinta la trayectoria hacia el otro estado con línea puntada, de lo contrario se pinta con línea continua.
- El estado inicial es el estado " $a$ " = 00 y las trayectorias se pintan de izquierda a derecha.
- Cada transición de tiempo indicará que un nuevo bit ha ingresado

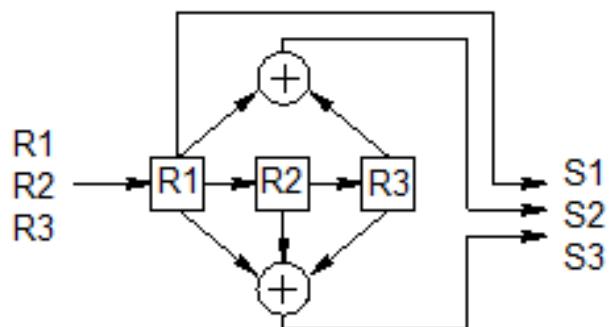
- El eje x es usado para representar el tiempo de todos los posibles estados y el eje “Y” muestra todos los posibles estados del código (ósea a , b ,c ,d).
- Partimos del estado A(00) y t0 a partir de aquí se trazan dos líneas desde este estado, una línea para el caso en que la entrada sea 0 y otra línea para el caso que a entrada sea 1:



Recordando las operaciones que realizaba el convolucionador básico, las cuales son:

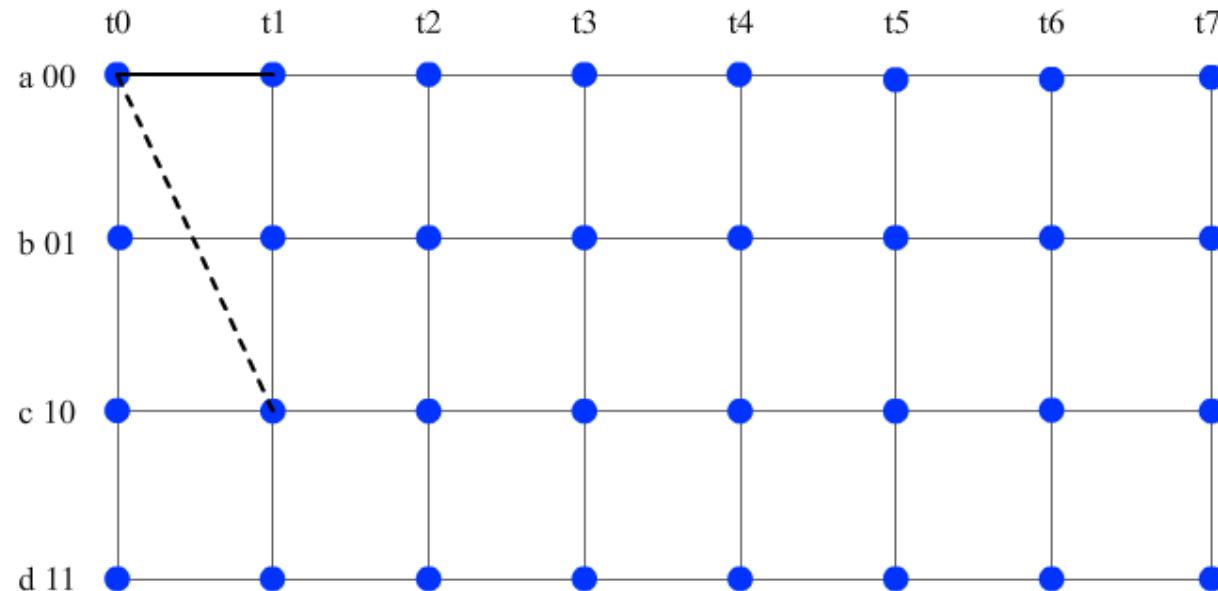
- $S_1 = R_1$
- $S_2 = R_1 \text{ xor } R_3$
- $S_3 = R_1 \text{ xor } R_2 \text{ xor } R_3$

Tenemos que  $R_1$  es la entrada del 1er bit,  $R_2$  y  $R_3$  son los registros de memoria,  $S_1$ ,  $S_2$  y  $S_3$  será la codificación de los datos.

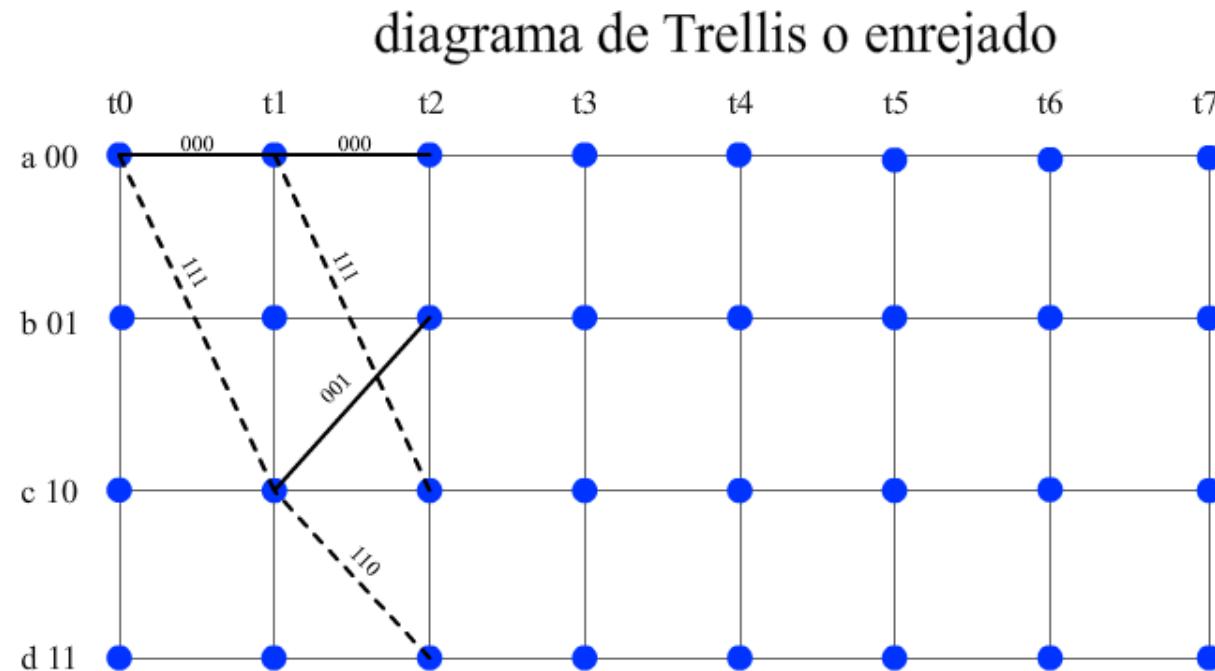


R	R	R	S	S	S
1	2	3	1	2	3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	0	1

Para  $t_1$  tenemos la siguiente grafica donde las líneas indican las salidas posibles en cada caso (0 o 1) y encima de cada línea se indica la salida del decodificador.



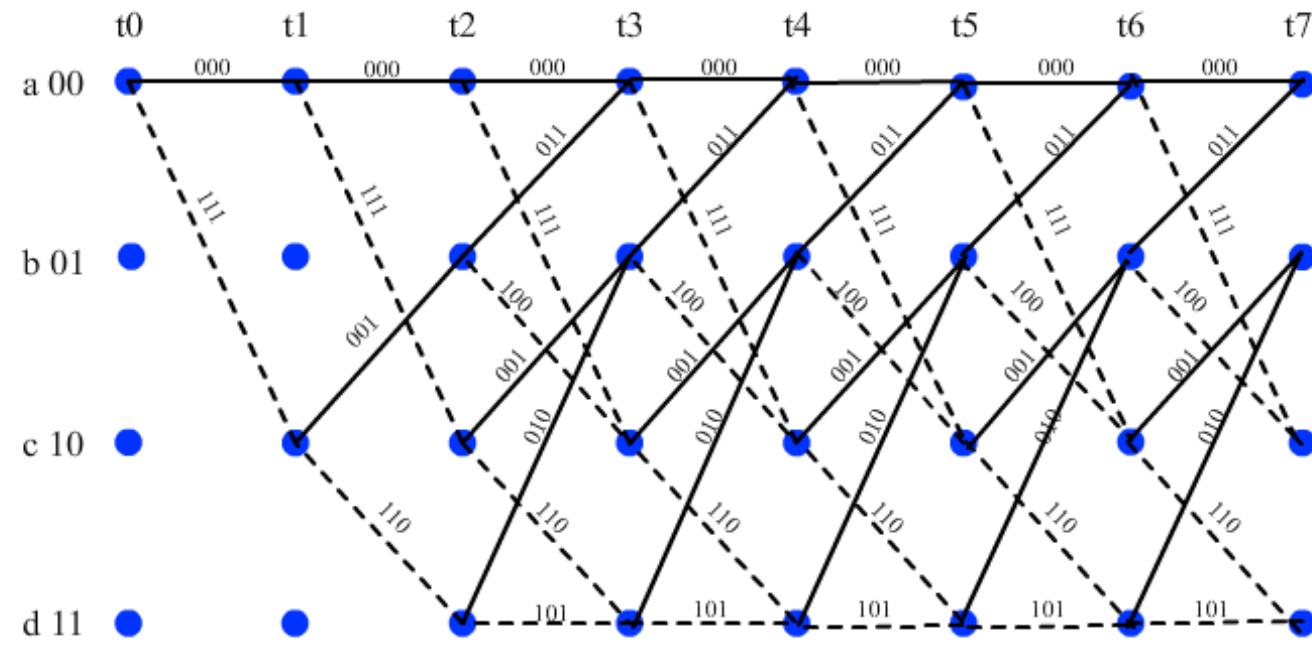
De esta manera se deberá construir el diagrama para cada nodo que se valla creando:  
En t2 el diagrama quedará así:



Cada transición de tiempo indicará que un nuevo bit ha ingresado.

Para este caso el diagrama se estabilizara en  $t_3$  dado que seguirá la misma secuencia y por consiguiente realizará los mismos recorridos.

Finalmente el diagrama quedara así:



# EJEMPLO DE DIAGRAMA DE TRELLIS

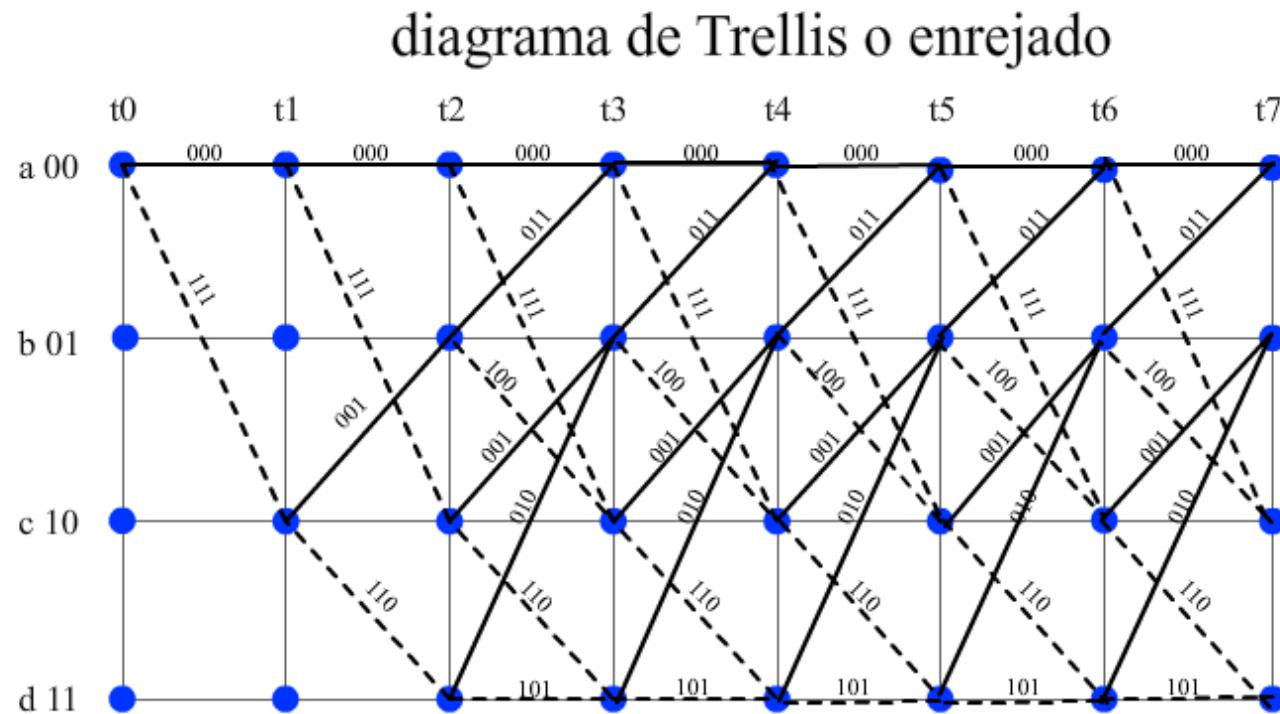
Si tenemos una secuencia de datos para el codificador 101011.

las entradas y salidas son

r1	r2	r3	s1	s2	s3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	0
1	1	1	1	0	1

Código de ejemplo =

101011



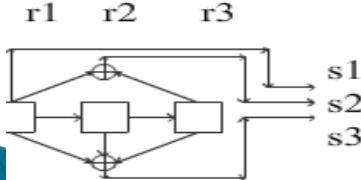
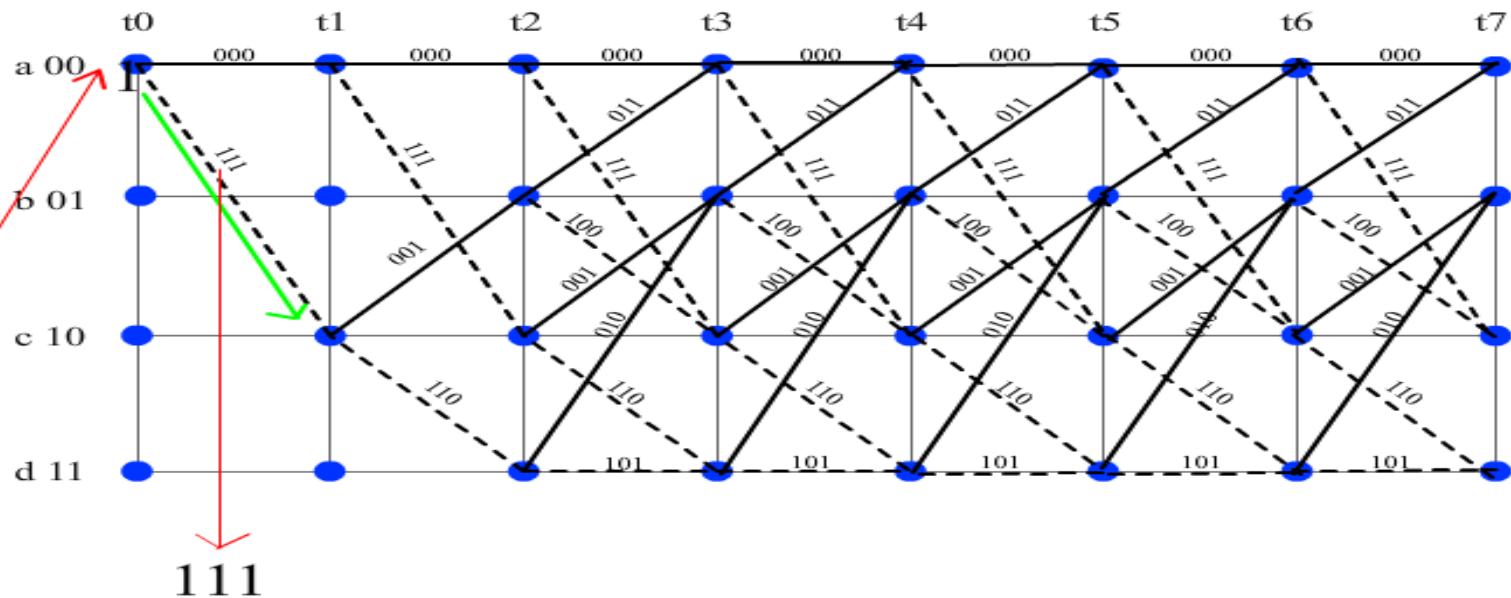
En t1 Si entra el 1 al diagrama el recorrido será de a ->c en t0, generando la secuencia codificada 111.

las entradas y salidas son  
 r1 r2 r3 s1 s2 s3  
 0 0 0 0 0 0  
 0 0 1 0 1 1  
 0 1 0 0 0 1  
 0 1 1 0 1 0  
 1 0 0 1 1 1  
 1 0 1 1 0 0  
 1 1 0 1 1 0  
 1 1 1 1 0 1

Código de ejemplo =

**10101**

diagrama de Trellis o enrejado

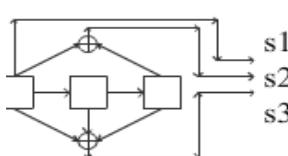
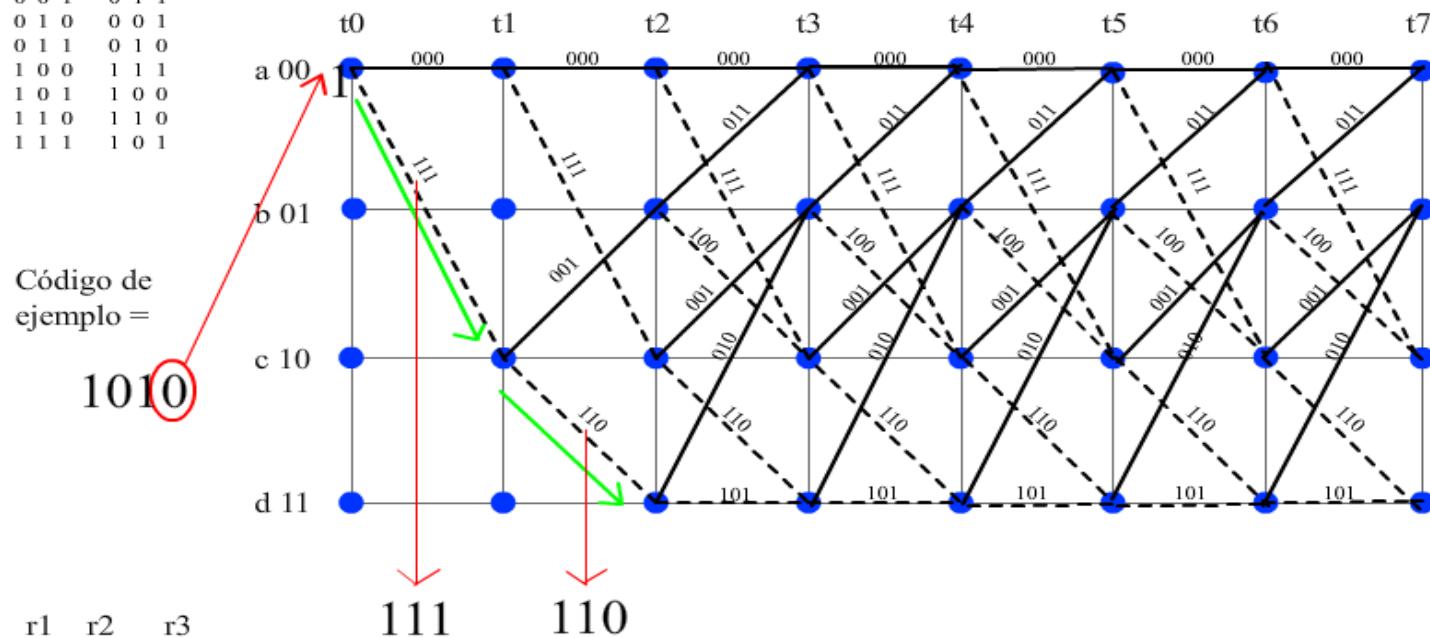


En  $t_2$  entrará otro 1 recorriendo la ruta  $c \rightarrow d$  en  $t_1$ , generando la secuencia codificada 111110.

las entradas y salidas son

r1	r2	r3	s1	s2	s3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	0	1

diagrama de Trellis o enrejado



En t3 entrará un 0 recorriendo la ruta d->b, generando la secuencia codificada 111110010.

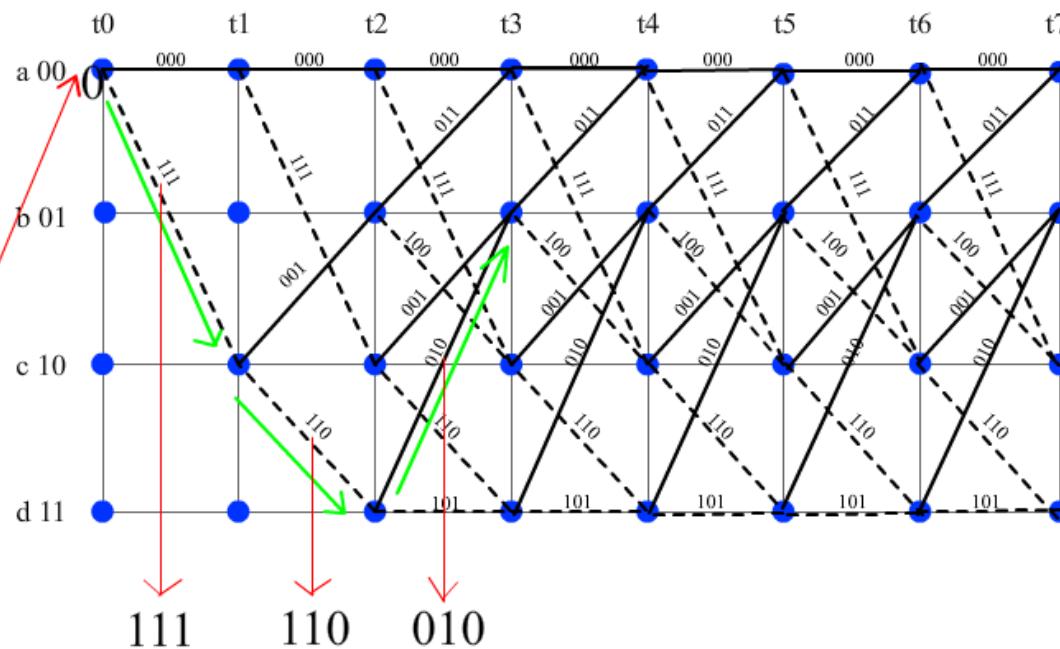
las entradas y salidas son

r1	r2	r3	s1	s2	s3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	0	1

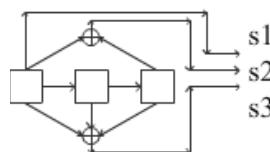
diagrama de Trellis o enrejado

Código de ejemplo =

101



r1 r2 r3



En t4 entrará un 1 recorriendo la ruta b->c, generando la secuencia codificada 111110010100.

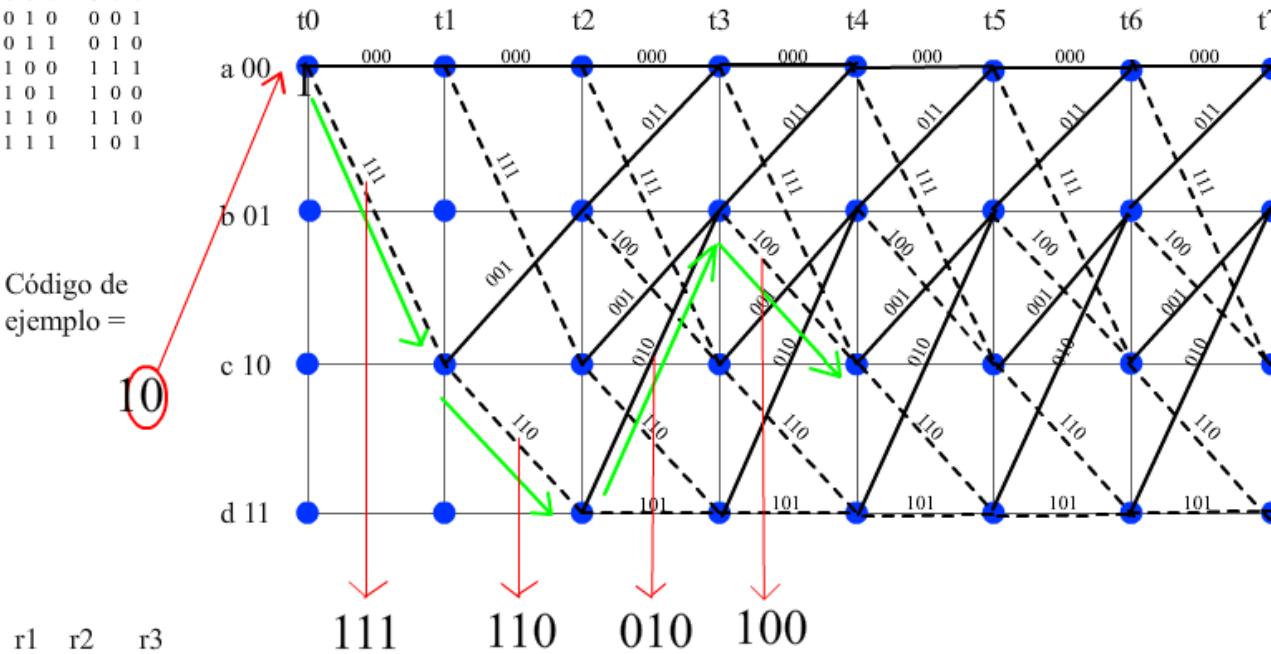
las entradas y salidas son

r1	r2	r3	s1	s2	s3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	0	1

diagrama de Trellis o enrejado

Código de ejemplo =

10



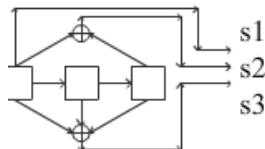
r1 r2 r3

111

110

010

100



En t5 entrará un 0 recorriendo la ruta c->b, generando la secuencia codificada 111110010100001.

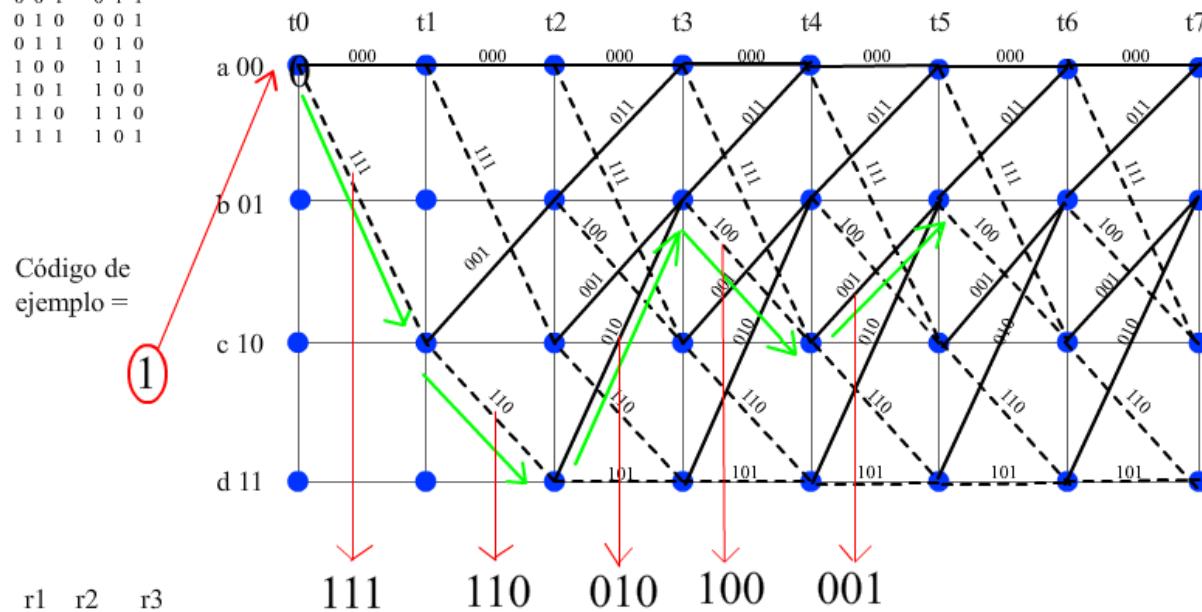
las entradas y salidas son

r1	r2	r3	s1	s2	s3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	0	1

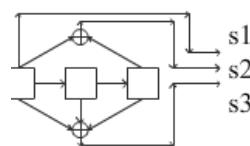
diagrama de Trellis o enrejado

Código de ejemplo =

1



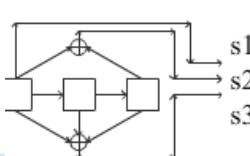
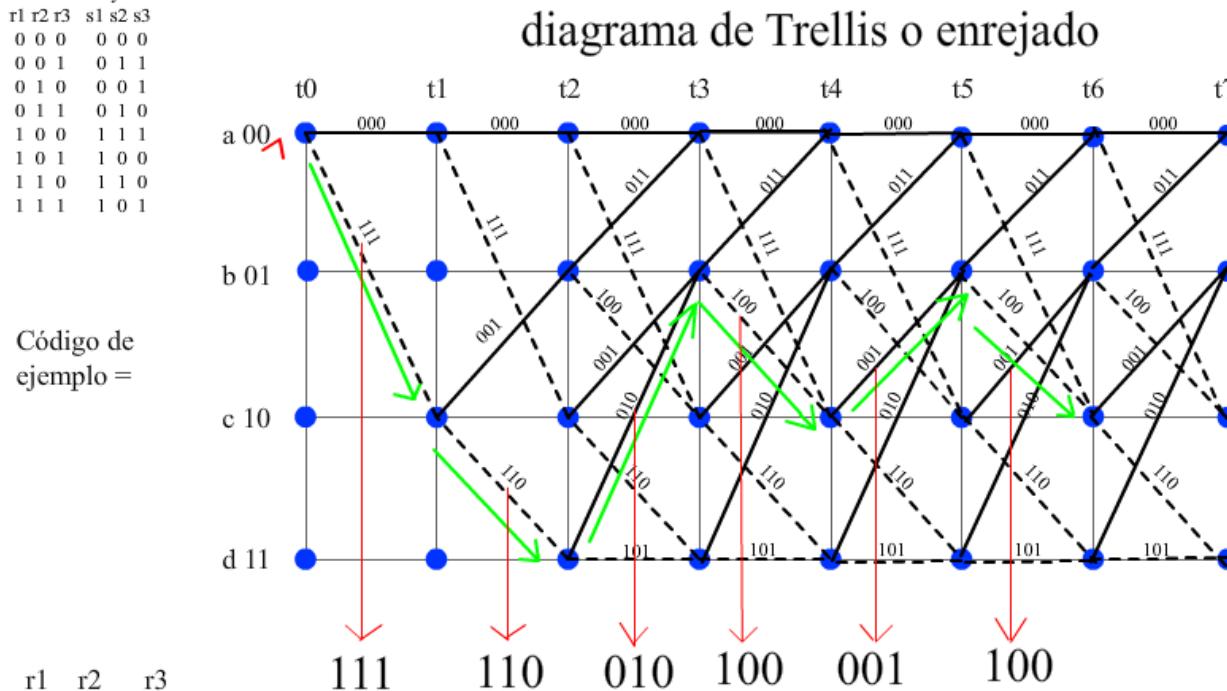
r1 r2 r3



Finalmente en t6 entrará un 1 recorriendo la ruta b->c, generando la secuencia codificada 111110010100001100, recorriendo las rutas: a->c->d->b->c->b->c.

las entradas y salidas son  
 r1 r2 r3 s1 s2 s3  
 0 0 0 0 0 0  
 0 0 1 0 1 1  
 0 1 0 0 0 1  
 0 1 1 0 1 0  
 1 0 0 1 1 1  
 1 0 1 1 0 0  
 1 1 0 1 1 0  
 1 1 1 1 0 1

Código de ejemplo =



secuencia codificada = 111110010100001100

# Algoritmo de viterbi

El problema que se aborda con este algoritmo es que desde la teoría de la codificación, considerando como transparente el canal y los bloques modulador-demodulador. Para ello consideramos los tres bloques como uno solo, el canal digital.

# Distancia de Hamming

- Se define como el número de bits que tienen que cambiarse para transformar una palabra de código válido en otra palabra de código válido.
- Si dos palabras de código difieren en una distancia **d**, se necesitan **d** errores para convertir una en la otra.
- Cuanto mayor sea esta diferencia, menor es la posibilidad de que un código válido se transforme en otro código válido por una serie de errores.

Por ejemplo:

- La distancia Hamming entre **1011101** y **1001001** es 2.
- La distancia Hamming entre **2143896** y **2233796** es 3.
- La distancia Hamming entre "tener" y "reses" es 3.

# Decodificación de Viterbi

- Fortaleza en capacidad de corregir errores y decodificar más de un error.
- El decodificador es también una máquina de estados similar al codificador, con la misma correspondencia de estados y transiciones.
- Se basa en el diagrama de trellis, incluyendo todas las posibles rutas y combinaciones, para finalmente, con la distancia de hamming de todas las posibles rutas, toma la de menor peso (dados por la menor distancia de hamming)

# Decodificación de Viterbi

## Ejemplo

Para nuestro ejemplo tenemos el dato codificado:

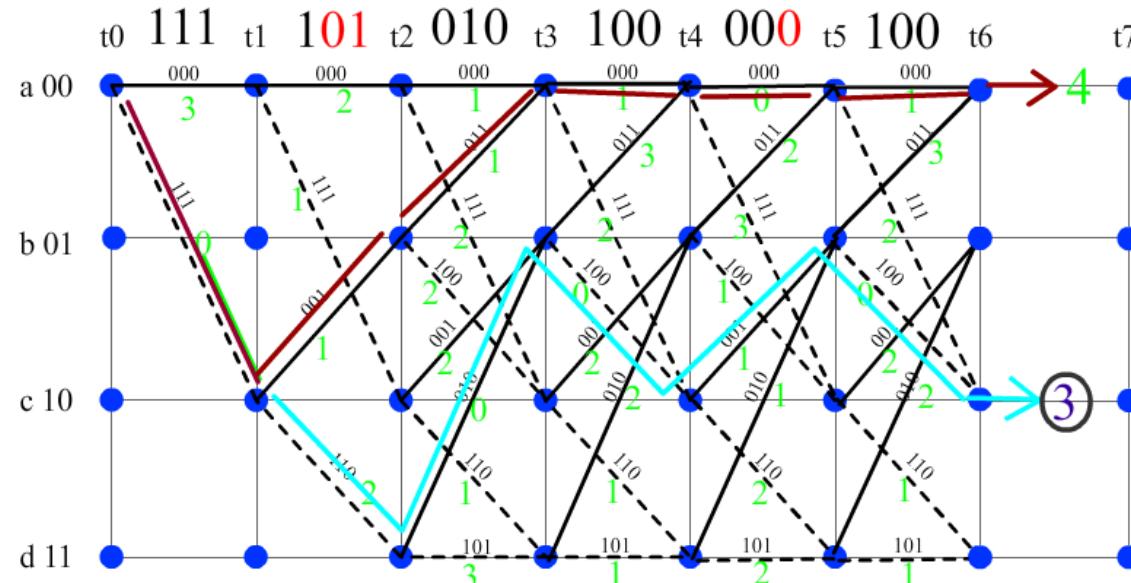
111 110 010 100 001 100

Ahora bien, supongamos que en el proceso de transmisión se dañaron 3 bits (los que están en rojo):

111 1**10** 010 100 00**1** 100

cambiando a 111 1**01** 010 100 00**0** 100

**La ruta con menor sumatoria de distancias de hamming contendrá la información correcta que fue transmitida.**



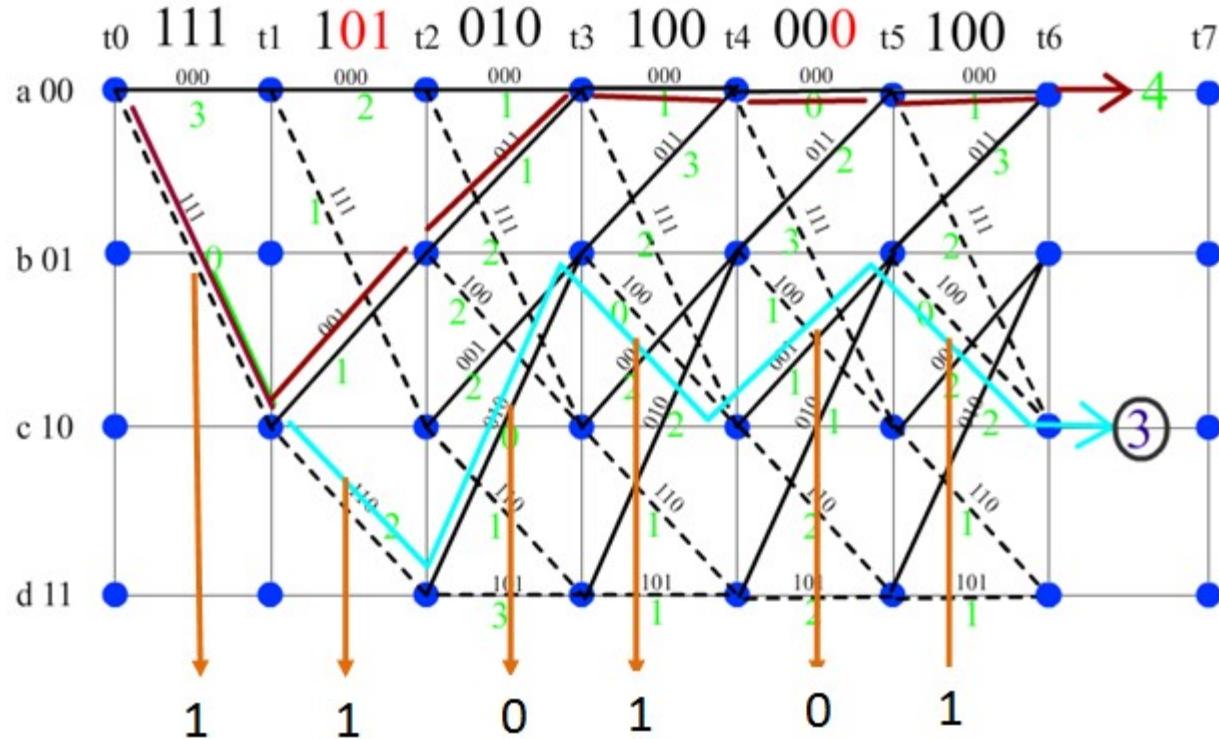
Para el ejemplo que venimos trabajando se determinaron dos posibles rutas donde los pesos que se calcularon son

Tiempos            t1    t2    t3    t4    t5    t6

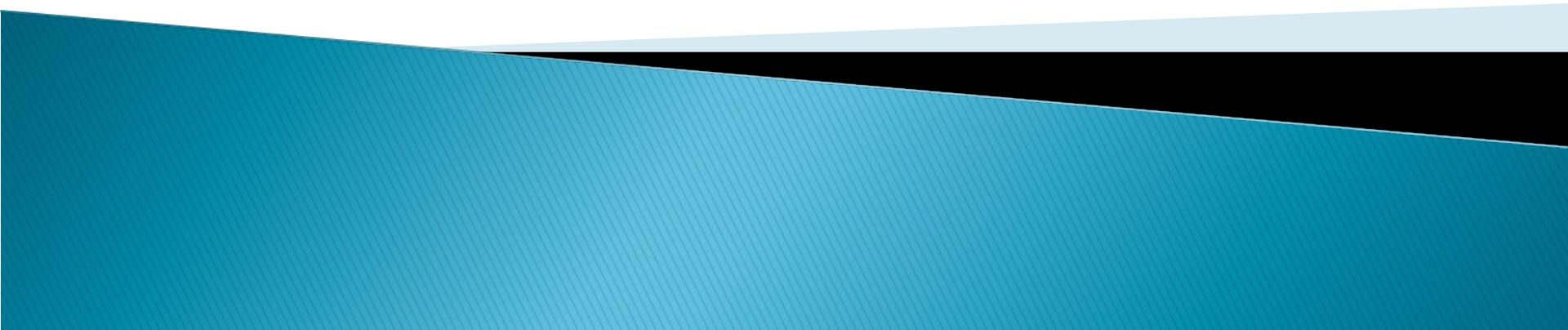
$$\text{Ruta vino tinto} = 0 + 1 + 1 + 1 + 0 + 1 = 4$$

$$\text{Ruta azul claro} = 0 + 2 + 0 + 0 + 1 + 0 = 3$$

A partir de la ruta azul claro, obtenemos la información corregida y decodificada.



# TURBO-CÓDIGO

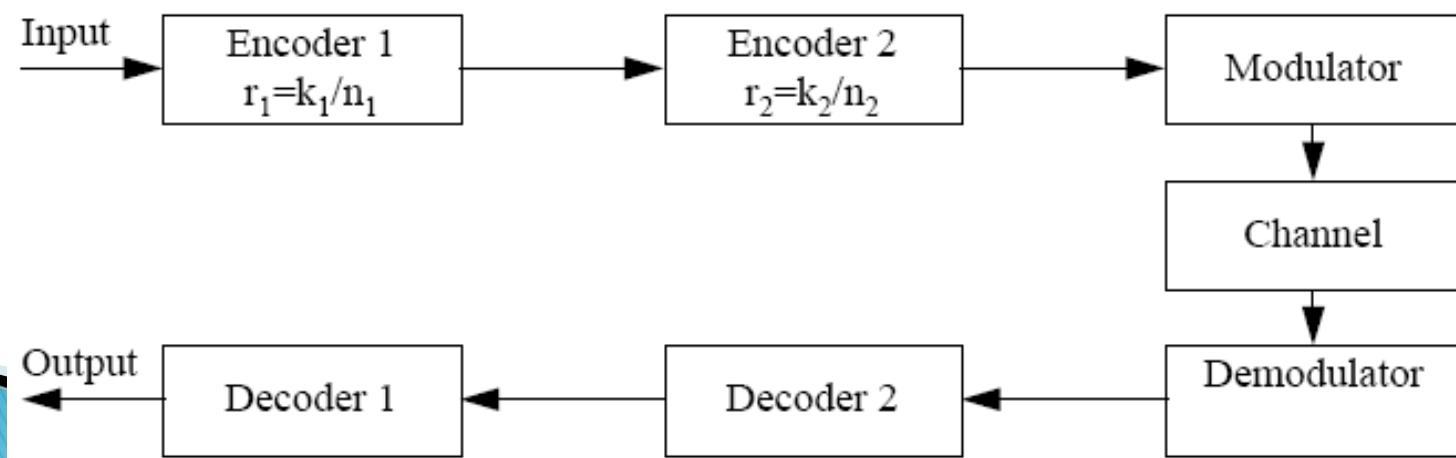


# Contextualización

- ▶ Las prestaciones de un codificador convolucional aumentan, al aumentar su tamaño de memoria.
- ▶ No se puede aumentar la memoria indiscriminadamente ya que la complejidad en el proceso de decodificación crece exponencialmente.

# Historia

- ▶ Dado el problema del sub-aprovechamiento de la memoria, sobre los 60s Forney introdujo el concepto de concatenación, según el cual es posible unir 2 codificadores en un solo bloque.



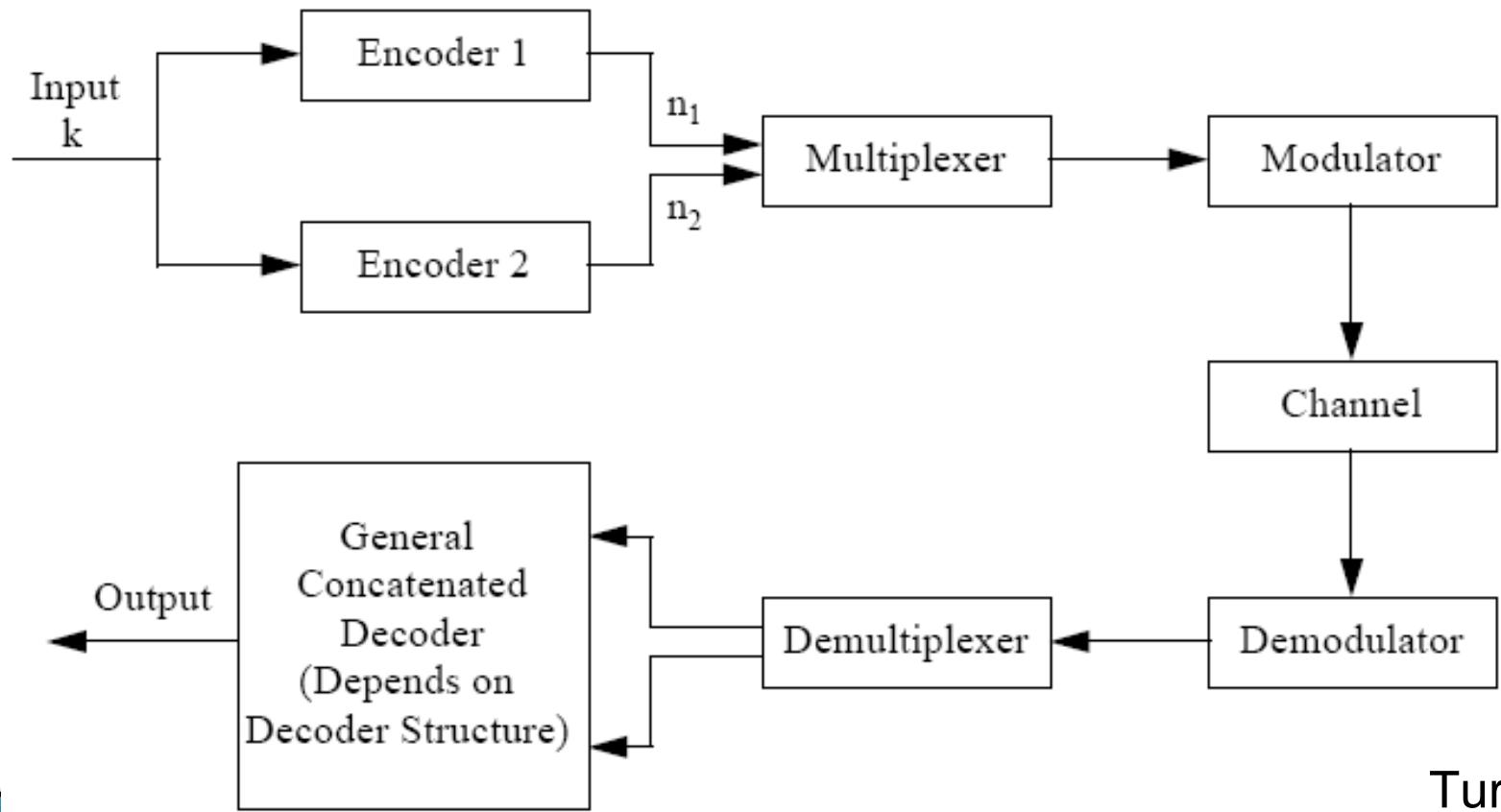
# Historia (2)

- ▶ En 1993, los turbo-códigos fueron presentados formalmente por 1era vez en un paper publicado por Berrou, Glavieux y Thitimajshima de Telecom-Bretagne, Francia.
- ▶ La novedad real de esta publicación era la posibilidad de utilizar la concatenación de bloques en forma paralela.

# Definición - Turbo-código

- ▶ Un turbo-código es una refinación de la estructura de codificación concatenada, ya que se hace utilizando concatenación en paralelo, más un algoritmo iterativo de decodificación.

# Definición - Turbo-código (2)



Turbo-  
códigos

# Ventajas

1. Con Turbo-códigos se consigue un aprovechamiento casi total de la capacidad del canal, a diferencia de otros esquemas de concatenación propuestos anteriormente.
2. Adicionalmente, se incrementa la tasa de código en comparación con la concatenación serial.

Turbo-  
códigos

# Capacidad de Canal (Shannon)

- ▶ La capacidad del canal está dada por:

$$C = B \log_2 \left( 1 + \frac{S}{N} \right)$$

En donde:

- ▶ C: Capacidad del canal
- ▶ B: es el ancho de banda del canal dado en Hertz
- ▶ S: es la potencia de la portadora del canal dado watts
- ▶ N: es la potencia del ruido y las interferencias del canal dado en watts
- ▶ S/N: Radio de señal a ruido dado en dB

Turbo-  
códigos

# Capacidad de canal (2)

- ▶ Al utilizar turbo-códigos se puede obtener una diferencia entre la capacidad y la utilización del canal de aproximadamente 0.7dB; en comparación con otros métodos, el turbo-código es el que más se acerca a la capacidad del canal.

# Tasas de código

- ▶ Para codificadores concatenados en serie:
- ▶ Para codificadores concatenados en paralelo:

$$R_s = R_1 R_2$$

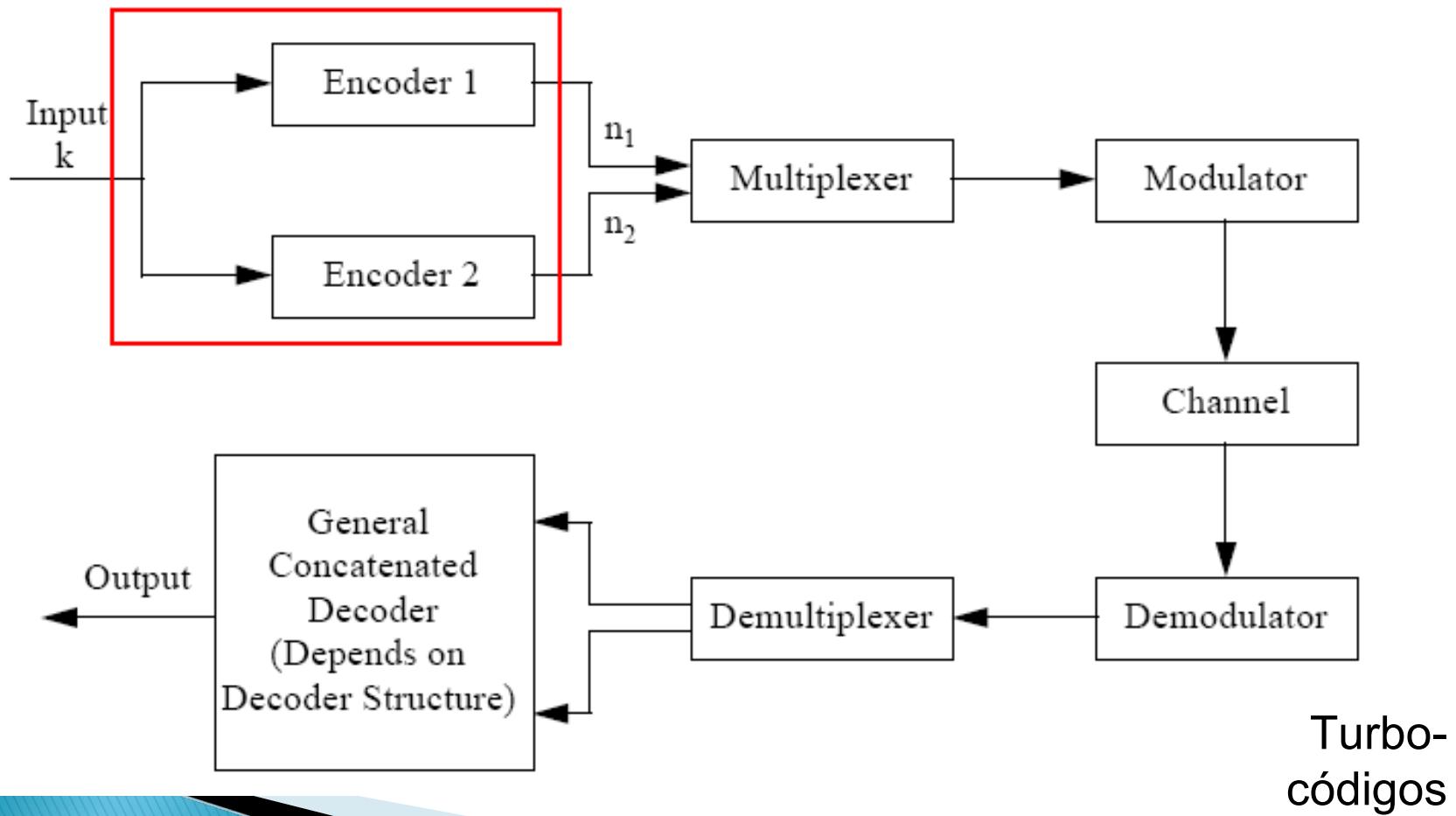
$$R_p = \frac{R_1 R_2}{1 - (1 - R_1)(1 - R_2)}$$

Turbo-  
códigos

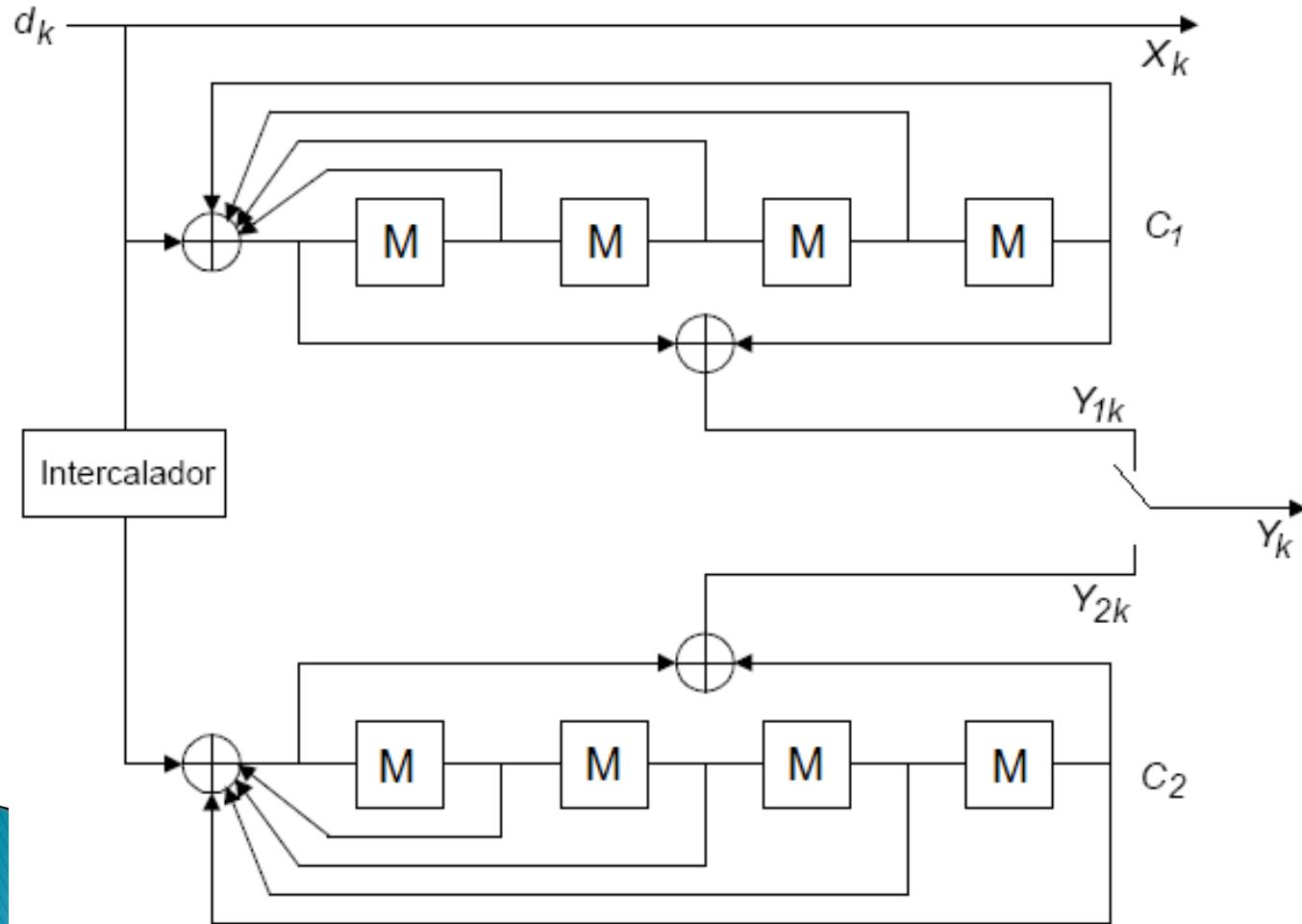
# Tasas de código (2)

- ▶ Utilizando los mismos valores  $R_1$  y  $R_2$  para bloques individuales, se encontrará que siempre  $R_p > R_s$  lo cual sugiere que hay una mejor tasa de código global cuando se concatenan bloques en paralelo

# Codificador



# Codificador (2)



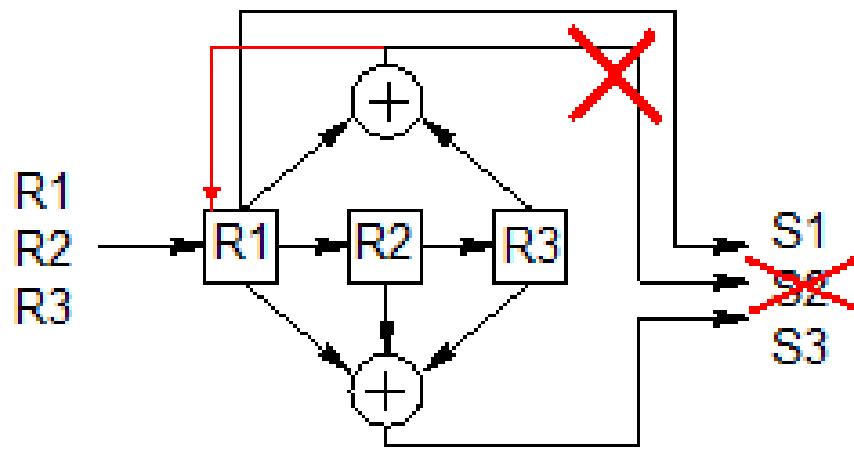
Turbo-  
códigos

# Codificador (3)

- ▶ La idea es que los datos sean codificados en su orden natural y en un orden intercalado (entrelazado) por 2 codificadores RSC que crean los bits de paridad  $Y_1$  y  $Y_2$
- ▶ El intercalado debe ser diseñado muy cuidadosamente porque tendrá un fuerte impacto en la eficiencia total del codificador.

# Códigos RSC

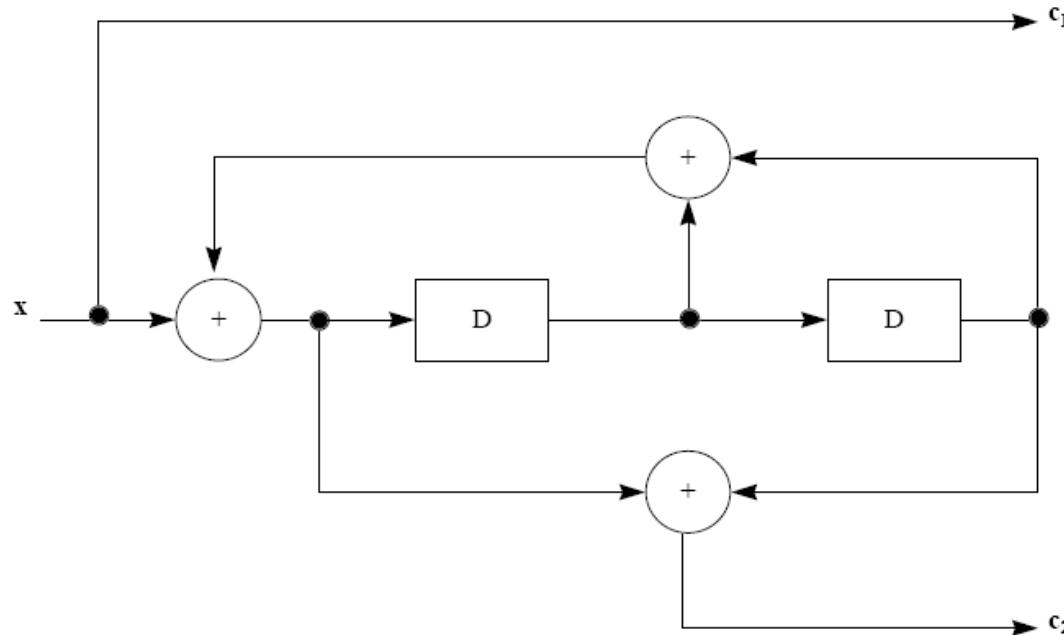
- El codificador RSC (Recursive Systematic Convolutional) se obtiene al realimentar una salida del codificador convolucional convencional hacia la entrada nuevamente.



Turbo-  
códigos

# Códigos RSC

- Bat93 sugiere que buenos códigos pueden ser obtenidos a partir de la realimentación.



Turbo-  
códigos

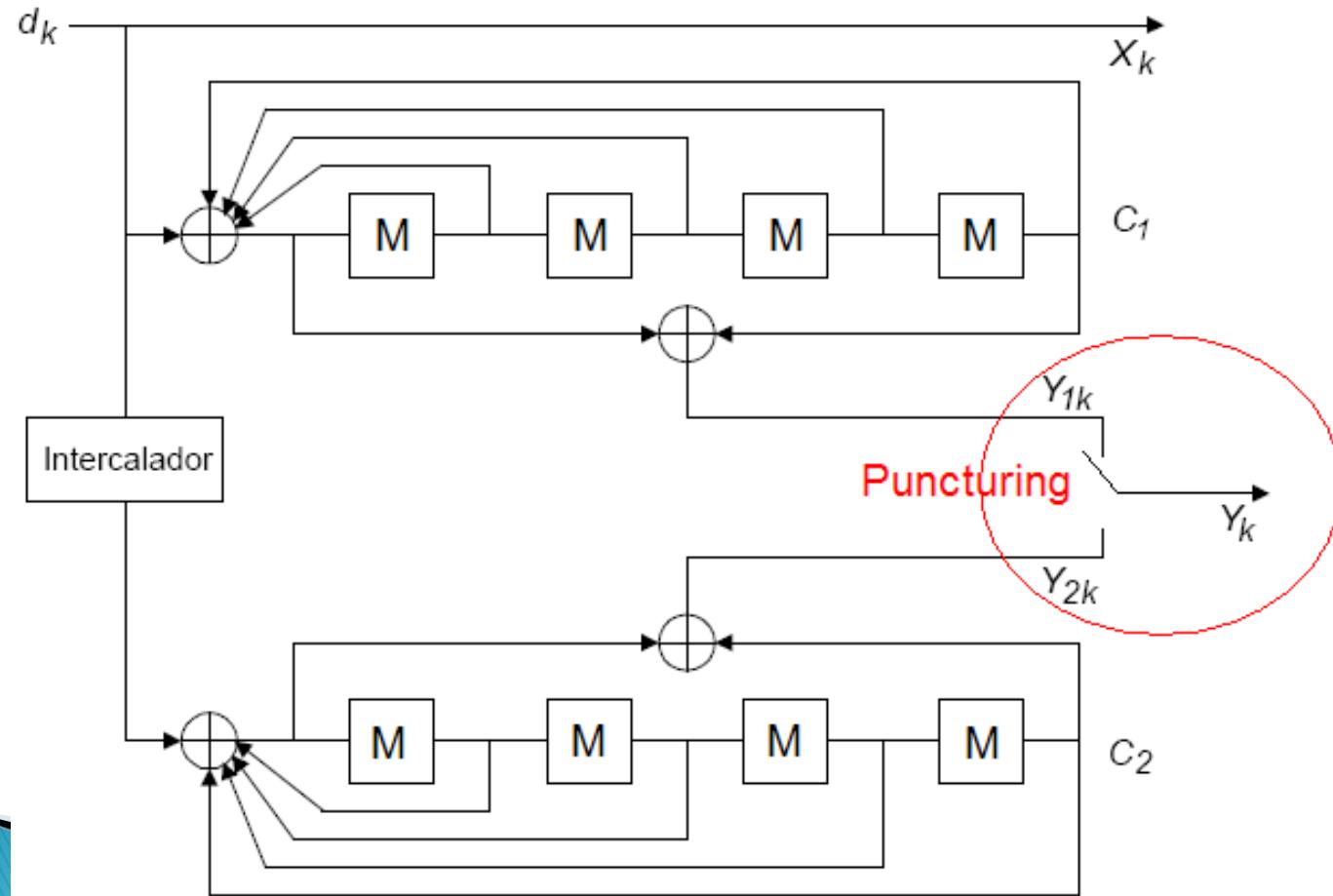
# Puncturing

- ▶ La tasa natural de un turbo-código es de  $1/3$ . Para mejorar las tasas de código se utiliza “puncturing”.
- ▶ Por ejemplo, transmitir alternadamente un  $Y_1$  y un  $Y_2$  puede llevar la tasa global a  $1/2$ .

# Puncturing

- ▶ Es el proceso de remover algunos bits de paridad tras la codificación; esto tiene el mismo efecto que codificar a una mayor tasa o menor redundancia. Sin embargo, tras utilizar puncturing es posible utilizar el mismo decodificador sin importar cuántos bits hayan sido removidos. Lo anterior agrega flexibilidad al sistema sin incrementar demasiado su complejidad.

# Escenario para puncturing



Turbo-  
códigos

# Decodificador

- ▶ El decodificador se construye de manera un poco diferente al codificador; consiste en 2 decodificadores elementales (DEC 1 y DEC 2) concatenados en serie, no en paralelo.
- ▶ DEC1 opera para el codificador C1 y DEC2 para C2 respectivamente.
- ▶ DEC1 toma una decisión suave (soft decision) lo cual causa un retardo L1. DEC2 por otro lado tomará una decisión dura (hard decision)

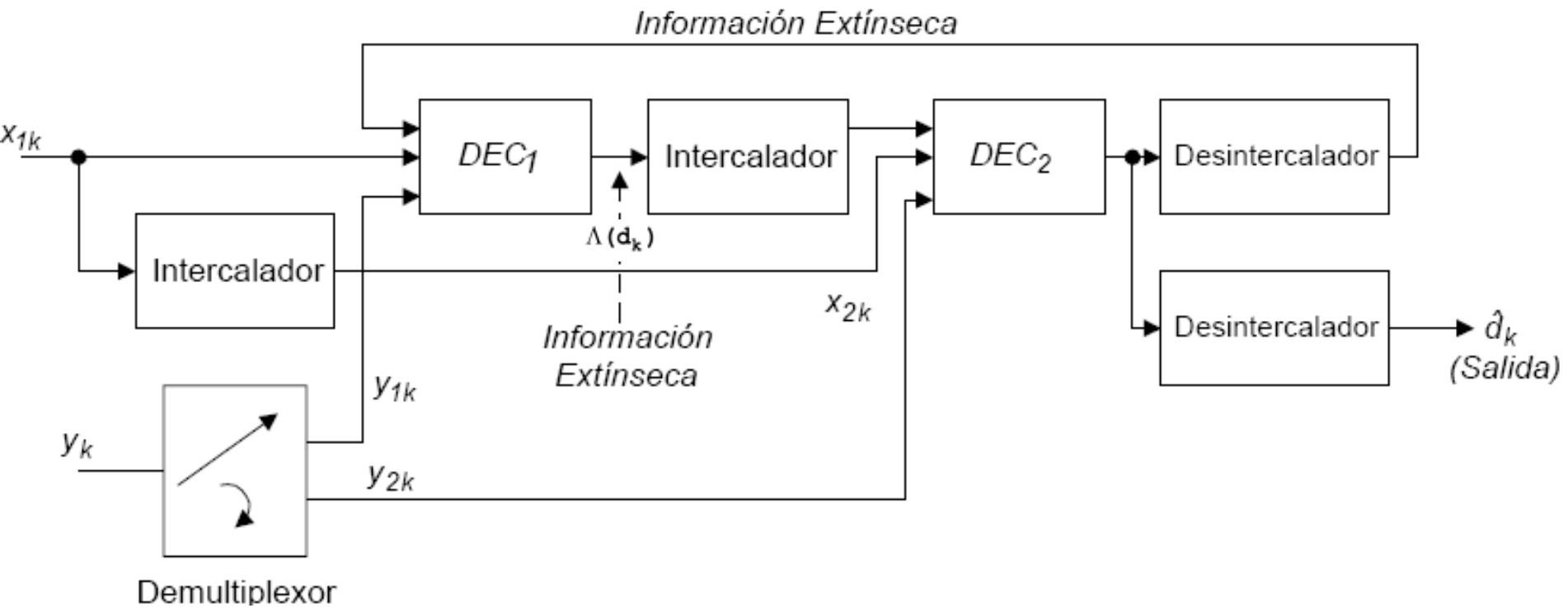
# Soft Decision (Decisión suave)

- ▶ Cuando se toma una soft-decision no se decide si el valor recibido es 0 o 1; en lugar de ello produce para cada bit, un entero en el intervalo [-127,127] el cual dará una “idea” de lo que podría ser el valor del bit así:
  - -127 significa “seguro es 0”
  - -100 significa “posiblemente es 0”
  - 0 significa “podría ser 0 o 1”
  - 100 significa “posiblemente es 1”
  - 127 significa “seguro es 1”

# Soft-decision (2)

- ▶ Soft-decision introduce un aspecto probabilístico a la cadena de bits pero transmite mayor información sobre un bit que simplemente 1 ó 0.
- ▶ Por otro lado un decodificador hard-decision (decisión dura) necesariamente cada vez que reciba un dato tiene que decidir si es 1 ó 0 para la salida

# Decodificador



Turbo-  
códigos

- ▶ Para que la estructura representada sea óptima se requiere que DEC1 utilice no sólo una fracción de la redundancia disponible en la información. Por ello se propone un lazo de realimentación entre la salida de DEC2 hacia DEC1, para que DEC1 pueda evaluar los datos una y otra vez

# Decodificación

- ▶ Si consideramos un canal con ruido blanco gaussiano y asumiendo la iteración k-ésima, el decodificador recibe un par de variables aleatorias:
- ▶ donde  $a_k$  y  $b_k$     $x_k = (2d_k - 1) + a_k$ , con varianza  $\sigma^2$ .  
                                 $y_k = 2(Y_k - 1) + b_k$

# Decodificación

- ▶ La información redundante es des-multiplexada y enviada a DEC1 (cuando  $y_k=y_{1k}$ ) ó DEC2 (cuando  $y_k=y_{2k}$ )
- ▶ DEC1 toma una decisión suave:

$$\Lambda(d_k) = \log \frac{p(d_k = 1)}{p(d_k = 0)}$$

# Decodificación

- ▶  $\Lambda(d_k)$  se denomina logaritmo del radio de posibilidad (LLR → Logarithm of the Likelihood ratio).
- ▶  $p(d_k = i), i \in \{0,1\}$  es la probabilidad a posteriori (APF) que muestra la probabilidad de interpretar un bit recibido como 0 ó 1. Al recibir el LLR, DEC2 toma una decisión dura, es decir, un bit decodificado !