UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA – PROCESAMIENTO DIGITAL DE SEÑALES

ESTUDIO DE ALGORITMOS DE DECODIFICACIÓN ITERATIVA EN TURBO CÓDIGOS Y SU IMPLEMENTACIÓN EN UN FPGA

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
ESQUIVEL JARAMILLO ALFREDO

TUTOR PRINCIPAL
DR. FRANCISCO JAVIER GARCÍA UGALDE, FACULTAD DE INGENIERÍA

MÉXICO, D.F., ENERO 2015

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

JURADO ASIGNADO:

Presidente:	DR. JESÚS SAVAGE CARMONA
Secretario:	DR. CARLOS RIVERA RIVERA
Vocal:	DR. FRANCISCO JAVIER GARCÍA UGALDE
1 er. Suplente:	DR. MARIO PEÑA CABRERA
2 d o. Suplente:	DR. HORACIO TAPIA RECILLAS

Lugar o lugares donde se realizó la tesis:

Laboratorio de codificación y seguridad de los sistemas de información. Posgrado de Ingeniería UNAM. México, D.F

Laboratoire de l'Intégration du Matériau au Système (IMS). Bordeaux, Francia.

TUTOR DE TESIS:

Dr. Francisco Javier García Ugalde

_____

FIRMA

Con números se puede demostrar cualquier cosa.

Thomas Carlyle
(1795-1881)

Estudio de algortimos de decodificación iterativa en turbo códigos y su implementación en un FPGA

Alfredo Esquivel Jaramillo.

DEDICATORIA

A mis padres Sandra y Alfredo

A mis hermanos Daniel y Diana Paulina

A mi abuelo Alfredo, quien ya no se encuentra con nosotros

AGRADECIMIENTOS

A CONACYT, por el apoyo económico que recibí para realizar mis estudios de maestría y por el apoyo de la beca mixta que me permitió realizar una estancia de investigación en el extranjero. Al programa PAPIIT IN–112513, debido al apoyo con el cual me fue posible culminar la realización de la tesis.

A la UNAM, por abrirme sus puertas para realizar mis estudios de maestría y por darme un nuevo panorama del mundo académico.

A mi mamá Sandra por quererme y estar al pendiente de mi.

A mi papá Alfredo por haberme dado lo necesario para que pudiera estudiar.

A mis hermanos con quienes he pasado tanto buenos como malos momentos, pero que siempre han estado a mi lado.

A mis familiares por darme el apoyo y la motivación necesarios. En especial a mis abuelas Silvia y Carmen, y a mi abuelo Vicente por su preocupación. También agradezco a mi tía Mónica el apoyo que he recibido de su parte.

A mi tutor de tesis, el Dr. Francisco García Ugalde por su paciencia y orientación con respecto a este trabajo.

Al Dr. Cristophe Jégo, de la Universidad de Bordeaux I, Francia, por recibirme en su laboratorio y por proporcionarme material y orientación sobre arquitecturas de turbo códigos en FPGA.

A mis profesores, por compartir sus conocimientos conmigo y por darme clases

de muy buena calidad.

A mis sinodales el Dr. Horacio Tapia Recillas, el Dr. Carlos Rivera Rivera, el Dr. Jesús Savage Carmona y el Dr. Mario Peña Cabrera por sus atinadas observaciones y comentarios que me permitieron mejorar mi trabajo.

A mis compañeros de aula y de laboratorio, por el apoyo recibido cuando así lo requería.

Índice general

1 Introducción

Las comunicaciones digitales, así como el almacenamiento de información, se han vuelto parte importante de nuestra vida cotidiana. En muchas ocasiones, se da por hecho que una transmisión de datos así como el almacenamiento de los mismos, se da de forma natural. Las personas rara vez se dan cuenta de que ocurren errores en la transmisión de datos y en los sistemas de almacenamiento, sin embargo si no fuera por el uso de técnicas de control de errores, sería imposible la transmisión y el almacenamiento confiables.

Los errores en los sistemas de transmisión o almacenamiento provienen de múltiples fuentes, tales como: ruido aleatorio, interferencia, desvanecimiento del canal, o defectos físicos, entre otros. Estos errores deben ser reducidos a un nivel aceptable con el objetivo de asegurar la calidad de la transmisión de datos/almacenamiento. Para controlar los errores, se pueden usar técnicas encaminadas a detectar los errores, o bien detectarlos y además corregirlos. En la literatura de comunicaciones digitales, estas técnicas de codificación de control de errores se implementan en la capa física de la red, en los módulos conocidos como codificador de canal y decodificador de canal.

La codificación para el control de errores está enfocada en métodos que entregan información desde una fuente a un destino con un mínimo de errores. Como tal, puede ser vista como una rama de la teoría de la información y remonta sus orígenes a los trabajos de Shannon [24] en la década de los 40 del siglo pasado. El trabajo teórico señala qué es posible, y el segundo teorema de Shannon de la codificación para un canal con ruido, define la existencia de control de errores. Por otro lado, los problemas involucrados en encontrar e implementar códigos eficientes, han tenido alguna implicación en que los efectos prácticos de utilizar la codificación sea de alguna manera diferente de lo planteado teóricamente por las limitaciones de memoria disponible y tiempo de transmisión de las comunicaciones.

El trabajo de Shannon demostró que cualquier canal de comunicación puede ser caracterizado por una capacidad teórica a la que la información puede ser transmitida de manera confiable. A cualquier tasa de transmisión menor a tal capacidad del canal, debería ser posi-

ble poder transferir información a tasas de error que puedan ser reducidas a prácticamente a cero. Es posible proveer el control de errores introduciendo redundancia en las transmisiones. Esto significa que símbolos adicionales se incluyen en el mensaje de datos, con el resultado de que sólo ciertos patrones llamados palabras del código, corresponden a transmisiones válidas. Una vez que se ha introducido un código para el control de los errores, las tasas de error pueden hacerse tan bajas como se requiera extendiendo la longitud del código, promediando así los efectos del ruido sobre un periodo más largo.

Diversas implementaciones prácticas se concentran en las posibles mejoras que se pueden obtener, en comparación de comunicaciones no codificada, o incluso de otros esquemas de codificación. El uso de la codificación puede incrementar el rango operacional en un sistema de comunicaciones, reducir las tasas de errores, reducir los requerimientos de potencia transmitida, o inclusive aprovechar todos estos beneficios.

La codificación para el control de errores es en principio una combinación de técnicas de procesamiento digital de señales encaminadas a reducir los efectos del ruido del canal sobre varias señales transmitidas. Por su carácter aleatorio, la cantidad de ruido que afecta a un único bit transmitido es menos predecible que la experimentada sobre un mayor intervalo de tiempo, esto nos lleva a considerar la información a codificar por bloques de gran tamaño.

1.1. Justificación.

Tradicionalmente, el diseño de buenos códigos ha sido realizado en base a estructuras algebraicas, para los cuales existen esquemas viables de decodificación. Dicho enfoque puede ser ejemplificado por los códigos de bloque lineales, códigos cíclicos y códigos convolucionales. La dificultad con estos códigos tradicionales es que, en un esfuerzo por acercarse al límite teórico de la capacidad de canal de Shannon, resulta necesario incrementar la longitud de la palabra de código en un código de bloque lineal, o la longitud de restricción en un código convolucional, lo cual, a su vez causa que la complejidad computacional de un decodificador basado en el criterio de máxima verosimilitud, o uno máximo *a-posteriori* se incremente exponencialmente. Al final, se alcanza un punto en el que la complejidad del decodificador es tan alta que se vuelve físicamente impráctica.

En su publicación de 1948 [24] , Shannon demostró que el desempeño promedio de un conjunto aleatorio de códigos resulta en un error de decodificación exponencialmente decreciente al aumentar la longitud del bloque. Desafortunadamente, como lo fue con su teorema de codificación, Shannon no ofreció una dirección en como construir códigos generados aleatoriamente.

El interés en el uso de códigos elegidos aleatoriamente permaneció inactivo por mucho tiempo hasta que una nueva idea de turbo codificación fue descrita por Berrou *et al.* (1993) [7]; dicha idea fue basada en dos principios de diseño:

1. El diseño de un buen código, cuya construcción está caracterizada por propiedades aleatorias.

2. El diseño iterativo de un decodificador que haga uso de valores de salida suave, aprovechando el algoritmo máximo *a-posteriori* (MAP) debido a Bahl *et al.* (1974) [47].

Conjuntando con estas dos ideas, se demostró experimentalmente que la turbo codificación podía acercarse al límite de Shannon a un costo computacional menor y que no sería

viable con los códigos algebraicos tradicionales. Por lo tanto, se puede decir que la invención de la turbo codificación merece ser catalogada entre los mayores logros técnicos en el diseño de sistemas de comunicación del siglo XX.

Lo que también es notable es el hecho de que la turbo codificación y la decodificación iterativa despertaron de nuevo el interés en trabajos previos de Robert G. Gallager (1962,1963) respecto a los códigos LDPC (*Low Density Parity Check*). Estos códigos también poseen el poder de procesamiento de la información para acercarse al límite teórico de Shannon. El punto importante a notar aquí es el hecho de que ambos: turbo códigos y códigos LDPC son capaces de acercarse al límite teórico de Shannon con un nivel similar de complejidad computacional, asumiendo que ambos tienen una palabra de código lo suficientemente larga. Específicamente, los turbo códigos requieren un tamaño largo de entrelazador, mientras que los códigos LDPC requieren una mayor palabra de código.

De esta manera se tienen dos clases básicas de técnicas de codificación con un enfoque probabilístico: turbo códigos y códigos LDPC, los cuales se complementan uno con el otro en el siguiente sentido:

Los turbo códigos son simples de diseñar pero el algoritmo de decodificación puede ser demandante. Por el contrario, los códigos LDPC son relativamente complejos pero son simples de decodificar.

En el presente trabajo de tesis, se realizará el estudio de los algoritmos de decodificación asociados a los turbo códigos, debido a que son una componente importante de varios estándares de comunicaciones digitales. Se tendrán en cuenta detalles de la implementación en *hardware* de un turbo decodificador. Específicamente, este consiste de algunos módulos de memoria y un decodificador SISO *Soft Input-Soft Output*(entrada suave-salida suave). Aunque la estructura conceptual emplea dos decodificadores SISO, uno de ellos permanece inactivo mientras el otro se encuentra en operación. El proceso de decodificación de cada código componente dependiente de la información *a priori* del otro código, y de la aleatoriedad del entrelazador hace difícil comenzar una media iteración de este código componente hasta que todas las entradas necesarias se encuentren disponibles. Más aún, los dos códigos componentes son idénticos en estructura, por lo que pueden compartir el mismo decodificador SISO. En una arquitectura práctica, las memorias almacenan los datos recibidos y los datos temporales de la decodificación, y un controlador global puede manejar la alternancia entre los códigos componentes. Es posible realizar mejoras o modificaciones al algoritmo MAP de la decodificación con la finalidad de poder optimizar el área y el consumo de potencia en el diseño, sin embargo, el presente trabajo se limita a realizar la implentación del algoritmo en su forma estándar. Cabe destacar, que el prototipo final y el aprendizaje en arquitecturas de mapeo de algoritmos de decodificación en *hardware* derivado de este trabajo de tesis se realizó tanto en la UNAM como en el laboratorio IMS de la Universidad de Burdeos I, Francia en una estancia de investigación.

1.2. Planteamiento del problema.

Resulta necesario implementar algoritmos de decodificación, ya que los sistemas receptores de comunicaciones deben de corregir los errores que aparecieron durante la transmisión de la información. Y de esa manera, se puede garantizar la confiabilidad en la recepción correcta de los datos. Normalmente, las funciones de corrección en dispositivos reales se

implementan en circuitos ASIC[1]. Por lo que resulta viable, realizar la simulación e implementación de algoritmos de decodificación iterativa para turbo códigos, en plataformas de *hardware* como un DSP[2] o un FPGA[3]. En el presente trabajo de tesis, se considera el uso de una plataforma FPGA.

1.3. Objetivo.

Realizar el estudio de algoritmos de decodificación iterativa con turbo códigos, específicamente el algoritmo Max-Log-MAP, así como el diseño e implementación de un decodificador en una plataforma FPGA para evaluar el desempeño en términos de tasa de error de bit (BER- bit error rate) para diferentes longitudes de trama.

[1] Application-Specific Integrated Circuit
[2] Digital Signal Processor
[3] Field Programmable Gate Array

2

Contexto y estado del arte

En el contexto de las comunicaciones digitales, la codificación para el control de errores tiene una historia que data de mediados del siglo XX. En años recientes, este campo ha sido revolucionado por códigos que son capaces de acercarse a los límites teóricos de desempeño que estableció Claude Shannon en 1948 [24]. Esto ha sido impulsado por una tendencia a alejarse de enfoques puramente discretos y combinatorios, yendo hacia códigos que estén más ligados al canal físico y a las técnicas con decodificación suave (*soft decision*).

La codificación para detección y corrección de errores es la forma por medio de la cual los errores que han sido introducidos en los datos digitales (como resultado de una transmisión a través de un canal con ruido) pueden ser corregidos, basándose en la señal recibida. Conjuntamente, la detección y corrección de errores se refieren como la codificación de control de errores (o codificación de canal), la cual provee la diferencia entre un sistema de comunicaciones operacional y uno disfuncional. Son técnicas necesarias en los sistemas modernos basados en la teoría de la información. Las aplicaciones son diversas, en efecto, se puede encontrar el control de errores en un disco compacto, en el disco duro, en cada llamada telefónica realizada sobre un teléfono celular digital, en cada paquete transmitido sobre internet. De hecho, casi todos los comercios de hoy en día toman una ventaja de la codificación de canal [1].

Los problemas de control de errores se pueden resolver desde los modos de operación ARQ (*Automatic Repeat Request*)y FEC(*Forward Error Correction*). La diferencia principal radica en que en ARQ hay retransmisión de la información en caso de que haya detección de una palabra corrompida. ARQ se aplica principalmente en redes de datos, donde hay un enlace físico del tipo full-dúplex. En FEC se confía en el uso controlado de redundancia en la información transmitida, lo cual es adecuado en corrección de errores para enlaces unidireccionales (half-dúplex) entre el transmisor y receptor. Un ejemplo es un sistema llamado paginación, donde se envían mensajes de texto a un usuario móvil. Los códigos que se abordan en la presente tesis operan en el modo FEC.

Los diversos códigos correctores de errores tienen sus bases sustentadas en ciertas áreas matemáticas. Tradicionalmente, estos códigos se han clasificado en **códigos de bloque** y en **códigos convolucionales**, los cuales se construyen con una fuerte estructura algebraica. Los primeros trabajan con bloques de información, mientras que los segundos operan sobre secuencias de información [2]. Estos códigos fueron eficientes en su tarea de mantener los errores dentro de un cierto límite de acuerdo con la relación señal a ruido (E_b/N_0) recibida, con una complejidad de implementación relativamente baja. Sin embargo, esta relación estaba aún lejana (> 3 dB) del límite teórico de Shannon. Los códigos concatenados disminuyeron esta frontera sin que se incrementara mucho la complejidad, introduciendo la idea de combinar las capacidades de control de errores de dos, o más códigos [4]. Sin embargo, esos códigos individuales no intercambiaban información en un modo de realimentación.

El descubrimiento de los turbo códigos fue un evento revolucionario en la disciplina de codificación de control de errores. Este avance se logró en el año 1993, con los trabajos de Berrou, Glavieux y Thitimashida [7]. El desempeño cercano al límite teórico de Shannon obtenido, y los algoritmos de decodificación iterativa han estimulado esfuerzos de investigación para su comprensión y realización, ya que las ideas bajo las cuales se sustenta su operación colaborativa fueron novedosas. Se demostró que la cooperación eficiente entre elementos simples podía conducir a resultados que superan la suma de las contribuciones individuales. Estas ideas, que ya eran conocidas por la comunidad científica al inicio de los 90, son:

- La adaptación de los códigos convolucionales **no sistemáticos** a códigos convolucionales **sistemáticos**.
- El uso de módulos de decodificación, que reciben a su entrada probabilidades de los símbolos recibidos (llamadas **entradas suaves**), en vez de decisiones duras sobre estos bits, que conllevan cierta pérdida de información (las llamadas **decisiones duras**). Más aún, la salida de la decodificación, entendida como un refinamiento de estas probabilidades, por el proceso de decodificación (llamadas **salidas suaves**).
- Dos codificadores operando en diferentes órdenes de la información de entrada, y dos decodificadores que pueden retroalimentar sus salidas entre ellos en un modo iterativo. El **entrelazador** es el elemento que construye una versión diferente de la entrada original mediante la permutación. Se demostró que el entrelazador era el elemento esencial cuando se lograba el desempeño teórico de los turbo códigos, siendo el dispositivo que tiene la propiedad aleatoria que había predicho Shannon. En la presente tesis, se pretende evaluar el desempeño de distintos entrelazadores.

2.1. Generalidades de codificación de canal.

Los sistemas de comunicaciones digitales están diseñados para transmitir información sobre canales ruidosos para proveer enlaces confiables entre la fuente y el destino. Debido a las perturbaciones del ruido, pueden ocurrir errores durante el proceso de la transmisión, ocasionando que los mensajes producidos por la fuente no sean correctamente interpretados en el lado del destino.

En la figura 2.1 se ilustra un marco general de un sistema de comunicaciones digitales. El sistema consta de tres partes principales: transmisor, canal y receptor. En este enlace,

los datos digitales de una fuente son codificados y modulados (y posiblemente cifrados) para la comunicación a través de un canal. En el otro extremo del canal, los datos son demodulados, decodificados y enviados a un utilizador. Todos estos elementos tienen descripciones matemáticas y teoremas en los que la teoría de la información establece su desempeño. A continuación se detallan algunos aspectos importantes de estos elementos.

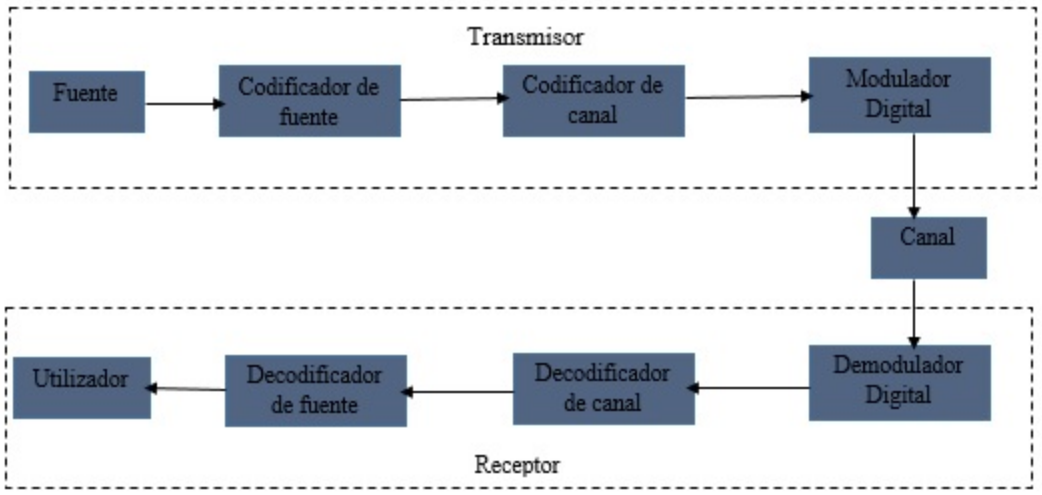

Figura 2.1: Marco general de un sistema digital de comunicaciones.

- La **fuente** representa los datos a ser enviados que se representan en forma digital. Desde el punto de vista de la teoría de la información son vistos como flujos de números aleatorios gobernados por una distribución probabilística.

- El **codificador de la fuente** comprime los datos de una fuente de información para evitar la ocupación de un ancho de banda en exceso, respecto al necesario, en consecuencia disminuye el consumo de potencia. Para facilitar el intercambio de contenidos, se usan estándares industriales (como JPEG, MPEG).

- El **codificador de canal**, representado por códigos FEC, es el primer elemento en la corrección o detección de errores. Por codificación de canal, se entiende como las transformaciones de señales diseñadas para mejorar el desempeño de las comunicaciones, permitiendo que las señales transmitidas resistan los efectos de diferentes imperfecciones del canal, como ruido, interferencia y desvanecimiento. Estas técnicas de procesamiento de señales son formas de lograr compensaciones deseables en los sistemas (por ejemplo, desempeño vs. error, potencia vs. ancho de banda). [6]. El uso de circuitos integrados a larga escala y técnicas de procesamiento digital de señales de alta velocidad ha hecho posible tener mejoras tan grandes como de 10 dB a través de estos métodos, a un costo mucho menor que el uso de otros métodos, como transmisores de alta potencia con grandes antenas.

A través de los códigos correctores se agrega redundancia al flujo original de información, con el fin de mapearla a un espacio vectorial de mayor dimensión; de esta manera, se aumenta la distancia entre palabras de código. La presente tesis se dedica al estudio de algoritmos que se implementan en el codificador de canal y su correspondiente decodificador de canal. Debido a la redundancia introducida, debe de haber más símbolos a la salida del codificador que a la entrada. Con frecuencia, un codificador de canal opera aceptando un bloque de k símbolos de entrada y produciendo a su salida un bloque de n símbolos, con $n > k$. La tasa de ese codificador de canal es $R = k/n$, por lo que siempre $R < 1$.

- La secuencia de bits que sale del codificador de canal se debe transformar en formas de onda que se adapten desde el punto de vista electromagnético a las características del canal, para facilitar su propagación. El cometido del **modulador digital** es hacer corresponder las formas de onda de señales de un conjunto finito y discreto, con los bits que tienen a su entrada.

- El **canal** consiste en el medio físico sobre el que se transmite la señal que sale del modulador. Por ejemplo, el canal puede ser el cable de pares de la línea telefónica, o el espacio libre en el caso de señales de radio. Por desgracia, todos los medios físicos deterioran la señal transmitida, ya sea distorsionando la forma de onda, o corrompiendo la señal con ruido.

  La teoría de la codificación empezó con los trabajos de C. Shannon (1948) [24], quien estableció el problema fundamental de la transmisión confiable de información. Shannon introdujo la capacidad de canal C (expresada en bits por uso del canal) como una medida de la información máxima que puede ser transmitida sobre un canal con ruido aditivo. Su principal teorema es como sigue: si la tasa de información R de la fuente es menor que la capacidad de canal C, luego es teóricamente posible lograr una transmisión confiable (libre de errores) sobre el canal a través de una codificación apropiada. Usando el conjunto de códigos aleatorios, este teorema demostró la existencia de un código que permitiera transmitir la información con cualquier probabilidad de error dada, tendiente a cero. Pero no da una idea de cómo encontrar esos códigos que lograsen la capacidad de canal. El trabajo de Shannon estimuló a la comunidad de la teoría de codificación para encontrar códigos correctores de errores capaces de lograr la capacidad de canal.

- Los bloques de la figura etiquetados como **demodulador digital, decodificador de canal** y **decodificador de fuente** realizan las funciones inversas al modulador digital, codificador de canal y codificador de fuente, respectivamente. Por ejemplo, el demodulador digital tiene que convertir la forma de onda recibida en grupos de bits, por lo que debe estimar la forma de onda transmitida que se parezca más a la recibida.

- Con respecto al bloque utilizador, éste implica la conversión digital-analógica, en aquellos casos en que la señal que se transmitió es de esta naturaleza. Y la presentación, o almacenamiento, en aquellos casos en que la fuente es digital.

Los bloques codificador de fuente, codificador de canal y modulador componen el **transmisor**, mientras que el demodulador, el decodificador de canal y el decodificador de fuente

componen el **receptor**. Los canales son caracterizados por modelos matemáticos que son suficientemente exactos para representar los atributos del canal actual. Para la presente tesis, se considerará uno de dos canales idealizados, el canal simétrico binario (BSC) y el canal de ruido blanco gaussiano aditivo (AWGN). Existen otros canales como el Rayleigh, el cual no se considerará, pero nuestra propuesta se puede extender a este caso.

2.2. Decodificación dura y decodificación suave.

La información que provee el demodulador digital al decodificador de canal puede tener diferentes niveles. El caso más sencillo de considerar es cuando el demodulador solamente puede determinar que cada forma de onda recibida tiene dos posibles niveles: alto o bajo. En este caso, el decodificador de canal recibe una secuencia de valores binarios (llamados **duros**). A partir de esta secuencia, el mensaje de la fuente debería ser recuperado. Este proceso de decodificación se llama decodificación por decisiones duras. Sin embargo, es posible incrementar las capacidades de información del demodulador para que pueda proveer, adicionalmente a la secuencia de valores binarios, valores (llamados **suaves**) de confiabilidad para cada valor duro. Los valores suaves representan: qué tan confiado está el demodulador de que cada valor duro esté correctamente estimado. Por lo tanto, el decodificador tiene más información que, de manera probabilista, le permite mejorar las decisiones tomadas en todo el proceso de decodificación. Más aún, el decodificador de canal puede generar valores de confiabilidad por sí mismo. Dicho decodificador se llama un decodificador SISO (*Soft Input Soft Output*), y el algoritmo que se utiliza es un algoritmo de decodificación tipo SISO.

Cuando los valores suaves son considerados en el sistema de comunicaciones completo, se pueden observar mejoras importantes en el desempeño del sistema. Se ha demostrado que estas decisiones suaves son convenientes para mejorar las capacidades de corrección de errores en el sistema de comunicación. Específicamente, los algoritmos de decodificación SISO han sido propuestos como parte del decodificador de canal, para llevar a cabo la decodificación iterativa. La idea principal es reemplazar códigos largos y complejos, por códigos simples que cooperen entre ellos, a través del intercambio de valores suaves [22]. Los turbo códigos, están basados en esta idea, conocida en los modelos Bayesianos como propagación de creencia (*belief propagation*).

2.3. Proceso de decodificación de canal.

Para encontrar los bits transmitidos más confiables, es posible aplicar dos principios de decodificación: MAP (Maximum A Posteriori) y ML (Maximum Likelihood). La decodificación MAP permite maximizar la probabilidad $P(\mathbf{u}_k|\mathbf{r}_k)$. Por lo tanto, a partir de los valores recibidos a la salida del canal $\mathbf{r}_k$, se trata de estimar las señales transmitidas más confiables $\mathbf{u}_k$. Usando la regla de Bayes, se tiene:

$$p(\mathbf{u}_k|\mathbf{r}_k) = \frac{p(\mathbf{r}_k|\mathbf{u}_k)P(\mathbf{u}_k)}{p(\mathbf{r}_k)} \tag{2.1}$$

con $p(\mathbf{r}_k|\mathbf{u}_k)$ la pdf (función de densidad de probabilidad) condicional de los valores de salida del canal dada $\mathbf{u}_k$, y $P(\mathbf{u}_k)$ la probabilidad de que las señales u_k fueron transmitidas.

$P(\mathbf{u}_k)$ también se conoce como la probabilidad a-priori. Proporciona información adicional al proceso de decodificación. Complementario al proceso de decodificación MAP, la decodificación ML busca maximizar $p(\mathbf{r}_k|\mathbf{u}_k)$, en vez de $p(\mathbf{u}_k|\mathbf{r}_k)$. Ambos principios de decodificación conducen a un desempeño de decodificación similar mientras no exista información a priori.

El principio de decodificación ML es equivalente a minimizar la distancia euclidiana como se verá en el siguiente apartado; permite la convergencia a las formas de onda transmitidas más cercanas $\mathbf{u}_k$, respecto a $\mathbf{r}_k$. La decodificación MAP es óptima en el sentido de que maximiza la probabilidad de símbolo por símbolo, es decir, encuentra el bit transmitido más probable. Por otro lado, el principio de decodificación ML minimiza los errores con respecto a la secuencia de información transmitida como un todo.

2.4. Modelo del canal.

Asumiendo una modulación, sincronización y demodulación perfectas, el canal es representado mediante un modelo equivalente en tiempo discreto. Para el propósito de nuestro estudio se utiliza un esquema de modulación y modelos de canal simple, sin comprometer el objetivo final del trabajo. Por lo tanto, se asume un canal AWGN (Additive White Gaussian Noise) de media 0 y varianza σ^2.

Para los propósitos de este trabajo, el esquema de modulación asumido es BPSK (*Binary Phase Shift Keying*) donde los valores binarios $c_k \in \{0, 1\}$ son mapeados en símbolos modulados en fase $u_k \in \{-1, 1\}$. La palabra de código modulada se denota como u y la salida del demodulador como r. Se asume además un canal sin memoria. Por lo tanto, la probabilidad condicional se expresa como:

$$p(r|c) = p(r|u) = \prod_{k=0}^{n-1} p(r_k|u_k) \tag{2.2}$$

donde $p(r_k|u_k)$ es obtenida utilizando

$$p(r_k|u_k) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(r_k - u_k)^2}{2\sigma^2}\right) \tag{2.3}$$

Por conveniencia, puede ser apropiado expresar el logaritmo de las probabilidades, en vez de la probabilidad misma. Combinando las dos expresiones anteriores y tomando el logaritmo natural de la expresión resultante se obtiene:

$$ln(p(r_k|u_k)) = -nln(\sigma\sqrt{2\pi}) - \frac{1}{2\sigma^2}\sum_{k=1}^{n-1}(r_k - u_k)^2 \tag{2.4}$$

La suma que aparece en la expresión anterior es la distancia euclidiana que se mencionó en la sección anterior.

El desempeño en BER(*Bit Error Rate*) y FER (*Frame Error Rate*) está dado generalmente como una función de la razón señal a ruido E_b/N_0, donde E_b es la energía promedio por bit de información y $\frac{N_0}{2}$ es la densidad espectral de potencia del ruido. La varianza del canal se puede expresar como una función de la razón señal a ruido y de la tasa del código R, usando:

$$\sigma^2 = \frac{1}{2R \cdot \frac{E_b}{N_0}} \tag{2.5}$$

Para entender más detalles en referencia a los modelos de canal, se puede consultar el apéndice A.

2.5. Límites fundamentales del canal.

Para un canal dado, existe un límite superior en la capacidad, relacionado con la relación señal a ruido (SNR, por sus siglas en inglés) y al ancho de banda del sistema. Shannon introdujo el concepto de capacidad del canal, C, como la máxima tasa a la cual se puede transmitir información sobre un canal ruidoso. La capacidad de información de un canal continuo de ancho de banda B Hertz, perturbado por ruido AWGN de densidad espectral de potencia $N_0/2$ y limitado en ancho de banda a B, está dada por

$$C = Blog_2\left(1 + \frac{P}{N_0 B}\right) \text{ bits por segundo.} \tag{2.6}$$

donde P es la potencia promedio recibida. Para entender cómo se obtiene la fórmula anterior se puede consultar el apéndice A.

Del teorema de codificación del canal ruidoso (apéndice A), se sabe que para que haya comunicación con una probabilidad de error arbitrariamente baja, la tasa de transmisión no debe exceder a la capacidad del canal, es decir, $R < C$. Bajo esta restricción y bajo el hecho de que $P = E_b R$ (donde R es la tasa de bits por segundo y E_b es la energía promedio por bit), se obtiene el límite inferior en E_b/N_0 para una transmisión libre de errores.

$$\frac{R}{B} < \frac{C}{B} = log_2\left(1 + \frac{R}{B}\frac{E_b}{N_0}\right) \tag{2.7}$$

Despejando E_b/N_0 y considerando que $\eta = \frac{R}{B}$ se tiene que

$$\frac{E_b}{N_0} > \frac{2^{\frac{R}{B}} - 1}{\frac{R}{B}} = \frac{2^{\eta} - 1}{\eta} \tag{2.8}$$

Esta relación establece la condición para una comunicación confiable en términos de eficiencia del ancho de banda η y E_b/N_0 (que es una medida de la eficiencia de potencia de un sistema). Una gráfica de esta relación se presenta en la figura 2.2.

El valor mínimo de E_b/N_0 para el cual es posible una comunicación confiable se obtiene haciendo $\eta \to 0$, o equivalentemente $B \to \infty$. Hasta el momento, los turbo códigos son los códigos correctores que más se acercan a este límite teórico, gracias a su carácter aleatorio y a los algoritmos de decodificación iterativa que ellos utilizan.

En general, la comunicación confiable es posible a través de la codificación de canal, lo cual implica asignar mensajes a bloques de entrada del canal y usando solamente un subconjunto (palabras del código) de todos los bloques posibles. Hasta el momento no se han estudiado mapeos específicos entre mensajes y secuencias a la entrada del canal. Se podría hacer un análisis de la capacidad del canal utilizando una codificación aleatoria (para ver detalles matemáticos, consultar [5]).

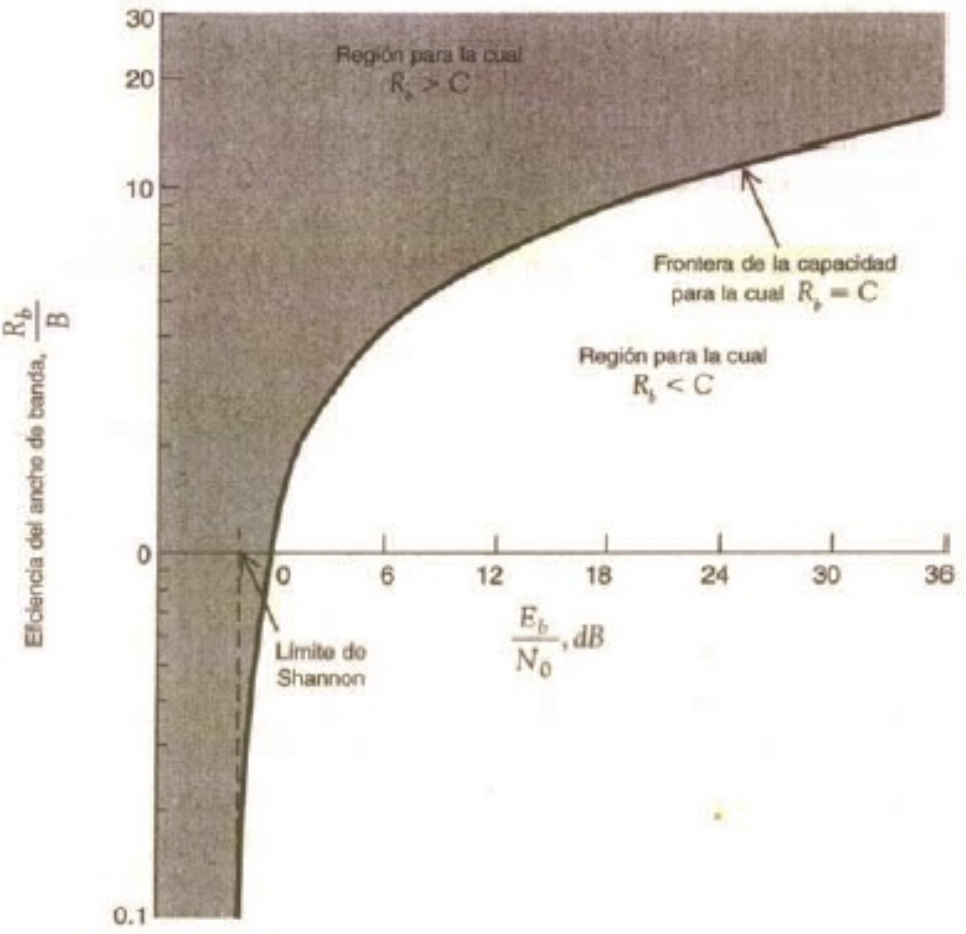

Figura 2.2: Diagrama de eficiencia del ancho de banda [2].

De modo general, en la codificación aleatoria no se encuentra el mejor mapeo del conjunto de mensajes a las secuencias de entrada al canal ni se analiza el desempeño de dicho mapeo; más bien, se promedia la probabilidad de error sobre todos los mapeos posibles, y se muestra que si la tasa de transmisión es menor a la capacidad del canal, el promedio de conjunto de la probabilidad de error, promediado sobre todos los mapeos posibles, tiende a cero conforme se incrementa la longitud de bloque. Se puede concluir que debe de existir al menos un mapeo de entre todos los mapeos, para los que la probabilidad tiende a cero, conforme se incrementa la longitud del bloque.

La demostración original del teorema de codificación de canal, presentado por Shannon en 1948, estaba basada en la codificación aleatoria, y por lo tanto no fue constructiva en el sentido de que proveía solamente la existencia de buenos códigos, pero no decía ningún

método para su diseño. Desde luego, basado en la idea de codificación aleatoria, se puede argumentar que hay una buena probabilidad de que un código generado aleatoriamente sea un buen código. El problema, sin embargo, es que la decodificación de un código generado aleatoriamente, cuando las secuencias de palabras de código son largas, se vuelve extremadamente compleja, haciendo imposible su uso en sistemas prácticos. El desarrollo de la teoría de la codificación en las décadas posteriores a 1948, se ha enfocado en diseñar esquemas de codificación que tengan suficiente estructura para hacer que su decodificación sea práctica, y al mismo tiempo cerrar la brecha entre un sistema no codificado y los límites derivados por Shannon.

El enfoque en el siguiente capítulo será sobre esquemas de codificación de canal con algoritmos de decodificación manejables, que son usados para mejorar el desempeño de sistemas de comunicación sobre canales ruidosos. Se dará una introducción a los códigos de bloque cuya construcción está basada en estructuras algebraicas familiares como grupos, anillos y campos. En seguida se detallarán algunos esquemas de codificación que se representan mejor en términos de enrejados “trellis”.

3

Códigos correctores de errores de bloque y convolucionales

Como se mencionó en el capítulo anterior, el teorema de la codificación del canal ruidoso indica que si se usan códigos muy largos, las comunicaciones confiables pueden ocurrir con la mínima SNR. Sin embargo, los códigos verdaderamente aleatorios no son prácticos para implementar. Los códigos deben de tener cierta estructura para que puedan tener algoritmos computacionalmente tratables de codificación y de decodificación. Los códigos de bloque lineales (incluyendo los cíclicos), los códigos convolucionales y los turbo códigos tienen una estructura algebraica subyacente que los hace prácticamente realizables. El siguiente capítulo se enfoca en el tema principal de esta tesis, los turbo códigos, los cuales comparten muchos de los componentes y terminología de los códigos de bloque y los convolucionales. Por esta razón, es útil tener un panorama de los códigos de bloque y los convolucionales antes de tratar los turbo códigos.

3.1. Códigos de bloque.

Un código binario de bloque es aquel que tiene todas sus palabras de la misma longitud. La secuencia binaria se divide en bloques secuenciales de k bits de largo, y cada bloque de k bits se convierte en un bloque codificado de n bits, donde $n > k$. Se dice que se tiene un código de bloque $C(n, k)$. Cada palabra de código depende solo del mensaje actual de la fuente, y no de mensajes previos. En general, los símbolos de información y codificados están definidos en un campo finito de Galois $GF(p^q)$, donde p es un número primo. Se considerará el caso binario, es decir, $p = 2$. El código puede generar $2^{q \cdot k}$ diferentes palabras de código. En este trabajo consideraremos únicamente el caso en el que $q = 1$, es decir, los códigos de bloque definidos sobre el campo binario $GF(2)$. Por lo cual, el conjunto $K = \{0, 1\}$

es un campo binario.

Los bloques de k bits forman 2^k secuencias de mensajes distintas llamadas k-tuplas. Los bloques de n bits pueden formar diversas 2^n secuencias distintas referidas como n-tuplas. El conjunto de todas las n-tuplas definidas sobre $K = \{0, 1\}$ es denotado por K^n. El codificador de canal realiza el mapeo $T : U \to V$, donde U es un conjunto de palabras de datos binarias de longitud k y V es un conjunto de palabras de código binarias de longitud n, con $n > k$. Cada una de las 2^k palabras de datos es mapeada a una palabra de código única. La razón k/n se conoce como la tasa de código.

3.1.1. Códigos de bloque lineales.

El campo binario $GF(2)$ tiene dos operaciones, suma y multiplicación, tales que el resultado de sus operaciones se encuentra en K. Las reglas de suma y multiplicación son similares a las de las correspondientes a las compuertas lógicas XOR (para la suma) y AND (para la multiplicación).

Sean $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ y $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ dos palabras de código en un código C. La suma de $\mathbf{a}$ y $\mathbf{b}$, denotada por $a \oplus b$ está definida por $(a_1 \oplus b_1, a_2 \oplus b_2, ..., a_n \oplus b_n)$. Un código C se dice que es lineal si la suma de dos palabras de código es también una palabra de código en C.Un código lineal debe contener la palabra de código nula $\mathbf{0} = (0, 0, ..., 0)$ ya que $\mathbf{a} \oplus \mathbf{a} = \mathbf{0}$. [18]

3.1.2. Codificación de códigos de bloque lineales.

El proceso de codificación consiste en descomponer inicialmente los datos en k-tuplas, llamadas mensajes $\mathbf{m}$, y luego realizar un mapeo uno-a-uno de cada mensaje m_i a una n-tupla x_i llamada una palabra de código. Luego, el proceso de codificación puede ser descrito mediante una multiplicación matricial $\mathbf{x} = \mathbf{mG}$, donde $\mathbf{G}$ es la matriz generadora $k \times n$ (cuyas filas forman una base para C); la multiplicación de matrices se hace sobre el campo $GF(2)$.

La distancia de Hamming $v(x_i, x_j)$ es el número de posiciones en las cuáles difieren dos palabras de código. Se puede encontrar primero tomando la suma módulo 2 de las dos palabras de código, y enseguida contar el número de unos que resultan. La distancia mínima d_{min} de un código es la distancia de Hamming más pequeña entre dos palabras de código cualesquiera:

$$\min_{i \neq j} v(x_i, x_j) \tag{3.1}$$

Un código con distancia mínima d_{min}, es capaz de corregir todas las palabras de código con t, o menos errores [1], donde

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor \tag{3.2}$$

donde $\lfloor \cdot \rfloor$ denota la parte entera. Es decir, alrededor de cada palabra del código existe una hiperesfera de radio t en el espacio vectorial de n dimensiones, cuyo centro es una palabra del código.

El peso de Hamming $w(x_i)$ de una palabra de código es la distancia de Hamming entre sí misma y la palabra de código nula, es decir,

$$w(\mathbf{x}_i) = v(\mathbf{x}_i, \mathbf{0}) \tag{3.3}$$

El peso de Hamming puede encontrarse simplemente contando el número de unos en la palabra de código. Para códigos lineales C, la distancia mínima es el peso de Hamming más pequeño de todas las palabras de código, excepto la palabra de código nula [9].

$$d_{min} = \underset{0 \neq x_i \in C}{min}\ w(\mathbf{x}_i) \tag{3.4}$$

Un código (n, k) tiene 2^k palabras de código, que pueden tener pesos entre 0 y n. En cualquier código de bloque lineal existe una palabra de código de peso 0, y los pesos de palabras de código diferentes de cero pueden ser entre d_{min} y n. El polinomio de distribución de pesos (WDP, por sus siglas en inglés), o función de enumeración de pesos (WEF) de un código, es un polinomio que especifica el número de palabras de código de diferentes pesos en un código. El polinomio de distribución de pesos, o función de enumeración de pesos, se denota por $A(Z)$ y está definido por:

$$A(Z) = \sum_{i=0}^{n} A_i Z^i = 1 + \sum_{i=d_{min}}^{n} A_i Z^i \tag{3.5}$$

donde A_i denota el número de palabras de código de peso i.

Se dice que un código es sistemático si el mensaje m está contenido dentro de la palabra de código x. La matriz generadora de códigos sistemáticos se puede expresar como:

$$\mathbf{G} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_k \end{bmatrix} \tag{3.6}$$

donde $\mathbf{I}_k$ es la matriz identidad $k \times k$, y $\mathbf{P}$ es una matriz $k \times (n-k)$. Los últimos k componentes de la palabra de código son iguales a la secuencia de información, mientras que los primeros $n-k$ componentes, llamados los bits de chequeo de paridad, proporcionan la redundancia para la protección contra los errores. La matriz de chequeo de paridad asociada con un código sistemático es:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{n-k} & \mathbf{P}^T \end{bmatrix} \tag{3.7}$$

donde $\mathbf{P}^T$ es la transpuesta de P. La matriz de chequeo de paridad da una forma eficiente de calcular la distancia mínima, que es el número más pequeño de columnas en $\mathbf{H}$ que suman cero. Entre las matrices $\mathbf{G}$ y $\mathbf{H}$ se tiene la siguiente propiedad: $\mathbf{G}\mathbf{H}^T = 0$, esto implica que $\mathbf{H}$ es también la matriz generadora de otro código, cuyas palabras son ortogonales al primero, y se le conoce como código dual [1].

3.1.3. Códigos cíclicos.

Otros códigos de gran importancia son los códigos cíclicos. Los códigos cíclicos son una clase importante de códigos de bloque lineales, caracterizados por el hecho de ser fácilmente implementados usando lógica secuencial o registros de corrimiento. Para un vector dado

de n componentes, $\mathbf{c} = (c_0, c_1, ..., c_{n-1})$, una rotación de corrimiento a la derecha de sus componentes genera un vector diferente. Si esta rotación de corrimiento a la derecha se realiza i veces, una versión cíclicamente rotada del vector original se obtiene como sigue: $c^{(i)} = (c_{n-i}, c_{n-i+1}, ..., c_{(n-1)}, c_0, c_1, ..., c_{(n-i-1)})$.

Un código de bloque lineal dado se dice que es cíclico si para cada uno de sus vectores código la rotación cíclica (i)-ésima es también un vector código del mismo código.

3.2. Códigos convolucionales.

Los códigos convolucionales fueron inventados por Elias (1955) como una alternativa a los códigos de bloque lineales. Los códigos convolucionales tienen una estructura que se extiende sobre la secuencia completa de bits transmitidos, en vez de estar limitado a bloques de palabras de código. La estructura convolucional es adecuada para sistemas de comunicación espacial, satelital y móvil, que requieren codificadores simples y que además logren un buen desempeño cuando se usan con técnicas de decodificación iterativa [12]. Sin embargo, siguen siendo códigos de tasa $R = k/n$, aceptando k nuevos símbolos cada intervalo de tiempo (ventana de tiempo) y produciendo n nuevos símbolos. La aritmética que se emplea en la mayoría de los casos se basa en el campo finito $GF(2)$.

Los códigos convolucionales son códigos lineales que tienen una estructura adicional en la matriz generadora, tal que la operación de codificación se pueda ver como una operación de filtrado, o convolución. Un código convolucional no es más que un conjunto de filtros digitales − sistemas lineales e invariantes en el tiempo, con la secuencia de código siendo la salida multiplexada de las salidas del filtro.

3.2.1. Representación genérica de un codificador convolucional.

Para un código convolucional en cada momento k, el codificador entrega un bloque de N símbolos binarios $\mathbf{c}_k = (c_{k,1}, c_{k,2}, ..., c_{k,N})$, que es una función del bloque de K símbolos de información $\mathbf{d}_k = (d_{k,1}, d_{k,2}, ..., d_{k,K})$ presente a su entrada y de m bloques precedentes. En consecuencia, introducen un efecto de memoria de orden m.

La cantidad $v = m + 1$ se llama la longitud de restricción del código. Si K símbolos de información en la entrada del codificador se encuentran explícitamente en el bloque codificado $\mathbf{c}_k$, es decir:

$$\mathbf{c}_k = (d_{k,1}, d_{k,2}, ..., d_{k,K}, c_{k,K+1}, c_{k,K+2}, ..., c_{k,N}) \tag{3.8}$$

luego se dice que el código es sistemático. En el caso contrario, se dice que es no sistemático [26]. El diagrama general de un codificador con tasa de código K/N y memoria m se representa en la figura 3.1.

En cada momento k, el codificador tiene m bloques de K símbolos de información en memoria. Estos mK símbolos binarios definen el estado $\mathbf{S}_k$ del codificador:

$$\mathbf{S}_k = (d_k, d_{k-1}, ..., d_{k-m+1}) \tag{3.9}$$

Aunque de manera teórica, un codificador convolucional pueda recibir una secuencia infinita de bloques de información $\mathbf{d} = (\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2, ...)$ y produce una secuencia infinita de

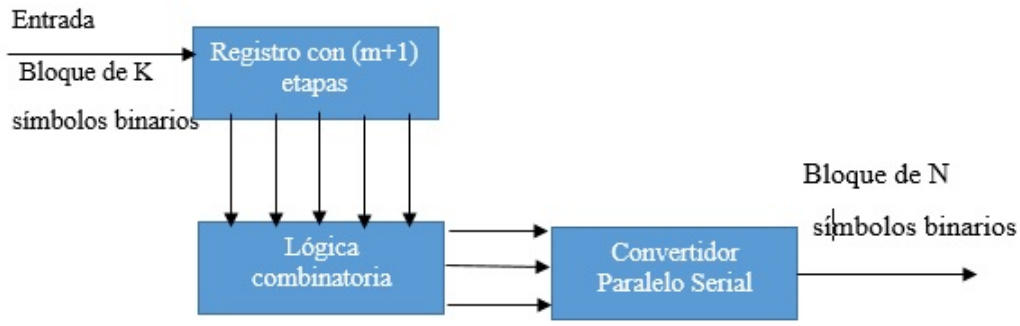

Figura 3.1: Diagrama general de un codificador convolucional con una tasa K/N y memoria m

bloques codificados $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, ...)$, en la práctica estas secuencias pueden ser finitas, para resolver problemas de sincronización.

El codificador convolucional puede ser implementado utilizando un conjunto de flip-flops y compuertas XOR. Una representación general de este codificador se muestra en la figura 3.2. [27]. Se tienen m flip-flops con la disposición mostrada, donde cada uno de ellos almacena un bit $s_{k,i} \in \{0, 1\}$.

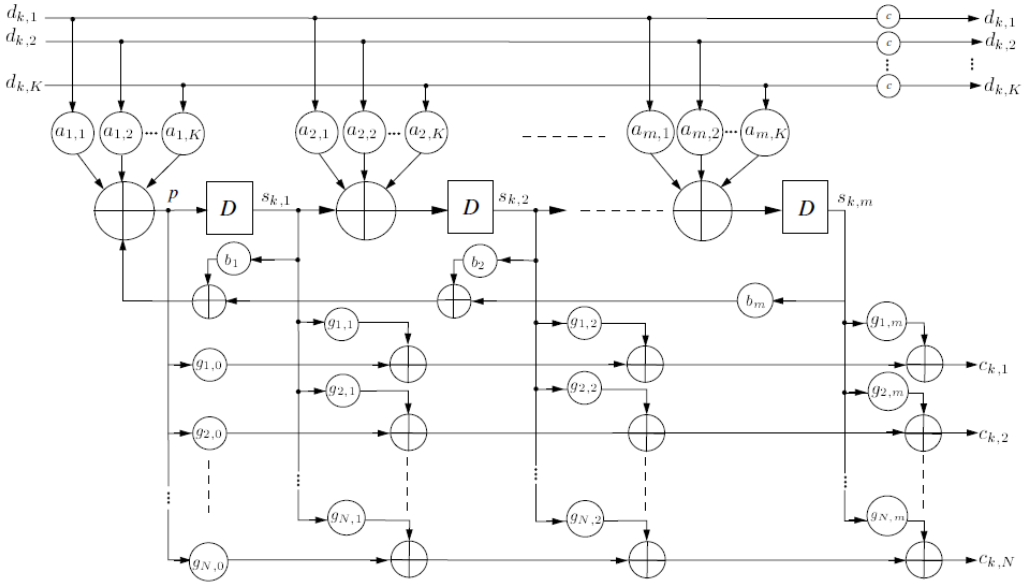

Figura 3.2: Representación genérica del codificador de un código convolucional.

Sea S el conjunto de posibles estados del codificador. Por lo tanto, el estado $s_k \in S$, el

estado del codificador en el tiempo k, puede ser expresado como:

$$s_k = \sum_{i=1}^{m} s_{k,i} \cdot 2^{m-i} \tag{3.10}$$

En donde $m+1$ es la longitud de restricción del código. Como K bits son codificados en cada instante de tiempo, se dice que el código es un código K-binario. Un código convolucional se denota como (m, K, N). Entre mayor sea el valor de m, es más alta la complejidad en el proceso de decodificación, pero se garantiza una mayor capacidad de corrección de errores.

Cuando el codificador se alimenta con el bloque de información d_k, ocurre una transición de estado $s_k \rightarrow s_{k+1}$. El siguiente estado s_{k+1} depende de d_k, del estado actual s_k y de los parámetros del código $a_{i,j}, b_i \in \{0, 1\}$ para $1 \leq i \leq m$ y $1 \leq j \leq K$.

Usando los coeficientes $a_{i,j}$, cada uno de los K componentes del vector d_k es seleccionado, o no, como el término de una suma con el contenido del *flip-flop* previo (excepto en el caso del primer *flip-flop*) para dar el valor a ser almacenado en el siguiente *flip-flop*. Por lo tanto, el nuevo contenido de un *flip-flop* depende de la entrada actual y del contenido del *flip-flop* previo. El caso del primer *flip-flop* debe de considerarse de manera diferente. Si todos los coeficientes b_i son nulos, la entrada es el resultado de la suma de únicamente los componentes seleccionados de d_k. En el caso opuesto, los contenidos de los *flip-flops* seleccionados por los coeficientes no nulos b_i son sumados a la suma de los componentes seleccionados de d_k. El código generado de tal manera es recursivo. Si todos los coeficientes b_i son nulos, el código es no recursivo. El parámetro $c \in \{0, 1\}$ define si el código convolucional es sistemático ($c = 1$) o no. Los símbolos de redundancia $c_{k,i}$ son finalmente producidos sumando el contenido de los *flip-flops* seleccionados por los coeficientes g.

Como ejemplo, la figura 3.4 presenta un codificador binario recursivo sistemático ($K = 1$, $c = 1$ y $a_{1,1} = 1$). Los coeficientes del lazo de recursivad son $b_1 = 1$, $b_2 = 0$, $b_3 = 1$ y los de la redundancia son $b_1 = 1$, $b_2 = 0$, $b_3 = 1$ y los de la redundancia son $g_{1,0} = 1$, $g_{1,1} = 0$, $g_{1,2} = 1$, $g_{1,3} = 1$.

Es importante enfatizar que los códigos convolucionales están bien adaptados a transmisiones con un flujo continuo de datos. De hecho, las secuencias de datos a ser codificadas pueden tener cualquier longitud.

3.2.2. Generador de código.

Para cada secuencia codificada $i = 1, ..., N$, se puede asociar su transformada en D definida por

$$C_i(D) = \sum_{k} c_{k,i} D^k \tag{3.11}$$

donde D ($(delay)$ es el operador retardo, equivalente a la variable z^{-1} de la transformación z. Cada símbolo codificado $c_{k,i}$ puede ser expresado como una combinación lineal de K símbolos de información presentes en la entrada del codificador y Km símbolos contenidos en su memoria:

$$c_{k,i} = \sum_{l=1}^{K} \sum_{j=0}^{m} g_{l,j}^{i} d_{k-j,l} \tag{3.12}$$

donde los coeficientes $g_{l,j}^{i}$ toman los valores 0 o 1 y las operaciones suma se hacen módulo 2. La relación anterior es un producto convolucional entre la secuencia de símbolos a ser codificada y la respuesta al impulso del codificador definido por los coeficientes $g_{l,j}^{i}$. [1]

Tomando en cuenta la expresión anterior, $C_i(D)$ también puede ser expresada como

$$C_i(D) = \sum_{l=1}^{K} G_l^i(D)\, d_l(D) \tag{3.13}$$

donde $G_l^i(D)$ y $d_l(D)$ son, respectivamente, las transformadas en D de la respuesta del codificador y la secuencia a ser codificada para la entrada $l, l = 1, ..., K$:

$$G_l^i(D) = \sum_{j=0}^{m} g_{l,j}^{i} D^j \tag{3.14}$$

$$d_l(D) = \sum_{p} d_{p,l} D^p \tag{3.15}$$

Las cantidades $G_l^i(D), 1 \leq l \leq K, 1 \leq i \leq N$ se llaman los polinomios generadores del código.

Introduciendo notación matricial, se tiene:

$$\mathbf{d}(D) = (d_1(D), ..., d_K(D)) \tag{3.16}$$

$$\mathbf{C}(D) = (C_1(D), ..., C_N(D)) \tag{3.17}$$

La salida del codificador se puede expresar como una función de su entrada por la siguiente relación matricial

$$\mathbf{C}(D) = \mathbf{d}(D)\mathbf{G}(D) \tag{3.18}$$

donde $\mathbf{G}(D)$ es una matriz con K filas y N columnas y la matriz generadora del código:

$$\mathbf{G}(D) = \begin{bmatrix} G_1^1(D) & \dots & \dots & G_1^i(D) & \dots & \dots & G_1^N(D) \\ \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \vdots \\ G_l^1(D) & \dots & \dots & G_l^i(D) & \dots & \dots & G_l^N(D) \\ \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \vdots \\ G_K^1(D) & \dots & \dots & G_K^i(D) & \dots & \dots & G_K^N(D) \end{bmatrix} \tag{3.19}$$

A continuación se considerarán dos ejemplos de códigos convolucionales (tanto no recursivo como recursivo) y se determinarán sus respectivas matrices generadoras.

[1] El nombre de codificación convolucional viene del hecho de que la ecuación (3.12) tiene la forma de una convolución binaria, análoga a la integral de convolución $c(t) = \int g(\lambda) d(t-\lambda) d\lambda$

3.2.3. Representación polinomial.

Códigos convolucionales no recursivos.

Considérese primero el caso de un código binario clásico (no sistemático y no recursivo), es decir, con $c = 0$, todos los coeficientes b_i son nulos y $K = 1$. El codificador, para realizar un ejemplo de análisis se muestra en la figura 3.3. Este codificador tiene una longitud de restricción 4 y una tasa de código $R = 1/2$.

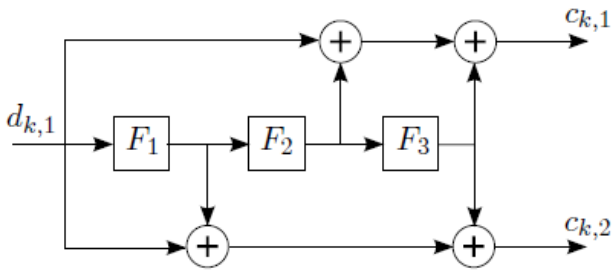

Figura 3.3: Codificador convolucional no recursivo $(3, 1, 2)$ con polinomios generadores, representados en octal, $(13, 15)$.

El codificador tiene los parámetros $(g_{1,0}, g_{1,1}, g_{1,2}, g_{1,3}) = (1, 0, 1, 1)$ y $(g_{2,0}, g_{2,1}, g_{2,2}, g_{2,3}) = (1, 1, 0, 1)$. Como no existe lazo de retroalimentación, la respuesta al impulso se puede expresar fácilmente. De hecho, corresponde a:

$$g_1^{(1)} = (g_{1,0}, g_{1,1}, g_{1,2}, g_{1,3}, 0, 0, ...) = (1, 0, 1, 1, 0, 0, ...) \tag{3.20}$$

$$g_2^{(1)} = (g_{2,0}, g_{2,1}, g_{2,2}, g_{2,3}, 0, 0, ...) = (1, 1, 0, 1, 0, 0, ...) \tag{3.21}$$

Por lo tanto, la matriz generadora polinomial está dada por:

$$\begin{aligned} \mathbf{G}^{NoRec}(D) &= \left[G_1^{(1),NoRec}(D)\ G_2^{(1),NoRec}(D)\right] \\ &= \left[g_{1,0} + g_{1,1}D + g_{1,2}D^2 + g_{1,3}D^3,\ g_{2,0} + g_{2,1}D + g_{2,2}D^2 + g_{2,3}D^3\right] \\ &= \left[1 + D^2 + D^3,\ 1 + D + D^3\right] \end{aligned} \tag{3.22}$$

Una representación binaria ($G_1^{(1),NoRec}(D) = 1011_2$ y $G_2^{(1),NoRec}(D) = 1101_2$) de los polinomios generadores especifica los valores de los coeficientes ordenados siguiendo las potencias crecientes de D. La representación de estos generadores en formato octal ($G_1^{(1),NoRec}(D) = 13_8$ y $G_2^{(1),NoRec}(D) = 15_8$) es comúnmente usada, y la notación se simplifica a $(13, 15)$ [27].

Códigos recursivos sistemáticos convolucionales.

Una clase importante de códigos convolucionales de especial interés para los turbo códigos es la clase de códigos recursivos sistemáticos convolucionales (RSC). Los códigos RSC introducen retroalimentación en la memoria de los códigos convolucionales.

Un código RSC se obtiene introduciendo un lazo de retroalimentación en la memoria (recursivo) y transmitiendo directamente la entrada del codificador como una salida (sistemático). La figura 3.4 muestra un código binario RSC construido aplicando algunas transformaciones al código no recursivo de la figura 3.3. La construcción es como sigue: una de las salidas del código no recursivo original se retroalimenta a la entrada del *flip-flop* F1, y se agrega una salida sistemática. Cabe destacar que el conjunto de palabras de código generadas por el codificador RSC es el mismo que el generado por el correspondiente codificador no sistemático. Sin embargo, la relación entre las secuencias de información y las palabras de código es diferente [23]. El codificador convolucional de la figura 3.4 es una parte del turbo código que se utiliza en el estándar 3GPP-LTE (3rd Generation Partnership Project -Long Term Evolution).

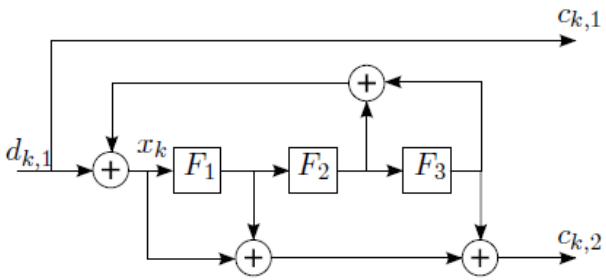

Figura 3.4: Codificador recursivo sistemático convolucional (RSC) (3,1,2) con polinomios generadores $[1, 15/13]$

Un codificador RSC tiene una respuesta al impulso infinita, y por lo tanto la definición de los polinomios generadores no es directa de inferir. Como el codificador es sistemático, el polinomio asociado a la salida sistemática $c_{k,1}$ es trivial: $G_1^{(1),Rec}(D) = 1$. Para deducir el segundo polinomio se introduce la variable x (entrada al *flip-flop* F_1). Por lo cual se tiene:

$$d_{k,1} = x_k + x_{k-2} + x_{k-3} \tag{3.23}$$

$$c_{k,2} = x_k + x_{k-1} + x_{k-3} \tag{3.24}$$

Por lo cual, es válida la siguiente ecuación:

$$d_{k,1} + d_{k-1,1} + d_{k-3,1} = c_{k,2} + c_{k-2,2} + c_{k-3,2} \tag{3.25}$$

Aplicando la transformación D (que es equivalente a la transformada z si se toma en cuenta que $D = z^{-1}$) a la ecuación anterior, se encuentra el segundo componente de la matriz generadora, el cual es una función racional:

$$G_2^{(1),Rec}(D) = \frac{C_2(D)}{D_1(D)} = \frac{1 + D + D^3}{1 + D^2 + D^3} \tag{3.26}$$

Se puede ver que esta matriz generadora corresponde a la matriz generadora no recursiva obtenida en el apartado anterior dividida por $G_1^{(1),NoRec}$(D).

Los códigos RSC se usan como códigos componentes para los turbo códigos (concatenación paralela de dos códigos RSC) debido a su respuesta al impulso infinita: si un codificador binario RSC se alimenta con un único "1" y luego solamente con entradas "0", debido a la existencia de retroalimentación, el peso de la secuencia de paridad producida será infinito. Para un codificador no recursivo solamente un máximo de bits "1"son generados, luego los "1"salen del registro de corrimiento.

Se debe notar que en los códigos convolucionales recursivos, la existencia de la retroalimentación causa que el código tenga respuestas al impulso infinitas. En general, son deseables los códigos convolucionales sistemáticos, pero desafortunadamente los que no son recursivos no pueden lograr la distancia libre mayor posible, con respecto a los códigos no sistemáticos y no recursivos de la misma tasa y longitud de restricción. Sin embargo, los códigos RSC pueden lograr la misma distancia libre que los códigos sistemáticos no recursivos para una tasa dada y longitud de restricción. El concepto de distancia libre se explicará más adelante.

3.2.4. Diagramas de representación.

La evolución de un codificador binario convolucional puede ser representada como una máquina de estados finitos, cuyas salidas son una función del estado actual y de sus entradas. El estado de esta máquina corresponde a los m bits de la memoria de los registros de corrimiento (o *flip-flops*), lo cual produce 2^m estados.

Considérese $\mathbf{S}_k = (s_{k,1}, s_{k,2}, ..., s_{k,m})^T$ el vector que representa el estado del codificador en el tiempo k, donde $s_{k,i}$ representa el valor del *flip-flop* i-ésimo en el tiempo k. Por conveniencia, el estado del codificador $\mathbf{S}_k$ también se denota por la cantidad escalar

$$s_k = \sum_{i=1}^{2^m} 2^{i-1} s_{k,j}, \tag{3.27}$$

tomando valores entre 0 y 2^{m-1}, y que luego se escribe como $\mathbf{S}_k = s_k$.

Es posible definir, a semejanza de los códigos de bloque lineales, una matriz de comprobación de paridad $\boldsymbol{H}$ e incluso un vector de síndrome. Sin embargo, resulta de mayor interés desarrollar el análisis de los códigos convolucionales en base a la consideración de los codificadores como máquinas de estado finitos, pues va a permitir de manera natural la aplicación del algoritmo de Viterbi como decodificador de máxima verosimilitud [15].

Diagrama de estados.

El diagrama que representa la evolución de la máquina de estados finitos es el diagrama de estados. Representa todos los 2^m estados del codificador, y también la transición entre estados. Las salidas dependen del estado actual y de los bits de entrada. Las transiciones en líneas sólidas corresponden a la presencia de un "1" mientras que las líneas punteadas corresponden a una entrada "0" del codificador [28]. Esta representación es particularmente

útil para determinar la función de transferencia y el espectro de distancias de un código convolucional.

A partir de esta representación es posible ver una diferencia notable entre la máquina de estados de un código recursivo y la de un código no recursivo: la existencia y el número de ciclos de una secuencia nula en la entrada (un ciclo es una sucesión de estados donde el estado inicial también es el estado final). La máquina de estados correspondiente al codificador no recursivo de la figura 3.3 se muestra en la figura 3.5, mientras que el del codificador recursivo de la figura 3.4 se muestra en la figura 3.6.

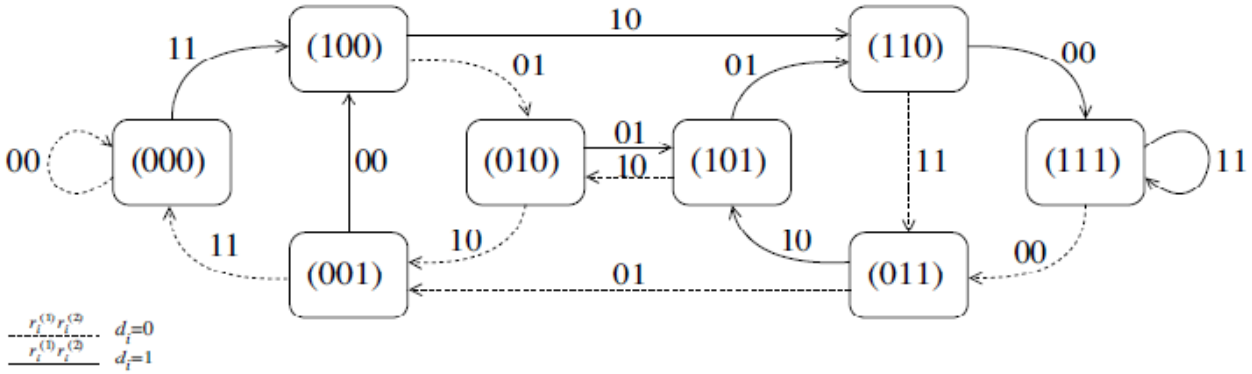

Figura 3.5: Máquina de estados para un código con una matriz generadora polinomial $\left[1 + D^2 + D^3, 1 + D + D^3\right]$.

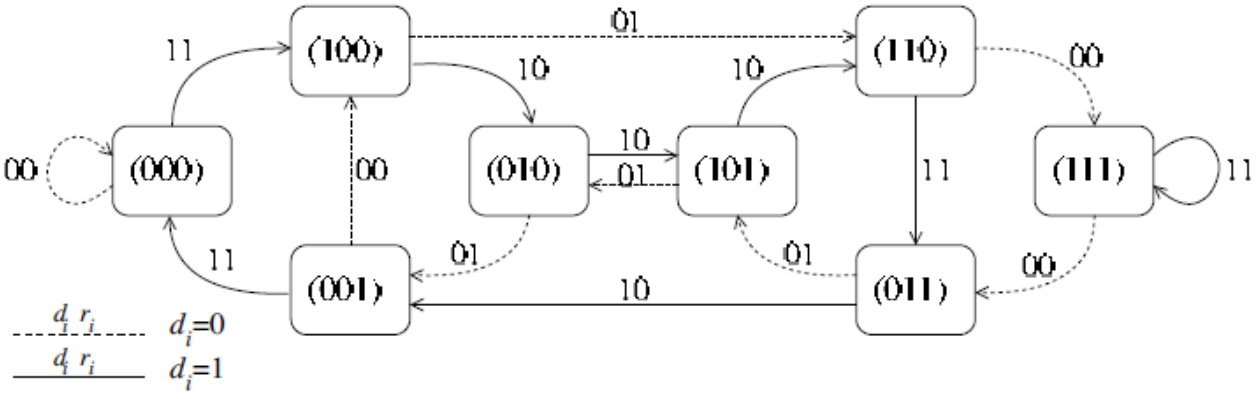

Figura 3.6: Máquina de estados para un código con matriz generadora $\left[1, \frac{(1+D^2+D^3)}{(1+D+D^3)}\right]$

Diagrama de "Trellis"

La representación más común de un código convolucional es el diagrama de "trellis". Es de gran importancia para definir las propiedades de un código y para ilustrar su proceso de decodificación, y corresponde al registro en el tiempo de los cambios de estados.

En cada instante i, el estado de un codificador convolucional puede tomar 2^m valores. Cada uno de esos posibles valores está representado por un nodo. A cada instante i se le asocia una columna de nodos de estado y de acuerdo a la entrada d_i, el codificador transita de un estado $\mathbf{s}_i$ a $\mathbf{s}_{i+1}$ mientras entrega los bits codificados. La transición entre dos estados está representada por un trazo entre los dos nodos asociados y está etiquetada con las salidas del codificador. En el caso de un código binario, la transición debida a una entrada "0" está representada por una línea punteada, mientras que la transición correspondiente a una entrada "1" es representada por una línea sólida. La sucesión de estados s_i hasta el instante t está representada por diferentes caminos entre el estado inicial y los diferentes posibles estados en el instante t.

Cuando una secuencia particular de bloques $\boldsymbol{d}$ es codificada, un camino en el diagrama de *"trellis"* está definido como una secuencia de transiciones de estado dadas por la estructura del codificador. Se asumirá que el estado del codificador en el tiempo $k = 0$ es $s_k = 0$. Por lo tanto, para el tiempo $k < 3$ hay algunos estados que no están permitidos. Para $k \geq 3$, el codificador puede tener cualquier estado, y por lo tanto cualquier transición de estados en el diagrama "trellis" puede ocurrir. En la figura 3.7, se muestran las posibles trayectorias del codificador RSC de la figura 3.4.

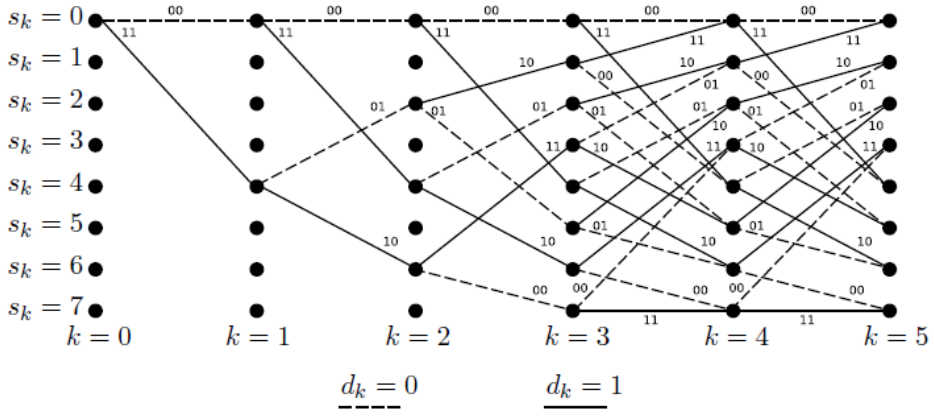

Figura 3.7: Diagrama de "trellis" para el código RSC de la figura 2.4

Debido a que el código es binario, en este diagrama de *"trellis"* hay siempre dos trazos que llegan a cada estado, uno correspondiente a un símbolo de entrada $d_k = 0$ y otro para un símbolo $d_k = 1$. También existe un patrón mariposa; cada sección del *"trellis"* del instante k al $k+1$ consiste de cuatro estructuras en mariposa. Por ejemplo, cada una de las secciones de la figura 3.7 está construida de 4 mariposas (las transiciones de los estados 0 y 1 hacia los estados 0 y 4 conforman una).

Un parámetro importante de un código convolucional es la distancia libre, d_f. La distancia libre es la mínima distancia de Hamming entre dos caminos que divergen en un instante dado y convergen en el instante de tiempo más breve. Debido a la linealidad, para su cálculo, es suficiente el considerar trayectorias en referencia con el camino nulo, es decir, encontrar el peso mínimo de cualquier camino divergiendo y convergiendo al camino nulo [16]. Para el código cuyo "trellis" se ilustra en la figura 3.7, la distancia libre es $d_f = 6$. Esto corresponde a la secuencia de entrada 1011 y secuencia de salida $11, 01, 10, 11$.

3.2.5. Códigos convolucionales perforados.

Para un código convolucional de tasa K/N, existen 2^K transiciones que comienzan en cada nodo. Para un código de tasa alta, es decir, aquél para el cual el coeficiente K es grande, la complejidad del diagrama *trellis* puede crecer considerablemente, y en consecuencia, el código se vuelve complejo de decodificar.

Es posible construir códigos convolucionales de mayor tasa a partir de códigos con tasa de salida $\frac{1}{2}$ ($K = 1$). Para obtener tales códigos se usa la técnica de perforación.

La perforación consiste en no transmitir todos los símbolos entregados por el codificador con tasa $\frac{1}{2}$; ciertos símbolos son suprimidos o perforados. Considérese el ejemplo de la figura 3.8. Esta figura representa la salida de un codificador con tasa $\frac{1}{2}$ conformada por una sucesión de bloques de dos símbolos codificados.

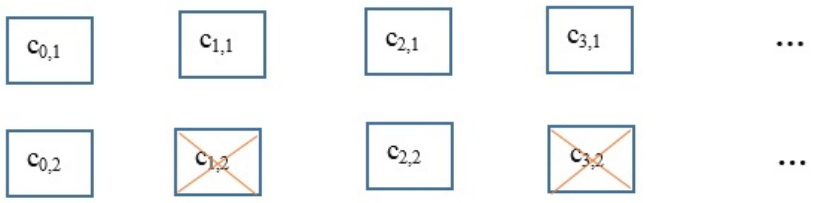

Figura 3.8: Patrón de perforado que permite construir un código convolucional con tasa $R = 2/3$ a partir del código madre (3,2,1) de la figura 3.4

Si se descarta un símbolo codificado en cuatro, por ejemplo, los símbolos cruzados en la figura 3.8, luego dos símbolos de información del codificador están asociados a tres símbolos codificados, es decir una tasa de 2/3. La regla de perforación está definida a partir de una máscara M que, para nuestro ejemplo, puede ser representada a través de una matriz con dos filas y dos columnas:

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{3.28}$$

El elemento de la matriz M igual a 0 indica el símbolo codificado a ser perforado. En este caso es uno sobre cada dos símbolos entregados a la salida número 2 del decodificador. Otra ventaja de esta solución es que el mismo decodificador se puede usar para decodificar el conjunto de códigos perforados derivados de un único código madre C [29]. A pesar de que los códigos perforados son sub-óptimos comparados con los códigos diseñados para una tasa

en particular, son populares debido al hecho de que un único par codificador/decodificador puede ser usado para implementar diferentes tasas. En el caso de códigos sistemáticos, es generalmente la redundancia la que es perforada.

La elección de la máscara de perforación obviamente influencia el desempeño del código. Es posible favorecer una parte de la trama, que transporte datos sensitivos, perforando ligeramente a comparación de otra parte que sea mayormente perforada.

La perforación es por lo tanto una técnica flexible, y sencilla de implementar. Conforme el codificador y el decodificador permanezcan idénticos en el tipo de perforación que se aplica, es posible modificar la tasa de codificación en cualquier momento. Algunas aplicaciones emplean esta flexibilidad para adaptar la tasa de acuerdo a la importancia de los datos transmitidos.

3.2.6. Códigos catastróficos.

Es importante enfatizar que algunos códigos convolucionales exhiben lo que se conoce como propagación catastrófica de errores. Esto ocurre cuando un número finito de errores de canal causa un número infinito de errores de decodificación, aún si los símbolos subsecuentes son correctos. Los codificadores que presentan este comportamiento mostrarán en su diagrama de estados un estado donde una entrada diferente de cero, causa una transición de regreso al estado mismo, produciendo una salida cero [12].

Otra manera de identificar a un código catastrófico es mediante sus polinomios generadores; si en un código de tasa $1/N$ existe un factor común distinto a la unidad entre sus polinomios generadores, calculado en aritmética módulo 2, se tiene que el código es catastrófico. Por ejemplo, si en un código los polinomios generadores son $g_1(D) = 1+D$ y $g_2(D) = 1+D^2$; entre estos polinomios existe el factor común $1 + D$, ya que $1 + D^2 = (1 + D)(1 + D)$ en la aritmética sobre $GF(2)$, con lo cual este código es catastrófico.

3.2.7. Terminación de códigos convolucionales.

En este escrito se considera el caso de códigos convolucionales terminados (o finitos) de tamaño L. Un código convolucional finito de tamaño n y tasa $R = K/N$ es por lo tanto un código de bloque lineal de dimensión $n \cdot K$ y longitud $n \cdot N$, denotado como $C(n \cdot N, n \cdot K)$.

Para un código convolucional finito, el estado inicial de los *flip-flops* generalmente se establece al estado nulo por el codificador. Esta información es usada por el decodificador para mejorar el desempeño de la decodificación. Sin embargo, como el estado final del codificador está indeterminado y depende de la secuencia de entrada, el decodificador no tiene información disponible con respecto al estado final del registro de corrimiento. Hay múltiples alternativas para terminar el código convolucional que han sido propuestas en la literatura [23].

Sin terminación.

El estado final del codificador, el cual depende de la secuencia de entrada, no es conocido en el lado del decodificador. Esta falta de información conlleva a una degradación en desempeño para los últimos datos decodificados y por lo tanto a una protección de errores menos efectiva. En particular, los últimos símbolos codificados están asociados con palabras de

código con bajos pesos de Hamming, resultando en una reducción de la ganancia asintótica de codificación. Esta degradación es más notable cuando aumenta el tamaño del bloque n.

Regreso del estado del codificador (al final de la codificación) a un estado final conocido.

Generalmente, el estado final es forzado al estado "nulo" codificando $n \cdot K$ símbolos y alimentando al registro de corrimiento con la secuencia apropiada de símbolos, llamados símbolos de cola. El bloque codificado resultante es de longitud $(n + m) \cdot K$. Por un lado, el estado inicial y final del registro de corrimiento son conocidos por el decodificador. Por el otro lado, la eficiencia espectral de la transmisión se reduce y la pérdida de tasa $m/(m + n)$ debida a estos símbolos de cola es incrementada conforme la longitud del bloque n es más corta.

Códigos circulares.

Con los códigos convolucionales circulares, el codificador no está inicializado al estado nulo pero a un estado inicial dado, y el codificador recupera este estado inicial al final del proceso de codificación. Este estado inicial llamado "estado de circulación" depende de los polinomios generadores del código y también de la secuencia de entrada del codificador. La técnica se propuso inicialmente para códigos convolucionales no recursivos y luego se adaptó a códigos recursivos. En el lado del codificador, el estado de circulación es no conocido, pero el "trellis" puede ser visto como un círculo, sin ninguna discontinuidad en las transiciones entre estados. Esta propiedad puede ser usada por el decodificador obteniéndose así una protección igual de errores para todos los símbolos del bloque. El diagrama de un código CRSC (Circular Recursivo Sistemático Convolucional) se muestra en la figura 3.9.

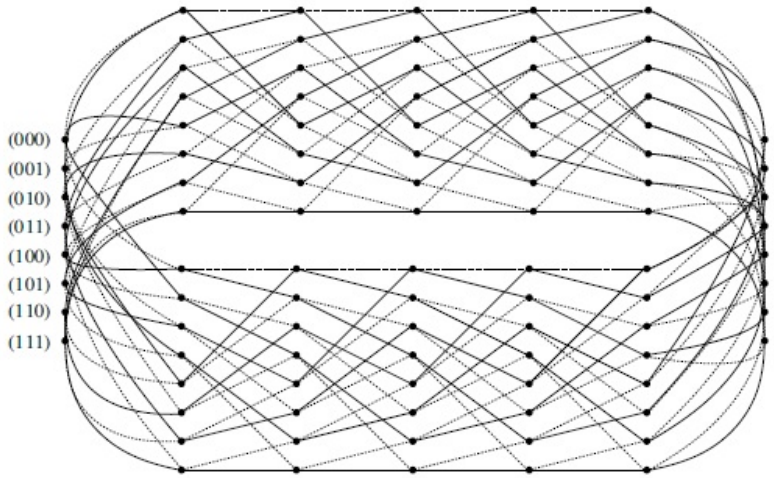

Figura 3.9: "Trellis" de un código CRSC de 8 estados.

Más detalles sobre como codificar en el caso de códigos circulares se pueden consultar en [27].

3.2.8. Concatenación de códigos convolucionales.

En la primera arquitectura propuesta por Forney [39], se concatenó un código interno con un código externo como se muestra en la figura 3.10. Esto se llama concatenación serial de códigos convolucionales (SCCC) donde la salida de un código externo se conecta a la entrada de un código interno. En seguida, se observó que al agregar una función de entrelazado entre los dos códigos, se incrementaba significativamente la robustez de los códigos concatenados. Este entrelazador juega un rol significativo en la construcción del código, ya que realiza una permutación pseudo-aleatoria de la secuencia de entrada tardía, lo cual introduce aleatoreidad en el esquema de codificación. Esto es lo que se llama hoy en día códigos convolucionales concatenados serialmente que se representa como en la figura 3.11.

Figura 3.10: Concatenación serial de códigos convolucionales (SCCC).

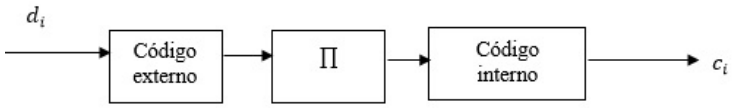

Figura 3.11: Concatenación serial de códigos convolucionales con entrelazador (SCCC).

Con la aparición de los turbo códigos [7], se introdujo una nueva estructura: la concatenación paralela de códigos convolucionales (PCCC) de dos codificadores RSC (RSC1, RSC2) presentada en la figura 3.12. Esta estructura está asociada a codificadores sistemáticos donde el primer codificador recibe los datos d_i en orden natural y al mismo tiempo el segundo codificador recibe los datos entrelazados. La secuencia de entrada d_i es codificada dos veces, una por cada codificador. Un entrelazador Π permuta la secuencia de información antes de la codificación mediante RSC2. En otras palabras, los dos codificadores componentes RSC codifican la misma secuencia de información pero en un orden diferente. La aleatoriedad para códigos correctores de errores es un atributo clave que permite acercarse a la capacidad del canal. Dado que el código es sistemático, la salida está compuesta por los datos de información, y las paridades asociadas en los dominios natural y entrelazado. De esta manera, en un instante de tiempo se transmite la paridad de dos símbolos distintos.

El uso de un entrelazador en la construcción de códigos concatenados en serie, o en paralelo, resulta en palabras de código que sean largas en longitud de bloque- y relativamente

dispersas. La decodificación de este tipo de códigos se realiza generalmente de forma iterativa, usando algoritmos de tipo MAP, SISO. La combinación de códigos con entrelazamiento y decodificación MAP iterativa resulta en un desempeño muy cercano al límite teórico de Shannon para tasas de error moderadas, tal como 10^{-4} a 10^{-5} (región baja de SNR). El esquema de de decodificación iterativa es justamente el que le da el nombre a los "turbo códigos" ya que se asemeja al comportamiento de una máquina *"turbo"* comprimida, en donde para hacer más eficiente el ciclo de combustión de la misma, los gases de escape de la máquina (energía residual) son retroalimentados a la entrada. En este esquema iterativo, la ganancia en desempeño obtenida en cada iteración disminuye gradualmente en relación con la iteración previa [20]. Más detalles de este tipo de concatenación se presentan en el siguiente capítulo.

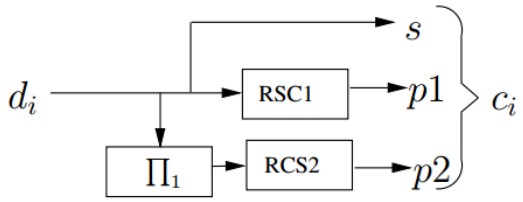

Figura 3.12: Concatenación paralela de códigos convolucionales con entrelazador (PCCC).

3.3. Decodificación de códigos convolucionales.

Existen varios algoritmos de decodificación para un código convolucional, entre los más conocidos se encuentra el algoritmo de Viterbi y el algoritmo MAP (Maximum a *posteriori*).El algoritmo de Viterbi minimiza la probabilidad de error en la secuencia de bloques decodificados, mientras que el algoritmo MAP trabaja en la probabilidad de error de cada bloque considerado. Hay que recordar que un bloque entrante se constituye de K símbolos binarios, $K = 1$ para el caso binario (nuestro caso). Consideremos que el decodificador recibe una secuencia ruidosa y demodulada, que tiene una longitud L, el algoritmo de Viterbi estima la verosimilitud a partir de la secuencia recibida proporcionando la secuencia más probable. Ese enfoque puede ser realizado de manera simple mediante una complejidad razonable (sección 3.3.1). Por otra parte, el algoritmo MAP busca la opción óptima de decodificación calculando la probabilidad de error en cada bloque considerado. Sin embargo, la implementación del algoritmo MAP es difícil debido a su complejidad computacional asociada (sección 3.3.2).

3.3.1. Algoritmo de Viterbi.

Sea
$\mathbf{d} = (d_0...d_k...d_{L-1})$ la secuencia de información;
$\mathbf{c} = (c_0...c_k...c_{L-1})$ la secuencia codificada;
$\mathbf{y} = (y_0...y_k...y_{L-1})$ la secuencia recibida por el decodificador (observación);
El algoritmo de Viterbi[30] consiste en buscar la secuencia $\widehat{\mathbf{d}}$ tal que

$$\widehat{\mathbf{d}} = \underset{\mathbf{d}}{Argmax} P(\mathbf{d}|\mathbf{y}) = \underset{\mathbf{d}}{Argmax} \frac{P(\mathbf{d},\mathbf{y})}{P(\mathbf{y})} \forall \mathbf{d} \tag{3.29}$$

donde Arg denota el argumento $\mathbf{d}$ de la probabilidad condicional $P(\mathbf{d}|\mathbf{y})$. Sabiendo que la probabilidad $P(\mathbf{y})$ es independiente de la información $\mathbf{d}$, el algoritmo de Viterbi busca el valor $\widehat{\mathbf{d}}$ que maximice la probabilidad $P(\mathbf{d},\mathbf{y})$. Se introduce una secuencia de estados $S = (S_0...S_k...S_{L-1})$ tal que $S_k = (s_k, s_{k-1}, ..., s_{k-v+2})$. El conocimiento de la secuencia S permite encontrar la información d. En consecuencia, la secuencia S es una imagen de $\mathbf{d}$. Por otra parte, la transición $S_{k-1} \rightarrow S_k$ depende solamente del valor de entrada d_k. Entonces el algoritmo de Viterbi busca $\widetilde{\mathbf{S}}$ tal que

$$\widetilde{\mathbf{S}} = \underset{\mathbf{d}}{Argmax} \left[P(\mathbf{S},\mathbf{y})/P(\mathbf{y})\right] \ \forall \mathbf{S} \tag{3.30}$$

Por lo tanto, es suficiente maximizar el término $P(\mathbf{S},\mathbf{y})$ ya que $P(\mathbf{y})$ es independiente de la entrada del codificador $\mathbf{d}$.

$$P(\mathbf{S},\mathbf{y}) = P(S_0) \prod_{k=1}^{L} P((y_k, S_k)|S_{k-1}) \tag{3.31}$$

donde $P(S_0)$ representa la distribución del estado inicial. Si el estado del codificador está inicializado en el estado 0, la distribución $P(S_0)$ es igual a 1 para $S_0 = (0...0...0)$, y 0 para el resto.

Introduciendo la métrica de rama γ_k definida por la siguiente ecuación:

$$\gamma_k(s', s) = -logP((y_k, S_k = s')|S_{k-1} = s), \tag{3.32}$$

y utilizando las ecuaciones (2.30) y (2.31) en el dominio logarítmico, se obtiene:

$$\widetilde{\mathbf{S}} = \underset{\mathbf{S}}{Argmin} \left(-logP(S_0) + \sum_{k=1}^{L} \gamma_k(S_{k-1}, S_k) \right). \tag{3.33}$$

El término $\gamma_k(S_{k-1}, S_k)$ se llama la métrica de rama en el instante k. El cual corresponde a la distancia de Hamming entre la observación y_k y la palabra del *trellis* c_k en un canal binario simétrico BSC, es decir $\gamma_k(S_{k-1}, S_k) = v(y_k, c_k)$. Para un canal gaussiano, la métrica de rama es igual al cuadrado de la distancia euclideana entre la observación y_k y la palabra c_k, $\gamma_k(S_{k-1}, S_k) = \|y_k - c_k\|^2$. Si ninguna transición es posible entre los estados s' y s en un cierto instante, la métrica $\gamma_k(s', s)$ es igual a $\gamma_k(s', s) = \infty$. Esta métrica se puede simplificar, de acuerdo con Glavieux [26] al siguiente producto escalar: $\gamma_k(s, s') = \langle y_k, c'(s, s')\rangle$. Este es el producto escalar de la observación y_k y el bloque modulado y codificado $c'(s, s')$, dando esta expresión que es parecida a un proceso de correlación. Considerando la tasa de 1/2 en el decodificador, un par de recepciones llegan en cada instante k. La métrica de rama es calculada tomando el producto de la información relacionada a los datos (sistemáticos) con cada una de las señales prototipo (llámese u_k), y similarmente el producto de la información asociada a la paridad (redundancia) con cada una de las señales prototipo (llámese por ejemplo v_k). Las métricas de rama también se utilizan en el algoritmo del siguiente apartado, que es la base de la turbo decodificación.

Luego, considerando que para una palabra codificada de longitud N se tiene un total de 2^N métricas de rama. La expresión (3.33) se construye con la búsqueda del camino más corto, llamado camino sobreviviente, si la métrica $\gamma_k(s', s)$ es considerada como la distancia asociada entre los estados $S_{k-1} = s'$ y $S_k = s$. Ese camino se compone de transiciones de estado $S_{k-1} \to S_k$ asociadas al símbolo d_k.

La métrica de nodo $M_{s,k}$ para el estado s en el instante k se obtiene por medio de:

$$M_{s,k} = \underset{s' \to s}{min}(M_{s',k-1} + \gamma_k(s', s)) \tag{3.34}$$

Esta métrica significa la máxima verosimilitud que parte del nodo $S_{k-1} = s'$ hasta el nodo $S_k = s$.

A nivel de implementación, el decodificador de Viterbi calcula primero las métricas de rama γ_k a partir de la palabra recibida. Para una decodificación binaria de tasa $1/2$ y $N = 2$, hay 4 métricas de rama que miden directamente la verosimilitud de la observación r_k respecto a la palabra codificada c_k. Posteriormente, un modulo ACS (Add-Compare-Select por sus siglas en inglés) selecciona la mínima métrica de nodo $M_{s,k}$ por cada nodo $S_k = s$ sumando la métrica de rama apropiada $\gamma_k(s', s)$ a la métrica de nodo precedente $M_{s',k-1}$. Los resultados de selección, 2^{v-1} caminos en total, son enseguida almacenados, lo cual permite recuperar la secuencia decodificada en una trayectoria en sentido inverso del *trellis*.

En su versión base, el decodificador de Viterbi arroja el resultado una vez que se tenga la recepción completa de la secuencia considerada. De hecho, los caminos sobrevivientes convergen a un nodo del instante $k+n$ al instante k, es decir, todos los caminos sobrevivientes a partir del instante k parten del mismo nodo. Por lo tanto, el decodificador puede emitir los resultados del instante 0 al instante k ya que la decodificación siguiente no está influenciada por los resultados precedentes. En general, el valor de n es $n = 5v$ o $6v$ para R = $1/2$ (v es la longitud de restricción). Cada nodo recibe 2^K ramas. En cada etapa, hay 2^N cálculos de métrica de rama y 2^{K+v-1} sumas en los módulos ACS. La complejidad de cálculo crece de manera exponencial. Es por ello que la implementación del algoritmo de Viterbi esta limitada a valores pequeños de K y v. Típicamente, $v \leq 8$. Más allá de ese límite, el algoritmo de decodificación secuencial como el de Fano es adoptado en razón de su complejidad reducida. Aunque el algoritmo de Viterbi corresponde con una medida de la probabilidad de error en una secuencia decodificada, conduce también a una buena aproximación de un mínimo de la probabilidad de error en los bloques de información cuando la relación señal a ruido excede ciertos decibeles.

3.3.2. Algoritmo BCJR.

El algoritmo BCJR (Bahl, Cocke, Jelinek, Raviv) con el criterio de máximo a posteriori, es un algoritmo óptimo de decodificación símbolo-a-símbolo para códigos convolucionales, y además es la componente central en muchos esquemas de detección y decodificación iterativa. En este algoritmo el decodificador usa el algoritmo MAP para decodificar cada símbolo de entrada al decodificador, en vez de buscar cuál es la secuencia más probable. Es óptimo para estimar los estados, o las salidas de un proceso de Markov observado en ruido blanco.

Para este criterio de decodificación, el decodificador busca el símbolo de información $\widehat{d_k}$ tal que

$$p(\widehat{d_k}|\mathbf{y}) \geq p(d_k|\mathbf{y}) \ \forall(d_k) \tag{3.35}$$

donde $p(\widehat{d_k}|\mathbf{y})$ es la probabilidad de la información $\widehat{d_k}$, condicionada a la observación $\mathbf{y}$.

Los siguientes términos deben ser definidos antes de detallar las características del algoritmo MAP:

- $\alpha_k(s)$ representa la probabilidad conjunta de las observaciones $y_1, ..., y_k$ y del estado del codificador $S_k = s$:

$$\alpha_k(s) = p(y_1, y_2, ..., y_k, S_k = s) \tag{3.36}$$

  Se le conoce como métricas de estado hacia adelante.

- $\beta_k(s)$ representa la probabilidad conjunta de las observaciones $y_{k+1}, ..., y_M$ condicionada al estado del codificador $S_k = s$:

$$\beta_k(s) = p(y_{k+1}, ..., y_M|S_k = s) \tag{3.37}$$

  La cantidad $\beta_k(s)$ se llama métrica de estado hacia atrás.

El algoritmo BCJR, junto con decisiones duras, produce valores suaves asociados a cada bit decodificado. Al tratarse de un algoritmo MAP, una vez que los L símbolos de una trama son decodificados, el camino que es reconstruido no necesariamente corresponde a una trayectoria conectada en el diagrama de *trellis*. Se calculan 2^m valores de probabilidad *a-posteriori* APP correspondientes a cada símbolo de información posible, permitiendo el conocimiento completo de la secuencia $\mathbf{y}$, recibida por el decodificador. La decisión dura corresponde al valor j que maximiza la probabilidad *a-posteriori*. Estas probabilidades se pueden expresar en función de las verosimilitudes conjuntas:

$$p(d_k = j, \mathbf{y}) = \frac{p(d_k = j, \mathbf{y})}{\sum_{k'=0}^{2^m-1} p(d_k = k', \mathbf{y})} \tag{3.38}$$

El denominador de la ecuación 3.38 es un factor de normalización, ya que afecta de la misma manera a todas las posibles probabilidades *a-posteriori*. Así, puede ser removido de esta ecuación sin afectar el resultado. Por lo tanto, en la práctica, solamente se calcula la probabilidad conjunta $p(d_k = j, \mathbf{y})$. El cálculo de las probabilidades conjuntas puede ser descompuesto entre observaciones pasadas y futuras debido a la estructura *trellis* del código (ver figura 3.13). Esta descomposición usa las métricas de recursión hacia adelante $\alpha_k(s)$ (la probabilidad de un estado del *trellis* en el instante k calculado a partir de valores pasados), las métricas de recursión hacia atrás $\beta_k(s)$ (la probabilidad de un estado del *trellis* en el instante k calculado a partir de valores futuros almacenados en memoria), y una métrica $\gamma(s', s)$ (la probabilidad de una transición entre dos estados del *trellis*). Usando estas métricas, se tiene:

$$\begin{aligned} p(d_k = j, \mathbf{y}) &= \sum_{(s' \to s)|d_k = j} p(s_k = s', s_{k+1} = s, \mathbf{y}) \\ &= \sum_{(s' \to s)|d_k = j} \alpha_k(s') \gamma_k(s', s) \beta_{k+1}(s) \\ &= \sum_{(s' \to s)|d_k = j} p(s_k = s', y_{j<k}) p(s_{k+1} = s, y_k | s_k = s') p(y_{j>k} | s_{k+1} = s) \end{aligned} \tag{3.39}$$

Con esta ecuación las APPs en 3.38 pueden ser expresadas como:

$$p(d_k = j, \mathbf{y}) = \frac{\sum_{(s' \to s)|d_k = j} \alpha_k(s') \gamma_k(s', s) \beta_{k+1}(s)}{\sum_{(s' \to s)} \alpha_k(s') \gamma_k(s', s) \beta_{k+1}(s)} \tag{3.40}$$

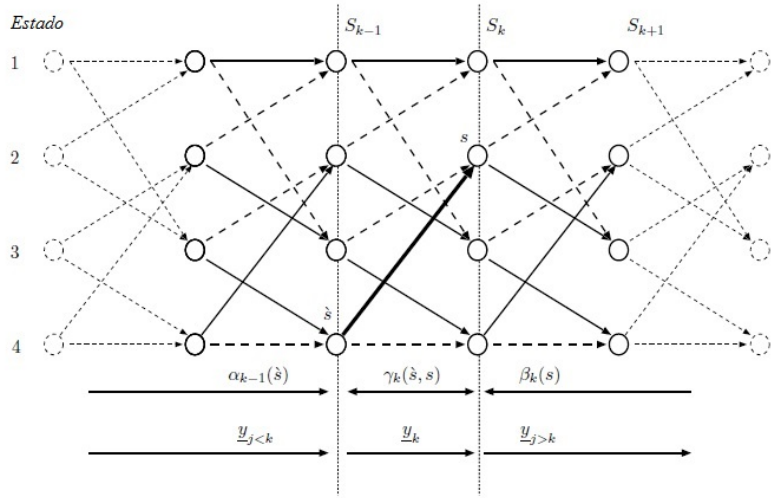

Figura 3.13: *Trellis* del decodificador MAP para un código RSC con tres elementos de memoria.

El algoritmo BCJR calcula las métricas de estado hacia adelante $\alpha_k(s)$ y hacia atrás $\beta_k(s)$ iterativamente:

$$\alpha_{k+1}(\cdot) \leftarrow \alpha_k(\cdot), y_{k+1} \tag{3.41}$$

$$\beta_k(\cdot) \leftarrow \beta_{k+1}(\cdot), y_{k+1} \tag{3.42}$$

En las ecuaciones anteriores, el "·" en $\alpha_{k+1}(\cdot)$ significa que consideramos las cantidades $\alpha_k(s)$ para los $2^{(v-1)K}$ posibles valores de s. Similarmente, para $\beta_k(\cdot)$, donde s representa un estado del codificador. A partir de tales ecuaciones, es posible ver que si se inicializa $\alpha_0(\cdot)$, se puede calcular $\alpha_1(\cdot)$ usando y_1 y $\alpha_0(\cdot)$, luego calcular $\alpha_2(\cdot)$ usando y_2 y $\alpha_1(\cdot)$, y así hasta obtener $\alpha_M(\cdot)$. Similarmente, se inicializa $\beta_M(\cdot)$, luego se calcula $\beta_{M-1}(\cdot)$ usando y_M y $\beta_M(\cdot)$; luego se calcula $\beta_{M-2}(\cdot)$ usando y_{M-1} y $\beta_{M-1}(\cdot)$, y así hasta obtener $\beta_0(\cdot)$.

Las métricas de recursión hacia adelante se calculan de la siguiente forma:

$$\alpha_{k+1}(s) = \sum_{s' \to s} \alpha_k(s')\gamma_k(s', s) \text{ para } k = 0...N-1 \tag{3.43}$$

$$\beta_k(s) = \sum_{s' \to s} \beta_{k+1}(s')\gamma_k(s', s) \text{ para } k = 0...N-1 \tag{3.44}$$

El valor de inicialización de estas métricas puede ser definido conociendo el estado inicial y final del *trellis*; si, por ejemplo, el codificador empieza en el estado s_0 luego $\alpha_0(s_0)$ tiene el valor de probabilidad más alta "1" mientras que las demás $\alpha_0(s)$ serán inicializadas en "0". Si el estado inicial es desconocido, luego todos los estados deben ser inicializados al mismo valor de probabilidad, es decir, $\alpha_0(s) = \frac{1}{2^p} \forall s$, donde p es el número de estados. Nótese que el estado inicial y final son establecidos por el método de terminación seleccionado, mismos que se presentaron en la sección 3.2.7. La figura 3.14 ejemplifica el cálculo de estos valores.

En la práctica, para evitar problemas de precisión o de *overflow* en la representación numérica de las probabilidades de estados hacia adelante y hacia atrás, tienen que ser regularmente normalizadas durante el cálculo recursivo.

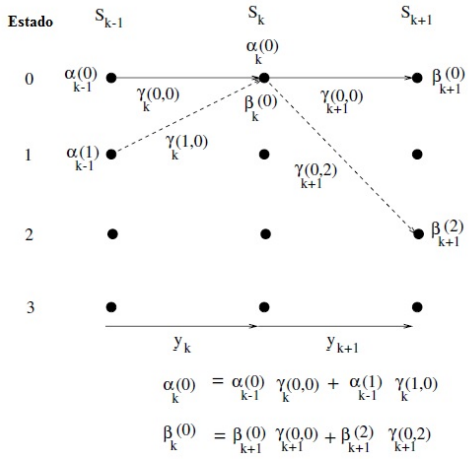

Figura 3.14: Cálculo recursivo de $\alpha_k(0)$ y $\beta_k(0)$

Para propósitos de notación, es conveniente expresar el algoritmo BCJR en una formulación matricial (esto permite verificar el algoritmo en un programa como Matlab). Sean

$$\mathbf{A}_k = \begin{bmatrix} \alpha_k(0) \\ \alpha_k(1) \\ \vdots \\ \alpha_k(N-1) \end{bmatrix} \tag{3.45}$$

y

$$\mathbf{B}_k = \begin{bmatrix} \beta_k(0) \\ \beta_k(1) \\ \vdots \\ \beta_k(N-1) \end{bmatrix} \tag{3.46}$$

vectores de las probabilidades hacia adelante y hacia atrás. Sea $\mathbf{G}_k$ la matriz de probabilidad con elementos $g_{k,i,j}$ definida por

$$g_{k,i,j} = \gamma_k(i,j) \tag{3.47}$$

La ecuación de la métrica hacia adelante se expresa como $\mathbf{A}_{k+1} = \mathbf{G}_k\mathbf{A}_k$. Complementariamente, las actualizaciones hacia atrás se pueden expresar como $\mathbf{B}_k = \mathbf{G}_k\mathbf{B}_{k+1}$.

Sea $\gamma_k(s',s) = p(s_{k+1} = s, y_k | s_k = s')$ la métrica de rama para la transición $(s_k = s') \rightarrow (s_{k+1} = s)$, igual a la probabilidad de que la transición ocurra. Si no ocurre esta transición entre s' y s en el diagrama *trellis*, $\gamma_k(s',s) = 0$. Usando dos veces la relación de Bayes, se tiene:

$$\gamma_k(s',s) = p(s_{k+1} = s, y_k | s_k = s') = p(y_k | s_k = s', s_{k+1} = s)p(s_{k+1} | s_k = s') \tag{3.48}$$

El primer término en esta ecuación es la probabilidad condicional del canal (ecuación 2.2). El segundo término corresponde a la transición $s' \rightarrow s$ dado que el codificador está en el estado s'.Por lo tanto, este término es igual a la probabilidad de que el símbolo $d_k = j$ fue transmitido, con j el símbolo que produce la transición de s' a s. Por lo tanto, la ecuación (3.48) se convierte en:

$$\gamma_k(s',s) = p(y_k|u_k)p(d_k = j) = p(d_k = j)\prod_{i=1}^{n}\left(\frac{1}{\sqrt{2\pi}\sigma_n} exp\left(-\frac{(y_{ki} - u_{ki})^2}{2\sigma_n^2}\right)\right) \tag{3.49}$$

Desarrollando esta expresión, y empleando la definición de confiabilidad del canal se tiene que:

$$\gamma_k(s,s') = B_k \cdot p(d_k = j) \cdot exp\left(\frac{1}{2}L_c\sum_{i=1}^{n} y_{ki} \cdot u_{ki}\right) \tag{3.50}$$

Donde L_c es la llamada confiabilidad del canal, que se obtiene como $L_c = \frac{2}{\sigma_n^2}$. En el resultado anterior, B_k es una constante.

Finalmente, se calcula la razón logarítmica de verosimilitud. En este trabajo, tomamos en cuenta a R_1 al conjunto de transiciones de s' a s que se deben a bits de mensaje $u_k = +1$, mientras que R_0 es el conjunto de ramas originadas por bits de mensaje $u_k = -1$ [32].

$$L(d_k = j|\mathbf{y}) = ln\frac{\sum_{R_1} \alpha_k(s')\gamma_k(s', s)\beta_{k+1}(s)}{\sum_{R_0} \alpha_k(s')\gamma_k(s', s)\beta_{k+1}(s)} \tag{3.51}$$

Desafortunadamente, la decodificación MAP requiere un número grande de operaciones, involucrando multiplicaciones y operaciones exponenciales, las cuales no se pueden transponer en una implementación directa en *hardware*. Una forma de simplificar el proceso de decodificación para propósitos de implementación es reescribir el algoritmo en el dominio logarítmico. Así, el decodificador provee estimados suaves proporcionales al logaritmo de las APPs (probabilidades *a posteriori*). Las multiplicaciones se vuelven sumas y las sumas detecciones de máximos con un término de corrección adicional. Para esto, el operador max^* se define de la siguiente manera:

$$max^*(x, y) = ln(exp(x) + exp(y)) = max(x, y) + ln(1 - exp(-|x - y|)) \tag{3.52}$$

El algoritmo resultante se conoce como algoritmo Log-MAP, el cual puede simplificarse aún más, dando el algoritmo Max-Log-MAP, si se invoca a la aproximación Max-Log-MAP que reemplaza la operación max^* por una operación clásica max, reduciendo fuertemente la complejidad:

$$log(exp(a) + exp(b)) \approx max(a, b) \tag{3.53}$$

Más detalles sobre estos algoritmos, y aspectos relacionados a su implementación en hardware, se presentarán en los siguientes capítulos.

4
Turbo Códigos

El concepto de turbo codificación fue propuesto en 1993 en una contribución seminal por Berrou et al. [7], quienes reportaron excelentes resultados de ganancia de codificación, aproximándose a las predicciones teóricas de Shannon [7]. La secuencia de información es codificada dos veces, con un entrelazador entre los dos codificadores, el cual sirve para que las dos secuencias de datos sean lo más independientes estadísticamente, una de la otra. Se usan codificadores RSC de tasa 1/2, produciendo una salida sistemática la cual es equivalente a la secuencia original de información, así como una secuencia de información de paridad por cada codificador RSC. Para adaptar la tasa de codificación a una aplicación dada, las dos secuencias de paridad pueden ser perforadas antes de ser transmitidas. La perforación de la información de paridad permite que un amplio rango de tasas de codificación sean implementadas sin degradar la probabilidad de error del código global, y a menudo solo la mitad de la información de paridad de cada codificador es enviada. Junto con la secuencia sistemática de datos originales, esto resulta en una tasa de codificación de 1/2.

El decodificador está compuesto a su vez por dos decodificadores SISO (Soft Input-Soft Output) convolucionales. Para el intercambio de información entre los dos decodificadores, se deben usar algoritmos especiales de decodificación que acepten entradas suaves y den salidas suaves para la secuencia decodificada. Estas entradas y salidas suaves proveen no solo un indicativo sobre si un bit en particular fue 0, o un 1, sino que también una razón de verosimilitud, la cual da la probabilidad de que el bit ha sido decodificado correctamente. Los algoritmos de decodificación pueden ser el SOVA (Viterbi de salida suave), MAP o alguna de sus variantes.

El turbo decodificador opera iterativamente. En la primera iteración, el primer decodificador SISO provee una salida suave la cual da una estimación de la secuencia original de datos basándose sólo en las entradas suaves del canal. También provee una **salida extrínseca**. La salida extrínseca para un bit dado está basada no sólo en la entrada del canal para ese bit, sino también en la información de los bits vecinos y en las restricciones impues-

tas por el código utilizado. La salida extrínseca del primer decodificador es utilizada por el segundo decodificador RSC como **información *a priori***, y esta información junto con las entradas del canal son usadas por el segundo decodificador RSC para generar su salida suave y la información extrínseca. En la segunda iteración la información extrínseca del segundo decodificador en la primera iteración es usada como la información a priori del primer decodificador, y usando esta información a priori el decodificador puede decodificar más bits correctamente con respecto a los de la primera iteración. El ciclo continúa, y en cada iteración ambos decodificadores RSC producen una salida suave e información extrínseca, basada en las entradas del canal y la información a priori obtenida de la información extrínseca provista por el decodificador previo. Después de cada iteración se reduce la tasa de errores BER de la secuencia decodificada, pero las mejoras obtenidas con cada iteración se reducen conforme el número de iteraciones se incrementa, por lo que por razones de complejidad usualmente sólo se usan entre 4 y 12 iteraciones [11]. Para poder hacer un intercambio coherente de información, un entrelazador/desentrelazador debe ser utilizado entre los decodificadores SISO.

Con una transmisión binaria sobre un canal AWGN, un esquema de codificación de tasa 1/2, basado en una combinación de dos códigos RSC de tasa 1/2 y 16 estados, conectados con un entrelazador de tamaño 256×256, logró tasas de errores de 10^{-5} a una SNR por bit, o E_b/N_0 de 0.7 dB, muy cercano al límite de Shannon (0.2 dB) [19].

4.1. Principio de turbo codificación.

La estructura general usada en los turbo codificadores se muestra en la figura 4.1. Dos códigos componentes son usados para codificar los mismos bits de entrada, pero un entrelazador se coloca entre los codificadores para crear la característica aleatoria de la codificación de acuerdo a la teoría de Shannon. Generalmente, se usan códigos RSC como códigos componentes, pero se ha visto que es posible lograr un buen desempeño usando una estructura con otros componentes, como por ejemplo, códigos de bloque. Generalizando, es también posible utilizar más de dos códigos componente. Sin embargo el tratamiento de la presente tesis se concentra en la estructura estándar del turbo codificador usando dos códigos RSC (una variante se encuentra en [31], donde se habla de turbo códigos 3D, los cuales manejan tanto concatenación paralela como serial, con la consecuente mayor complejidad).

Para cambiar la tasa, las salidas de los dos códigos componentes se pueden luego perforar y multiplexar. En este caso, para obtener una tasa de codificación global de 1/2, la mitad de los bits de salida de los dos codificadores se perforan. El arreglo que ha sido escogido por preservar mejor las características de distancia de las secuencias codificadas, es el de transmitir los bits sistemáticos del primer codificador RSC, y la mitad de los bits de paridad de cada codificador. Los bits sistemáticos son rara vez perforados, ya que esto degradaría el desempeño del código con mayor grado que el de perforar los bits de paridad.

Un entrelazador Π (también llamado un permutador) se aplica en la secuencia de información antes de la codificación RSC_2. Este entrelazador juega un papel crucial en la construcción del código, ya que aplica una permutación de la secuencia de entrada, lo cual introduce aleatoriedad en el esquema de codificación. En otras palabras, los dos codificadores constitutivos codifican la misma secuencia de información pero en un orden diferente. La aleatoriedad para los códigos correctores de errores es un atributo clave que permite

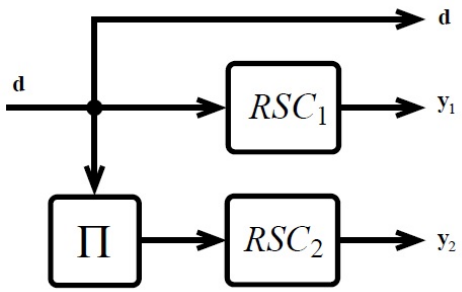

Figura 4.1: Esquemático del turbo codificador.

acercarse a la capacidad del canal. Como se mencionó en la primera sección, el trabajo de C. Shannon mostró que era posible lograr la capacidad de canal (la llamada "cota de Shannon") usando códigos aleatorios. El teorema de Shannon es una prueba de la existencia de tal código que logre la capacidad del canal.

El concepto de códigos concatenados ha sido adoptado en muchos estándares de comunicaciones digitales debido a sus excelentes capacidades de corrección de errores. Por ejemplo, en comunicaciones satelitales, tal como DVB-RCS (*Digital Video Broadcasting - Return Channel via Satellite*), Eutelsat (skyplex) e Inmarsat. Adicionalmente, está presente en estándares de comunicaciones móviles como UMTS(*Universal Mobile Telecommunications Systems*), CDMA2000, WiMAX (*WorldWide Interoperability for Microwave Access*) (IEEE 802.16), 3GPP-LTE (*Long Term Evolution*) y *LTE-Advanced*.

4.2. Decodificación de turbo códigos.

Para esta etapa la aplicación directa de un algoritmo de decodificación es impráctica ya que el entrelazador haría la estructura concatenada *trellis* mucho más compleja. En consecuencia, en vez de que se combinaran los *trellis* de ambos codificadores al mismo tiempo, se propuso decodificar cada código constitutivo por separado usando decodificadores SISO. Estos decodificadores luego intercambiarían información probabilística de una manera iterativa.

Como se mencionó previamente, los turbo códigos se usan para construir "códigos largos" con propiedades aleatorias, que pueden ser decodificados con una complejidad razonable usando decodificación iterativa de los códigos constitutivos que conforman a los turbo códigos. El elemento importante en la turbo decodificación son los decodificadores SISO (*soft-in soft-out*) (ver figura 4.2). Cada SISO toma como entradas:

- La salida del demodulador correspondiente a la parte sistemática transmitida $\mathbf{y}^s$.
- La salida del demodulador correspondiente a las paridades transmitidas asociadas con el SISO (es decir, $\mathbf{y}^{P1}$ para SISO1 y $\mathbf{y}^{P2}$ para SISO2).

- Información *a priori*. Esto es, $L^k_{apr} = ln\frac{p(u_k=1)}{p(u_k=0)}$. Aquí $k = 0, \ldots, N-1$ donde N es el tamaño de la secuencia de información en símbolos.

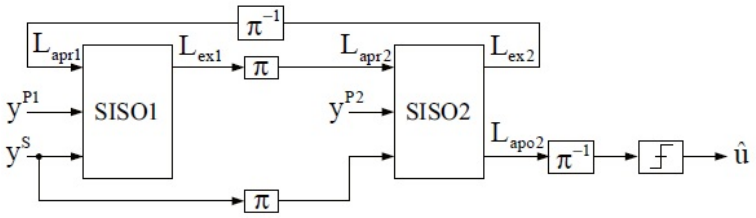

Figura 4.2: Estructura del turbo decodificador.

Sea $\mathbf{y}_{SISO}$ la parte de la secuencia recibida correspondiente al módulo SISO (es decir, $\mathbf{y}_{SISO} = \mathbf{y}^S, \mathbf{y}^{P_1}$ para SISO1 y $\mathbf{y}_{SISO} = \mathbf{y}^{P_2}$ para SISO2). Cada módulo SISO es capaz de producir dos salidas suaves:

- Una salida suave llamada información extrínseca, L_{ex}, necesaria para el intercambio entre decodificadores constitutivos.
- Información *a posteriori* que es adecuada para decisiones duras. Esta se define como: $L^k_{app} = ln\frac{p(u_k=1|\mathbf{y}_{SISO})}{p(u_k=0|\mathbf{y}_{SISO})}$.

El módulo SISO considerado en esta tesis está basado en el algoritmo *máximo a posteriori* (MAP). En el caso de que la palabra de código sea perforada antes de la transmisión, la salida del demodulador correspondiente a los bits perforados es nula.

La decodificación iterativa se realiza como sigue. El módulo SISO1 toma como entrada la parte sistemática $\mathbf{y}^S$, la información de paridad $\mathbf{y}^{P1}$ perteneciente al RSC1, y la información *a priori* L_{apr1}. En la primera iteración los valores L_{apr1} son inicializados a cero, asumiendo que todos los símbolos entrando a RSC1 son igualmente probables. El módulo SISO1 produce la información extrínseca L_{ex1} que es entrelazada, usando el mismo entrelazador π considerado en el turbo codificador, y luego es pasada al módulo SISO2. El módulo SISO2 toma como entradas la parte sistemática entrelazada $\mathbf{y}^S_\pi$, la información de paridad $\mathbf{y}^{P2}$ que pertenecen al RSC2 y la información extrínseca L_{ex1} provista por el módulo SISO1 como información *a priori* L_{apr2}. El módulo SISO2 produce la información extrínseca L_{ex2} que es desentrelazada, usando el entrelazador inverso π^{-1}, y es pasada al módulo SISO1 como información *a priori*. Este procedimiento se repite iterativamente. Después de un determinado número de iteraciones, SISO2 produce información *a posteriori* L_{app2}. Los valores L_{app2} son desentrelazados, usando igualmente π^{-1}, para que el orden coincida con el de la parte sistemática. Luego, para cada instante de tiempo $k = 0, \ldots, N-1$ una decisión "dura" se toma basándose en los valores L^k_{app2}. De esta manera, una iteración, también llamada una iteración completa, se refiere a dos decodificaciones SISO. Mientras que, una media iteración se refiere a una única decodificación SISO.

El desempeño respecto a la corrección de errores se puede dividir en las tres regiones señaladas en la figura 4.3: (a) región de SNR baja, (b) región de SNR media y (c) región de SNR alta [33].

- En la región (a), el desempeño es muy pobre y no adecuado para la mayoría de los sistemas de comunicación, aunque se llegase a aumentar el número de iteraciones.
- En la región (b), la curva de desempeño de error desciende rápidamente, teniendo así muy bajas tasas de error, para valores de SNRs moderados. Esta se llama la región *cascada*, en donde el desempeño lo determina principalmente el comportamiento convergente del decodificador. Generalmente, mientras mayor sea el entrelazador, será mejor la convergencia por el caracter aleatorio del codificador, y así más pronunciada la cascada. Esto es debido a que la naturaleza de la decodificación iterativa tiene poca mejora potencial en las propiedades de distancia de entrelazadores largos.
- La región (c) se conoce como *piso de ruido* y se caracteriza por un aplanamiento severo de las curvas de tasa de error. En esta región el decodificador iterativo típicamente converge después de pocas iteraciones, y más iteraciones generalmente no mejoran mucho el desempeño. El piso de ruido depende principalmente de la distancia mínima del turbo código, lo cual a su vez es determinado por el entrelazador utilizado. Mientras mejor sean las propiedades de distancia, bajará más el piso de ruido. En otras palabras, distancias mínimas altas con bajas multiplicidades son importantes para disminuir este piso de ruido.

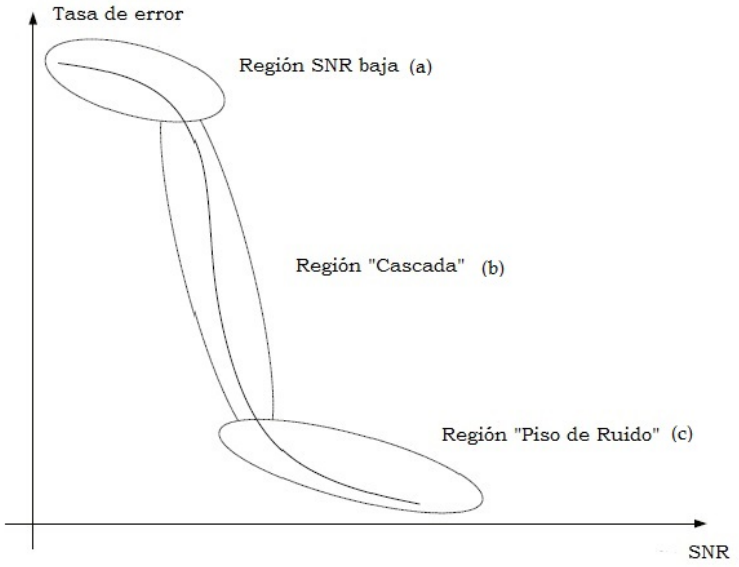

Figura 4.3: Regiones de desempeño de turbo códigos. Las tasas de error están en escala logarítmica base 10 y las SNRs (dB) en escala lineal.

La turbo decodificación se asemeja a un efecto tipo "dominó" en el que los bits más fiables (aquellos con probabilidades *a posteriori* próximas a 0, o a 1) fuerzan a los bits vecinos a tomar valores acordes con estas probabilidades para cumplir las restricciones del *trellis*. El

entrelazador consigue que esta fiabilidad se comparta por todos los bits al esparcir los bits de información entre los dos códigos. Al realizar la iteración entre los decodificadores de los dos códigos convolucionales, el algoritmo de decodificación permite que la gran fiabilidad de algunos bits se extienda a todos los bits de información y que después de varias iteraciones se obtengan estimados de muy alta fiabilidad para toda la secuencia de información, con probabilidades de error muy bajas a la salida del decodificador.

Para códigos sistemáticos, la salida suave para un bit de información será representada por la suma de tres términos:

$$L_{app}^{k} = L_c \cdot y + L_{apr}^{k} + L_{ex}^{k} = L_{in}^{k} + L_{apr}^{k} + L_{ex}^{k} \tag{4.1}$$

La LLR *a posteriori* está compuesta de tres LLRs. La primera, llamada información *intrínseca* contiene la contribución de la parte sistemática (del bit actual en cuestión). La segunda, es la LLR *a priori*. Y la tercera, llamada información *extrínseca*, provee la contribución probabilística pura de la paridad, y del resto de símbolos sistemáticos (no el actual) dada la estructura del codificador y la secuencia recibida $\mathbf{y}$. Cuando se itera, la información extrínseca del decodificador MAP previo se vuelve la información *a priori* del decodificador MAP actual. Esto se puede expresar como,

$$L_{app}^{k}(actual) = L_{ex}^{k}(anterior) + L_{in}^{k}(actual) + L_{ex}^{k}(actual). \tag{4.2}$$

Para cada módulo SISO, la información *a priori* es provista por el otro decodificador a través del entrelazador, o desentrelazador. La información extrínseca anterior se convierte en la información *a priori* actual; y se calcula una proporción de la información *a posteriori* tomando lo que ya es conocido por el decodificador. Estos valores extrínsecos son una estimación del valor "duro" (la certidumbre de que el valor duro es correcto), cuando no se consideran ni la información del canal ni la *a priori*.

4.3. Algoritmo de decodificación Max-Log-MAP

.

Desde el punto de vista de implementación, el algoritmo BCJR es relativamente complejo debido al largo número de multiplicaciones y operaciones exponenciales que deben realizarse. Por esta razón, una versión logarítmica del algoritmo se prefiere, en donde las multiplicaciones se vuelven sumas y se eliminan las operaciones exponenciales. Se usa la siguiente aproximación, cuya demostración se justifica en el apéndice B.

$$ln\left(\sum_{i} e^{x_i}\right) \approx max(x_i) \tag{4.3}$$

Defínase:

- El logaritmo de la métrica de estado hacia adelante $\alpha_k(s)$: $A_k(s) = ln(\alpha_k(s))$
- El logaritmo de la métrica de estado hacia atrás $\beta_k(s)$: $B_k(s) = ln(\beta_k(s))$
- El logaritmo de la métrica de rama $\gamma_k(s)$: $\Gamma_k(s) = ln(\gamma_k(s))$

Las ecuaciones de recursividad hacia adelante y hacia atrás se escriben como:

$$\begin{aligned} A_k(s) &= ln\left(\sum_{(s',s)} \alpha_{k-1}(s')\gamma_k(s',s)\right) = ln\left(exp\left[A_{k-1}(s') + \Gamma_k(s',s)\right]\right) \\ &\approx max(A_{k-1}(s') + \Gamma_k(s',s)) \end{aligned} \tag{4.4}$$

$$\begin{aligned} B_k(s) &= ln\left(\sum_{(s',s)} \beta_{k-1}(s')\gamma_k(s',s)\right) = ln\left(exp\left[B_{k-1}(s') + \Gamma_k(s',s)\right]\right) \\ &\approx max(B_{k-1}(s') + \Gamma_k(s',s)) \end{aligned} \tag{4.5}$$

La siguiente explicación hace referencia a $A_k(s)$, pero la explicación es similar para $B_k(s)$. La ecuación (4.4) implica que para cada trayectoria del *trellis* desde el estado anterior al estado en la etapa presente, el algoritmo suma un término de métrica de rama $\Gamma_k(s',s)$ al valor previo $A_{k-1}(s')$ para encontrar un nuevo valor $A_k(s)$ de ese camino. El nuevo valor de $A_k(s)$ es el máximo de los valores $A_k(s)$ de los múltiples caminos que se alcanzan en el estado $S_k = s$. Esto puede ser visualizado como seleccionar un camino como el "sobreviviente", y descartar cualesquiera otros caminos que alcanza el estado.

El valor de $A_k(s)$ debe de corresponder al logaritmo natural de que el *trellis* esté en el estado $S_k = s$ en la etapa k, dado que la secuencia de canal recibida hasta ese punto haya sido $y_{j<k}$. Sin embargo cuando se calcula esta probabilidad, debido a la aproximación, solamente el camino de máxima verosimilitud a través del estado $S_k = s$ es considerado. Por lo cual, el valor de A_k permite obtener la probabilidad del camino más probable a través del *trellis* hacia el estado $S_k = s$, en vez de la probabilidad de cualquier camino a través del *trellis* en el estado $S_k = s$. Esta aproximación es una de las razones por la cual el desempeño del algoritmo Max-Log-MAP es subóptimo, comparado con el algoritmo MAP. La métrica de rama es equivalente a la que se usa en el algoritmo de Viterbi con la suma del término *a priori* $u_k L(u_k)$.

Finalmente, se pueden calcular las LLRs *a posteriori* $L(d_k|\mathbf{y})$ como (tomando la misma convención que la señalada en la ecuación (2.51)):

$$L(d_k|\mathbf{y}) = ln\left(\frac{\sum_{R_1} exp(A_{k-1}(s') + \Gamma_k(s',s) + B_k(s))}{\sum_{R_0} exp(A_{k-1}(s') + \Gamma_k(s',s) + B_k(s))}\right) \tag{4.6}$$

que se puede aproximar como

$$L(d_k|\mathbf{y}) \approx \underset{R_1}{max}\ (A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)) - \underset{R_0}{max}\ (A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)) \tag{4.7}$$

Esto significa que en el algoritmo Max-Log-MAP para cada bit u_k la LLR *a posteriori* $L(u_k|\mathbf{y})$ se calcula considerando cada transición de la etapa S_{k-1} del *trellis* a la etapa S_k. Estas transiciones son agrupadas en aquellas que pudieron ocurrir si $u_k = +1$, y en aquellas

que pudieron ocurrir si $u_k = -1$. Para ambos grupos la transición que arroje el valor máximo de $A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)$ es encontrada, y la LLR *a posteriori* es calculada basada solamente en estas dos "mejores" transiciones.

La complejidad del algoritmo Max-Log-MAP no es mucho mayor que la del algoritmo de Viterbi, es decir, en vez de una recursión, dos son llevadas a cabo.

4.4. Algoritmo de decodificación Log-MAP.

El algoritmo Max-Log-MAP presenta una degradación leve en desempeño comparada con el algoritmo MAP debido a la aproximación que utiliza. Cuando se usa en la decodificación iterativa de turbo códigos, esta degradación resulta en una caída en desempeño de alrededor de 0.35 dB [11]. Sin embargo, esa aproximación puede hacerse exacta si se utiliza el logaritmo Jacobiano:

$$log(exp(x) + exp(y)) = max(x, y) + log(1 + exp(-|y - x|)) = max^*(x, y) \tag{4.8}$$

para $x, y \in \mathbb{R}$. La función $max^*(x, y)$ puede ser implementada usando la arquitectura de la figura 4.4, donde una tabla *look-up* se usa para calcular la función $log(1+exp(-|y-x|))$. Se ha encontrado en [25] que esa tabla *look-up* necesita contener solamente ocho valores para las posibles diferencias, que estén en un rango dentro de 0 y 5. Más aún, para varios operandos, esta función puede ser calculada recursivamente gracias a su propiedad asociativa, usando múltiples operadores max^* de 2 operandos: $max^*(x, y, z) = max^*(max^*(x, y), z)$.

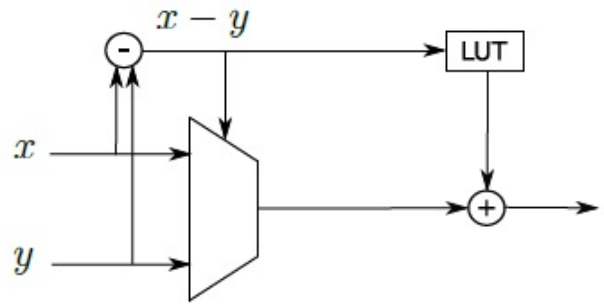

Figura 4.4: Arquitectura para la implementación de la función $max^*(x, y)$.

Las expresiones utilizadas para el algoritmo Log-MAP son las mismas utilizadas en el algoritmo Max-Log-MAP, pero utilizando el operador max^* en vez del operador max. El uso de la tabla *look-up* hace que el resultado del algoritmo BCJR y el Log-MAP sean iguales, sin embargo se introduce mayor complejidad. El algoritmo subóptimo Max-Log-MAP es comúnmente usado en las implementaciones prácticas ya que su desempeño es aceptable comparado con el del algoritmo Log-MAP. Además, pueden surgir problemas de precisión numérica debido a la implementación de un término de corrección. Esto podría incrementar considerablemente el número de bits requeridos para codificar los valores internos de este

algoritmo, y en consecuencia, la complejidad del algoritmo. Además de que reduce la máxima frecuencia de reloj [23].

En ambos algoritmos, Log-MAP y Max-Log-MAP, si el estado inicial (final) del codificador es conocido, la recursión hacia adelante (respectivamente hacia atrás) es inicializada con 0 para el estado inicial y $-\infty$ para los otros estados. De lo contrario, todas las métricas de estado son inicializadas a un valor idéntico, generalmente 0.

Otra causa de degradación en el desempeño del algoritmo Max-Log-MAP se debe al hecho de que la información extrínseca es menos confiable, ya que al no ser probabilidades como tal (si no sus logaritmos) en las primeras iteraciones se tienen valores muy optimistas. Sin embargo, reducir la confianza en la información pasada permite compensar este efecto. Esto se logra multiplicando la información extrínseca por un factor de escalamiento menor a 1.0. El mejor desempeño observado se obtiene cuando se usa un factor de escalamiento de 0.7, para todas las iteraciones, excepto para la última. Otra ventaja del algoritmo Max-Log-MAP es que no requiere el conocimiento de la varianza del ruido en el canal de transmisión y por lo tanto es independiente de la SNR [21].

Otro algoritmo para decodificar turbo códigos es el algoritmo SOVA (*Soft Output Viterbi Algorithm*)(que es menos complejo), pero presenta una degradación de alrededor 0.3 dB comparado con el algorito Max-Log-MAP. SOVA toma en cuenta cuatro caminos en el *trellis* para calcular su salida suave, mientras que Max-Log-MAP toma en cuenta todos los posibles caminos desde el estado inicial al estado final. Como resultado, las salidas suaves producidas por SOVA son menos confiables que las de Max-Log-MAP, y por lo tanto en este trabajo de tesis no se considerará al algoritmo SOVA para la implementación.

Un resumen del algoritmo Log-MAP se puede ver en la figura 4.5.

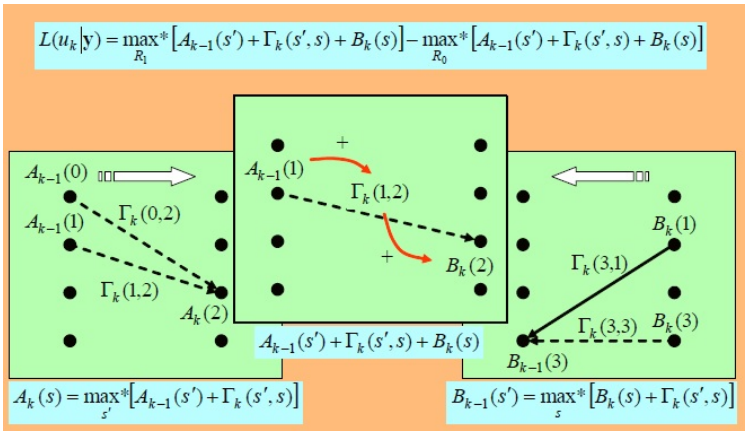

Figura 4.5: Resumen de las expresiones usadas en los algoritmos MAP simplificados [32].

4.5. Entrelazadores.

Para proveer el caracter aleatorio a la codificación, el entrelazador es un componente esencial en el rendimiento de los turbo códigos. Dolinar et al. [10]. han encontrado que el

desempeño de un esquema de turbo decodificación puede ser mejorado si el entrelazador es capaz de eliminar secuencias de entrada que producen palabras de código de bajo peso de Hamming. La región "cascada" está caracterizada por la ganancia del entrelazador, el cual corresponde a la ganancia en dB sobre la curva de desempeño de la transmisión no codificada.

El entrelazado es una técnica muy utilizada en sistemas de comunicación digital y de almacenamiento. Un entrelazador toma una secuencia de símbolos y permuta sus posiciones, acomodándolas en un orden temporal diferente. El objetivo básico de un entrelazador es procurar aleatoriedad a la secuencia de datos. Cuando se usan contra errores de ráfaga, los entrelazadores se diseñan para convertir patrones de errores que contienen secuencias largas de datos seriales erróneos en un patrón de error más aleatorio, por lo tanto distribuyendo los errores entre muchos vectores de código. Los errores de ráfaga son característicos de algunos canales, como el inalámbrico. También ocurren en códigos concatenados en serie, donde un decodificador interno sobrecargado con errores puede pasar una ráfaga de errores al decodificador externo [17].

El entrelazador es en sentido estricto, un permutador de tamaño N, que se puede describir como un proceso que cambia el orden de cierta secuencia, en un proceso uno a uno. Matemáticamente se tiene,

$$\begin{aligned} &\Pi : \{1, ..., N\} \rightarrow \{1, ..., N\} \\ &\Pi : i \rightarrow \pi(i) \end{aligned} \tag{4.9}$$

Los entrelazadores pueden ser definidos e implementados de distintas maneras; la figura 4.6. muestra una descripción gráfica que ilustra una implementación general. El entrelazador lee un vector de símbolos de entrada, v_{in}, y escribe en un vector de muestras de salida permutadas v_{out} [34].

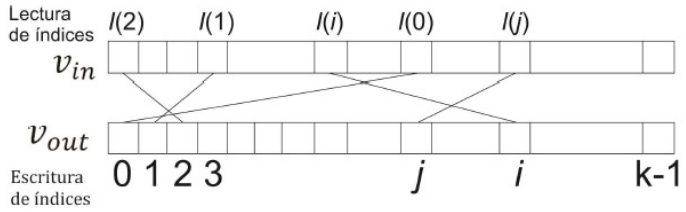

Figura 4.6: Descripción general de un entrelazador [34].

Otra tarea de gran importancia para el entrelazador es romper la correlación que existe entre la entrada de ambos codificadores. Esto para que en el decodificador, en caso de que un error no haya podido ser corregido por un módulo SISO, la probabilidad de que sea corregido por el otro módulo sea alta. Romper la correlación se puede interpretar como romper entradas de peso bajo; hay algunas entradas de peso bajo (particularmente las de peso Hamming 2) que provocan que el turbo codificador entregue una salida de peso bajo afectando así toda su distancia libre. Logrando que solo uno de los codificadores RSC genere

una secuencia de peso bajo, hace que la distancia libre del turbo codificador sea globalmente mejorada [20].

Existen varias formas de especificar una permutación, la primera es mediante ecuaciones que relacionen las direcciones previa, y posterior a la permutación. El segundo método es mediante una tabla *look-up* que provea la correspondencia entre direcciones. El primer método es preferible desde el punto de vista de simplicidad en la especificación del turbo código (los comites de estandarización son muy sensibles en este rubro) pero la segunda forma puede conducir a mejores resultados ya que el grado de libertad es generalmente mayor cuando se diseña la permutación.

El tipo de entrelazador más común, usado en sistemas de comunicación en general, es el entrelazador de bloque donde los elementos son escritos fila por fila en una matriz (la matriz *madre*) con N elementos y luego leídos columna por columna [3]. Por ejemplo, el entrelazador de bloque de longitud 16 escribirá la secuencia de mensaje $\mathbf{u} = [u_1 u_2 u_3 u_4 u_5 \dots u_{16}]$ en una matriz como

$$\begin{bmatrix} u_1 & u_2 & u_3 & u_4 \\ u_5 & u_6 & u_7 & u_8 \\ u_9 & u_{10} & u_{11} & u_{12} \\ u_{13} & u_{14} & u_{15} & u_{16} \end{bmatrix}$$

y se lee como $[u_1\ u_5\ u_9\ u_{13}\ u_2\ u_6\ u_{10}\ u_{14}\ u_3\ u_7\ u_{11}\ u_{15}\ u_4\ u_8\ u_{12}\ u_{16}]$.

Los entrelazadores aleatorios pueden utilizarse para satisfacer el criterio de convergencia de la decodificación. Desafortunadamente, la probabilidad de encontrar un entrelazador aleatorio, para el cual no haya parejas de símbolos que permanezcan cercanas antes y después del entrelazado, es cercana a 0. Por lo tanto, el desempeño del código es mejorado gracias a ciertas restricciones impuestas al entrelazador. Por esta razón, los entrelazadores *S-random* (*Spread Random*) fueron introducidos en 1995 por Dolinard [10]. La construcción de este entrelazador aleatorio introduce una restricción en la elección de índices permutados. En cada paso de la construcción incremental, un índice permutado aleatorio es seleccionado, que verifique la llamada condición de *dispersión* (*spread*): dos símbolos están al menos S símbolos distantes en el orden natural o en el entrelazado. Se demostró que para una eleción de S de alrededor de $\sqrt{N/2}$, este algoritmo de construcción incremental es exitoso en una medida razonable de tiempo.

En el 2000, Crozier [8] propone el entrelazador HSR (*High Spreand Random*) cuya construcción está basada en el algoritmo del entrelazador *S-random*. La diferencia es que en vez de generar números enteros, se trabaja con números reales. La métrica para la distancia S se redefine con la siguiente ecuación:

$$S(i,j) = |\pi(i) - \pi(j)| + |i - j| \tag{4.10}$$

Se tiene en este caso un factor de dispersión de hasta $\sqrt{2N}$. Los patrones de permutación regulares (aquellos que se pueden describir por medio de expresiones analíticas) son más fáciles de implementar, con respecto a estructuras no determinísticas como las generadas por el entrelazador *S-random* o HSR (aunque presente un muy buen desempeño), esto requiere almacenar el patrón de permutación en tablas *look-up*. Estas tablas representan una sobrecarga especial en recursos de *hardware* cuando diferentes tamaños de entrelazador se requieren en el mismo sistema. Otro punto que debe ser considerado es la integración entre el entrelazador y el resto de la arquitectura del codificador/decodificador.

El estándar 3GPP-LTE Advanced considera el uso de entrelazadores basados en polinomios de permutación cuadrática, mismos que se describen a continuación.

4.5.1. Permutación cuadrática.

Recientemente, Sun y Takeshita [35] propusieron una nueva clase de entrelazadores determinísticos basados en polinomios de permutación (PP) sobre anillos de enteros modulares. El uso de PP reduce el diseño de entrelazadores a simplemente la selección de los coeficientes polinomiales. Más aún, los turbo códigos basados en PP han demostrado tener (a) buenas propiedades de distancia [36], que son deseables para disminuir el piso de ruido y (b) una propiedad de máxima contención libre [37] la cual es deseable para procesamiento paralelo que permita una implementación en hardware con alta velocidad de decodificadores iterativos turbo.

Antes de tratar el PP cuadrático, definiremos una forma general de un polinomio y se discutirá como se puede verificar si un polinomio es PP sobre el anillo de enteros módulo N, $\mathbb{Z}_N$.

Polinomios de permutación.

Dado un entero $N \geq 2$, un polinomio

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m mod N \tag{4.11}$$

donde los coeficientes $a_0, a_1, a_2, \ldots, a_m$ y m son enteros no negativos, se dice que es un polinomio de permutación sobre $\mathbb{Z}_N$ cuando $f(x)$ permuta $\{0, 1, 2, \ldots, N-1\}$. Como se tiene una operación módulo N, es suficiente que los coeficientes $a_0, a_1, a_2, \ldots, a_m$ estén en $\mathbb{Z}_N$. Recordemos que la derivada formal del polinomio $f(x)$ está dada por:

$$f'(x) = a_1 + 2a_2 x + 3a_3 x^2 + \ldots + m a_m x^{m-1} mod N \tag{4.12}$$

Para verificar si un polinomio es un PP sobre $\mathbb{Z}_N$, revisaremos los tres siguientes casos: (a) cuando $N = 2^n$, donde n es un elemento de los enteros positivos $\mathbb{Z}_+$, (b) cuando $N = p^n$ donde p es cualquier número primo y (c) cuando N es un elemento arbitrario de $\mathbb{Z}_+$.

1. Caso I ($N = 2^n$): un teorema en [38] establece que $f(x)$ es un PP sobre el anillo entero $\mathbb{Z}_{2^n}$ si y sólo si: 1) a_1 es impar, 2)$a_2 + a_4 + a_6 + \ldots$ es par, y $a_3 + a_5 + a_7 + \ldots$ es par.
   Ejemplo 1 para $N = 2^3 = 8$: $f(x) = 1 + 5x + x^2 + x^3 + x^5 + 3x^6$ es un PP sobre $\mathbb{Z}_{N=8}$ ya que mapea la secuencia $\{0, 1, 2, 3, 4, 5, 6, 7\}$ a $\{1, 4, 7, 2, 5, 0, 3, 6\}$. Notemos que $a_1 = 5$ es impar, $a_2 + a_4 + a_6 = 1 + 0 + 3 = 4$ es par, y $a_3 + a_5 = 1 + 1 = 2$ es par.
   Ejemplo 2 para $N = 2^3 = 8$: $f(x) = 1 + 4x + x^2 + x^3 + x^5 + 3x^6$ no es un PP sobre $\mathbb{Z}_{N=8}$ ya que mapea la secuencia $\{0, 1, 2, 3, 4, 5, 6, 7\}$ a $\{1, 3, 5, 7, 1, 3, 5, 7\}$. Notemos que $a_1 = 4$ es par.

2. Caso II ($N = p^n$): un teorema en [59] garantiza que $f(x)$ es un PP módulo p^n si y sólo si $f(x)$ es un PP módulo p y $f'(x) \neq 0$ modulo p, para cada entero $x \in \mathbb{Z}_{p^n}$. Nótese que el Caso I es simplemente un caso especial del caso II ya que $p = 2$ es un número primo.
   Ejemplo 1 para $N = 3^n (p = 3)$: $f(x) = 1 + 2x + 3x^2$ es un PP sobre $x \in \mathbb{Z}_{3^n}$ ya que

$f(0) = 1 \mod 3 = 0$, $f(1) = 6 \mod 3 = 0$, $f(2) = 17 \mod 3 = 2$, y $f'(x) = 2+6x = 2 \mod 3 = 2$ es una constante diferente de cero para toda x en $\mathbb{Z}_{3^n}$.
Ejemplo 2 para $N = 3^n (p = 3)$: $f(x) = 1+6x+3x^2$ no es un PP sobre $x \in \mathbb{Z}_{3^n}$ ya que $f'(x) = 6 + 6x = 0 \mod 3 = 0$ para toda x en $\mathbb{Z}_{3^n}$. En particular, para $N = 3^2 = 9$, $f(x)$ mapea la secuencia $\{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ a $\{1, 1, 7, 1, 1, 7, 1, 1, 7\}$.

3. Caso III (N arbitraria): sea $P = \{2, 3, 5, 7, \ldots\}$ el conjunto de números primos. Luego, para cada $N \in \mathbb{Z}_+$ puede ser factorizada como $N = \prod_{p \in P} p^{n_{N,p}}$, donde todos los valores de p son números primos distintos, $n_{N,p} \geq 1$ para un cierto número de p y $n_{N,p} = 0$ de otra manera. Este es el teorema fundamental de la aritmética. Por ejemplo, si $N = 2500 = 2^2 \times 5^4$, luego tenemos que $n_{2500,2} = 2$ y $n_{2500,5} = 4$. Un teorema en [35] establece que para cualquier $N = \prod_{p \in P} p^{n_{N,p}}$, $f(x)$ es un PP módulo N si y sólo si $f(x)$ es también un PP módulo $p^{n_{N,p}}, \forall p$ tal que $n_{n,P} \geq 1$. Con este teorema, verificar si un polinomio es un PP módulo N se reduce a verificar el módulo polinomial para cada factor $p^{n_{N,P}}$ de N. Para $p = 2$, se usa el teorema reportado en el caso I, que es una simple prueba de los coeficientes polinomiales.
Para $p \neq 2$, se debe usar el teorema reportado en el caso II, lo cual no puede hacerse por simple verificación de los coeficientes polinomiales. Para una N arbitraria, es difícil desarrollar una simple prueba de coeficientes para checar si un polinomio arbitrario de grado m $f(x)$ es un PP módulo N. Sin embargo, para un polinomio cuadrático $(m = 2)$, $f(x) = a_0 + a_1 x + a_2 x^2$, una prueba simple de coeficientes ha sido propuesta en [40]. La siguiente sección detallará esta prueba de coeficientes.

.

Polinomios de permutación cuadrática (QPP).

Como la constante q_0 en el polinomio cuadrático $q(x) = q_0 + q_1 x + q_2 x^2$ solamente causa un "corrimiento cíclico" a los valores permutados, se definirán sin pérdida de generalidad, los polinomios cuadráticos como $q(x) = q_1 x + q_2 x^2$. Considerando la notación de que b es divisible por a como $a|b$, y lo opuesto como $a \not| b$. El máximo común divisor de a y b se denota como $mcd(a, b)$. Si $mcd(a, b) = 1$, entonces a y b son primos relativos. Como se verá a continuación en la Proposición 1, estamos interesados principalmente en la factorización del coeficiente q_2, que puede ser escrito de acuerdo a la notación previa como $q_2 = \prod_{p \in P} p^{n_{N,p}}$. La siguiente Proposición 1 provee una condición necesaria y suficiente para verificar si un polinomio cuadrático es un PP módulo N.

Proposición 1: Sea N = $\prod_{p \in P} p^{n_{N,p}}$. Para que un polinomio cuadrático $q(x) = q_1 x + q_2 x^2$ módulo N sea un PP, las siguientes condiciones necesarias y suficientes deben de satisfacerse [41]:

1. $2|N$ y $4 \not| N$ (es decir, $n_{N,2} = 1$). $q_1 + q_2$ es impar, $gcd(q_1, N/2) = 1$, y $q_2 = \prod_{p \in P} p^{n_{q_2,p}}, n_{q_2,p} \geq 1, \forall p$ tal que $p \neq 2$ y $n_{N,p} \geq 1$.

2. Ya sea que $2 \not| N$ o $4|N$ (es decir, $n_{N,2} \neq 1$) $mcd(q_1, N) = 1$ y $q_2 = \prod_{p \in P} p^{n_{q_2,p}}, n_{q_2,p} \geq 1, \forall p$ tal que $n_{N,p} \geq 1$.

La expresión $q_2 = \prod_{p \in P} p^{n_{q_2,p}}, n_{q_2,p} \geq 1, \forall p$ tal que $n_{N,p} \geq 1$, puede ser expresada en palabras simples como sigue: *cada factor p de N debe ser también un factor de* q_2. Es

importante notar que esta expresión sigue permitiendo que q_2 tenga factores primos que difieran de todos los p factores de N.

Ejemplo 1: Si $N = 36$, luego se tiene el caso I de la Proposición 2 ya que $4|N$. Todos los posibles valores de q_1 son simplemente el conjunto de números que son relativamente primos a N. En consecuencia, $q_1 = \{1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35\}$. Ya que $36 = 2^2 \times 3^2$ ($p_1 = 2$ y $p_2 = 3$), luego 2 y 3 deben ser un factor de q_2. Esto es, $q_2 = \{(2 \times 3), (2^2 \times 3), (2^3 \times 3), (2 \times 3^2), 5 \times (2 \times 3)\} = \{6, 12, 24, 18, 30\}$. Como se mencionó arriba, el uso del número primo 5 en $5 \times (2 \times 3)$ no viola la condición en el caso I de la Proposición 2. En total, para $N = 36$ hay $12 \times 5 = 60$ posibles QPPs.

La expresión $q_2 = \prod_{p \in P} p^{n_{q_2,p}}, n_{q_2,p} \geq 1, \forall p$ tal que $p \neq 2$ y $n_{N,p} \geq 1$ del caso 2, también puede ser expresada en palabras simples como sigue: *cada factor $p \neq 2$ de N debe ser también un factor de q_2*. Es importante notar que esta expresión sigue permitiendo que q_2 tenga el factor primo 2 y todos los factores primos que difieren de todos los p factores de N (q_2 puede, o no, tener 2 como factor).

Ejemplo 2: Si $N = 90$, luego tenemos el caso 2 de la Proposición 2 ya que $2|N$ y $4 \not| N$. Como $N = 90 = 2 \times 3^2 \times 5$, todos los valores de p que difieren de 2 son $p_1 = 3$ y $p_2 = 5$. Por lo tanto, los valores potenciales para q_2 son $\{(3 \times 5), (3^2 \times 5), (3 \times 5^2), 2 \times (3 \times 5), 2^2 \times (3 \times 5)\} = \{15, 45, 75, 30, 60\}$.

Bajo la condición $mcd(q_1, N/2 = 45) = 1$, los valores potenciales para q_1 hacen que se tengan 120 posibles QPPs:

1, 2, 4, 7, 8, 11, 13, 14, 16, 17, 19, 22, 23, 26, 28, 29, 31, 32, 34, 37, 38, 41, 43, 44, 46, 47, 49, 52, 53, 56, 58, 59, 61, 62, 64, 67, 68, 71, 73, 74, 76, 77, 79, 82, 83, 86, 88, 89.

A pesar de las condiciones impuestas sobre q_1 y q_2, el espacio de búsqueda de los QPPs es muy largo, especialmente para entrelazadores de tamaño medio a largo. Es deseable reducir más el espacio de búsqueda. Una solución podría ser considerar aquellos QPPs que tengan un cuadrático inverso. Esta solución que reduce el espacio de búsqueda está basada en el descubrimiento interesante reportado por Rosnes y Takeshita [36] para el caso $32 \leq N \leq 512$ y $N = 1024$, en el que los entrelazadores basados en QPP con inversos cuadráticos son estrictamente superiores (en términos de distancia mínima) que aquellos que no poseen un cuadrático inverso. Usando una búsqueda computacional exhaustiva, Rosnes y Takeshita provieron para turbo códigos de 8 y 16 estados, una lista muy útil para los mejores QPPs (en términos de distancia mínima) para un amplio rango de N ($32 \leq N \leq 512$ y $N = 1024$). La especificación 3GPP LTE-Advanced define 188 posibles tamaños de bloque, que van de 40 a 6144, y estos posibles tamaños $\mathbf{N}$ se pueden resumir como sigue:

$$\mathbf{N} = \begin{cases} 40 + 8 \times (\Bbbk - 0) & \text{para } \Bbbk = 0 - 58 \\ 512 + 16 \times (\Bbbk - 59) & \text{para } \Bbbk = 59 - 90 \\ 1024 + 32 \times (\Bbbk - 91) & \text{para } \Bbbk = 91 - 122 \\ 2048 + 64 \times (\Bbbk - 123) & \text{para } \Bbbk = 123 - 187 \end{cases}$$

Se puede verificar fácilmente que un entrelazador descrito por $f(x)$ y su correspondiente desentrelazador descrito por $f^{(-1)}(x)$ cumple la siguiente propiedad:

$$f(f^{(-1)}(x)) \equiv f^{(-1)}(f(x)) \equiv x (mod\ N). \tag{4.13}$$

En [42] se encuentran tablas de algunos entrelazadores QPP, junto con sus inversos, lo cual puede considerarse para la implementación. Hoon [43] identificó la existencia de sólo 6 entrelazadores autoinvertibles (es decir, aquellos cuya función de entrelazador es la misma

que se utiliza para el desentrelazador), esto es importante ya que se simplifican los recursos de hardware. La tabla de entrelazadores autoinvertibles se presenta a continuación.

longitud	QPP
48	$7x + 12x^2$
176	$21x + 44x^2$
240	$29x + 60x^2$
304	$37x + 76x^2$
864	$17x + 48x^2$
2688	$127x + 504x^2$

Tabla 4.1: Polinomios cuadráticos auto-invertibles para entrelazadores 3GPP LTE.

Implementación en hardware del entrelazador QPP.

Basándose en un análisis algebraico, se garantiza que el entrelazador QPP siempre genera una única dirección, lo cual simplifica bastante la implementación en *hardware*. En la decodificación sobre el *trellis* MAP (o cualquiera de sus variantes), las direcciones QPP de entrelazado son normalmente generadas en un orden consecutivo (con un tamaño de paso d) [14]. Tomando ventaja de esto, la dirección de entrelazado QPP puede calcularse de una manera recursiva. Supongamos que el entrelazador comienza en x_0, primero se pre-calcula $f(x_0)$ como:

$$f(x_0) = (f_2 x_0^2 + f_1 x_0) mod(N) \tag{4.14}$$

En los siguientes ciclos, conforme x es incrementada en d, $f(x+d)$ es calculada recursivamente como sigue:

$$f(x+d) = (f_2(x+d)^2 + f_1(x+d)) mod(N) \tag{4.15}$$

$$= (f(x) + g(x)) mod(N), \tag{4.16}$$

donde $g(x)$ está definida por

$$g(x) = (2df_2 x + d^2 f_2 + df_1) mod(N) \tag{4.17}$$

Notemos que $g(x)$ también puede calcularse de manera recursiva:

$$g(x+d) = (g(x) + 2d^2 f_2) mod(N) \tag{4.18}$$

$$= (g(x) + (2d^2 f_2 mod(N))) mod(N) \tag{4.19}$$

El valor inicial $g(x_0)$ necesita ser precalculado como

$$g(x_0) = (2df_2 x_0 + d^2 f_2 + df_1) mod(N) \tag{4.20}$$

La operación módulo en (4.16) y (4.19) puede ser difícil de implementar en *hardware* si no se conocen los operandos con anticipación. Sin embargo, por definición sabemos que $f(x)$ y $g(x)$ son ambos menores a N, por lo que calcular (4.16) y (4.19) se puede realizar mediante sumas. En el método propuesto en [14], se requiere tener tres números pre-calculados: $(2d^2 f_2) mod(N)$, $f(x_0)$, y $g(x_0)$. La figura (4.7) muestra una arquitectura de hardware para calcular la dirección de entrelazado $f(x)$, donde x comienza desde x_0 y es incrementada en d en cada ciclo de reloj. Por ejemplo, estableciendo d en 1, este circuito puede generar direcciones de entrelazado en cada paso de 1. Si n direcciones consecutivas de entrelazado se requieren durante cada ciclo de reloj, este circuito se puede replicar n veces con n diferentes valores iniciales: $x_0, x_0 + 1, \ldots,$ y $x_0 + n - 1$.

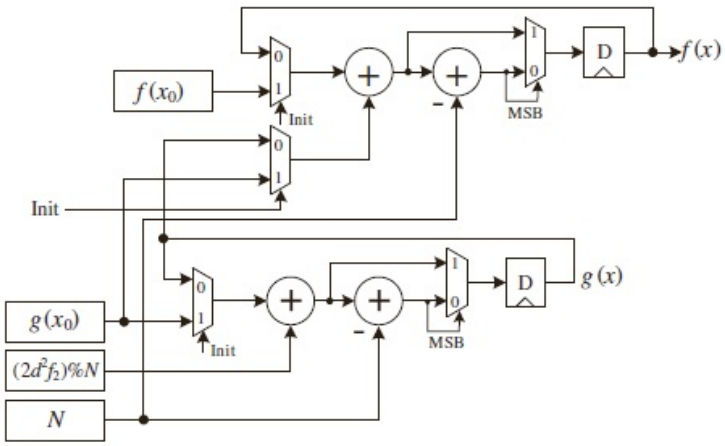

Figura 4.7: Diagrama del circuito generador de direcciones QPP, con tamaño de paso d [14].

4.5.2. Permutación casi regular (ARP).

Entre los modelos de permutación que se han sugerido hasta el momento, los aparentemente más prometedores en términos de distancias mínimas de Hamming están basados en una permutación regular que involucra un corrimiento circular [10] o el principio co-primo. Después de escribir los datos en una memoria lineal, con dirección i, $(0 \leq i \leq N-1)$, el bloque de información es ligado a un círculo; por lo que ambas extremidades de ese círculo ($i = 0$ e $i = N - 1$) son contiguas. Los datos son leídos de tal forma que el dato j-ésimo leído fue escrito en la posición i dada por:

$$i = \Pi(j) = (Pj + i_0)(mod(N)) \tag{4.21}$$

donde el valor de salto P es un entero, relativamente primo a N, e i_0 es el índice de comienzo. Esta permutación no requiere que el bloque sea visto como un rectángulo, esto es, N puede ser cualquier entero.

El dilema en el diseño de una buena permutación para turbo códigos reside en la necesidad de obtener una distancia mínima suficiente para dos clases distintas de palabras de código,

las cuales requieren un tratamiento diferente. La primera clase contiene todas las palabras de código construidas de una única secuencia (irreducible) de retorno a cero (RTZ) [1]. La expresión de permutación regular expresada por (4.21) es apropiada para esos patrones de error. La segunda clase abarca todas las palabras de código construidas a partir de combinaciones de secuencias sencillas RTZ. Para esta clase, se debe de introducir una *no uniformidad* en la función de permutación para obtener distancias mínimas largas.

En [44] y [45], dos modificaciones muy similares de (4.21) se propusieron para introducir un desorden controlado en la permutación regular. Ellas generalizan el principio turbo adoptado en los turbo códigos DVB-RCS/RCT, o del estándar IEEE802.16a. Se considera así el modelo ARP (*Almost Regular Permutation*) detallado en [45], que cambia la relación (4.21) a:

$$i = \Pi(j) = (Pj + Q(j) + i_0) mod(N) \tag{4.22}$$

donde $Q(j)$ es un entero, cuyo valor es tomado en un conjunto limitado $\{0, Q_1, Q_2, \ldots, Q_C - 1\}$, en una forma cíclica. C, llamado el *ciclo* de la permutación, debe ser un divisor de N y tiene un valor típico de 4 u 8. Por ejemplo, para $C = 4$, la ley de permutación está definida por:

$$\begin{array}{l} \text{Si } j = 0(mod4), i = \Pi(j) = Pj + 0 + i_0 \ (mod(N)) \\ \text{Si } j = 1(mod4), i = \Pi(j) = Pj + Q_1 + i_0 \ (mod(N)) \\ \text{Si } j = 2(mod4), i = \Pi(j) = Pj + Q_2 + i_0 \ (mod(N)) \\ \text{Si } j = 3(mod4), i = \Pi(j) = Pj + Q_3 + i_0 \ (mod(N)) \end{array} \tag{4.23}$$

y N debe ser un múltiplo de 4, la cual no es una condición muy restrictiva, con respecto a flexibilidad. Para asegurar la propiedad de biyección de Π, los valores de Q no son cualesquier valores. Una forma directa de satisfacer la condición de biyección es escoger todas las Qs como múltiplos de C.

La implementación de este entrelazador en *hardware* puede realizarse siguiendo un criterio recursivo, similar al que se uso en los entrelazadores QPP del apartado anterior.

[1] Una secuencia RTZ es cualquier secuencia finita de entrada que hace a un codificador RSC salir del estado cero y recuperarlo de nuevo

5

Turbo Decodificador: Diseño a nivel algorítmico.

En este capítulo se tratará con el diseño a nivel algorítmico del turbo decodificador y se subrayarán las consideraciones a tomar en cuenta para la implementación en *hardware*. Cuando circuitos digitales y elementos de almacenamiento son requeridos para implementar el algoritmo teórico, se debe de poner atención en cómo el proceso basado en ciclos y la limitación de recursos afectan la turbo decodificación [13]. El diseño práctico de un turbo decodificador contiene diversas unidades funcionales para los principales cálculos dentro del algoritmo simplificado; además requiere de las unidades de control necesarias para la propagación de mensajes entre los dos códigos componentes. En este capítulo se introducen algunas de las unidades principales del turbo decodificador que se implementará. Al trabajar con aritmética de punto fijo, es posible la introducción del problema de saturación *overflow*, por lo cual se explica cómo debe ser tratado en el manejo de las métricas de estado, que como ya se vio en el capítulo anterior, aumentan de manera recursiva.

5.1. Planificadores de algoritmos basados en MAP.

Las ecuaciones del algoritmo Max-Log-MAP dadas en el capítulo anterior pueden ser implementadas de acuerdo a varios planificadores que difieren en la secuenciación de las operaciones básicas: recursión hacia adelante, recursión hacia atrás y cálculo de la salida suave. Se consideran tres casos particulares para códigos RSC convolucionales terminados con bits de relleno. Tales planificadores se muestran en la figuras 5.1 a 5.3. El eje horizontal representa el tiempo, con unidades de periodo de símbolo. El periodo de símbolo corresponde al tiempo que se requiere para calcular una etapa de las recursiones hacia adelante, o hacia atrás. El eje vertical representa los índices de las etapas del *trellis*. Se asume que las entradas

suaves (datos de observación e información *a priori*) asociadas a los N símbolos de entrada están guardadas en una memoria y están todas disponibles desde el tiempo $t = 0$. Las líneas continuas simbolizan exclusivamente el cálculo de métricas de estado (hacia adelante o hacia atrás), mientras que las líneas punteadas representan un cálculo simultáneo de métricas de estado junto con información *a-posteriori*.

Planificador *Forward-Backward*. El primer planificador primero calcula la recursión hacia adelante de acuerdo con la ecuación (4.4) y almacena todas las métricas de estado hacia adelante que son generadas. Luego, una vez que la recursión hacia adelante es completada, la recursión hacia atrás se calcula empleando la ecuación (4.5). Al mismo tiempo que esta recursión hacia atrás, las salidas suaves son producidas combinando las métricas de estado hacia atrás actuales y las métricas de estado hacia adelante, siguiendo las ecuaciones (4.7) y (4.1). Se asume que el *trellis* comienza en el estado cero. Las recursiones se inicializan apropiadamente. El retardo de decodificación, definido como el tiempo requerido para producir N salidas suaves es igual a $2N$, y el tamaño de memoria requerido para almacenar las métricas de estado hacia adelante es igual a N conjuntos de métricas de estados. Los puntos negros (•) representan los valores iniciales de métricas de estado hacia adelante o hacia atrás. Se muestra en la figura 5.1.

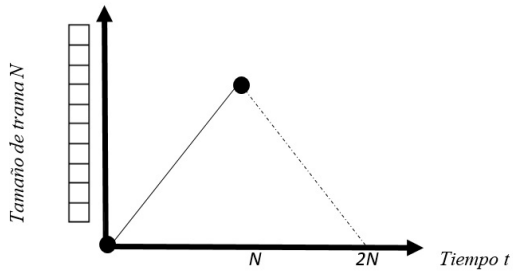

Figura 5.1: Planificador *Forward-Backward*.

Planificador *Backward-Forward*. Este planificador es el inverso del planificador anterior: la recursión hacia atrás se realiza primero, y luego las salidas suaves son producidas junto con la recursión hacia adelante. Ambos planificadores tienen el mismo retardo de decodificación y requerimientos de memoria. En este planificador las entradas suaves se leen en orden decreciente a partir del símbolo $N - 1$ hasta el símbolo 0, y las salidas suaves son producidas en orden creciente desde el símbolo 0 al $N - 1$. Se muestra en la figura 5.2.

Planificador mariposa. Con el fin de reducir el retardo de decodificación, las recursiones hacia adelante y hacia atrás pueden ser calculadas de manera concurrente. Sin embargo el costo es duplicar los recursos de hardware, pero se obtienen dos salidas suaves para dos símbolos de manera concurrente. Se muestra en la figura 5.3.

La figura 5.4 presenta el diagrama de bloques de un decodificador SISO con el planificador *backward-forward* que es el que se considerará en la implementación. La memoria SISO interna es un *buffer* que almacena los valores extrínsecos de manera temporal. La unidad de métricas de rama (UMR, abreviado) permite calcular valores de transición de métrica, que son utilizados por las unidades *Add-Compare-Select* (ACS) para ejecutar recursiones de métricas de estado. Las memorias de métricas de estado permiten almacenar los valores

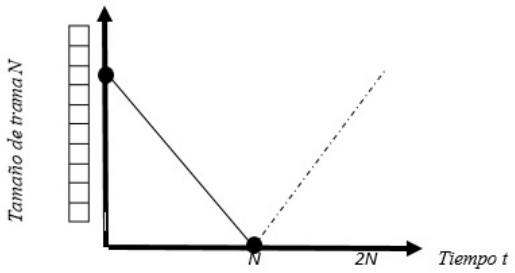

Figura 5.2: Planificador *Backward-Forward*.

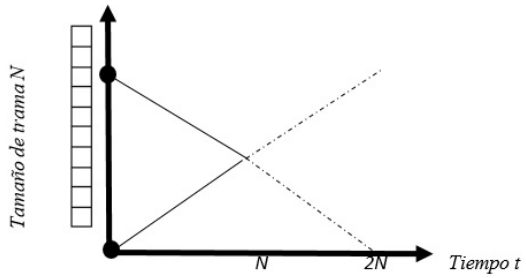

Figura 5.3: Planificador mariposa.

$A_k(s)$ y $B_k(s)$ de las ecuaciones (4.4) y (4.5). Por último, la unidad de salida suave (SOU - *Soft Output Unit*) calcula los valores extrínsecos y toma las decisiones duras. Recibe las métricas de rama y las métricas de estado calculadas previamente en las unidades ACS. Los bloques de la arquitectura se conectan en una estructura tipo *pipeline* para poder tener el camino crítico en la unidad ACS.

Figura 5.4: Arquitectura SISO con el planificador *Backward-Forward*.

5.2. Cuantización y aritmética de punto fijo.

Un análisis en punto fijo del turbo decodificador es necesario para la implementación en hardware dedicado. Cuando se realiza el mapeo de un algoritmo en una arquitectura, la cuantización tiene un efecto importante en el desempeño, en el área ocupada y en la disipación de potencia. De esta manera, surgen las siguientes preguntas:

- ¿Cuál es el ancho de bits de datos optimizado, que alcance un compromiso entre desempeño, velocidad y memoria requerida? Más aún, la pregunta de cuantización misma se subdivide en un problema de rango dinámico y de granularidad [46] en el sentido de que mientras mayor sea la granularidad, es mayor la precisión alcanzable; por otro lado, a menor granularidad, el mayor rango dinámico posible que puede ser representado, mientras que la precisión se mantiene menor. El problema de rango dinámico se representa por $q - f$ bits, como se verá más adelante, y el problema de granuralidad se representa mediante f bits.

- ¿Cómo una normalización apropiada puede ayudar a solucionar el problema de rango dinámico? En los capítulos anteriores, se vio cómo las métricas de estado continúan aumentando mientras sigue la recursión. Por lo tanto, es necesario adoptar un esquema de normalización para evitar la saturación *overflow*.

- ¿Cómo puede solucionarse el problema potencial de *overflow*? Existen dos enfoques principales: se direcciona completamente el problema de *overflow* (por medio de una normalización apropiada); el problema de *overflow* no se evita completamente (para todas las variables), pero en vez de ello se acomoda de tal manera que no afecte en los resultados correctos. Esto se logra utilizando aritmética en complemento a dos en una representación a punto fijo, y en la literatura esto es conocido como la técnica de normalización de módulo [48].

Para la descripción de números en punto fijo (con representación en complemento a dos), se usa la notación (q, f), donde q es el número total de bits (incluyendo el bit de signo) y f es la parte fraccional (por lo tanto, $q - f$ es la parte entera). También adoptaremos la notación del número de bits para la parte entera y la parte fraccional como $\rho_I(\cdot)$ y $\rho_F(\cdot)$, respectivamente.

El problema de la longitud de los datos involucra los bits de cuantización y un formato de expresión para los símbolos recibidos, las probabilidades *a priori*, las métricas y los valores extrínsecos. Si se consideran datos modulados con BPSK que se transmiten a través de un canal AWGN, casi cualquier símbolo recibido necesita sólo un número de un único dígito para su parte entera (sin contar el bit de signo). Si la observación y_i está más allá del rango que $\rho(y_i)$ puede representar, luego el valor de y_i cuantizado será recortado al valor más largo, o más pequeño. A pesar del error de redondeo, el desempeño de degradación puede ser despreciable con una longitud suficiente de los datos. De acuerdo con [13], las relaciones entre L_{apr}, $L(d_k|\mathbf{y})$, $A_k(s)$, $B_k(s)$ y $\Gamma_k(s)$ deben de satisfacer:

$$\rho_I(L(u_i)) \geq \rho_I(A_k(s)) = \rho_I(B_k(s)) \geq \rho_I(\Gamma_k(s', s)) \geq \rho_I(L_{apr}(u_i)) \geq \rho_I(r_i) \tag{5.1}$$

Una $\rho_F(y_i)$ mayor implica mejor exactitud de los datos de entrada. Para la parte fraccionaria, se debe de verificar:

$$\{\rho_F(L(u_i)), \rho_F(A_k(s)), \rho_F(B_k(s)), \rho_F(\Gamma_k(s', s), \rho_F(L_{apr}(u_i)))\} \leq \rho_F(r_i) \tag{5.2}$$

5.2.1. Normalización de métricas de estado.

En el algoritmo Max-Log-MAP, las métricas de estado se acumulan en el curso de la decodificación. Con la finalidad de prevenir situaciones de *overflow* aritmético durante las recursiones de las métricas de estado, y para mantener el retardo combinacional lo más pequeño posible, se usan técnicas de normalización de métricas [51]. Se conocen varios métodos para normalización de métricas de estado, que están basados en dos hechos[13]:

- Las diferencias entre todas las métricas de estado en cualquier etapa k del *trellis* están acotadas en magnitud por una cantidad fija $\Delta_{me,max}$ independientemente de la etapa actual k del *trellis*.
- Un valor común puede ser sustraido de todas las métricas de estado para cualquier etapa k del trellis, ya que la resta de un valor común no tienen ningún impacto en los resultados de la comparación de las siguientes métricas y del cálculo de la salida.

El *overflow* aritmético puede ser prevenido mediante un enfoque de re-escalamiento o utilizando aritmética módulo. Las siguientes subsecciones resumen el estado del arte y enfatizan los efectos en la complejidad de implementación.

Normalización mediante la substracción de una constante.

Un primer enfoque intuitivo sería un re-escalamiento periódico substractivo de las métricas. Después de un cierto número de etapas del *trellis*, se determina la mínima métrica de estado y se sustrae de todas las otras métricas de estado. Este esquema conlleva a la mínima longitud de palabra de métrica de estado. La principal desventaja de esta técnica de normalización es un camino crítico prolongado y por lo tanto una degradación en la máxima frecuencia de reloj. Este enfoque requiere cálculos extra para determinar los mínimos y realizar las restas.

El re-escalamiento por medio de la substracción de las mínimas métricas de estado, es más eficiente si es combinado con la saturación de las métricas para minimizar la longitud de la palabra. Si fuera posible admitir una reducción tolerable en el desempeño de la comunicación, tres o cuatro bits de la longitud de palabra de la métrica de estado pueden ser reducidos con saturación [48]. Esto es benéfico además debido a que un conjunto de las métricas de estado son almacenadas en memoria, por lo cual este re-escalamiento substractivo puede usarse antes del almacenamiento. Una implementación con este esquema se detalla en [49]. Otra variante considera la sustracción de la mínima métrica de estado [50].

Normalización de módulo.

La técnica de normalización de módulo se presentó con la finalidad de reemplazar la normalización substractiva de métricas de estado en la decodificación de Viterbi [51]. Este enfoque aprovecha dos propiedades del algoritmo de Viterbi (VA):

1. La salida del VA depende solamente en la diferencia de las métricas.

2. La diferencia entre las métricas está siempre acotada.

Si cada métrica se reemplaza mediante su representación en complemento a dos, y mientras el rango numérico de tal representación incluya las fronteras, la diferencia reducida obtenida en el complemento a dos es igual a la diferencia original de las métricas originales. De esta forma, los cálculos pueden hacerse utilizando aritmética complemento a dos, sin que esto afecte a los resultados correctos. Por lo tanto, la normalización de módulo no afecta al camino crítico como la técnica de substracción, ya que la diferencia se mantiene. Sin embargo, al menos un bit más es requerido cuando se compara con tal técnica.

Para la normalización de módulo cada métrica de estado candidata en una etapa k del trellis puede ser pensada como *corredores en una carrera* [48]. La figura 5.5 muestra un ejemplo de dos corredores, donde la aritmética en complemento a dos se obtiene al hacer curva la recta real y doblarla alrededor de un círculo de circunferencia C.

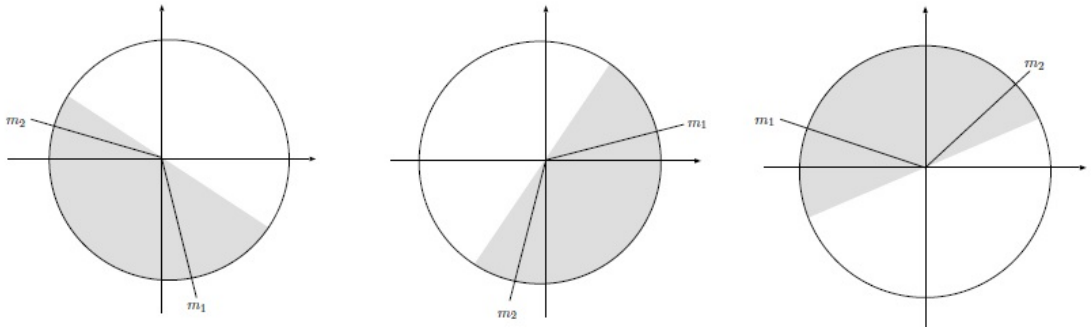

Figura 5.5: Normalización de módulo.

La restricción es que $C \geq 2\Delta$, donde C es la circunferencia y $\Delta = 2mB$ es la diferencia acotada entre las métricas de estado candidatas m_1 y m_2 (m es el número de elementos de memoria en el codificador, y B es una cota superior para los valores absolutos de la métrica de rama). La expresión $\Delta = 2mB$ ha sido probada en [51]: cada estado en un instante k tiene un camino a todos los estados $c = m + 1$ pasos después. Durante la recursión de las métricas de estado, debido a la restricción impuesta en C, la diferencia entre dos métricas de estado se preserva al moverse alrededor del círculo, es decir, las dos métricas candidatas siempre pertenecen a una mitad del círculo numérico (resaltado en gris en la figura 5.5). Sin embargo, debido a que $C \geq 2\Delta$, la normalización de módulo requiere de más bits cuando se compara con la técnica de substracción.

Aplicación de la normalización de módulo a decodificadores SISO.

Esta sección aplica la técnica de normalización de módulo a turbo decodificadores. Las siguientes ecuaciones fueron desarrolladas por [52].

Se puede demostrar que las métricas de trayectoria candidatas están acotadas superiormente como sigue:

$$\left|(A_{k-1}(s_{k-1}^1, s_k) + \Gamma_k(s_{k-1}^1, s_k) - A_{k-1}(s_{k-1}^2, s_k) + \Gamma_k(s_{k-1}^2, s_k)\right| \leq 2(m+1)B \tag{5.3}$$

donde s_{k-1}^1, s_{k-1}^2, $s_k \in S$ y B es un límite superior para los valores absolutos de las métricas de rama con signo:

$$|\Gamma_k(s_{k-1}, s_k)| \leq B \quad s_k, s_{k-1} \in S \tag{5.4}$$

La diferencia entre cualquier par de métricas de estado hacia adelante de la misma etapa de *trellis* k está también acotada superiormente, donde $m = c-1$ es el número de elementos de memoria para un código con longitud de restricción c:

$$|\alpha_k(s_1) - \alpha_k(s-2)| \leq 2mB \quad s_1, s_2 \in S \tag{5.5}$$

Por lo tanto la longitud candidata en bits de la métrica de estado puede ser calculada como:

$$b_{me} = \lceil log_2(2(m+1)B))\rceil + 1 \tag{5.6}$$

Y para la información a posteriori $L_u(u_k)$ debe de tener una longitud de bits:

$$b_{L_u(u_k)} = b_{me} + 1 \tag{5.7}$$

Valor de inicialización.

Como se vio en el capítulo anterior, los valores de inicialización para la recursión de las métricas de estado necesitan establecerse a $-\infty$. Cuando se usan b bits, el valor de inicialización resulta ser $-2^{(b-1)}$. Con la normalización de módulo otro valor aproximado de inicialización debe de ser usado: $-2^{(b-1)}$ se establece como la frontera entre valores positivos y negativos. Sin embargo, la métrica de estado puede cruzar accidentalmente la frontera de transición. Para evitar esto, es suficiente establecer el valor de inicialización a $-2^{(b-1)}/2$ [48].

5.3. Arquitectura del decodificador SISO.

En la presente sección, se detallarán las arquitecturas de los diferentes bloques del decodificador SISO, cuyo diagrama a bloques se ilustró en la figura 5.4. Para ilustrar la arquitectura, se tomará en cuenta el turbo código binario utilizado en el estándar 3GPP-LTE, detallado en capítulos anteriores.

5.3.1. Unidad de métricas de rama.

La ecuación utilizada para calcular las métricas de rama es la siguiente [11]:

$$\Gamma_k(s', s) = \frac{1}{2}u_k L(u_k) + \frac{L_c}{2}\sum_{l=1}^{n} x_{kl}y_{kl} \tag{5.8}$$

Expandiendo esta expresión, se puede ver que solamente hay cuatro valores posibles para $\Gamma_k(s', s)$ ya que $x_{kl} \in \{-1, +1\}$:

$$x_{k1} = -1, x_{k2} = -1 \quad \Rightarrow \Gamma_k^{00} = -\tfrac{1}{2}L(u_k) - \tfrac{1}{2}L_c y_{k1} - \tfrac{1}{2}L_c y_{k2}$$

$$x_{k1} = -1, x_{k2} = +1 \quad \Rightarrow \Gamma_k^{01} = -\tfrac{1}{2}L(u_k) - \tfrac{1}{2}L_c y_{k1} + \tfrac{1}{2}L_c y_{k2}$$

$$x_{k1} = +1, x_{k2} = -1 \quad \Rightarrow \Gamma_k^{10} = +\tfrac{1}{2}L(u_k) + \tfrac{1}{2}L_c y_{k1} - \tfrac{1}{2}L_c y_{k2}$$

$$x_{k1} = +1, x_{k2} = +1 \quad \Rightarrow \Gamma_k^{11} = +\tfrac{1}{2}L(u_k) + \tfrac{1}{2}L_c y_{k1} + \tfrac{1}{2}L_c y_{k2}$$

donde "00", "01", "10", "11" son $\{x_{k1}, x_{k2}\} = \{(-1,-1), (-1,+1), (+1,-1), (+1,+1)\}$, respectivamente.

Por lo tanto, la unidad de métrica de ramas (UMR) consiste de un conjunto de sumadores para calcular las métricas de rama de cada transición posible del diagrama de *trellis* del código. De esta manera, la UMR puede ser implementada con la arquitectura propuesta en la figura 5.6.

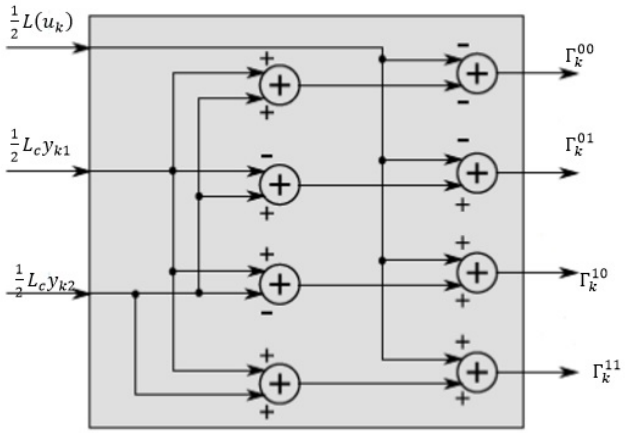

Figura 5.6: Arquitectura de UMR (unidad de métricas de rama).

5.3.2. Unidades de métricas de estado.

Las unidades de métrica de estado son unidades ACS (Add-Compare-Select por sus siglas en inglés). Como ya se vio, las métricas de estado aumentan durante cada recursión y es necesario manejar el *overflow*. En la sección anterior se vio que dos de las técnicas principales para re-escalar la información son restando la mínima métrica en cada etapa o haciendo uso de la normalización de módulo. La primer forma de re-escalamiento no afecta el desempeño de la decodificación. Sin embargo, se implementa dentro de la unidad ACS, y por lo tanto, tendría un impacto dentro del camino crítico de la unidad ACS [22]. Realmente,

la unidad ACS contiene el camino crítico del turbo decodificador, y por lo tanto esta forma de re-escalamiento substractivo es inconveniente para lograr valores de alto desempeño, en aspectos como la latencia (ya que se tomarían dos ciclos de reloj para calcular una cierta métrica de estado).

El método tradicional de acortar los retardos de trayectoria consiste en usar la técnica de *pipeline*. Sin embargo, esta es solo efectiva para el circuito que calcula el valor de verosimilitud o de salida suave. Mientras se insertan registros adicionales dentro de los circuitos de métricas hacia adelante, o hacia atrás, las otras unidades funcionales estarán estancadas durante algunos ciclos, debido a la dependencia de datos del cálculo recursivo de las métricas.

Otra forma de manejar el desbordamiento es el uso de la técnica de normalización de módulo. Se ha probado que tal técnica es efectiva para reducir el camino crítico de la unidad ACS [53]. De hecho con esta técnica, sería posible diseñar decodificadores de alto desempeño con un mayor número de estados. La implementación correspondiente se presenta en la figura 5.7, donde se logra la renormalización con un *overflow* controlado en la trayectoria de datos y requiere solamente una compuerta XOR adicional de tres entradas en cada unidad ACS. De esta manera, se logra una mejora de velocidad en 50 % y una reducción en el área, comparado a las técnicas tradicionales de re-escalamiento substractivo.

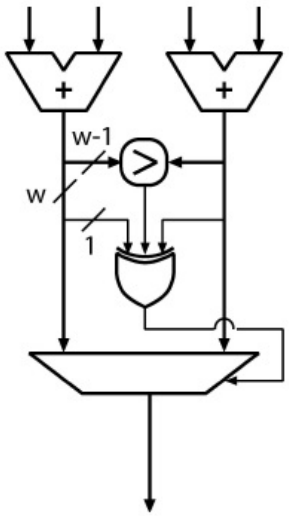

Figura 5.7: Implementación de la unidad ACS en la recursión de métricas de estado.

5.3.3. Unidad de salida suave (SOU).

Esta unidad es la encargada de calcular la información extrínseca y de obtener la decisión dura de manera paralela. Está conformada por una estructura de árbol comparativo, como se presenta en la figura 5.8. La suma de las métricas de estado (tanto hacia adelante, como hacia atrás) con las métricas de rama es calculada para cada una de las transiciones en el diagrama *trellis*. Enseguida, se calculan los valores máximos correspondientes a cada uno de

los símbolos de información $d_k = 0(LLR_+)$ y $d_k = 1(LLR_-)$.

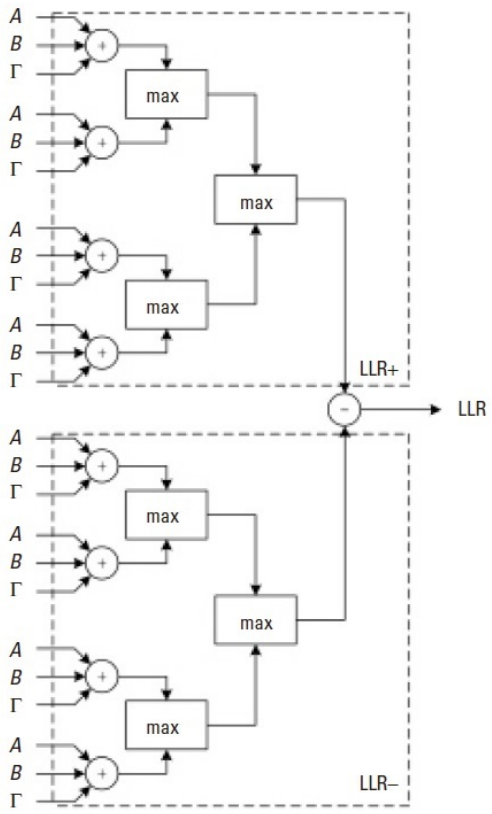

Figura 5.8: Implementación de la unidad de salida suave.

Debido a que se ha aplicado la técnica de normalización de módulo, las unidades que obtienen los valores máximos pueden ser implementadas con bloques CS (Compare-Select). Estos bloques se conforman por lo mostrado en la figura 5.7, a excepción de los sumadores. Una vez que se han determinado LLR_+ y LLR_-, la información a-posteriori se obtiene mediante su diferencia. Sin embargo, debido al uso de la normalización de módulo y dependiendo de en qué cuadrante quedaron los resultados, la resta pudiera conducir a un resultado incorrecto. Por lo tanto, antes de la resta sería necesario aplicar una rotación para desplazar los dos valores a dos cuadrantes adyacentes (esto no se muestra en el diagrama). Esta operación se realiza sumando 01 a los dos bits más significativos de LLR_+ y LLR_-. Ya aplicada la rotación, se calculan tanto la información a-posteriori como la decisión dura.

6

Prototipo de implementación y resultados.

6.1. Generalidades del hardware y del lenguaje de descripción de hardware.

En esta sección se pretende dar un panorama general de los FPGA, y del lenguaje utilizado en el que es programado: VHDL (*Very High-Level Design Language*). No se pretende profundizar mucho en sus detalles, si no más bien en dar una visión general, así como mencionar elementos muy particulares del lenguaje, que permitan el desarrollo e implementación de un turbo decodificador.

6.1.1. FPGA's.

Un FPGA es un circuito integrado diseñado para ser configurado después de su fabricación. Los dispositivos FPGA se basan en arreglos de compuertas, las cuales consisten en la parte de arquitectura que contiene tres elementos configurables: bloques lógicos configurables (CLB), bloques de entrada y de salida (IOB) y canales de comunicación. Cada FPGA contiene una matriz de bloques lógicos idénticos, por lo general de forma cuadrada, conectados por medio de líneas metálicas que corren vertical y horizontalmente entre cada bloque, como en la figura 6.1. La lógicca de un FPGA se implanta en una matriz de bloques lógicos programables llamados CLBs (Configurable Logic Blocks). Las líneas de entrada y salida de tal matriz se controlan mediante los bloques de entrada/salida (IOBs) que se colocan en los bordes de la matriz [60].

En la figura 6.2 [58], se puede observar la arquitectura de un FPGA XC4000 de Xilinx. Este circuito muestra a detalle la configuración interna de cada uno de los componentes principales que conforman este dispositivo. Los bloques lógicos, o celdas generadoras de funciones, están configurados para procesar cualquier aplicación lógica. Tales bloques son

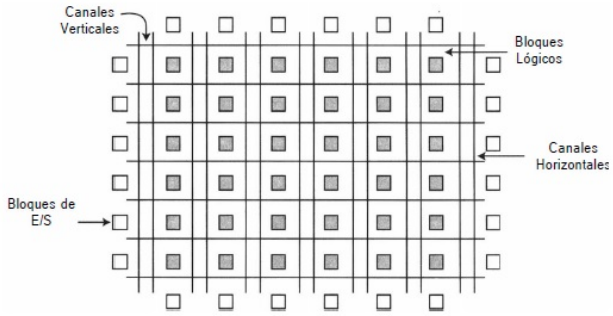

Figura 6.1: Arquitectura básica de un FPGA [58].

funcionalmente completos, lo cual implica que pueden implementar cualquier función booleana representada en la forma de suma de productos. El diseño lógico se implementa mediante bloques llamados generadores de funciones o LUT (Look Up Table: tabla de búsqueda), los cuales pueden almacenar la lógica requerida, ya que cuentan con una pequeña memoria interna. Al aplicar alguna combinación en las entradas de la LUT, el circuito la traduce en una dirección de memoria y envía fuera del bloque el dato almacenado en dicha dirección. En la figura 6.3 se observan tres LUT, etiquetados con las letras G,F y H.

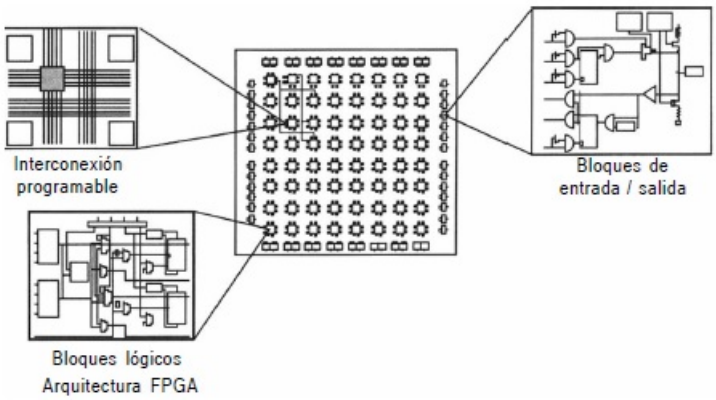

Figura 6.2: Arquitectura del FPGA XC4000 de Xilinx.

6.1.2. Lenguajes de descripción de hardware.

Debido a la necesidad de integrar un mayor número de componentes en un mismo circuito, surgió la necesidad de tener herramientas de apoyo en el diseño de circuitos digitales de alta complejidad que normalmente requerirían una mayor inversión. En los años 80 surgieron lenguajes como Verilog y VHDL, referidos como lenguajes descriptores de hardware (HDL's, por sus siglas en inglés), los cuales permiten describir hardware usando un lenguaje.

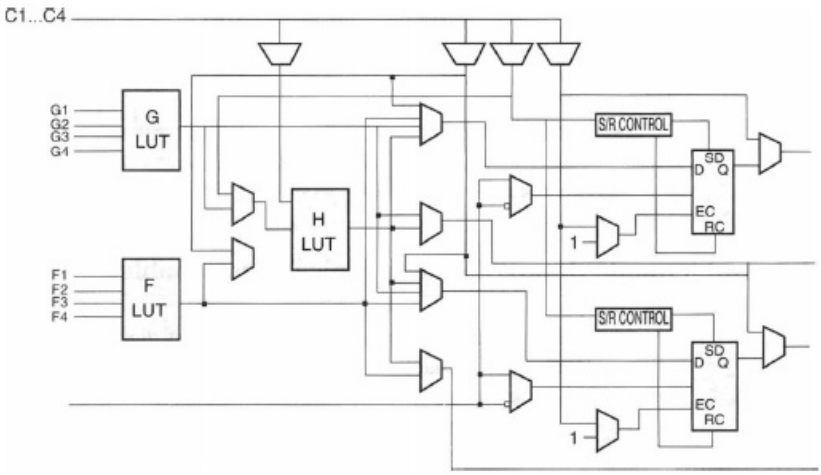

Figura 6.3: Arquitectura de un bloque lógico configurable FPGA [58].

Un HDL permite interpretar un circuito digital como un conjunto de declaraciones que se pueden describir mediante diferentes niveles de abstracción, tales como: por comportamiento, o por estructura. VHDL es un lenguaje de alto nivel de abstracción que permite el uso de un código estructurado mediante palabras reservadas y sentencias de control.

En particular, el lenguaje VHDL permite que los diseños se puedan descomponer de forma jerárquica, de tal forma que cada elemento de diseño tenga una interconexión bien definida con otros elementos, además de una especificación precisa de su comportamiento. Estas pueden usar un algoritmo, o estructura de *hardware*, y con ello se define la operación que realiza cada elemento. Es posible modelar la concurrencia inherente al hardware.

VHDL puede manejar estructuras de circuitos secuenciales síncronos y asíncronos. Además, es posible simular la operación lógica y el comportamiento en el tiempo de los diseños. VHDL ha permitido un gran avance en el área de diseño electrónico ya que facilita una inversión mínima de tiempo y recursos, obteniendo buenos resultados.

Generalidades del lenguaje VHDL.

La *entidad* es el bloque básico de diseño en VHDL. Es una abstracción de un diseño, pudiendo representar desde una única compuerta lógica hasta un sistema completo. Una entidad en VHDL está compuesta por un par de componentes llamados la declaración de la entidad y cuerpo de la arquitectura [54]. La declaración de una entidad describe las entradas y salidas del diseño de una entidad, a menudo referidas como puertos. Los puertos declarados deben tener un nombre, modo y tipo de dato. La sintaxis general es la siguiente:

```
ENTITY nombre_de_entidad IS
PORT (nombre_puerto_1: modo tipo_de_dato;
...
```

```
nombre_puerto_N: modo tipo_de_dato);
END nombre_de_entidad
```

El modo del puerto puede ser IN, OUT, INOUT o BUFFER; donde el modo IN y OUT son pines unidireccionales, mientras que INOUT es pin bidireccional. El modo BUFFER se aplica para señales que son de salida con realimentación a la misma entidad. El tipo de dato puede ser BIT, STD_LOGIC, INTEGER, STD_LOGIC_VECTOR, etc. Para mayor información en referencia a ellos, conviene consultar textos como [55].

Una arquitectura es la estructura que define el funcionamiento de una entidad, de modo que describa los procedimientos que la entidad ejecutará. Dentro de la arquitectura se destacan dos partes: la primera, la parte declarativa, la cual es opcional, es donde se declaran señales y constantes, y la segunda, que es donde se escribe el código (después del BEGIN). La sintaxis general es:

```
ARCHITECTURE nombre_de_arquitectura OF nombre_de_entidad IS
[declaraciones]
BEGIN
(instrucciones)
END nombre_de_arquitectura
```

La ventaja del lenguaje VHDL es que permite varias formas de describir el funcionamiento de una entidad, teniéndose así tres diferentes estilos de programación:

- *Descripción funcional.* Es una forma similar a la descripción de un lenguaje de alto nivel. Se usan las instrucciones típicas en los lenguajes de programación de software, tales como: if-then-else, case-when, for-loop, wait.
- *Descripción por flujo de datos.* Indica la forma en que los datos se pueden transferir de una señal a otra sin el uso de sentencias secuenciales, permitiendo así definir el flujo que tomarán los datos entre módulos encargados de realizar operaciones.
- *Descripción estructural.* En ella se declaran los componentes que se usan, se crean dichos módulos y después, mediante los nombres de los nodos, se realizan las conexiones entre los módulos de los componentes. Esta forma es muy útil al tratar con diseños jerárquicos bien modularizados.

Durante la implementación del decodificador, resulta necesario guardar información que deberá ser utilizada en algún otro punto del proceso de la decodificación. De esta manera, es importante explicar los componentes de memoria, que se encuentran embebidos dentro de las plataformas FPGA.

6.1.3. Manejo de memoria RAM en un FPGA.

Un sistema digital frecuentemente requiere memoria para almacenamiento. Para facilitar esta necesidad, la mayoría de los dispositivos FPGA contienen módulos de memoria

embebidos [56]. Aunque estos módulos no pueden reemplazar los dispositivos masivos de memoria externa, son útiles para aplicaciones que requieren memoria de tamaño pequeño a intermedio.

Para usar la descripción de HDL por comportamiento, para inferir módulos de memoria, los "templates" que proponen fabricantes como Xilinx, deben seguirse cuidadosamente. A continuación, se presentan "templates" que son iguales a los originales, excepto en que se utilizan genéricos para la longitud de los bits de dirección, y la longitud de los bits de datos. Es una buena práctica confinar la descripción de memoria en un módulo HDL separado para que el módulo pueda ser fácilmente identificado y reemplazado cuando se requiera.

RAM de puerto único.

En una memoria embebida, la operación de escritura es siempre síncrona. Durante el flanco de subida del reloj, la dirección, los datos de entrada, y señales de control relevantes (como we-"write enable"), son muestreadas. Si *we* está activa, se realiza una operación de escritura (es decir, los datos de entrada se almacenan en la ubicación de memoria indicada por la señal de dirección).

La operación de lectura puede ser síncrona o asíncrona. Para la lectura asíncrona, la señal de dirección se usa directamente para acceder al arreglo RAM. Después de que la señal de dirección cambia, los datos se encuentran disponibles después de un retardo corto. Para la lectura síncrona, la señal de dirección se muestrea en el flanco de subida del reloj y se almacena en un registro. La dirección registrada se usa luego para acceder el arreglo RAM. Debido al registro, la disponibilidad de los datos es retardada y sincronizada por la señal de reloj.

Un "template" de RAM de puerto único con lectura síncrona se presenta a continuación.

```
library ieee;
use ieee.std-logic-1164.all ;
 use ieee. numeric-std.all ;

entity one-port-ram-sync is
generic (
ADDR_WIDTH :  integer := 12;
DATA_WIDTH : integer := 8;
);

port (
clk: in std_logic;
we: in std_logic;
addr: in std_logic_vector (ADDR-WIDTH-1 downto 0 ) ;
dout : out std_logic_vector (DATA_WIDTH -1 downto 0 )
din: in std_logic_vector (DATA_WIDTH-1 downto 0) ;
) ;
end one-port-ram-sync;

architecture beh_arch of one-port-ram-sync is
```

```
type ram-type is array (2**ADDR_WIDTH -1 downto 0 ) of
std-logic-vector (DATA_WIDTH-1 downto 0 ) ;
signal ram: ram_type;
signal addr_reg: std_logic_vector (ADDR_WIDTH-1 downto 0 ) ;

begin
process (clk)
 begin

if (clk'event and clk = '1') then
if (we='l') then
ram(to-integer(unsigned(addr))) <= din;
end if ;
addr_reg <= addr;
end if ;
end process ;

dout <= ram(to_integer (unsigned(addr_reg))) ;
end beh_arch ;
```

Este código contiene un tipo de datos de arreglo bi-dimensional para almacenamiento y emplea indexación dinámica para acceder al elemento en el arreglo.

RAM de puerto dual.

Una RAM de puerto dual incluye un segundo puerto para acceso a memoria. De forma ideal, el segundo puerto debe de poder hacer de forma independiente una operación de lectura y escritura y tener su propio conjunto de direcciones, datos de entrada y salida, así como señales de control. Un "template" con lectura síncrona se ejemplifica a continuación:

```
library ieee;
use ieee.std-logic-1164.all ;
use ieee.numeric-std.all ;
entity dual-port-ram-sync is
generic (
ADDR_WIDTH : integer := 6 ;
DATA_WIDTH: integer:= 8
) ;

port (
clk: in std_logic;
we: in std_logic;
addr_a: in std_logic_vector (ADDR_WIDTH-1 downto 0) ;
addr_b: in std_logic-vector (ADDR_WIDTH-1 downto 0) ;
din-a: in std-logic-vector (DATA_WIDTH-1 downto 0 ) ;
```

```
dout-a: out std-logic-vector (DATA_WIDTH-1 downto 0 ) ;
dout-b : out std-logic-vector (DATA_WIDTH -1 downto 0 )
) ;
end dual-port-ram-sync;

architecture beh-arch of xilinx-dual-port-ram-sync is
type ram_type is array ( 0 to 2**ADDR_WIDTH-l) of
std-logic-vector (DATA-WIDTH-1 downto 0) ;
signal ram: ram_type;
signal addr_a_reg , addr_b_reg :std_logic_vector (ADDR_WIDTH -1 downto 0 ) ;

begin
process (clk)
begin

if (clk'event and clk = ' 1 ' ) then
if (we = '1') then
end if ;

addr_a_reg <= addr_a;
addr_b_reg <= addr_b ;
ram(to_integer(unsigned(addr_a))) <= din_a;
end if ;
end process ;

dout_a <= ram(to_integer(unsigned(addr_a_reg));
dout_b <= ram(to_integer(unsigned(addr_b_reg));
end beh_arch;
```

6.2. Controlador de un decodificador SISO.

El módulo decodificador SISO está compuesto de dos partes: una sección de control y una sección de procesamiento de la información. La sección de control permite sincronizar los componentes mientras que la sección de procesamiento se aplica a la realización de operaciones aritméticas. Los módulos que componen a la sección de procesamiento, fueron descritos en el capítulo anterior: unidad de métricas de rama, unidades de métricas de estado y la unidad de salida suave. Otro componente del módulo SISO es la memoria que almacena las métricas de estado hacia atrás, la cuál es descrita en *hardware* de acuerdo a lo estipulado en el apartado anterior. A continuación haremos énfasis en el módulo controlador.

El controlador de un decodificador SISO es modelado por medio de una máquina de estados finitos. Se manejan un total de cinco estados, y para cada estado se encuentran asociadas señales de control. El diagrama de estados se presenta en la figura 6.4.

El controlador es inicializado al estado "Idle". El decodificador SISO espera a la llegada de los datos entrantes; una vez que los datos se encuentran disponibles se activa una señal "s0",

y el controlador pasa al estado siguiente: "init_backward". Durante este estado, todas las métricas de estado $B_k(s)$ son inicializadas en cero para cada uno de los estados del "trellis" (start_backward = '1'). Estas primeras métricas de estado hacia atrás son almacenadas en la memoria de métricas de estado, por lo que $W/R = W$. Dado que dentro de este estado las métricas iniciales se escriben en la primera dirección de memoria, la señal Writeadd_SM permanece en un valor inicial de cero.

Durante el siguiente ciclo, se llega al estado "backward". Durante ese estado, la señal start_backward se desactiva ya que se está ejecutando de manera normal (sin condiciones de inicialización), el cálculo recursivo de las métricas de estado. Además, la señal de direccionamiento Writeadd_SM que está en modo "buffer", es incrementada en 1, con el fin de que las métricas puedan ser escritas en cada una de las posiciones de memoria. Un contador (count_0) se encarga de realizar el conteo del número de etapas recorridas a través del *trellis*, cuando llega al conteo máximo, la máquina de estados pasa al estado "init_forward "para recomenzar la decodificación en el sentido "hacia adelante". Además, durante los estados "init_backwardz backward"se activa una señal read_data_bwd, la cual se encarga de leer los datos necesarios (ya sea en el orden natural, o en el orden dado por el entrelazador) necesarios para realizar todas las operaciones. Los datos que se lean dependerán de la iteración que se vaya a realizar y por la señal dada por un controlador de lectura externo (explicado en una sección posterior).

Durante el estado "init_ forward", las métricas de estado $A_k(s)$ son inicializadas al valor 0 para el estado 0 del *trellis* y a un valor mínimo para los otros estados del *trellis*. De la misma manera que para la decodificación en el sentido "hacia atrás", la señal "init_ forward.^{es} desactivada durante el siguiente ciclo. Un contador (count_1) realiza el conteo del número de etapas recorridas en el sentido "hacia adelante", durante el estado "Forward". Las métricas de estado $B_k(s)$ que habían sido almacenadas en una memoria son leídas para calcular la salida suave. En tal caso, la señal $W/R = R$. Se requiere que la señal Readadd_SM (que se encuentra en modo "buffer") se decremente en 1 durante cada ciclo perteneciente a este estado, ya que la primera dirección de lectura será la última posición de memoria de las métricas hacia atrás (en otras palabras, la primera etapa del "trellis") debido a que se considera el uso del planificador *backward-forward* dentro de la implementación. Antes de este estado, la señal Readadd_SM debe estar puesta al valor máximo, que corresponde a la primera dirección de memoria a ser leída. Además, durante los estados "init_forwardz "forward"se activa una señal read_data_fwd, la cual se encarga de leer los datos necesarios (ya sea en el orden natural o en el orden entrelazado) necesarios para realizar todas las operaciones. Al mismo tiempo de las operaciones "hacia adelante", se realiza el cálculo de los valores de verosimilitud, por lo cual la unidad SOU (*Soft Output Unit*) presenta a su salida la decisión dura correspondiente.

Una vez que el contador count_1 ha llegado a su límite, es decir, cuando se ha terminado una iteración completa, la máquina de estados regresa al estado inicial "Idle", esperando a que llegue un nuevo bloque de datos o a proseguir la ejecución de las operaciones de la siguiente iteración.

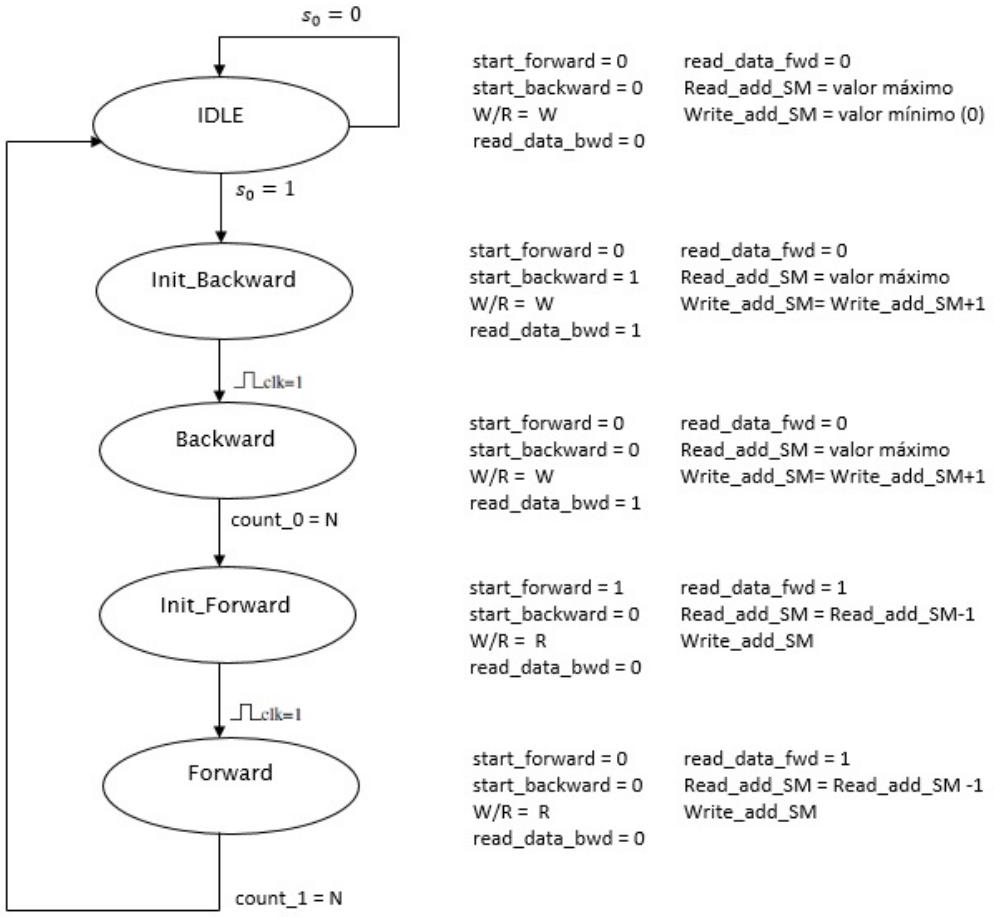

Figura 6.4: Máquina de estados que modela el controlador del decodificador SISO.

6.3. Gestión de memorias de datos.

Durante la recepción de los datos demodulados, los mensajes recibidos son primeramente almacenados en memorias de datos, que serán representadas como $\mathbb{M}_X$, $\mathbb{M}_{Y_1}$ y $\mathbb{M}_{Y_2}$, las cuales alojan a los elementos recibidos X, Y_1 y Y_2 (información sistemática y de paridad de cada codificador). Se considerará de manera adicional, una memoria $\mathbb{M}_{LLR}$ la cual almacenará los valores de información extrínseca. Con respecto al entrelazador, se considerará el uso de un entrelazador QPP, cuya arquitectura se muestra en la figura 4.7. Como ya se vio, este entrelazador es capaz de generar durante ciclo reloj la dirección que se requiere para acceder a los datos de las memorias correspondientes, sin tener que almacenar dichas direcciones en tablas *look-up* de memoria, lo cual implicaría la ocupación de una mayor área del decodificador. Es importante tomar en cuenta la gestión de las memorias tanto para lectura, como escritura. El tamaño de la memoria depende directamente de la longitud de la trama

a memorizar. Cada escritura (o lectura) incrementa (o decrementa) la dirección actual en una unidad.

Durante el transcurso de la decodificación, las memorias $\mathbb{M}_X$, $\mathbb{M}_{Y_1}$, $\mathbb{M}_{Y_2}$ y $\mathbb{M}_{LLR}$ proporcionan los datos necesarios al módulo decodificador SISO. El módulo SISO es controlado por la señal *Active*, que a su vez, es entregada por el controlador de lectura. Cuando la señal *Active* es igual a 1, el decodificador SISO se encuentra en el estado de ejecución, es decir, se está realizando la decodificación. De otra manera (*Active* =0), el decodificador no opera sobre ningún dato. En la práctica, es posible implementar un único módulo decodificador SISO que trabaja con los datos tanto en el orden natural como en el orden entrelazado.

A continuación se describe cómo es la lectura de los datos en las memorias. En el orden natural, la señal P es conectada a la configuración Π^{-1} (usando el desentrelazador o entrelazador inverso). La dirección de lectura A_r es enviada a las memorias $\mathbb{M}_X$, $\mathbb{M}_{LLR}$, $\mathbb{M}_{Y_1}$. Debido al multiplexor, que es también controlado por la señal P, el dato Y_s a la entrada del decodificador SISO recibe la salida Y_{1s} de la memoria $\mathbb{M}_{Y_1}$. En el orden entrelazado, la señal P es conectada a la configuración Π (usando el entrelazador). En dicho caso, las memorias $\mathbb{M}_X$, $\mathbb{M}_{LLR}$ reciben una dirección de entrelazado generada por el módulo entrelazador Π a partir de la dirección A_r. Finalmente, la salida Y_{2s} de la memoria $\mathbb{M}_{Y2}$ es seleccionada para la entrada Y_s del módulo decodificador SISO.

Por su parte, el decodificador SISO, produce a la vez la información extrínseca y la decisión dura $\hat{d}$ sobre el bit considerado. A nivel de la turbo decodificación, la información extrínseca resultante del decodificador es en un principio multiplicada por un factor de escala (por ejemplo, $\zeta = 0.75$) definido en el tercer capítulo y posteriormente saturado (descartando los dos bits decimales menos significativos). Posteriormente, es almacenada en la memoria $\mathbb{M}_{LLR}$. Cuando todas las iteraciones de la turbo decodificación han finalizado, el controlador activa la señal "Disponible" indicando que la información extrínseca se encuentra disponible. De manera paralela, se produce la decisión dura correspondiente $\hat{d}$.

6.4. Implementación del emulador del canal AWGN.

Después de la codificación de una trama, los símbolos codificados son transmitidos a través de un canal de transmisión con ruido aditivo, blanco, gaussiano (AWGN). Por lo cual, resulta necesario el implementar el emulador de tal canal en el FPGA. La figura 6.6. presenta la interfaz del emulador del canal AWGN implementado. La varianza del ruido σ_n es precalculada de acuerdo a la potencia deseada, es decir, la relación señal-a-ruido es almacenada en una memoria $\mathbb{M}_\sigma$. Su tamaño es igual a $2^6 \times 17$ bits. Cada palabra de la memoria es codificada con 17 bits, de los cuales 16 bits se dedican a la parte fraccionaria. La dirección de la memoria A_σ requiere de 6 bits, teniendo así 2^6 direcciones disponibles. La dirección 0 está reservada a una $SNR = \infty$, y las demás corresponden a las razones señal a ruido $\frac{E_b}{N_0}$ que va de 0.25 dB a 15.75 dB con un tamaño de paso de 0.25 dB. Los valores ruidosos X, Y_1 y Y_2 cuantificados a 5 bits, son obtenidos a partir de las entradas d, y_1 y y_2, obtenidas de la parte emisora. Para la realización de pruebas no se considerará al codificador, ya que se supondrá la transmisión de una trama que consiste en únicamente símbolos nulos '0', por lo cual la salida del turbo codificador en cada uno de los casos será igual a 0.

Durante el curso de la transmisión, el emulador de canal añade ruido AWGN a los símbolos codificados. El emulador de canal gaussiano utilizado es descrito con mayor detalle

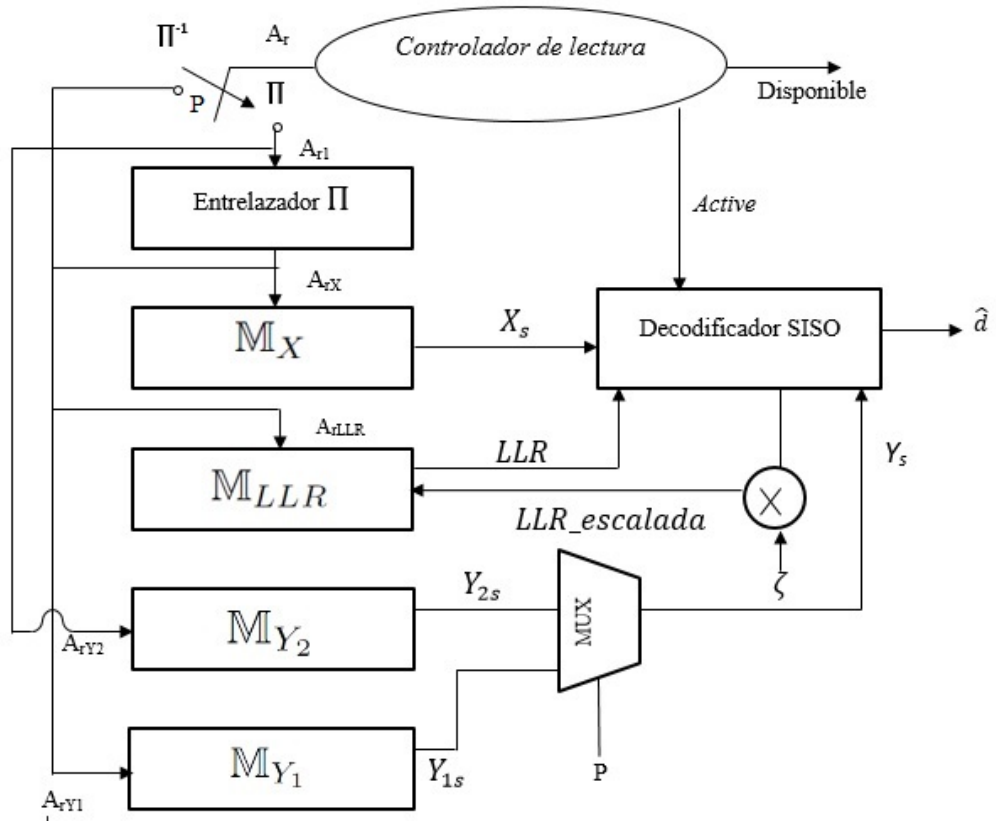

Figura 6.5: Lectura de las memorias durante una iteración en la decodificación.

en [57]. Se considera el ruido como una variable aleatoria V con una varianza σ y media cero. Es decir, una distribución $N(0, \sigma)$. En los software de simulación, el método de Box-Muller es comúnmente empleado para generar una variable aleatoria gaussiana con una distribución $N(0, 1)$. Este método consiste de dos etapas: en la primera se generan dos variables aleatorias independientes v_1 y v_2 que están distribuidas de manera uniforme en el intervalo [0,1]. En seguida, se generan las funciones $f(v_1)$ y $g(v_2)$ definidas como:

$$f(v_1) = \sqrt{-ln(v_1)} \tag{6.1}$$

$$g(v_2) = \sqrt{2}cos(2\pi v_2) \tag{6.2}$$

El producto de las funciones genera una muestra v de la variable aleatoria V de la distribución gaussiana $N(0, 1)$:

$$v = f(v_1)g(v_2) \tag{6.3}$$

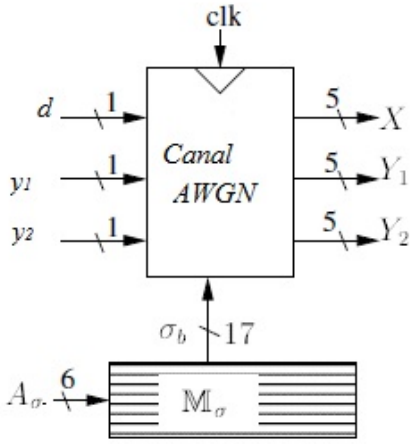

Figura 6.6: Interfaz de un canal AWGN.

Es difícil implementar las funciones $ln(v_1)$ y $cos(2\pi v_2)$ en un procesador de punto flotante de 32 bits (ya que se requiere de mucho *hardware*). Más aún, es más complejo de implementar tales funciones en una plataforma de hardware, como un FPGA. Por lo cual, una técnica consiste en pre-calcular, y después memorizar los valores $f(v_1)$ y $g(v_2)$ a partir de valores conocidos v_1 y v_2.

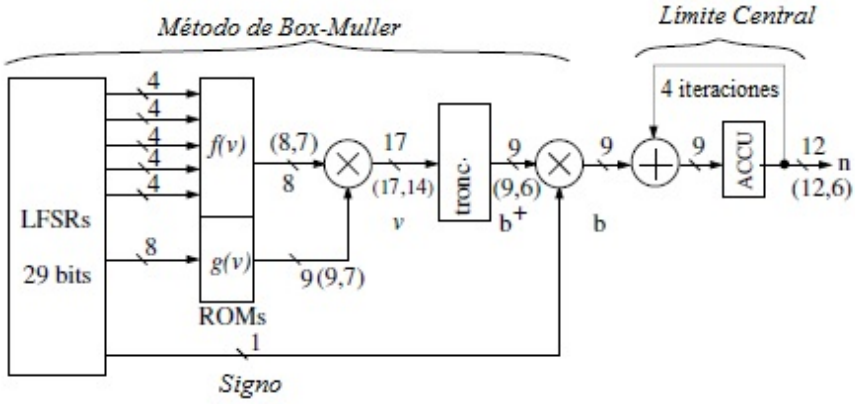

Figura 6.7: Arquitectura del generador de ruido AWGN.

La figura 6.7. muestra la arquitectura general del generador de ruido AWGN que sigue una distribución normal $N(0,1)$. Esta arquitectura está basada en el método de Box-Muller y en el Teorema de límite central. A partir de LFSRs (registros de corrimiento de realimentación lineal) de 29 bits, una dirección de 20 bits y una dirección de 8 bits son transmitidas a memorias ROM, en donde los valores $f(v_1)$ y $g(v_2)$ se encuentran almacenados. El último

bit, denotado como *signo*, representa el signo de la muestra producida. Se usa la notación (m, n) que indica que un valor está cuantizado a m bits de los cuales n bits constituyen la parte fraccionaria. Los valores $f(v_1)$ y $g(v_2)$ se cuantizan respectivamente, a 8 y 9 bits con los formatos (8,7) y (9,7). En seguida, el valor v cuantizado con (17,14) bits es truncado a un valor de (9,6) bits. Por lo tanto, se obtiene una muestra b a partir de la siguiente relación:

$$b = (1 - 2 \cdot signo) \times b^{+} \tag{6.4}$$

donde b^{+} es un valor positivo.

La multiplicación por la señal *signo* permite obtener una densidad de probabilidad centrada en 0 empleando una representación en complemento a 2. En ese caso, la probabilidad del valor 0 es dos veces más grande que la de una distribución normal, ya que el complemento a 2 del valor 0 es también igual a 0. Para resolver esa limitación, se aplica el teorema del límite central.

Si V es una variable aleatoria caracterizada por su valor esperado m_v y su varianza σ_v, una variable aleatoria V_N puede ser definida a partir de la siguiente expresión:

$$V_N = \frac{1}{\sigma_v \sqrt{N}} \sum_{i=0}^{N-1} (v_i - m_v) \tag{6.5}$$

donde v_i con $i = 0, \ldots, N-1$ son N realizaciones de la variable aleatoria V. El Teorema del límite central establece que sí N tiende a infinito, la variable aleatoria sigue una distribución normal $N(0, 1)$. En el presente caso, y de acuerdo a Boutillon [57], es suficiente acumular las muestras b durante 4 iteraciones para así obtener la realización n (ver figura 6.7).

Para obtener el canal AWGN, es necesario ahora multiplicar la varianza del ruido σ_b (según la potencia deseada) por la variable aleatoria que posee una distribución normal $N(0, 1)$. Después, el ruido es agregado a los mensajes codificados. El valor resultante será luego truncado antes de entrar al decodificador SISO. En el presente caso, los elementos ruidosos X, Y_1 y Y_2 son saturados y cuantizados a 5 bits según el formato (5,3).

6.5. Implementación de un sistema de transmisión y de recepción.

Debido a que en nuestras pruebas se considera la transmisión de una trama que consiste en únicamente símbolos nulos '0', la salida del turbo codificador consiste en únicamente en símbolos 0. Después del proceso de codificación, los símbolos deben ser transmitidos a través de un canal con ruido. En el presente caso, ese canal es simulado por el emulador de canal AWGN descrito en la sección anterior, y que es implementado en el FPGA. La interfaz es la que se presentó en la figura 6.6. Una vez que se le ha agregado el ruido a los datos, y estos ya se encuentran cuantizados con 5 bits, en la recepción estos datos deben ser almacenados con el objetivo de realizar el proceso de decodificación.

En el receptor, la arquitectura del turbo decodificador se conecta a diferentes elementos: las memorias de datos $\mathbb{M}_d$ y a la memoria de información extrínseca $\mathbb{M}_{LLR}$. El proceso de turbo decodificación realiza el procesamiento con datos que provienen de tales memorias

y transmite la información extrínseca a la memoria $\mathbb{M}_{LLR}$ durante el curso de las iteraciones. De manera paralela, el BER (*Bit Error Rate*) es calculado y desplegado en tiempo real mediante leds de la tarjeta FPGA que realizan el conteo de errores resultantes, en este caso, corresponden a los bits que resultarán ser decodificados como '1' al finalizar la última iteración del proceso de turbo decodificación. La arquitectura global del receptor se muestra en la figura 6.8. Se considera que el turbo decodificador implementado, corresponde al empleado por el estándar 3GPP-LTE, el cual utiliza 8 estados, y cuyo codificador componente de la concatenación paralela es el que se presentó en la figura 3.4. El entrelazador y desentrelazador utilizados son del tipo QPP. Las memorias de datos recibidos X, Y_1 y Y_2 son consideradas memorias externas al decodificador SISO. De la misma manera, la memoria de información extrínseca es externa al decodificador. La memoria que almacena las métricas de estado hacia atrás es considerada un componente interno del decodificador elemental SISO.

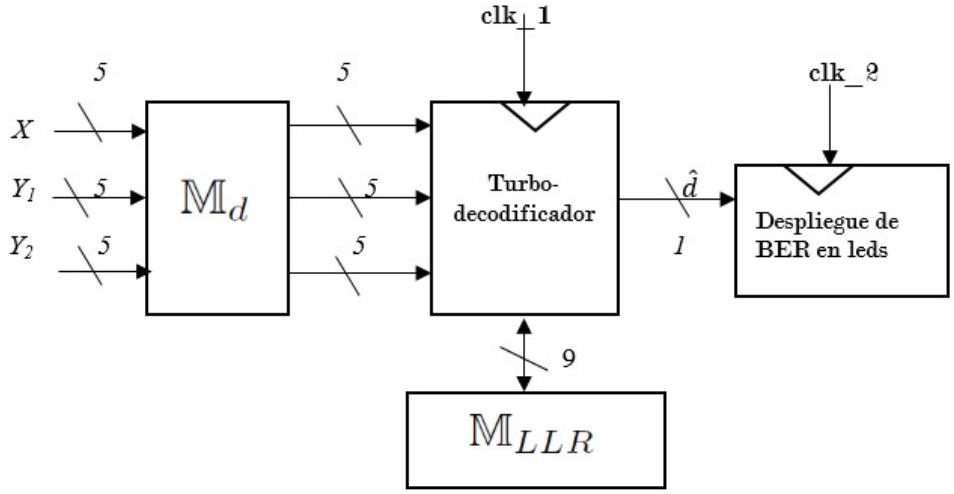

Figura 6.8: Arquitectura global del receptor.

6.6. Resultados de la implementación del turbo decodificador.

A continuación se detallan las consideraciones para la implementación en punto fijo de la arquitectura del decodificador SISO en el FPGA. Se toman en cuenta las restricciones impuestas por las ecuaciones (5.1) y (5.2), así como las restricciones que fueron descritas en la sección 5.2.

Se considerarán los siguientes parámetros:

- Modulación BPSK y un canal AWGN.

- No se toma en cuenta el perforado, es decir la tasa de transmisión es $R = 1/3$. De cualquier forma, se transmite una trama que consiste exclusivamente de ceros.
- Posibles tamaños de trama: N = 128, N = 256, N = 176 y N = 240 (estos dos últimos tamaños son aquellos donde la misma implementación del entrelazador se puede usar como desentrelazador).
- Número de iteraciones: 10 (5 por cada decodificador)
- Los datos recibidos serán cuantizados con 5 bits: 2 bits para la parte entera y 3 para la parte decimal. Es decir, un esquema de cuantización (5,3).
- Las métricas de rama serán cuantizadas con 6 bits: 3 bits para la parte entera y 3 bits para la parte decimal. Que corresponde a un esquema de cuantización (6,3).
- Las métricas de estado serán cuantizadas con 10 bits: 7 bits para la parte entera y 3 bits para la parte decimal. Un esquema de cuantización (10,3).
- Los valores de información extrínseca calculados son cuantizados con 11 bits. Sin embargo, es posible realizar una saturación con los 9 bits más significativos sin afectar el desempeño. Se tiene así un esquema de cuantización (9,1).

A continuación se presentan las curvas de desempeño en BER para diferentes escenarios. En las gráficas se muestran los puntos resultantes de la simulación, correspondientes a $\frac{E_b}{N_0}$ con valores de 0 a 2 dB, en incrementos de 0.25 dB. Dados esos puntos, se pueden extrapolar a una gráfica general de tasa de error de bit, misma que se muestra además en las gráficas.

1. Curvas BER para diferentes iteraciones (iteración 1,2,3,5,7 y 10) con una trama de longitud N = 256 (ver figura 6.9). Se puede observar como el desempeño mejora gradualmente con cada iteración.
2. Curvas BER para diferentes tamaños de entrelazador (N=128, N = 176, N = 240 y N = 256), tomando en cuenta 10 iteraciones (ver figura 6.10). La curva que se desplaza más hacia abajo es la de la trama con longitud N=256, y la que está desplazada más hacia arriba es la que corresponde a la trama con menor longitud (N=128).
3. Curvas BER para diferentes factores de escalamiento de la información extrínseca, $\zeta = 0.75$ (el más conveniente de acuerdo a la literatura), $\zeta = 0.875$ y $\zeta = 0.625$ para una trama de longitud $N = 240$ (ver figura 6.11). El desempeño más pobre corresponde al caso del algoritmo Max-Log-MAP sin ningún factor de escalamiento. De ahí, los factores de escalamiento con el siguiente orden son los que presentan gradualmente un mejor desempeño: 0.875, 0.625 y 0.75. En resumen, el mejor desempeño resulta de aplicar un factor de escalamiento de 0.75 para la información extrínseca.

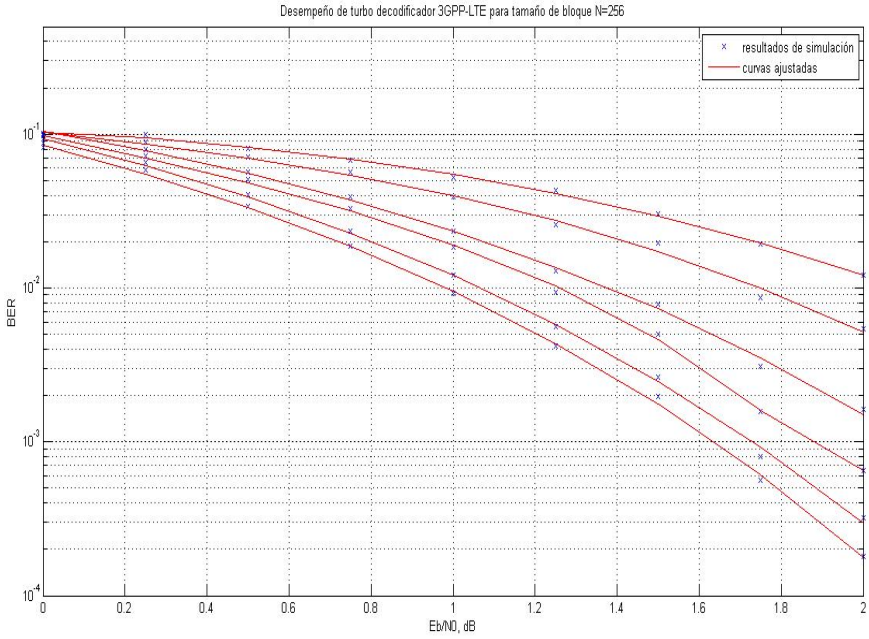

Figura 6.9: Curvas BER para diferentes iteraciones (se muestran la 1,2,3,5,7 y 10) para trama de longitud N=256. La curva que se encuentra más arriba corresponde a la de la iteración 1, y descendentemente hasta la curva de la iteración 10, que es la que se encuentra más abajo.

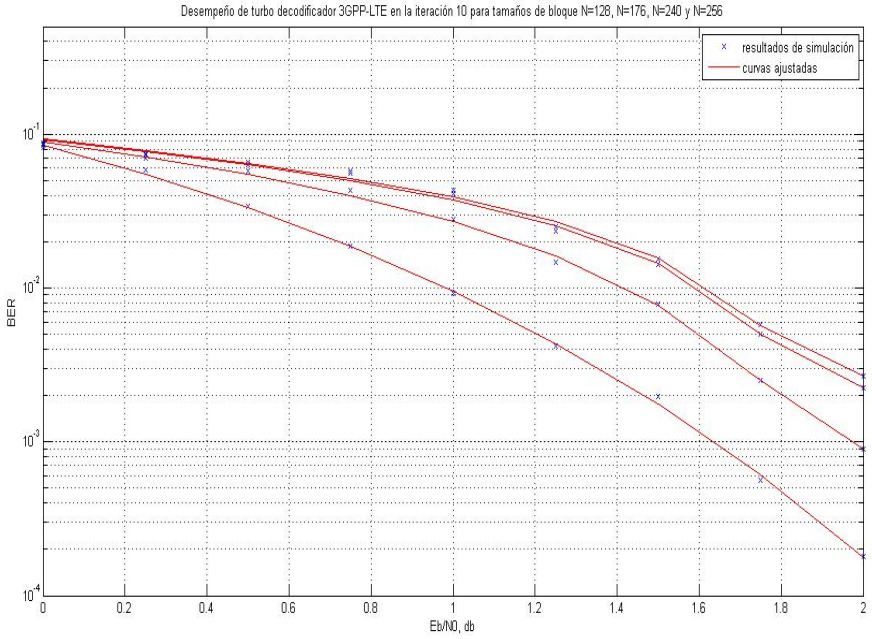

Figura 6.10: Curvas BER para la iteración 10 de tramas con diferente longitud (128, 176, 240 y 256).

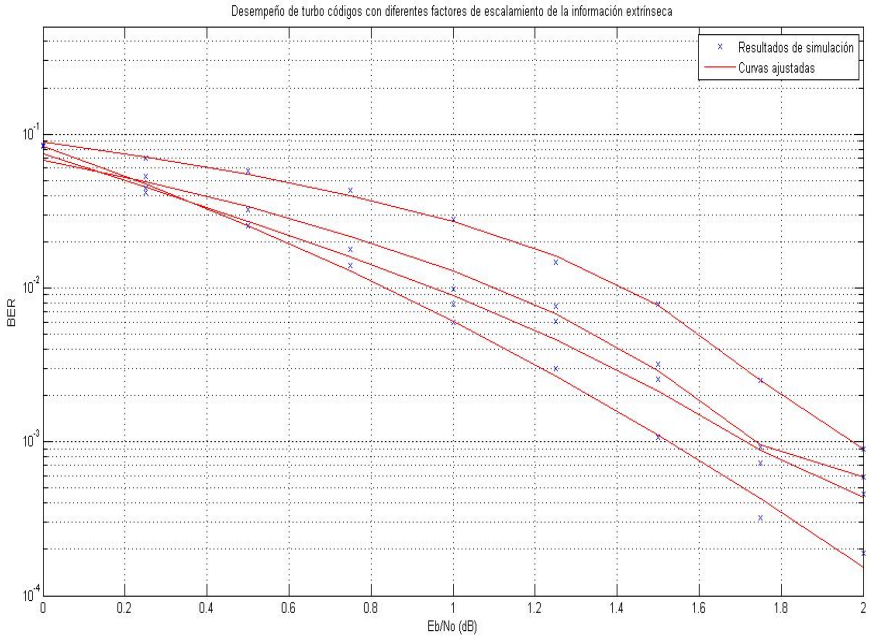

Figura 6.11: Curvas BER para diferentes factores de escalamiento, para tramas de longitud 240.

7

Conclusiones y trabajo a futuro.

7.1. Conclusiones

.

Este trabajo de tesis se enfocó en desarrollar en una plataforma FPGA el prototipo de un turbo decodificador elemental del estándar 3GPP-LTE. Se tomó como punto de referencia el diagrama general de un sistema de comunicaciones, y durante el resto del trabajo se hace énfasis en aspectos relacionadas a los módulos codificador de canal y decodificador de canal. Los turbo códigos presentan un buen desempeño en la corrección de errores, además de que se utilizan en múltiples estándares de comunicaciones digitales, de ahí su interés por realizar el trabajo de tesis en esta área.

Se presentaron los elementos más importantes de los códigos convolucionales, tanto como su construcción como sus diferentes diagramas de representación, ya que son de vital importancia para entender los elementos que componen a los turbo códigos así como sus algoritmos de decodificación. De gran importancia resulta el entrelazador QPP, ya que este permite generar las direcciones de entrelazado durante cada ciclo de reloj sin tener que almacenar los valores en una memoria. Además de que es un entrelazador que presenta buenas propiedades de distancia, lo cual se sugiere para reducir el piso de ruido. Se realizaron experimentos considerando tamaños de trama en donde la función de entrelazador es la misma que la del desentrelazador, y de esa manera, comprobar el ahorro en recursos de *hardware* en el FPGA.

El algoritmo MAP sirve como punto de referencia para realizar la decodificación, sin embargo este cuenta con una complejidad computacional elevada en términos del número de operaciones que requieren altos recursos de *hardware* así como de tiempo de ejecución. Por lo cual, una variante conocida como el algoritmo Max-Log-MAP es considerada para la implementación. Debido a la implementación en *hardware*, fue necesario realizar las con-

sideraciones para representar la información utilizada por el decodificador, en un formato de punto fijo. De ahí que resulte necesario hacer un análisis de la cuantización y de esquemas para normalizar los datos con el objetivo de solucionar el problema de rango dinámico. El enfoque considerado fue la normalización de módulo, ya que este esquema a diferencia de un esquema substractivo, no afecta el camino crítico de un decodificador, y además no incrementa de manera considerable la complejidad computacional.

Se realizó la integración de un módulo decodificador SISO, con su respectivo controlador e integración con un módulo emulador de ruido AWGN, teniendo así un sistema completo transmisor-receptor. La arquitectura fue sintetizada en un FPGA Altera DE2 Cyclone II y se puede alcanzar una tasa de reloj de 50 MHz. La eficiencia de operación está definida como:

$$\eta = \frac{\Gamma_N}{\Gamma_R} \tag{7.1}$$

En donde Γ_R es el número de ciclos de reloj utilizados para una iteración completa de la decodificación y Γ_N es el periodo en ciclos para producir los resultados de N símbolos. Se tiene así, una eficiencia de operación del 46.7 por ciento. El desempeño correspondiente en bits por segundo es 1.17 Mbits/seg.

7.2. Trabajo a futuro.

En la presente tesis, se consideró la implementación del algoritmo en su forma estándar, sin embargo el cálculo de las métricas hacia adelante requiere que se hayan obtenido previamente todas las métricas hacia atrás de la trama completa, y si el tamaño de la trama es grande, puede haber un alto retardo en la decodificación, así como una cantidad inasequible de memoria de métricas de estado. En diversas consideraciones prácticas, se han aplicado técnicas de división en ventanas "windowing", en donde la longitud de decodificación se limita a una ventana truncada de la trama total, con longitud L, en vez de hacer todo el proceso sobre la trama de longitud N. Para un trabajo a futuro, sería conveniente adoptar un esquema de *ventanas deslizantes*, y de esa manera lograr una mejora en la velocidad de decodificación.

Otra variante en la que puede ser posible realizar una mejora, implica tomar un criterio dinámico de finalización en la decodificación, es decir, aquel en el que el número de iteraciones del algoritmo de decodificación sea dependiente de las características del canal sin que esto repercuta en el desempeño de la decodificación. Esto se podría hacer fácilmente en *hardware* comparando la amplitud logarítmica de verosimilitud con un cierto umbral, donde dicho umbral se puede determinar de acuerdo a las condiciones del canal y una tasa de errores BER deseada. En ese caso, el número promedio de iteraciones se incrementará si este umbral se incrementa.

Otro de los criterios que se han adoptado es el uso de varios decodificadores SISO trabajando en paralelo para decodificar una única trama. En estas técnicas, para tener un mayor paralelismo en el algoritmo Max-Log-MAP, el *trellis* correspondiente a la trama codificada es dividido en un número de secciones de *trellis*, las cuales son procesadas independientemente y simultáneamente por varias unidades de recursión y de salida suave. Esto es posible debido al principio de ventana deslizante. Otra variante de paralelismo puede ser aplicada a nivel

del *trellis*, en donde las unidades funcionales dentro del decodificador SISO se duplicarían para completar los cálculos asociados a dos, o más etapas del *trellis* en un sólo ciclo de reloj. Para esas implementaciones, que se pueden utilizar en el estándar 3GPP-LTE, se debe tomar en cuenta las modificaciones a la función del entrelazador QPP.

Un área de estudio a futuro también consideraría la implementación de turbo códigos de bloque y de códigos LDPC, así como la aplicación del concepto de codificación conjunta fuente-canal. También se pudiera considerar implementar la implementación completa de un sistema de comunicaciones, que incluya el desarrollo de módulos de comunicación inalámbrico empleando *software radio*, en donde se haga una consideración de la modulación de la información, así como la transmisión de información en tiempo real.

A

Modelos del canal y la capacidad del canal.

En general, un canal de comunicación es descrito en términos de un conjunto posible de entradas, denotado por X, y llamado el alfabeto de entrada; el conjunto de salidas posibles denotado por Y, y llamado el alfabeto de salida; y la probabilidad condicional que relaciona las secuencias de entrada y de salida de cualquier longitud n, la cual se denota por $P\left[y_1, y_2, ..., y_n | x_1, x_2, ..., x_n\right]$, donde $x = (x_1, x_2, ...x_n)$ y $y = (y_1, y_2..., y_n)$ representan las secuencias de entrada y salida de longitud n, respectivamente. Se dice que un canal no tiene memoria si hay independencia estadística entre los símbolos, lo cual se expresa como:

$$P\left[y|x\right] = \prod_{i=1}^{n} P[y_i|x_i] \ \forall n \tag{A.1}$$

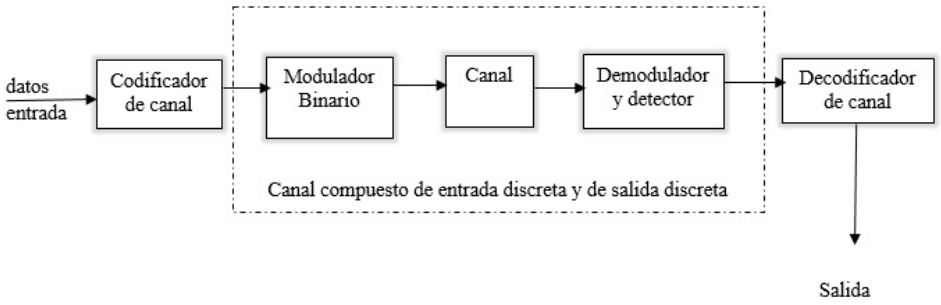

Figura A.1: Un canal compuesto de entrada discreta y salida discreta, incluyendo al modulador y demodulador como parte del canal.

El modelo de canal más simple es el canal simétrico binario (BSC), que corresponde al caso donde $X = Y = \{0, 1\}$. Este es un modelo de canal apropiado para modulación binaria y decisiones duras en el detector (ver figura A.1). El canal BSC está caracterizado por un conjunto de probabilidades que relacionan las posibles salidas con posibles entradas. Si el ruido de canal causa errores simétricos en la secuencia binaria transmitida con probabilidad p, se tiene (ver figura A.2) que:

$$\begin{aligned} P[Y = 0|X = 1] &= P[Y = 1|X = 0] = p \\ P[Y = 1|X = 1] &= P[Y = 0|X = 0] = 1 - p \end{aligned} \tag{A.2}$$

De esta manera, se ha reducido la cascada del modulador binario, el canal de la forma de onda, y el demodulador binario con el detector, a un canal en tiempo discreto equivalente, representado por el diagrama de la figura A.2.

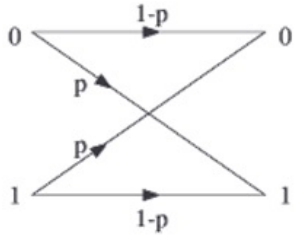

Figura A.2: Canal simétrico binario.

El canal BSC es un caso especial de un canal más general de entrada discreta y salida discreta. El canal discreto sin memoria (DMC) es un modelo de canal donde los alfabetos de entrada y salida, estadísticamente independientes, son conjuntos discretos. Este es el caso de cuando el canal usa un esquema de modulación M-ario y la salida del detector consta de símbolos Q-arios. Se puede ver el modelo de este canal en la figura A.3. Después de algunas manipulaciones matemáticas (consultar [5]), se puede demostrar que la información transmitida por cada uso del canal está expresada como $R = 1 - H_b(p)$, donde $H_b(p)$ es la llamada entropía de la fuente. Esta es la máxima tasa para una comunicación confiable en un canal BSC y se le conoce como la capacidad del canal C. La importancia de la capacidad del canal se establece en el siguiente teorema fundamental.

Segundo Teorema de Shannon -El teorema de la codificación del canal ruidoso (Shannon, 1948). La comunicación confiable sobre un canal discreto sin memoria es posible si la tasa de comunicación R satisface $R < C$, donde C es la capacidad teórica del canal. A tasas mayores que esta capacidad, la comunicación confiable no es posible.

Este teorema expresa un límite para las comunicaciones fiables y provee un criterio para medir el desempeño en sistemas de transmisión de información. Un sistema que opera cerca de la capacidad es un sistema casi óptimo y no da mucho espacio para una posible mejora. Si por otro lado, un sistema funciona alejado de este límite, su desempeño puede ser mejorado a través de diferentes técnicas de decodificación de canal.

Si en la entrada del modulador los símbolos son escogidos de un alfabeto discreto finito x, con $|x| = M$ y con la salida del detector no cuantizada, es decir $y = \mathbb{R}$, se tiene un canal compuesto caracterizado por la entrada discreta X, la salida continua Y, y un conjunto de

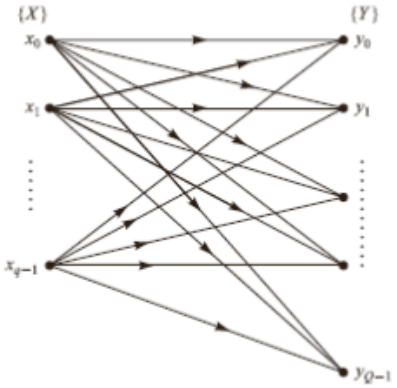

Figura A.3: Canal discreto sin memoria [5].

funciones de densidad de probabilidad PDF, condicionales. El canal más importante de este tipo es el AWGN, para el que $Y = X + N$, donde N es una variable aleatoria gaussiana de media cero y varianza σ^2. Bajo estas condiciones se tiene que:

$$p(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-x)^2}{2\sigma^2}} \tag{A.3}$$

En un canal AWGN en tiempo discreto se tiene que $X = Y = \mathbb{R}$. En cada instante i, una entrada x_i se transmite por lo que el símbolo recibido es $y_i = x_i + n_i$ donde las $n_i's$ son variables aleatorias gaussianas, independientes e idénticamente distribuidas i.i.d, de media cero y varianza σ^2. Se supone que la entrada del canal cumple con una restricción de potencia de la forma $E\left[X^2\right] \leq P$. Bajo la cual, cualquier secuencia de entrada de la forma $\mathbf{x} = (x_1, x_2, ..., x_n)$ se tiene que:

$$\frac{1}{n}\sum_{i=1}^{n} x_i^2 = \frac{1}{n}\left\|\mathbf{x}\right\|^2 \leq P \tag{A.4}$$

Para n muy grande, la ley de los números grandes establece que:

$$\frac{1}{n}\left\|\mathbf{y}\right\|^2 \to E\left[X^2\right] + E\left[N^2\right] \leq P + \sigma^2 \tag{A.5}$$

Lo cual significa que si $\mathbf{x}$ es transmitida, entonces y estará con una alta probabilidad en una esfera de n dimensiones de radio $\sqrt{n(\sigma^2)}$ y centrada en x [5]. El número máximo de esferas de ese radio que se pueden empaquetar en una esfera de radio $\sqrt{n(P+\sigma^2)}$ es la razón de los volúmenes de las esferas. El volumen de una esfera de n dimensiones está dado por $V_n = B_n R^n$, en donde $B_n = \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}+1\right)}$ donde Γ es la función gamma. Por lo tanto, el número máximo de mensajes que pueden ser transmitidos y ser resueltos en el receptor es

$$M = \frac{B_n\left(\sqrt{n(P+\sigma^2)}\right)^n}{B_n\left(\sqrt{n(\sigma^2)}\right)^n} \tag{A.6}$$

Lo cual resulta en una tasa de

$$R = \frac{1}{n} log_2 M = \frac{1}{2} log_2 \left(1 + \frac{P}{\sigma^2}\right) \tag{A.7}$$

con unidades de bits/transmisión. Si el canal es usado K veces para la transmisión de K muestras de un proceso aleatorio, ergódico $X(t)$ en T segundos, se encuentra que la capacidad de información por unidad de tiempo es (K/T) veces el resultado de la ecuación A.7. El número K es igual a $2BT$ (de acuerdo al teorema de muestreo de Nyquist) [2]. Por lo cual, se puede expresar la capacidad de información de un canal en la forma equivalente

$$C = B log_2 \left(1 + \frac{P}{N_0 B}\right), \tag{A.8}$$

donde P es la potencia promedio de la señal recibida.

A partir de esta ecuación es claro que la capacidad es incrementada aumentando P; si la potencia aumentara de forma infinita, la capacidad también podría hacerse infinita. Sin embargo, la tasa a la que la capacidad se incrementa en valores largos de P es logarítmica. El incremento en B tiene un rol dual en la capacidad. Por un lado, causa que la capacidad sea incrementada ya que un mayor ancho de banda implica más transmisiones sobre el canal por unidad de tiempo. Por el otro lado, incrementar B también reduce la SNR definida por P/N_0B. Esto es debido a que incrementar el ancho de banda aumenta la potencia efectiva de ruido entrando al receptor. Para ver cómo la capacidad cambia conforme $B \to \infty$, se emplea la relación $ln(1 + x) \to x$ conforme $x \to 0$ para obtener

$$C_\infty = \lim_{B\to\infty} B log_2 \left(1 + \frac{P}{N_0 B}\right) = (log_2 e) \frac{P}{N_0} \approx 1.44 \frac{P}{N_0} bits/s \tag{A.9}$$

B
Función max.

Defínase $E(x,y) = ln(e^x + e^y)$. Se tiene que:

$$ln(e^x + e^y) = lne^x + ln(e^x + e^y) - lne^x = x + ln\frac{e^x + e^y}{e^x} = x + ln(1 + e^{y-x}) \tag{B.1}$$

De manera similar

$$ln(e^x + e^y) = lne^y + ln(e^x + e^y) - lne^y = y + ln\frac{e^x + e^y}{e^x} = y + ln(e + e^{x-y}) \tag{B.2}$$

Por lo tanto, $E(x,y) = ln(e^x + e^y) = max(x,y) + ln(1 + e^{-|x-y|})$.
Y se toma $E(x,y) = ln(e^x + e^y) \approx max(x,y)$.
De manera similar, $E(x,y,z,w) = ln(e^x+e^y+e^z+e^w) = lne^x+ln(e^x+e^y+e^z+e^w)-lne^x$.

$$E(x,y,z,w) = x + ln\frac{e^x + e^y + e^z + e^w}{e^x} = x + ln(1 + e^{y-x} + e^{z-x} + e^{w-x}) \tag{B.3}$$

O $ln(e^x + e^y + e^z + e^w) = y + ln(1 + e^{x-y} + e^{z-y} + e^{w-y})$
O $ln(e^x + e^y + e^z + e^w) = z + ln(1 + e^{x-z} + e^{y-z} + e^{w-z})$
O $ln(e^x + e^y + e^z + e^w) = w + ln(1 + e^{x-w} + e^{y-w} + e^{z-w})$
Por lo tanto,
$E(x,y,z,w) = max(x,y,z,w)+ln\left[e^{x-max(x,y,z,w)+e^{y-max(x,y,z,w)+e^{z-max(x,y,z,w)+e^{w-max(x,y,z,w)}}}}\right] \approx$
$max(x,y,z,w)$.

En general, se tiene que

$$\begin{aligned} E(x_1, x_2, \ldots, x_k) &= ln \sum_{i=1}^{k} e^{x_i} = ln(e^{x_1} + e^{x_2} + \ldots + e^{x_{k-1}} + e^{x_k}) \\ &= ln\left(e^{x_1} + e^{x_2} + \ldots + e^{x_k - 2} + e^{ln(e^{x_{k-1}} + e^{x_k})}\right) \\ &= ln\left(e^{x_1} + e^{x_2} + \ldots + e^{x_k - 2} + e^{E(x_{k-1}, x_k)}\right) \\ &= ln\left(e^{x_1} + e^{x_2} + \ldots + e^{ln(x_{k-2} + e^{E(x_{k-1}, x_k)})}\right) \\ &= ln\left(e^{x_1} + e^{x_2} + \ldots + e^{E(x_{k-2}, E(x_{k-1}, x_k))}\right) \\ &= E\left(x_1, E\left(x_2, \ldots, E\left(x_{k-2}, E\left(x_{k-1}, x_k\right)\right)\right)\right) \end{aligned} \tag{B.4}$$

Por lo tanto,

$$E(x_1, x_2, \ldots, x_k) = ln \sum_{i=1}^{k} e^{x_i} = max(x_i) + ln \sum_{i=1}^{k} e^{x_i - max(x_i)} \approx max(x_i) \tag{B.5}$$

Bibliografía

[1] Moon, T. (2005). Error Correction Coding: Mathematical Methods and Algorithms. New Jersey: Wiley.

[2] Haykin, S. (2001) Communication Systems. 4th Edition. Wiley.

[3] Giuletti, A., Bougard, B., v.d. Perre, L. (2004). Turbo Codes: Desirable and Designable. Kluwer Academic Publishers.

[4] Forney, G., Burst-Correcting Codes for the Classic Bursty Channel. IEEE Transactions on Communications, Oct 1971, Vol. 19, Issue 5, pp.772-781.

[5] Proakis, J., Salehi, M. (2008). Digital Communications. 5th edition. McGraw-Hill.

[6] Sklar, B. (2001) Digital Communications. Fundamentals and Applications. Second Edition. Prentice Hall.

[7] Berrou, C., Glavieux, A., Thitimajshima, P. (1993) Near Shannon limit error-correcting coding and decoding: Turbo Codes. Proceedings of the International Conference on Communications. Geneva, Switzerland

[8] Crozier, S. (2000) New High-Spread High-Distance Interleavers for Turbo-Codes. 20th Biennal Symposium on Communications, Kingston, Ontario, Canada, pp.3-7,May 28-31,2000.

[9] Valenti, M.C., (1999) Iterative detection and decoding for wireless communications. Ph.D. Dissertation, Bradley Dept. of Elect. & Comp. Eng., Virginia Tech.

[10] Dolinar, S., Divsalar, D. (1995) Weight distribution for Turbo Codes using Random and NonRandom Permutations. JPL Progress Report.

[11] Hanzo, L., Liew, T.H., Yeap, B.L., Tee, R.Y.S., S.X.Ng. (2011) Turbo Coding, Turbo Equalisation and Space-Time Coding. Second Edition. UK: IEEE. Wiley.

[12] Carlson, A.B. (2002) Communication Systems. Fourth Edition. McGrawHill.

[13] Wong, C., Chang, H. (2014) Turbo Decoder Architecture for Beyond-4G Applications. New York: Springer.

[14] Sun, Y., Cavallaro, J. (2011). Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advanced turbo decoder. Elsevier.

[15] Artés, A., Pérez, F. (2007) Comunicaciones digitales. Pearson. Prentice Hall.

[16] Soleymani, M.R., Gao, Y., Vilaipornsawai, U. (2002). Turbo Coding for Satellite and Wireless Communications. Kluwer Academic Publishers.

[17] Castiñeira, J., Guy, P. (2006) Essentials of Error-Control Coding. Wiley.

[18] Hsu, H. (2003). Schaum's Outlines. Analog and Digital Communications. Second Edition.

[19] Morelos-Zaragoza, R.H. (2006) The Art of Error Correcting Coding. Second Edition. John Wiley & Sons.

[20] Lazcano, S. (2011) Algoritmos iterativos de control de errores para sistemas de transmisión de información con razón señal–ruido en la región de piso de ruido. Tesis de doctorado en Telecomunicaciones. UNAM.

[21] Douillard, C. (2007) COCA Project. Channel coding: State of the art and perspectives (technical and strategic aspects). Chapter 2: Convolutional Turbo Codes. ENST de Bretagne.

[22] Sánchez, O. (2013). La montée en debit dans les architectures de turbo décodage de codes convolutifs. Thése de Doctorat. Télécom Bretagne. Soutenue le 15 Mars 2013. École Doctorale –SICMA.

[23] Gnaedig, D. (2005). High-speed decoding of convolutional Turbo Codes. These présentée à L'Université de Bretagne Sud. Pour obtener le titre de Docteur de l'UBS.

[24] Shannon, C.E.(1948). A Mathematical Theory of Communication. Reprinted with corrections from *The Bell System Technical Journal*, Vol. 27, pp.379-423, 623-656, July, October, 1948.

[25] Robertson, P., Villebrum, E., Höher, P. (1995) A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain. IEEE International Conference On Communications. ICC '95 Seattle.

[26] Vaton, S., Glavieux, A. (2001) Codes Convolutifs. Notas de la Université Telecom Bretagne.

[27] Berrou, C. (2010) Codes and turbo Codes. Springer. Iris International Series. Ínstitut Telecom.

[28] Glavieux, A. (2005) Channel Coding in Communication Networks. From Theory to Turbocodes. ISTE. Digital Signal and Image Processing Series.

[29] Hagenauer, P., (1988) Rate-compatible punctured convolutional codes (RCPC codes) and their applications. IEEE Transactions on Communications, vol. 36.

[30] Benedetto, S., Montorsi. (1996). Serial concatenation of block and convolutional codes. Electronics Letters, vol. 32.

[31] Ben, D. (2011). Towards ideal codes: looking for new turbo code schemes. Thése de Doctorat. Télécom Bretagne.

[32] Abrantes, S. (2004). From BCJR to turbo decoding: MAP algorithms made easier. Faculdade de Engenharia da Universidade de Porto (FEUP), Porto, Portugal.

[33] Cheik, Y. (2005). On Distance Measurement Methods for Turbo Codes. Department of Electrical & Computer Engineering. McGill University. Montreal, Canada.

[34] Balbuena, C., Ugalde, F. (2013). Performance of HSR and QPP-based interleavers for turbo coding on power line communication systems. Signal, Image and Video Processing. Springer-Verlag London.

[35] Sun, J., Takeshita, O. (2005). Interleavers for turbo codes using permutation polynomials over integer rings. IEEE Transactions on Information Theory, Vol.51, No.1.

[36] Rosnes, E., Takeshita, O. (2006). Optimum distance quadratic permutation polynomial-based interleavers for turbo codes. Proceedings of IEEE International Symposium on Information Theory, pp. 1988-1992.

[37] Takeshita, O. (2006). On maximum contention-free interleavers and permutation polynomials over integer rings. IEEE Transactions on Information Theory, vol.52, no.3, pp.1249-1253.

[38] Rivest, R. (2001). Permutation polynomials modulo 2w. Finite fields and their applications, 7(2):287-292. CiteSeer.

[39] Forney, D. (1965). Concatenated Coes. Technical Report 440. Massachussets Institute of Techonlogy. Research Laboratory of Electronics. Cambridge, Massachusetts.

[40] Ryu, J., Takeshita, O. (2006). On quadratic inverses for quadratic permutation polynomials over integer rings. IEEE Transactions on Information Theory, 52:1254-1260.

[41] Chi, C. (2013). Quadratic Permutation Polynomial Interleavers for LTE Turbo Coding. International Journal of Future Computer and Communication, Vol.2, No.4, August 2013.

[42] Takeshita, O. (2006). Permutation Polynomial Interleavers: An Algebraic-Geometric Perspective. Ohio State University. IEEE Transactions on Information Theory.

[43] Hoon, J. (2007). Permutation Polynomial Interleavers for Turbo Codes over Integer Rings: Theory and Applications. Dissertation in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graudate School of The Ohio State University.

[44] Crozier, S., Lodge, P., Hunt, A. (1999). Performance of turbo codes with relatively prime and golden interleaver strategies. Proc. 6th Int. Mobile Satellite Conf. Ottawa, Canada.

[45] Berrou, C., Saouter, Y., Douillard, C., Kérouedan, S., Jézéquel, M. (2004) Designing good permutations for turbo codes: towards a single model. Proceedings of the IEEE ICC 2004.

[46] Heiko, M., Worm, A., Wehn, N. (2000) Influence of quantization on the bit-error performance of turbo decoders. Vehicular Technology Conference Proceedings. VTC 2000-Spring Tokyo.

[47] Bahl, L., Cocke, J., Felinek, F., Raviv, J. (1974) Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate, IEEE Trans. Info. Theory, Vol IT-20, pp 284-287, March 1974.

[48] Lestable, T., Ran, M. (2011) Error control coding for B3G/4G Wireless Systems. Wiley: UK.

[49] Jego, C., Liu, H., Diguet, J.P., Jezequel, M., Boutillon, E. (2007) Energy efficient turbo decoder with reduced state metric quantization. IEEE Workshop on Signal Processing Systems.

[50] Jung, C., Jung, J. Kim, N. (2005) Complexity-reduced algorithms for LDPC decoder for DVB-S2 systems. ETRI Journal, vol.27, no.5.

[51] Hekstra, A.P. (1989) An alternative to metric rescaling in Viterbi decoders. IEEE Transactions on Communications, vol.27, no.11, pp.1220-1222.

[52] Worm, A., Michel, H., Gilbert, F., Kreiselmaier, G., Thul, M., Wehn, N. (2000). Advanced implementation issues of turbo-decoders. Proc. 2nd International Symposium on Turbo-Codes and Related Topics.

[53] Benkeser, C., Burg, A., Cupaiuolo, T., Huang, Q. (2009). Design and Optimization of an HSDPA Turbo Decoder ASIC. IEEE Journal of Solid-State Circuits. Volume:44. Issue 1.

[54] Chávez, N. et al. (2006) Dispositivos lógicos programables VLSI. Universidad Nacional Autónoma de México. Facultad de Ingeniería.

[55] Brown, S., Vranesic, Z. (2009) Fundamentals of Digital Logic with VHDL Design. Third Edition. McGrawHill. Higher Education.

[56] Chu, P. (2008). FPGA Prototyping by VHDL Examples. Xilinx Spartan 3-Version. Wiley.

[57] Boutillon, E., Danger, J., Ghazel, A. (2003). Design of High Speed AWGN Communication Channel Emulator. University of Bretagne Sud.

[58] Maxines, D., Alcalá, J. (2002). VHDL. El arte de programar sistemas digitales. CECSA.

[59] Hardy, G., Wright, E., (1979) An Introduction to the Theory of Numbers. Oxford University Press, Oxford, UK.

[60] Savage, J., Vázquez, G., Chávez, N. (sin año). Diseño de Microprocesadores. Universidad Nacional Autónoma de México. Facultad de Ingeniería.