UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA

IMPLEMENTACIÓN DE UN ESQUEMA DE CODIFICACIÓN DE CANAL CONCATENADO PARA UN SISTEMA INALÁMBRICO BASADO EN LA RASPBERRY PI B.

Br. Stephanie Rocío Araque Hernández.

Mérida, Diciembre, 2020.

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA

IMPLEMENTACIÓN DE UN ESQUEMA DE CODIFICACIÓN DE CANAL CONCATENADO PARA UN SISTEMA INALÁMBRICO BASADO EN LA RASPBERRY PI B.

Trabajo de Grado presentado como requisito parcial para optar al título de Ingeniero Electricista.

Br. Stephanie Rocío Araque Hernández.
Tutor: Prof. Emigdio Antonio Malaver.
Tutor: Prof. Luis Ramón Araujo Rangel.

Mérida, Diciembre, 2020.

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA

IMPLEMENTACIÓN DE UN ESQUEMA DE CODIFICACIÓN DE CANAL CONCATENADO PARA UN SISTEMA INALÁMBRICO BASADO EN LA RASPBERRY PI B.

Br. Stephanie Rocío Araque Hernández.

Trabajo de Grado, presentado en cumplimiento parcial de los requisitos exigidos para optar al título de Ingeniero Electricista, aprobado en nombre de la Universidad de Los Andes por el siguiente Jurado.

_____

Prof. Edinson Del Carme Dugarte Dugarte.
Jurado

_____

Prof. Emigdio Antonio Malaver.
Jurado

_____

Prof. Luis Ramón Araujo Rangel.
Jurado

Stephanie Rocío Araque Hernández. Implementación de un esquema de codificación de canal concatenado para un sistema inalámbrico basado en la Raspberry Pi B. Universidad de Los Andes. Tutor(es): Prof. Emigdio Antonio Malaver; Prof. Luis Ramón Araujo Rangel. Diciembre 2020.

RESUMEN

El presente trabajo de grado tiene como propósito identificar la influencia de introducir a un sistema de comunicación, técnicas de codificación de canal. Las técnicas de codificación de canal tienen el objetivo de reducir el número de errores que pueden ocurrir en la transmisión de información. El sistema de comunicación que se implementará en el presente trabajo para efectuar la transmisión de información aplicando técnicas de codificación de canal, se realizará integrando un emisor representado por una Raspberry Pi B, un receptor representado por otra Raspberry Pi B y finalmente un módulo que permitirá la comunicación de manera inalámbrica entre las dos Raspberry Pi. Cada Raspberry Pi se programará en el lenguaje de programación C, para que una actúe como emisor y la otra como receptor. Las técnicas de codificación de canal que serán implementadas en el sistema de comunicación serán, el código de bloques y el código de convolución, dos técnicas que operando por separado tienen diferentes resultados en cuanto al número de errores que pueden corregir en la transmisión de información. En este trabajo de grado se evaluará, el aporte por separado de cada técnica de codificación de canal al sistema de comunicación implementado, luego se evaluará el aporte que generan dichas técnicas estando concatenadas, y finalmente el aporte que generará integrar a estas dos técnicas concatenadas, un entrelazador, con el objetivo de optimizar el funcionamiento de estas técnicas de codificación de canal.

Descriptores: corrección de errores, codificador de canal, código de bloques, código de convolución, entrelazador.

INDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCION

Los sistemas de comunicación juegan un papel fundamental en el creciente desarrollo tecnológico de la humanidad debido a todas las ventajas que ofrecen. Ventajas como, permitir la comunicación a distancias que parecían casi imposible transmitir o permitir la automatización de procesos. Esto ha generado la continua necesidad de mejorar tanto la calidad como la velocidad de transmisión de información.

Los sistemas de comunicación están conformados básicamente por el emisor, el receptor y el canal de comunicaciones, siendo el canal la conexión entre el emisor y el receptor. Una característica inherente al canal de transmisión que existe entre el emisor y el receptor de un sistema de comunicación es que la señal transmitida se degrada al ser propagada por el canal de comunicación. El grado de degradación está relacionado a diferentes factores, algunos de esos factores son, la distancia que existe entre el emisor y el receptor, esto es debido a que la señal puede sufrir una mayor atenuación cuando la distancia existente entre emisor y receptor, es mayor. Otro factor de degradación ocurre cuando el canal del sistema es guiado o no guiado, esto es debido a que el medio no guiado es más sensible al ruido, ruido sobre el cual no se tiene control, por tanto, la señal siempre sufrirá algún tipo de degradación.

La codificación de canal se puede definir como el producto de detectar y en algunos casos corregir los errores que se producen en la transmisión de información de un sistema de comunicación debido al ruido. El codificador de canal en el emisor toma el mensaje digital que se requiere transmitir al receptor, y le añade a este mensaje los denominados bits de paridad, estos bits de paridad en conjunto con el decodificador de canal, van a permitir que el receptor tenga la capacidad de detectar si el mensaje que le fue enviado tiene bits erróneos y además podrá corregir algunos de estos bits erróneos, mejorando así la calidad de la transmisión de información.

El presente trabajo de grado se refiere a la implementación de técnicas de codificación de canal en un sistema de comunicación entre dos Raspberry Pi con un medio no guiado como canal de comunicación, donde se seleccionan dos técnicas de codificación de canal y posteriormente se realiza la implementación de estas dos técnicas concatenadas, con el

objetivo de reconocer si este método funciona como una técnica robusta para mejorar la calidad de la transmisión de información, y si lo hace, reconocer que tanto mejora la calidad de transmisión. La implementación se realiza a partir de programar las dos técnicas de codificación de canal concatenadas en una Raspberry Pi que será el emisor del sistema de comunicación, y las dos técnicas de decodificación de canal concatenadas en otra Raspberry Pi que será el receptor del sistema de comunicación. Finalmente, el canal de comunicación entre las dos Raspberry Pi será un medio no guiado, o inalámbrico, empleando para ello el módulo RF 433. Luego de transmitir la información de la Raspberry Pi emisor a la Raspberry Pi receptor, se obtendrá el número de errores que ocurrieron en la transmisión de información, con un código que comparará la información enviada con la recibida.

El trabajo de grado se divide en cuatro capítulos los cuales se resumen de la siguiente manera:

El capítulo I, expone el planteamiento del problema, los objetivos, la justificación, el alcance y limitaciones de la implementación del codificador de canal concatenado.

El capítulo II, presenta antecedentes relacionados a la codificación de canal los cuales señalan ventajas y diferencias de algunas técnicas de codificación de canal, y expone una breve presentación de los fundamentos teóricos necesarios para entender e implementar el codificador de canal robusto.

El capítulo III, señala la metodología seguida para implementar el codificador de canal concatenado, también presenta de forma desglosada las distintas etapas de programación del codificador de canal concatenado implementado.

El capítulo IV, señala los resultados obtenidos de las diferentes etapas de la implementación descritas en el capítulo III, con sus respectivos análisis.

CAPITULO I
PROBLEMA EN ESTUDIO

En este capítulo se presentan los objetivos que se desean alcanzar mediante la implementación de un esquema de codificación de canal concatenado.

1.1 PLANTEAMIENTO DEL PROBLEMA.

El hombre desde el inicio de los tiempos ha sentido la necesidad de comunicar sus sentimientos y pensamientos, permitiendo enriquecer la experiencia humana. Según Briceño [1] y [2], el propósito de un sistema de comunicación es el de transmitir información, este sistema comprende un transmisor, que emite la información, un canal sobre el cual la información se transmite, y un receptor para recoger la información. El canal de transmisión puede ser un simple par de conductores, un cable coaxial, una fibra óptica, una guía de ondas o el espacio libre.

Para que el sistema de comunicación cumpla su objetivo debe poseer propiedades muy específicas, una de estas, es que debe ser capaz de resistir condiciones ambientales realmente duras y una gran cantidad de ruido electromagnético. Ahora bien, según Haykin [3], la razón entre la energía de la señal por bit y la densidad espectral de potencia del ruido, E_b/N_0, determina singularmente la tasa de error de bits de un esquema de modulación particular, pero las consideraciones practicas suelen poner un límite en el valor que se puede asignar a E_b/N_0, lo que conduce que no sea posible proporcionar una calidad de datos aceptable, por tanto, para un E_b/N_0 fijo, la única opción practica disponible para cambiar la calidad de los datos a aceptable consiste en utilizar la codificación de control de errores. En efecto existen diversas técnicas de codificación de canal que permiten la corrección de errores producidos por el ruido y distorsión introducidos en el canal de comunicación, y por las no linealidades

del sistema de transmisión. En virtud de lo anteriormente expuesto surgen las siguientes incógnitas, ¿Qué tanto se afectaría la integridad de la comunicación si no se aplica ninguna técnica de codificación de canal?, ¿Es posible obtener una correcta transmisión y reproducción de datos a partir de agregar una técnica de codificación de canal al sistema de comunicación usando microcontroladores o tarjetas de desarrollo?, Y, ¿Será suficiente la concatenación de dos técnicas de codificación de canal, un código de bloque con un código de convolución, como técnica robusta de codificación de canal para realizar una buena comunicación?.

Por todo lo anterior se destaca la significación y necesidad de implementar un esquema de codificación de canal robusto que permita comprobar si dicho esquema puede generar una transmisión y reproducción confiable de los datos.

1.2 OBJETIVOS.

A continuación, se exponen los objetivos generales y específicos que se persiguen cumplir con el desarrollo de este proyecto.

1.2.1 Objetivo General.

Implementar un esquema de codificación de canal concatenado en un sistema inalámbrico basado en la Raspberry Pi B.

1.2.2 Objetivos Específicos.

- Determinar la arquitectura del codificador concatenado que mejor se adapte para realizar la corrección de errores en un sistema de comunicación inalámbrico basado en la Raspberry Pi B.
- Implementar y verificar el funcionamiento del codificador/decodificador de bloques.
- Implementar y verificar el funcionamiento del codificador/decodificador de convolución.
- Implementar y verificar el funcionamiento del codificador concatenado.

1.3 JUSTIFICACIÓN.

En los entornos industriales el intercambio de datos entre dispositivos, subsistemas y el centro de control de la planta es una tarea fundamental. En plantas grandes, este intercambio de información se lleva a cabo a través de sistemas SCADA, redes de control (redes de campo) o redes de dispositivos. En el proceso de transmitir la información, la integridad de la señal transmitida se ve afectada por fenómenos de distorsión, tanto lineal como no lineal, distintos tipos de interferencia, eventos transitorios y por el ruido en el canal de comunicación. La pérdida de la integridad de la señal se refleja en la tasa de errores del sistema. Hasta cierto punto, el efecto de la distorsión y ciertos tipos de interferencia se puede minimizar diseñando apropiadamente la forma de onda a transmitir y los circuitos electrónicos que las procesan. Los eventos transitorios y el ruido, dada su naturaleza aleatoria, son más difíciles de controlar. El efecto del ruido se puede minimizar empleando filtros acoplados a la forma de onda transmitida; sin embargo, esto no elimina su influencia en la tasa de errores. Por otro lado, los errores causados por eventos transitorios son muy difíciles de controlar. Adicionalmente, la atenuación y el hecho de que los canales son de ancho de banda limitado, también deben ser considerados como factores que afectan negativamente la tasa de errores. Para que la transmisión de datos se lleve a cabo con una tasa de errores aceptable, se deben implementar ciertas técnicas que ayudan a detectar y corregir errores. Estas técnicas se conocen en conjunto como técnicas de codificación de canal. En los sistemas SCADA, en las redes de campo y en las redes de dispositivos, la codificación de canal se implementa en las capas de transporte de la red y generalmente no está bajo el control del usuario.

En situaciones donde no se dispone de la infraestructura de red, el uso de microcontroladores o de pequeñas tarjetas de desarrollo suele ser una alternativa para realizar las tareas de control y le da al usuario la posibilidad de implementar la técnica de codificación de canal apropiada para los niveles de interferencia y ruido que se tenga en el entorno de trabajo. Por consiguiente, se refleja la importancia de la implementación de un esquema de codificación de canal que permita la transmisión y reproducción de información veraz.

Finalmente, la implementación del esquema de codificación de canal concatenado

aplicado, así como toda la investigación teórica desarrollada en el presente proyecto, servirán a futuros investigadores para la creación de nuevas investigaciones.

1.4 ALCANCE.

En este trabajo de grado se implementará un esquema de codificación de canal que concatena dos técnicas de codificación de canal, un código de bloque con un código de convolución, como estrategia robusta de corrección de errores en la transmisión de datos, haciendo uso de dos Raspberry Pi, y un módulo RF 433 que permita la conexión entre las dos Raspberry Pi de manera inalámbrica.

1.5 LIMITACIONES.

Si el canal de transmisión se requiere de manera inalámbrica o de medio no guiado, se debe tener un módulo de transmisión y recepción que no realice control de errores, y además que trabaje de manera síncrona, debido a que la comunicación en estos entornos se basa en tramas, y estas tramas se envían una tras otra en un intervalo de tiempo preestablecido.

El proyecto tenía como posibilidad hacerse con microcontroladores, específicamente con el PIC 18F4620, este PIC cuenta con una RAM de 3.968 bytes, 64.000 bytes de memoria de programas y podía trabajar hasta 8 MHz, pero cuando se empezó a realizar la implementación del código de bloques para palabras de código de tan solo 7 bits, ya este código ocupaba alrededor del 20% de la memoria de programas, y un 10% de la memoria RAM de este PIC, y esta era tan solo la primera fase del proyecto de grado, por tanto, este dispositivo se descartó y en su lugar se usó la Raspberry Pi 1 modelo B, la cual tiene 512 MB de RAM y funciona a 700 MHz, a una frecuencia muy superior a la que trabaja el PIC 18F4620. Es importante acotar que la memoria es necesaria para la implementación de este codificador robusto debido a que se va a guardar toda la información codificada en el emisor antes de ser enviada al receptor, de igual manera para el receptor, también es necesario el uso de memoria para guardar todos los datos decodificados, y así comparar los datos transmitidos del emisor con los recibidos y decodificados por el receptor.

CAPITULO II
MARCO TEORICO

El presente capítulo muestra el soporte teórico del proyecto. Los siguientes conceptos y fundamentos teóricos son necesarios para entender el contexto del trabajo y lograr los objetivos propuestos para el mismo.

2.1 ANTECEDENTES.

A continuación, se presentan algunos antecedentes relacionados a la codificación de canal.

2.1.1 Inicios de la codificación de canal.

El matemático e ingeniero estadounidense C. E. Shannon propuso la teoría de la información con la publicación de su artículo *A Mathematical Theory of Communication* en la revista *Bell Systems Technical Journal* en 1948. En este artículo, Shannon introdujo los conceptos de entropía de la fuente y capacidad de canal, conceptos que son usados en dos teoremas fundamentales para el desarrollo de las comunicaciones. Estos teoremas son conocidos como Teorema de Shannon de codificación de fuente y Teorema de Shannon de codificación de canal [4]. En el primer teorema Shannon establece lo siguiente:

> "Sea una fuente con entropía H (bits por símbolo) y un canal con capacidad C (bits por segundo). Entonces es posible codificar la salida de la fuente de tal modo que se pueda transmitir a un régimen de $C/H-\varepsilon$ símbolos por segundo sobre el canal, donde ε es tan pequeño como se quiera. No es posible transmitir a un régimen promedio mayor que C/H" [4].

En este teorema Shannon expresa que para realizar la codificación de fuente se debe tomar en cuenta la existencia de un límite de eficiencia para la transmisión. Indica que el número de bits necesarios para describir un mensaje de la fuente debe ser mayor a la Entropía (H) y se puede aproximar a la correspondiente entropía tanto como se desee. La entropía de una

fuente caracterizada por la emisión de un número finito de símbolos, es la información promedio asociada con la fuente, esta equivale al mínimo número de dígitos binarios que podrían emplearse para la codificación de fuente. En conclusión, este teorema establece la longitud mínima de codificación sin perdidas de los mensajes de una fuente dada [4].

El segundo teorema establece lo siguiente:

> "Sea un canal discreto con capacidad C (bits por segundo) y una fuente discreta con entropía H (bits por segundo). Si $H \leq C$ entonces hay un sistema de codificación que permitiría transmitir la salida de la fuente sobre el canal con una tasa de errores tan pequeña como se desee (o una equivocación arbitrariamente pequeña). Si $H > C$ es posible codificar la fuente de modo que la equivocación sea menor que $H - C + \varepsilon$, donde ε es tan pequeño como se quiera. No hay ningún método de codificación que proporcione una equivocación menor que $H – C$" [4].

Este teorema añade que hay un límite también para la velocidad de transmisión de información, este límite depende también de la entropía (H), indica que para la transmisión debe emplearse un canal de capacidad C mayor que la entropía de la fuente, siendo C la capacidad del canal en bits por segundo y define una medida de la eficacia con la cual un canal transmite información, y H la entropía en bits por símbolo. Si se intentara transmitir mediante un canal de menor capacidad que entropía, el exceso de entropía de la fuente respecto a la capacidad del canal perjudicaría la transmisión de información ya que generaría un aumento de errores. En conclusión, nos señala que los errores que se obtienen en el receptor pueden disminuirse tanto como deseemos siempre y cuando la capacidad del canal sea mayor que la velocidad de transmisión (H), es decir, $C > H$. Tomando en cuenta este teorema se podrá obtener una transmisión de datos confiable [4].

Teniendo este teorema en cuenta, se puede introducir redundancia que permita al receptor identificar y corregir los errores usando la distancia existente entre la entropía de la fuente H y la capacidad del canal usado C. A la codificación empleada para introducir redundancia con el fin de detectar y corregir errores se le denomina codificación de canal. Existen diversas técnicas para introducir dicha redundancia, entre estas técnicas se encuentran los códigos de bloque y los códigos de convolución. En los primeros se usan bloques de datos consecutivos para la determinación de la redundancia que se añadirá a estos bloques de datos

y en los de convolución se utilizan máquinas de estados cuya salida depende del estado y de los datos de entrada. La corrección de errores consiste en una búsqueda de los bloques válidos más parecidos para el primer tipo o de las secuencias de código más probables en los de convolución [3].

2.1.2 Estudios Recientes.

A continuación, se presentan algunos estudios realizados a sistemas de comunicación en los que se aplican diferentes técnicas de codificación con la finalidad de mejorar la confiabilidad de la transmisión.

Balderas (2008), en su tesis presentada en el Instituto Politécnico Nacional, sobre "La Modulación Codificada Entramada (TCM) en los nuevos estándares de comunicación", cuyos objetivos fueron presentar una revisión de la modulación codificada entramada, así como los conceptos que involucra, desarrollar modelos de software en MATLAB útiles para la implementación de un sistema TCM, y proveer un material útil para la fácil comprensión del funcionamiento de los esquemas de modulación de Trellis. En esta tesis Balderas realizó simulaciones en MATLAB, en las que se enfocó en realizar, la codificación/modulación, el canal AWGN, y la demodulación/decodificación. En primer lugar, el programa hacia la codificación con una tasa de R= ½, luego realizó la simulación del paso de los datos codificados a través de un canal con AWGN, y posteriormente realizó la decodificación usando el algoritmo de Viterbi con decisión dura y suave. Aplicó esta codificación y decodificación a una cadena de datos que el usuario introdujo junto con las características básicas del codificador convolucional como son: el número de entradas, salidas y elementos de memoria. Las conclusiones de esta tesis fueron que, a partir de los resultados de las simulaciones, los códigos de convolución en combinación con el algoritmo de Viterbi proporcionan un buen desempeño en cuanto a la tasa de error que manejan [5].

Victorino (2008), en su tesis de posgrado presentada en el Centro de Investigación Científica y de Educación Superior de Ensenada, sobre "Concatenación de Códigos Convolucionales de Canal con Códigos Espacio-Tiempo y Espacio-Frecuencia", desarrolló una plataforma de simulación en MATLAB, que permite realizar y evaluar, el impacto de concatenar la codificación de canal convolucional a un sistema MISO (*multiple imput, single*

output) que usa como esquema de diversidad tanto la codificación espacio-tiempo como de espacio-frecuencia. En esta tesis, Victorino hizo pruebas del funcionamiento en MATLAB de sistemas de comunicaciones con solo los esquemas de diversidad de codificación espacio-tiempo y de codificación de espacio-frecuencia, estas son técnicas para mejorar la transmisión de información. Victorino comparó ambos esquemas realizando variaciones de algunos parámetros, evaluó las diferencias de cada modelo y también los comparó con un sistema que no tenía estos esquemas de diversidad, resultando que las curvas de desempeño de los esquemas con diversidad señalaban una disminución aproximadamente de 10 dB de la relación señal-ruido a una probabilidad de error de $BER=10^{-4}$, en comparación con las que no tenían técnicas de diversidad. Posteriormente agrego la etapa de codificación de canal con un código convolucional, ubicándolo antes del esquema de diversidad de codificación espacio-tiempo, con el fin de evaluar que tanto podía mejorar la transmisión. Encontró que la codificación de canal con código convolucional en conjunto con el esquema de diversidad ofrecía una reducción considerable en el número de errores, aproximadamente de 2 dB con $BER=10^{-4}$. Después, él comentó que el uso de codificadores de canal comúnmente es conjuntado a una etapa de entrelazado, y evaluó el aporte que daba el codificador de canal en conjunto con la etapa de entrelazado al esquema de diversidad, y lo comparó con el sistema que tenía solo el esquema de diversidad. Dando como resultado una contribución más importante, una reducción alrededor de 7 dB con $BER=10^{-4}$. Sus conclusiones fueron que la etapa de entrelazado juega un papel determinante en la disminución de los errores. La codificación de canal probó ser una técnica que redujo el número de errores del sistema al que fue concatenada, conforme se incrementa la longitud restringida de los códigos convolucionales se obtiene una mejora en el desempeño del sistema, pero solo hasta 2 dB, por tanto, el incremento de la complejidad esta pobremente remunerada. Victorino expone la conveniencia de usar códigos convolucionales con longitudes limitadoras pequeñas [6].

Derteano (2012), en su trabajo presentado a la Universidad de Chile, titulado “Aplicación y Evaluación de Estrategias para el Control de Errores en Canales Satelitales mediante BICM”, realiza simulaciones de Monte Carlo para evaluar el desempeño de las distintas opciones consideradas en un canal con ráfaga y también en uno de ruido blanco aditivo Gaussiano, realizando comparaciones y análisis respecto al desempeño de un esquema clásico de codificación y la codificación modulada con entrelazado de bit (BICM). Empieza

realizando simulaciones de distintas familias de codificación, empezando con el código Bose-Chaudhuri-Hocquenghem (BCH), que son códigos de bloques lineales, en el cual encontró que tenía una capacidad de corrección de errores moderada a una tasa de R=0.88. Luego simuló un esquema Reed-Solomon (RS), en el que encontró un buen desempeño a partir de una E_b/N_0 igual a 10 dB con una tasa de código R=0.87, pero con el inconveniente de requerir longitudes de mensaje muy grandes, de 2040 bits, constituyendo una desventaja de este esquema. Derteano acotó que el desempeño de este modelo es prácticamente el mismo para los casos con entrelazador o sin entrelazador, debido a la naturaleza de esta codificación, que permite corregir errores de ráfaga. Posteriormente, simuló un código convolucional obteniendo un buen desempeño y con una tasa de R=0.5, notando que en esta codificación el desempeño aumentó con el uso de un entrelazador. Finalmente, Derteano resaltó las siguientes ventajas de los codificadores convolucionales, buen desempeño para niveles bajos de E_b/N_0 y su simpleza. Además, acotó que el sistema demostró ser robusto frente a cambios en las condiciones del canal, mostrando muy poca degradación en el rendimiento, y constituyendo una alternativa posible de utilizar considerando variaciones presentadas por el canal satelital. También añadió que la principal desventaja de esta técnica es que el entrelazador agrega latencia al sistema, además que el receptor emplea un decodificador de Viterbi con decodificación suave lo que implica que ocupa recursos considerables, por tanto, se debe contar con una mayor potencia computacional [7].

Como se puede observar a lo largo de los antecedentes, el propósito de estos es ver el aporte a la transmisión de la señal, cuando se aplica una técnica de codificación de canal. El presente proyecto se realizó con el objetivo de ver que tanto se puede mejorar la transmisión de información aplicando una codificación de canal robusta, donde se usan no solo una técnica de codificación de canal, sino dos técnicas de codificación de canal concatenadas.

2.2 SISTEMA DE COMUNICACIÓN.

La comunicación es la acción de transmitir información de un punto a otro a través de un medio de transmisión denominado canal, esta se puede realizar analógicamente o digitalmente. En la comunicación analógica se transmiten mensajes que pertenecen a un conjunto infinito y continuo de valores, y en la comunicación digital los mensajes pertenecen

a un conjunto finito y discreto de valores. Independientemente del tipo, los elementos básicos de todo sistema de comunicaciones son: fuente, transmisor, canal, receptor, y el usuario, tal como se indica en la figura 2.1. Según [1] y [3], el transmisor se localiza en un punto del espacio separado del receptor. El canal es el medio físico que los conecta adecuadamente.

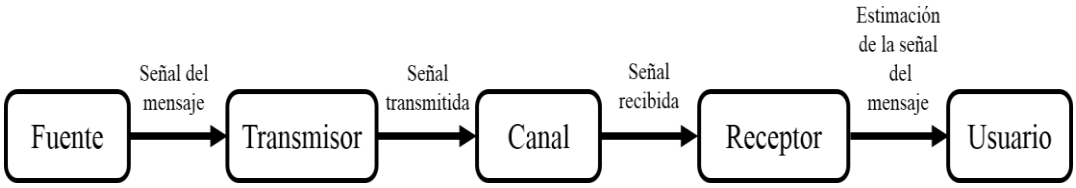


Figura 2.1. Elementos básicos en un sistema de comunicación [1].

Según [8], la comunicación digital a diferencia de la comunicación analógica, durante la transmisión es menos sensible al ruido, y además permite modificar la señal de transmisión para aplicar técnicas de control de error. Los sistemas de comunicación digital poseen un conjunto de subsistemas que permiten la comunicación entre el transmisor y el receptor como se puede observar en la figura 2.2. El transmisor estará compuesto por el codificador de fuente, el codificador de canal, y el modulador, y el receptor estará compuesto por el demodulador, el decodificador de canal y el decodificador de fuente.

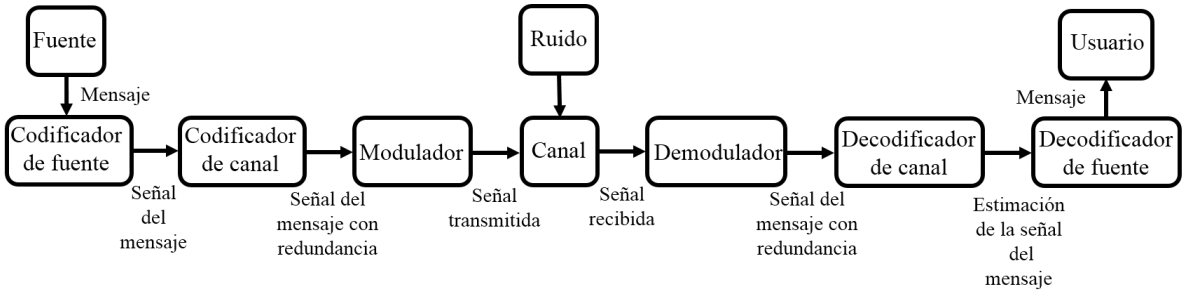


Figura 2.2. Elementos en un sistema de comunicación digital [1], [3], [8].

A continuación se hará una breve descripción de los elemtentos en un sistema de comunicación digital.

2.2.1 Fuente.

Según [1] y [3], la fuente es un mecanismo generador de información. Es posible caracterizar la fuente de información en términos de la señal que porta la información. Esta información

se materializa como un conjunto de N símbolos o mensajes distintos e independientes. Hay muchas fuentes de información, incluyendo personas y máquinas, lo que significa que los símbolos o mensajes pueden tomar una gran variedad de formas, como una secuencia de símbolos discretos o letras, una magnitud que varía en el tiempo, etc., pero cualquiera que sea el mensaje, el propósito del sistema de comunicación es el de proporcionar una réplica más o menos exacta del mismo en el destino. En este proyecto se considera una fuente de información discreta, la cual escoge y transmite secuencias de símbolos de un alfabeto finito fijo. Cada selección se hace al azar.

2.2.2 Mensaje y señal del mensaje.

Según [1] y [3], como regla general, el mensaje que produce la fuente no es de naturaleza eléctrica, o de una naturaleza que permita la transmisión de información en un canal, y, por lo tanto, es necesaria la presencia de un transductor o codificador que convierta el mensaje en una señal que pueda ser enviada posteriormente. En el caso del habla, el mensaje se produce en la mente del hablante que quiere expresarse, y se representa mediante una señal del habla que está compuesta por sonidos y cuyo arreglo está gobernado por las reglas del lenguaje.

Si se observa la figura 2.2, el mensaje es la materialización de la información en una cantidad mensurable, es el soporte de la información proveniente de la fuente, y la señal mensaje es la magnitud eléctrica que resulta de la transformación de una magnitud no eléctrica portadora de información en una magnitud eléctrica variable en el tiempo, constituyendo una señal analógica, de otro modo, los datos pueden ser transformados en un conjunto de valores discretos, en lo que se denomina una señal digital. En este proyecto se trabajará con una señal de mensaje digital, para poder aplicar los métodos de detección y corrección de errores.

2.2.3 Codificación y decodificación de fuente.

De acuerdo a [3] y [8], el convertidor de fuente convierte la secuencia de mensajes o símbolos que provienen de la fuente en una secuencia binaria. En el caso de que los símbolos tengan las mismas probabilidades, se les asignan a las palabras de código binario que produce el

codificador de fuente, una longitud fija, en el caso de que no tengan las mismas probabilidades, se encarga de eliminar información redundante de la señal de mensaje. La secuencia de símbolos resultante de la codificación de fuente se denomina palabra código de fuente.

El decodificador de fuente convierte la salida binaria del decodificador de canal en una secuencia de símbolos, en el mensaje, de manera que pueda ser interpretado por el usuario.

2.2.4 Codificación y decodificación de canal.

De acuerdo a [3] y [8], la codificación de canal toma la señal de mensaje proveniente del codificador de fuente y le añade bits de redundancia o también llamados bits de paridad, de acuerdo con una regla preestablecida, estos bits de redundancia no transmiten información, su función es permitirle al receptor detectar y/o corregir algunos errores que ocurrieron en la transmisión de la información. La secuencia de símbolos que produce el codificador de canal se denomina palabra de código de canal.

El decodificador de canal recibe la señal de mensaje y detecta y/o corrige la información a partir de códigos de decodificación de canal, generando una estimación de la señal del mensaje.

2.2.5 Modulación y demodulación digital.

El objetivo del modulador a partir de [1] y [7], es convertir la señal del mensaje en una forma adecuada de señal para la transmisión por el canal, en el caso de la modulación digital, es un proceso que transforma cada símbolo de la palabra de código de canal en una señal apta para ser transmitida en un medio físico específico. Para la transmisión de información digital en distancias cortas, se usa una modulación en banda base, llamada codificación de línea, y en distancias largas, y transmisión inalámbrica en particular, se usa la modulación pasa-banda, también conocida como modulación por portadora.

Ahora una vez definido el objetivo del modulador se procede a definir el demodulador. De [1] y [7], el objetivo del demodulador es actuar sobre la señal recibida y extraer el mensaje

de la forma de onda producida por el modulador, el proceso de demodulación es el inverso a la modulación, debido a la degradación de la señal recibida, el receptor no puede reconstruir exactamente la señal original, aunque el nivel de degradación que resulta depende del sistema de modulación que se utilice.

2.2.6 Canal de comunicación.

Según [2], el canal de transmisión es un enlace eléctrico entre el transmisor y el receptor, donde la señal se propaga en forma de una onda electromagnética. Todos los medios de transmisión tienen las siguientes características, están basados en ondas electromagnéticas, transmiten a velocidad de la luz en el medio considerado, presentan una atenuación proporcional a la distancia, están sujetos a interferencias y ruido, y son limitados en el ancho de banda sobre el que pueden transmitir.

Para establecer el medio de trasmisión para una aplicación dada, el diseñador del sistema se enfrenta a una serie de decisiones críticas, tanto el equipamiento como la propia instalación del medio de transmisión deberán satisfacer los requerimientos actuales y futuros relativos a la transmisión de datos, las características eléctricas y la topología. Dependiendo del modo de transmisión que se use, se distinguen dos grupos básicos de canales de comunicación, los que se basan en propagación guiada, como lo son los canales telefónicos, un par de conductores, cables coaxiales y fibra óptica, y los que se basan en propagación no guiada, como lo son los canales de transmisión inalámbrica, canales de radio móvil y canales de satélite.

Se describirá a continuación, que es un medio de comunicación no guiado debido a que este será el canal de comunicación para la implementación de este proyecto. Se describirá también, que es un canal discreto sin memoria, ya que el mismo también se aplicará en este proyecto.

- *Medios no guiados.* De acuerdo a [2] y [9], los medios no guiados usan un medio no físico, es decir, la transmisión y recepción de información se realiza en el espacio libre por medio de radiación electromagnética. La radiación electromagnética son las ondas electromagnéticas generadas por las cargas

eléctricas oscilantes, estas ondas se propagan a la velocidad de la luz y pueden detectarse a grandes distancias. Las ondas de radio, las cuales se utilizarán en este proyecto, son el resultado de cargas que se aceleran en alambres conductores, cuyos intervalos de longitud de onda llegan más allá de 10^4 m a casi 0,1 m, cuyas correspondientes frecuencias son desde 10^3 Hz a 10^{10} Hz, estas ondas se utilizan en radio y televisión.

- *Canal discreto sin memoria*. En [3], se afirma que el canal de la forma de onda es sin memoria si la salida del detector en un intervalo dado depende únicamente de la señal transmitida en ese intervalo y no de la transmisión previa. Bajo esta condición, es posible modelar la combinación del modulador, el canal de la forma de onda y el detector como un canal sin memoria discreto.

2.2.7 Ruido.

En [1], el termino ruido comúnmente se utiliza para identificar aquellas señales que perturban la transmisión y procesamiento de señales en los sistemas de comunicación y sobre las cuales no se tiene un control completo. El efecto no deseado del ruido se define como interferencia, el cual es la contaminación de la señal original con otras señales. El ruido que afecta a un sistema de comunicación se clasifica de acuerdo a su origen. Cuando proviene de los componentes del sistema como resistencias, dispositivos de estado sólido, tubos al vacío, componentes electrónicos, se denomina como ruido interno. La segunda categoría de ruido denominado ruido externo, se suma a la señal útil distorsionándola severamente, resulta de fuentes externas e incluye el ruido atmosférico, extraterrestre y el producido por el hombre.

Es imposible eliminar totalmente el ruido sin embargo es posible limitar su valor de manera que la calidad de la comunicación resulte aceptable, esto es posible a partir de introducir redundancia en la señal del mensaje.

2.3 RECURSOS PRIMARIOS DE LA COMUNICACIÓN.

En un sistema de comunicación se emplean dos recursos primarios, los cuales son, la potencia transmitida y el ancho de banda del canal. La potencia transmitida corresponde a la potencia promedio de la señal transmitida. El ancho de banda del canal se define como la banda de frecuencias destinadas a la transmisión de la señal del mensaje. En la mayor parte de los

canales de comunicación, uno de los recursos está más limitado que el otro y es posible clasificar estos canales como limitados en potencia o limitados en banda según como corresponda. Por ejemplo, el circuito telefónico es un típico canal limitado en banda, en tanto que los enlaces de comunicación espaciales o el canal de satélite son comúnmente limitados en potencia. La meta ideal en el diseño de un sistema de comunicación es la de transmitir información a la máxima velocidad con el mínimo de potencia y ancho de banda. De [1] y [3].

2.4 CARACTERIZACIÓN DEL CANAL DE COMUNICACIÓN.

Teniendo en cuenta las definiciones anteriores, se puede ahora definir algunas características del canal.

2.4.1 Ancho de banda del canal.

Recordando que la velocidad de fuente Vs es la velocidad con la que genera una secuencia de símbolos por segundo, la duración de cada secuencia de símbolos es de Ts segundos, entonces la duración de cada símbolo de esa secuencia tiene una duración de τ segundos. Según [1], en general, los codificadores son dispositivos comandados por un reloj, de modo que los impulsos tienen la misma duración τ.

De acuerdo a [1], desde el punto de vista eléctrico, el canal se puede considerar como un filtro que deja pasar solamente aquellas componentes de señal que están comprendidas dentro de su banda de paso o ancho de banda. La fidelidad de la salida depende tanto del ancho de banda del canal B como de la duración del impulso de entrada τ, donde se debe cumplir que:

$$B \geq \frac{1}{\tau} \tag{2.1}$$

Definiéndose la velocidad de modulación como V_b, y es igual a:

$$V_b = \frac{1}{\tau} \quad [baudio] \tag{2.2}$$

De modo que, una secuencia de impulsos cuya velocidad de modulación es V_b, puede transmitirse sin perder información por un canal ideal con un ancho de banda mínimo B_m

numéricamente igual a V_b. Es evidente que si $V_b \leq B$ donde B es el ancho de banda real del canal, no habrá problemas en la recuperación de la información. Sin embargo, si $V_b > B$, se perderá información y habría que buscar otros medios para evitar esa pérdida [1].

2.4.2 Capacidad del canal.

Para definir una medida de la eficacia con la cual un canal transmite información y para determinar su límite superior, como se mencionó en los antecedentes, Shannon introdujo el concepto de capacidad de un canal que comúnmente se representa con la letra C, y se expresa en bits por segundo (bps). A su vez Shannon introdujo el segundo teorema donde establece que si la velocidad de información Vi de la fuente es igual o menor que la capacidad C de un canal sin memoria discreto, entonces existe una técnica de codificación con redundancia que permite la transmisión sobre el canal con una frecuencia de errores arbitrariamente pequeña. De tal forma, la capacidad de canal es la máxima velocidad a la cual el canal puede transportar información confiable hasta el destinatario, en [3] y [4].

- *Capacidad del canal con ruido.* La capacidad de un canal, según [1] y [3], disminuye por los errores incurridos en la transmisión debido a señales perturbadoras o ruido, y como consecuencia se produce una pérdida de información. Shannon estableció en su segundo teorema que, si el canal tiene un ancho de banda B, la potencia promedio de la señal transmitida es S, y la potencia promedio del ruido en el canal es N, entonces la capacidad del canal en presencia de ruido aditivo y gaussiano viene dada por:

$$C = B\,log_2(1 + \frac{S}{N}) \quad [bps] \tag{2.3}$$

De la ecuación 2.3 se puede observar la importancia del ancho de banda B del sistema y la relación señal-ruido S/N presente en un sistema de comunicación. Esta ecuación señala que para una pequeña reducción en el ancho de banda B, es necesario incrementar considerablemente la potencia de la señal transmitida y, por otra parte, para un pequeño aumento en el ancho de banda B, se puede reducir considerablemente la potencia de la señal transmitida. Por lo tanto, en la práctica, el compromiso es en el sentido de reducir la potencia de transmisión a costas de

un aumento en el ancho de banda de transmisión, esto lo produce la redundancia que se agrega en la modulación y en aplicar las técnicas de codificación de canal. Esta redundancia estará limitada por C-Vi, como se indicó anteriormente, por consiguiente, la información contenida en una señal se puede recobrar aun si la relación S/N es muy pequeña, aumentando el ancho de banda B.

Aunque la ecuación 2.3 ha sido deducida para un canal gaussiano, esta es de gran importancia en los sistemas de comunicación porque muchos canales prácticos se pueden modelar como un canal gaussiano. A su vez, ha sido demostrado que el resultado obtenido para el canal gaussiano proporciona una cota superior en el funcionamiento de un sistema que opera sobre un canal no gaussiano. Por otro lado, Shannon ha demostrado que el ruido gaussiano es el peor ruido entre todos los ruidos posibles, y que la potencia del ruido gaussiano dentro de un ancho de banda dado es también la más alta de todos los ruidos posibles.

Cuando el ancho de banda de un canal está limitado por sus propias características o regulaciones, es necesario elegir un esquema de codificación de canal que optimice el rendimiento (η_B) del canal con la mínima probabilidad de error y el menor costo posible. Por consiguiente, el rendimiento máximo de un canal viene dado por:

$$\eta\text{Bmax} = \frac{C}{B} = log_2\left(1 + \frac{S}{N}\right) \quad \left[\frac{bps}{Hz}\right] \tag{2.4}$$

La teoría de Shannon no especifica cual es el mejor esquema de codificación que permita alcanzar un rendimiento máximo, pero si establece que para transmitir sin error los símbolos o muestras codificadas deben poseer una cierta redundancia. Los sistemas prácticos que tienen un rendimiento aproximado al máximo, incorporan mucha redundancia usando esquemas de codificación que incluyen codificaciones multinivel m-arias y procedimientos para la detección y/o corrección de errores.

2.5 INDICADORES DE DESEMPEÑO.

Se requieren de ciertos indicadores para evaluar el funcionamiento de un esquema de comunicaciones, estos indicadores permiten caracterizar el comportamiento del mismo y compararlo adecuadamente con otras alternativas. Algunos indicadores son los que se expondrán a continuación [10].

2.5.1 Relación señal ruido SNR.

Según [3] y [10], la salida del canal esta perturbada por ruido blanco gaussiano aditivo (AWGN) de media cero y densidad espectral de potencia $N_o/2$. Ruido blanco por analogía con la luz blanca, la cual contiene iguales cantidades de todas las frecuencias que pertenecen al espectro visible de la radiación electromagnética. Este ruido es aditivo y la señal es independiente al mismo, tal como se muestra en la ecuación 2.5, donde Y(t) es el resultado de la distorsión de la señal original X(t) por el ruido N(t).

$$Y(t) = X(t) + N(t) \tag{2.5}$$

X(t) con potencia promedio S, es acotada en el canal de banda B, con ancho de banda W, medido en Hz. Los parámetros que permiten caracterizar un canal AWGN continuo, son su ancho de banda W y su razón señal a ruido SNR. Este último está dado por:

$$SNR = \frac{S}{N} = \frac{S}{N_o W} \tag{2.6}$$

Donde N_oW es la potencia promedio del ruido dentro del ancho de banda considerado B.

2.5.2 Energía de un bit por densidad espectral.

Una manera de describir el efecto de ruido sobre señales discretas, según [3] y [10], es la razón energía-bit por densidad espectral de ruido, denotada por E_b/N_0, y corresponde a la energía portada por cada bit transmitido, en razón a la densidad espectral del ruido del canal. Se obtiene:

$$\frac{E_b}{N_o} = \frac{S}{N_o r_b} \ [dB] \tag{2.7}$$

Donde r_b es la tasa de símbolos, medida en bits por segundo, y S la potencia promedio de la señal en banda base portadora de 1 bit, de duración $T \leq T_b = 1/r_b$. Importante notar que el nivel E_b/N_0 depende tanto de la potencia de la portadora, dependiente a su vez de la modulación banda base usada para representar las secuencias binarias, como de la tasa de símbolos, siendo distinto al caso de la SNR que depende de la potencia de la señal y del ancho de banda.

2.5.3 Probabilidad de error de símbolos.

Una forma de medir el desempeño es considerar los errores cometidos en la demodulación de una señal binaria. En una transmisión de símbolos binarios en un canal AWGN, el demodulador consiste esencialmente de tres etapas, un filtro paso bajo que elimina cierta parte del ruido de entrada, un módulo de muestreo sincronizado de la señal filtrada, y un módulo de decisión de símbolos, donde se decide si el símbolo recibido corresponde a un 1 o un 0 binario, esta decisión se lleva a cabo comparando el valor de entrada de la señal con un valor denominado umbral, el cual se escoge de modo que los errores cometidos sean equiprobables para ambos símbolos binarios, este método de decisión se conoce como decisión dura, y es válido solo en condiciones de ruido gaussiano y con símbolos equiprobables [10].

Un canal para el que las probabilidades condicionales de error de los símbolos binarios son iguales se dice que será simétrico binario. En forma correspondiente, teniendo en cuenta que la energía de la señal transmitida por bit está definida por [3]:

$$E_b = A^2 T_b \tag{2.8}$$

Siendo A la amplitud del impulso y Tb la duración del impulso. La probabilidad promedio del error de símbolo será la siguiente [3]:

$$P_e = \frac{1}{2} ferc\left(\sqrt{\frac{E_b}{N_o}}\right) \tag{2.9}$$

Donde ferc es la función de error complementaria definida por la ecuación 2.10 [3].

$$ferc(u) = \frac{2}{\sqrt{\pi}} \int_{u}^{\infty} \exp(-z^2)\, dz \tag{2.10}$$

2.5.4 Tasa de errores de símbolos.

La tasa de error de símbolo es la tasa de error en el mensaje recibido en comparación con el mensaje enviado, esta tasa de error se denota como *bit error rate* (BER), al recibir una secuencia de símbolos de longitud n en el receptor, la BER se obtiene dividiendo el número de bits errados en la transmisión entre el largo total de la secuencia (n) como se indica en la ecuación 2.11 [10].

$$BER = \frac{Número\ de\ bits\ erróneos}{n} \tag{2.11}$$

Con los parámetros E_b/N_0 y BER, es posible comparar esquemas de modulación digital o codificación. Es importante señalar que, bajo un sistema digital de comunicaciones sin codificación, el BER converge a la probabilidad de error P_e dada por el esquema de modulación empleado, a medida que aumenta la cantidad de transmisiones de secuencias binarias. Pues ambos indicadores son relativos al demodulador. En un esquema de comunicaciones con codificación de canal esto no es así, puesto que el BER se mide entre la secuencia binaria inicial y la secuencia final decodificada, en tal caso se está midiendo el desempeño del decodificador y del demodulador en su conjunto [10]. En ambientes industriales, con requerimientos bajos de datos, un BER de 10^{-4} puede ser bastante aceptable (p, 125) [11].

2.6 DETECCIÓN Y CORRECCIÓN DE ERRORES.

De acuerdo a [2] y [3], en los sistemas de comunicaciones actuales el control de error se efectúa mediante la aplicación de códigos especiales que agregan redundancia. Estos códigos son de la clase de códigos de canal, esta redundancia agregada permite detectar y/o corregir los errores ocurridos durante la transmisión de los bloques de datos.

Existen dos técnicas de control de error comúnmente utilizadas las cuales son, la corrección directa de error (*Forward Error Correction*, FEC), donde se utilizan códigos para detectar y corregir errores en el receptor, solo requiere un enlace de una vía entre el

transmisor y el receptor. Y la solicitud de repetición automática (*Automatic Repeat Request*, ARQ), donde solamente se detectan la presencia de errores en los datos recibidos y se solicita de alguna forma la repetición de los bloques que tienen errores. En el presente trabajo se trabajará con la técnica FEC.

Cualquiera que sea la estrategia de control de error, FEC o ARQ, en un sistema de comunicaciones cuyas señales son digitales, el transmisor y el receptor cuentan con módulos que procesan la señal para ser enviada a través de un medio físico. En la figura 2.2 se muestra el esquema de un sistema de comunicación con codificador y decodificador, que permiten el control de error. La inclusión de redundancia en la transmisión de datos tiene como consecuencia un aumento en la velocidad de modulación de las secuencias transmitidas, lo cual produce un aumento del ancho de banda requerido y además el uso de la codificación de control de errores añade complejidad al sistema, en especial en la puesta en práctica de las operaciones de decodificación en el receptor. Se pueden definir dos parámetros de gran interés en la codificación que son, el rendimiento E del código y la redundancia R. En efecto:

$$E\% = \frac{Informacion\ realmente\ transmitida}{Maxima\ informacion\ posible\ de\ transmitir} * 100 \qquad (2.12)$$

$$R\% = (1 - E) * 100 \qquad (2.13)$$

Y la redundancia se define en la ecuación 2.13. La redundancia es agregada a cada palabra de datos o palabra mensaje y la expresión total de datos más redundancia se denomina palabra código o palabra codificada. El número de dígitos en cada palabra código es la longitud de bloque y habrá tantas palabras códigos como palabras mensaje o palabras de datos. Si la palabra mensaje tiene m dígitos y la redundancia agregada es de k dígitos, la longitud de bloque de la palabra código será de n dígitos, que corresponde con la suma de m dígitos de la palabra mensaje más k dígitos de la redundancia agregada.

2.7 TÉCNICAS DE CODIFICACIÓN DE CONTROL DE ERRORES FEC.

Existen muchos códigos diferentes de corrección de errores, FEC, que se pueden usar. Históricamente, estos códigos se han clasificado en códigos de bloque y códigos de

convolución. El rasgo definitivo para esta clasificación particular es la presencia o ausencia de memoria en los codificadores de los dos códigos [3].

2.7.1 Códigos de bloques lineales.

En [3], para generar un código de bloque (n, k), el codificador del canal acepta información en bloques sucesivos de k bits, en cada bloque agrega n-k bits redundantes que se relacionan algebraicamente con los k bits del mensaje, produciendo por ello un bloque codificado completo de n bits, donde n es mayor que k, por tanto, el código solo genera 2^k palabras de código válidas, que van a transmitir un mensaje. El bloque de n bits se denomina una palabra de código, y n es la longitud del bloque de código. El codificador de canal produce bits a razón de Ro como se indica en la ecuación 2.14, donde Rs es la tasa de bits de la fuente de información. La tasa de bits Ro, proveniente del codificador, se denomina tasa de datos del canal.

$$Ro = \left(\frac{n}{k}\right) Rs \ [\frac{bit}{segundo}] \tag{2.14}$$

La razón adimensional r, indicada en la ecuación 2.15, se conoce como la tasa de código, y está acotada entre 0 y 1. Es la relación entre el número de bits de información y el número de bits totales en la palabra de código.

$$r = \frac{k}{n} \tag{2.15}$$

Estos son códigos sin memoria, pues los bits codificados no dependen de la secuencia de bits de datos que hayan pasado por el codificador.

Un código de bloque es lineal si cualesquiera dos palabras de código en el código pueden sumarse en aritmética módulo 2 para producir una tercera palabra de código en el código. Considere entonces un código de bloque lineal (n, k), en el cual k bits de los n bits de código son siempre idénticos a la secuencia de mensaje que se va a transmitir. Los n-k bits en la porción restante se calculan a partir de los bits de mensaje de acuerdo con la regla de codificación prestablecida que determina la estructura matemática del código. Por tanto, estos n-k bits se conocen como bits de verificación de paridad generalizados o simplemente bits

de paridad. Los códigos de bloque en los cuales se transmiten los bits de mensaje en una forma inalterada se denomina códigos sistemáticos, un ejemplo de esto se puede observar en la figura 2.3, donde primero están los bits de redundancia y posteriormente los bits del mensaje como fueron enviados por la fuente. El arreglo sistemático también puede ser con los bits del mensaje primero y luego los bits de paridad. El uso de códigos sistemáticos simplifica la puesta en práctica del decodificador.

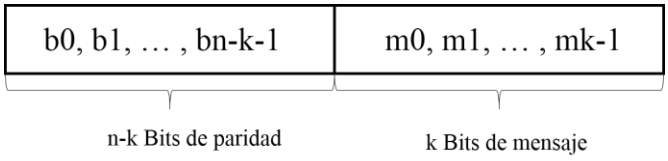


Figura 2.3. Código sistemático [3].

Una familia de códigos de bloque lineales (n, k), que tiene una longitud de bloque igual a n, siendo esta n igual a 2^m-1, un numero de bits de mensaje igual a k, y un numero de bits de paridad igual a m, siendo esta m igual a n-k, donde m≥3, son los llamados códigos de Hamming. A continuación, se definirá el código de bloques que se implementó en el presente trabajo.

- *Códigos cíclicos.* De [3], los códigos cíclicos forman una subclase de códigos de bloques lineales. En realidad, muchos de los códigos lineales importantes descubiertos hasta la fecha son cíclicos o bastante relacionados con este tipo de códigos. Una ventaja de los códigos cíclicos sobre la mayor parte de los otros tipos de códigos es que son más fáciles de codificar, por ejemplo, es más fácil que el código matricial. Además, los códigos cíclicos poseen una estructura matemática perfectamente definida, la cual ha conducido al desarrollo de esquemas de decodificación muy eficientes para ellos. Un código binario es cíclico si tiene la propiedad de linealidad, que la suma de cualesquiera dos palabras de código en el código es también una palabra de código, y la propiedad cíclica, que cualquier corrimiento cíclico de una palabra de código en el código también es una palabra de código.

  Para desarrollar las propiedades algebraicas de los códigos cíclicos, se utilizan los elementos, c0, c1, …, cn-1, de una palabra de código para definir el

polinomio del código como indica la ecuación 2.16, donde X es una indeterminación.

$$c(X) = c_0 + c_1X + c_2X^2 + \cdots + c_{n-1}X^{n-1} \quad (2.16)$$

Naturalmente para los códigos binarios, los coeficientes son unos y ceros. Cada potencia de X en el polinomio c(X) representa un corrimiento a tiempo de un bit. Por tanto, la multiplicación del polinomio c(X) por X puede verse como un corrimiento hacia la derecha.

El polinomio X^n+1 y sus factores desempeñan un papel fundamental en la generación de códigos cíclicos, advierta que en la aritmética de módulo 2, X^n+1 tiene el mismo valor que X^n-1. A partir de este polinomio se puede obtener el polinomio generador g(X) factorizando este polinomio a un polinomio irreductible, como se observa en la ecuación 2.18.

$$X^n - 1 = g(X)h(X) \quad (2.17)$$

$$g(X) = 1 + \sum_{i=1}^{n-k-1}(g_i\ X^i) + X^{n-k} \quad (2.18)$$

La ecuación 2.17 indica que el polinomio X^n-1 es igual al producto entre el polinomio generador g(X) que es de grado n-k, y el polinomio de chequeo de paridad h(X) de grado k, siendo k la longitud del mensaje, y n la longitud de la palabra de código que se desea obtener. La ecuación 2.18, indica la forma que debe tener el polinomio g(X) para que sea irreductible, tomando el coeficiente g_i el valor de 0 o 1, como corresponda, y g_0 y g_{n-1} el valor de 1.

Ahora suponiendo que dan el polinomio generador g(X) y el requerimiento es codificar la secuencia de mensaje (m_0, m_1, …, m_{k-1}) en un código cíclico sistemático (n, k). Es decir, los bits de mensaje se transmiten en forma inalterada, como se indica en la figura 2.3. Se define el polinomio de mensaje como la ecuación 2.19.

$$m(X) = m_0 + m_1X + \cdots + m_{k-1}X^{k-1} \quad (2.19)$$

Y el polinomio b(X) que contiene los bits de redundancia se define como la ecuación 2.20.

$$b(X) = b_0 + b_1X + \cdots + b_{n-k-1}X^{n-k-1} \tag{2.20}$$

Como el código es sistemático, la estructura de código puede ser como la ecuación 2.21, para el caso de que primero se encuentren los bits de redundancia y posteriormente se encuentren los bits de mensaje:

$$c(X) = b(X) + m(X)X^{n-k} \tag{2.21}$$

Si se desea ubicar primero los bits del mensaje y luego los bits de redundancia, la estructura de la palabra de código es la siguiente:

$$c(X) = m(X) + b(X)X^{k} \tag{2.22}$$

Para determinar el polinomio de los bits de redundancia para el caso de la ecuación 2.21, se realiza el siguiente procedimiento. Primero se debe generar el polinomio $m(X)X^{n-k}$ y luego se divide entre g(X) y el residuo de esta división es el polinomio de los bits de redundancia b(X).

Cuando se obtiene el polinomio de los bits de redundancia b(X), se suma este polinomio a $X^{n-k}m(X)$ y se obtiene el polinomio de código c(X). Este proceso lo realiza el codificador de canal.

Una vez que se envía la información, el polinomio de código c(X), y la recibe el receptor, el decodificador de canal debería recibir c(X), pero por el ruido se recibe r(X), que está conformado por c(X) y el error producido a lo largo de la transmisión, e(X), como:

$$r(X) = c(X) + e(X) \tag{2.23}$$

Para obtener el error correspondiente, se debe calcular el síndrome, el cual depende únicamente del error y no de la palabra de código transmitida. Si el síndrome es cero, no hay errores de transmisión en la palabra recibida. Si, en cambio, el síndrome es distinto de cero, la palabra recibida contiene errores de

transmisión que son necesarios corregir. En el caso de un código cíclico en la forma sistemática, el síndrome puede calcularse con facilidad. Para la palabra recibida de la ecuación 2.24.

$$r(X) = r_0 + r_1X + \cdots + r_{n-1}X^{n-1} \tag{2.24}$$

El síndrome se obtiene de dividir el polinomio r(X) entre el polinomio generador g(X), el residuo de esta división es el polinomio del síndrome s(X), si este es diferente de 0, hay errores en el polinomio de la palabra de código transmitida. El polinomio del síndrome s(X) tiene como propiedad, que el síndrome de un polinomio de palabra recibida es también el síndrome del correspondiente polinomio de error e(X), por tanto, se debe calcular el síndrome de todos los posibles errores que puedan ocurrir y que pueda corregir el decodificador, para que cuando se obtenga el síndrome del polinomio de la palabra recibida se busque el error que corresponda al mismo síndrome. Y por tanto ese error se sume al polinomio de la palabra recibida y se obtenga el polinomio de la palabra de código sin errores, como se observa a continuación:

$$r(x) = c(X) + e(X) + e(X) = c(X) \tag{2.25}$$

El polinomio de error e(X) es de grado n. Este código tiene limitaciones en cuanto al número de errores t que puede corregir, y está definido en la ecuación 2.26. Esta ecuación depende de la distancia mínima (dmin), que indica el mínimo número de errores necesarios para convertir una palabra de código en otra.

$$t \leq \frac{1}{2}(dmin - 1) \tag{2.26}$$

La distancia mínima se obtiene a partir de calcular la distancia de Hamming. La distancia de Hamming básicamente calcula el número de componentes en los cuales son diferentes cualquiera de las palabras de código válidas con otra palabra de código válida, por tanto, para obtener la menor distancia de Hamming de cada palabra de código se tendría que calcular las distancias de Hamming con respecto a todas las palabras de código válidas. Una manera más simple de obtener la distancia mínima es la siguiente. Se compara cada palabra de código

válida con la palabra de código cero, y el menor valor que se obtiene de todas, es la distancia mínima. Con el valor de t, se generan todos los posibles patrones de error que pueden ocurrir en el sistema de transmisión, pero solo aquellos que correspondan con la cantidad de errores que puede corregir el decodificador de canal, es decir que, si la cantidad de errores da uno, se generaran patrones de error con solo un error.

Y posteriormente como ya se mencionó, se calculan los síndromes de todos los patrones de errores que se generaron. El patrón de error que tenga el mismo síndrome que el síndrome de la palabra de código recibida, se sumará a la palabra de código recibida y con esto se obtendrá la palabra de código recibida sin errores, como lo expresa la ecuación 2.25. En el caso de que ningún síndrome corresponda con el síndrome de la palabra de código recibida, entonces el número de errores que tiene la palabra código es superior al número de errores que puede corregir el decodificador, como solución a esto se deberá sumar el patrón de error cero a la palabra de código recibida, ver ecuación 2.27, con el fin de evitar agregar más errores a la palabra código recibida.

$$e(X) = 0 + 0X + \cdots + 0X^n \tag{2.27}$$

2.7.2 Códigos convolucionales.

Según [3], en el codificador de bloques, el codificador acepta un bloque de mensaje de k bits y genera una palabra de código de n bits. De esa manera, las palabras de código se producen en un esquema bloque por bloque, deben tomarse previsiones en el codificador para retener en un búfer un bloque de mensaje completo antes de generar la palabra de código asociada. Sin embargo, hay aplicaciones en las que los bits de mensaje vienen en forma serial y no en bloques grandes, en cuyo caso el uso de un búfer puede ser innecesario. Un codificador convolucional genera bits redundantes utilizando convoluciones módulo 2. Estos son códigos con memoria, pues los bits codificados dependen de la secuencia de bits de datos que hayan pasado por el codificador.

El codificador de un código convolucional, puede considerarse como una máquina de

estado finito que consiste en un registro de corrimiento de M etapas con conexiones preestablecidas a n sumadores módulo 2 y a un multiplexor que pone en serie las salidas de los sumadores. Una secuencia de L bits produce una secuencia de salida codificada de longitud n(L+M) bits. La tasa de código está dada entonces por:

$$r = \frac{L}{n(L+M)} \quad \left[\frac{bits}{símbolo}\right] \tag{2.28}$$

Siendo n el número de bits de salida, M es el número de elementos de memoria. Por lo general L>>M, por lo tanto, la tasa de código se simplifica a:

$$r \simeq \frac{1}{n} \quad \left[\frac{bits}{símbolo}\right] \tag{2.29}$$

La longitud de restricción de un código de convolución, expresada en términos de bits de mensaje, se define como el número de corrimientos sobre el cual un bit de un solo mensaje puede influir en la salida del codificador. En un codificador con un registro de corrimientos de M etapas, la memoria del codificador es igual a M bits de mensaje y se requieren K=M+1 corrimientos para que un bit de mensaje entre al registro de corrimiento y finalmente salga. Así, la longitud de restricción del codificador es igual a K. La figura 2.4 señala un codificador convolucional con dos salidas, n=2, de longitud de restricción igual a tres, K=3, y de tasa de código de un medio, r=1/2. En el caso de los códigos convolucionales es preferible el uso de códigos no sistemáticos, donde se transmiten los bits del mensaje en una forma alterada.

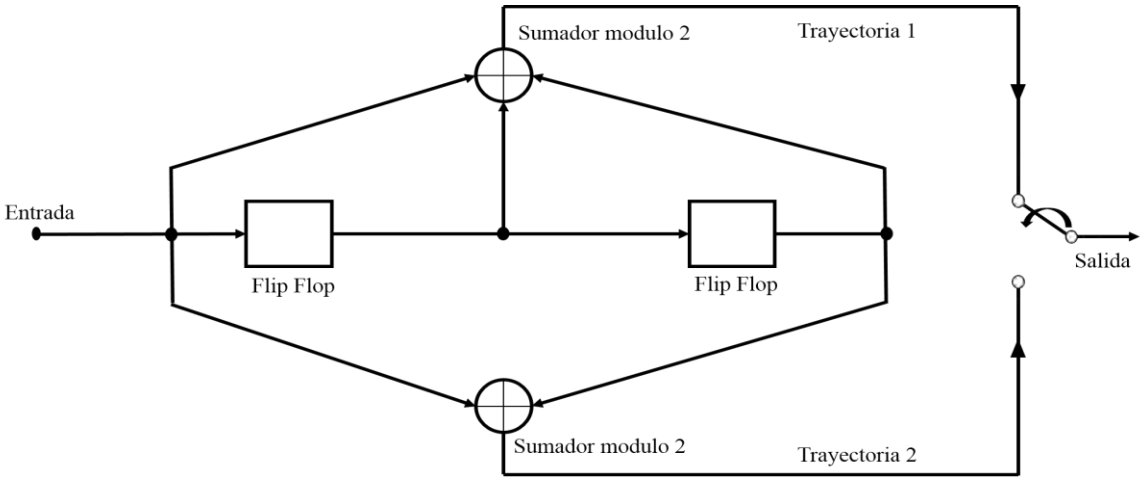


Figura 2.4. Codificador convolucional de tasa ½ y longitud de restricción 3 [3].

Cada trayectoria que conecta la salida con la entrada de un codificador convolucional puede caracterizarse en términos de su respuesta al impulso, definida como la respuesta de esa trayectoria a un símbolo 1 aplicada en su entrada, con cada flip flop en el conjunto codificador inicialmente en el estado cero. De manera equivalente, se puede caracterizar cada trayectoria en términos de un polinomio generador, definido como la transformada de retorno unitario de la respuesta al impulso. Específicamente, dejemos que la secuencia generadora ($g_0^{(i)}$, $g_1^{(i)}$, $g_2^{(i)}$,…, $g_M^{(i)}$) denote la respuesta al impulso de la i-ésima trayectoria, donde los coeficientes $g_0^{(i)}$, $g_1^{(i)}$, $g_2^{(i)}$,…, $g_M^{(i)}$ son iguales a 0 o 1. De manera correspondiente, el polinomio generador de la i-ésima trayectoria está definido por:

$$g^{(i)}(D) = g_0{}^{(i)} + g_1{}^{(i)}D + g_2{}^{(i)}D^2 + \cdots + g_M{}^{(i)}D^M \qquad (2.30)$$

Donde D denota la variable de retardo unitario. El codificador completo se describe mediante un conjunto de polinomios generadores {$g^{(1)}(D)$, $g^{(2)}(D)$, …, $g^{(n)}(D)$}.

- *Técnicas de codificación convolucional.* Tradicionalmente, las propiedades estructurales de un codificador convolucional se representan en forma gráfica utilizando uno de los siguientes tres diagramas, el árbol de código, el enramado o el diagrama de estado [3]. Para efectos del presente trabajo es conveniente utilizar la implementación basada en el diagrama de estado del codificador, y el enramado como técnica de decodificación, por tanto, se detallarán estos métodos a continuación.

  - *Enramado o Trellis:* en la figura 2.5 se tiene el enramado del codificador convolucional de la figura 2.4, en este codificador se tiene una longitud de restricción igual a 3, K=3, y dos elementos de memoria, M=2, el estado de este codificador puede asumir uno cualquiera de 2^M valores posibles, es decir, 4 valores, el valor 00 que será el estado a, el valor 10 será el estado b, el valor 01 será el estado c, y finalmente el valor 11 será el estado d. Los nodos del enramado representan los estados presentes del codificador. El enramado contiene (L+K) niveles, donde L es la longitud de la secuencia de mensaje entrante. Los niveles del enramado se indican como j=0, 1, …, L+K-1. Los primeros (K-1) niveles corresponden a la

desviación del codificador a partir del estado inicial a. El estado a es 00, es lo que tienen los elementos de memoria al iniciar el enramado, y los últimos (K-1) niveles corresponden al regreso del codificador al estado a. La convención utilizada para describir entre los símbolos de entrada 1 y 0 es la siguiente, una rama de código producida por una entrada 0 se dibuja como una línea continua, en tanto que una producida por una entrada 1 se traza como una línea punteada. Como antes, cada secuencia de entrada (mensaje) corresponde a una trayectoria específica en el enramado, se sigue de izquierda a derecha, y los símbolos codificados correspondientes sobre las trayectorias constituyen la secuencia de salida [3].

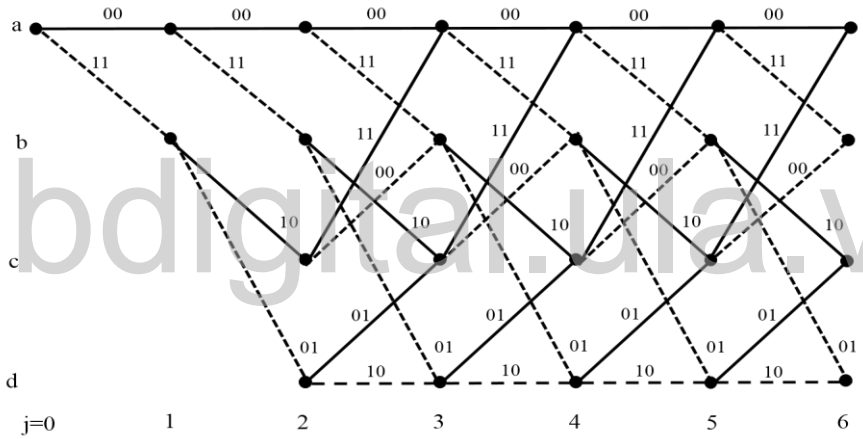


Figura 2.5. Enramado del codificador convolucional de la figura 2.4 [3].

- *Diagrama de estado:* En la figura 2.6 se presenta el diagrama de estados del codificador convolucional de la figura 2.4. En este esquema se realiza el diagrama de estado del codificador convolucional. En el caso del codificador de la figura 2.4, este tiene 2 elementos de memoria, por lo tanto, tiene 4 estados que serán el estado 00 representado por la letra a, el estado 10 representado por la letra b, el estado 01 representado por la letra c y el estado 11 representado por la letra d. Los nodos del diagrama representan los estados presentes del codificador, teniendo cada nodo dos ramas entrantes y dos ramas salientes. [3].

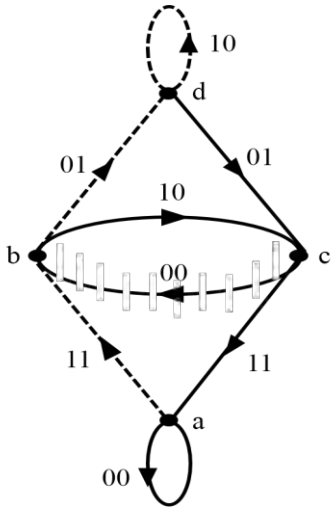


Figura 2.6. Diagrama de estados del codificador convolucional de la figura 2.4 [3].

Una transición de un estado a otro en respuesta a la entrada 0 se representa mediante una rama continua, en tanto que una transición en respuesta a una entrada 1 se ilustra mediante una rama punteada. La marca binaria sobre cada rama representa la salida del codificador cuando esta se mueve de un estado a otro. Para obtener la secuencia de salida, se empieza con el estado inicial de puros ceros que es el estado a, y se continua a través del diagrama de estado de acuerdo con la secuencia de mensaje. Seguimos una rama continua si la entrada es cero y una rama punteada si esta es uno. A medida que se recorre cada rama, se genera la marca binaria correspondiente sobre ellas. Para sintetizar, primero se ubica en el estado inicial y se introduce la secuencia de entrada siguiendo las transiciones del diagrama de estado y cada una de las transiciones especifica los símbolos codificados correspondientes de la secuencia de salida [3].

- *Técnicas de decodificación convolucional.* Ahora teniendo en cuenta cómo funciona el codificador convolucional, el siguiente aspecto a considerar es la decodificación del mismo, debido a esto, se procederá a hablar del algoritmo de Viterbi, a partir de [3].

Se puede observar en la figura 2.5 una representación del algoritmo de Viterbi para decodificar las secuencias de código que genera el codificador de la figura 2.4, en este caso se realiza un enramado, en el cual los símbolos codificados que corresponden a cada trayectoria, ubicados en la parte superior de cada trayectoria, se comparan con los bits de la secuencia de entrada, de dos bits en dos bits, debido a que el codificador de la figura 2.4 que es el que se usó para el enramado, tiene 2 salidas. Por tanto, el algoritmo opera al calcular una métrica o discrepancia para cada trayectoria posible en el enramado respecto a la secuencia de entrada. La métrica para una trayectoria particular se define como la distancia de Hamming entre la secuencia codificada representada por esa trayectoria y la secuencia recibida.

En cada nodo (estado) del enramado de la figura 2.5, el algoritmo compara las dos trayectorias que entran al nodo. Se retiene la trayectoria con métrica inferior y se descarta la otra. Este cálculo se repite para cada nivel j del enramado en el intervalo $M \leq j \leq L$, donde M es la memoria del codificador, y L es la longitud de la secuencia de mensaje entrante. Las trayectorias que retienen el algoritmo reciben el nombre de trayectorias sobrevivientes o activas. Para un código convolucional de longitud de restricción $K=3$, por ejemplo, nunca se almacenará más de $2^{k-1}=4$ trayectorias sobrevivientes y sus métricas. Se garantiza siempre que esta lista de 2^{k-1} trayectorias contendrá la elección de máxima verosimilitud.

El algoritmo continua hasta que completa su búsqueda hacia adelante, a través del enramado, y por tanto llega al nodo de terminación, que es el estado de puros ceros, en ese momento toma una decisión acerca de la trayectoria de máxima verosimilitud. Entonces, al igual que un decodificador de bloque, la secuencia de símbolos asociada con una trayectoria se libera hacia el destino como la versión decodificada de la secuencia recibida, en este sentido resulta entonces más correcto referirse a este algoritmo como un estimador de secuencia de máxima verosimilitud. Una dificultad que puede surgir en la aplicación del algoritmo de Viterbi es la posibilidad de que cuando se comparan las trayectorias que entran

a un estado, se encuentra que sus métricas son idénticas. En esta situación, se hace simplemente una suposición.

Sin embargo, cuando la secuencia recibida es muy larga, el requerimiento de almacenamiento del algoritmo de Viterbi se vuelve demasiado alto y debe establecerse algunos compromisos. El método que suele sugerirse consiste en truncar la memoria de la trayectoria del decodificador, a partir de especificar una ventana de decodificación de longitud l y el algoritmo opera sobre una trama correspondiente de la secuencia recibida, interrumpiéndose siempre después de l pasos. Se realiza entonces una decisión respecto a la mejor trayectoria y se libera al usuario el símbolo asociado con la primera rama sobre la trayectoria. Se elimina el símbolo asociado con la última rama de la trayectoria. Y luego la ventana de decodificación se mueve hacia adelante un intervalo de tiempo y se toma una decisión sobre la siguiente trama de código, etc. Las decisiones de decodificación que se llevan a cabo de esta manera ya no son en verdad de máxima verosimilitud, pero pueden ser casi tan buenas siempre que la ventana de decodificación sea suficientemente larga. La experiencia y el análisis han demostrado que se obtienen resultados satisfactorios si la longitud de la ventana de decodificación l es del orden de 5 veces la longitud restringida K del código convolucional o mayor.

El desempeño de un código convolucional no depende únicamente del algoritmo de decodificación utilizado, sino también de las propiedades de distancia del código. La medida más importante de la capacidad de un código convolucional para combatir el ruido del canal es la distancia libre, denotada por dlibre, y esta se define como la distancia de Hamming mínima entra cualesquiera dos palabras de código en el código. Según la ecuación 2.31, un código convolucional con distancia libre dlibre, puede corregir t errores si y solo si dlibre es mayor que 2t, donde t es un numero entero. Esta distancia libre puede obtenerse en forma bastante simple a partir del diagrama de estado del codificador convolucional. A partir de la figura 2.7, la cual enseña el diagrama de estado del codificador de la figura 2.4, cualquier secuencia de código distinta

de cero corresponde a una trayectoria completa que empieza y termina en el estado 00, por esta razón, se puede dividir este nodo de la manera que se ilustra en el diagrama de estado de la figura 2.7. Si se aprecia la figura 2.7, el exponente de D sobre una trayectoria en ese diagrama describe el peso de Hamming de la salida del codificador correspondiente de esa trayectoria. La distancia libre será el recorrido del estado a0 al estado a1 más corto, pasando por la menor cantidad de nodos.

$$t \leq \frac{d_{libre}}{2} \tag{2.31}$$

Ahora, sumando los pesos de Hamming de cada una de las trayectorias que se necesiten para cumplir el recorrido del estado inicial a0 al final a1 más corto será la distancia libre del codificador. Expresado de otra manera, la distancia libre se obtiene si se realiza el recorrido que implique la menor cantidad de nodos del estado inicial a_0 al estado final a_1, la cantidad de unos en este recorrido será la distancia libre. El diagrama de estado para obtener la distancia libre del codificador de la figura 2.4 se encuentra en la figura 2.7.

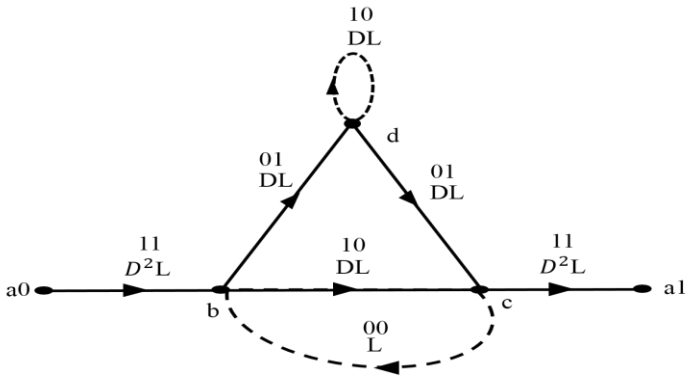


Figura 2.7. Diagrama de estado para obtener la distancia libre del codificador de la figura 2.4 [3].

Este diagrama indica que la distancia libre del codificador convolucional de la figura 2.4 es igual a 5, por consiguiente, este código puede corregir hasta dos errores en la secuencia recibida, ya que dos o un número menor de errores de transmisión provocarán que la secuencia recibida se encuentre a lo más a una

distancia de Hamming de 2 de la secuencia transmitida, aunque a menos de una distancia de Hamming de 3 desde cualquier otra secuencia de código. A pesar de la presencia de cualquier par de errores de transmisión, la secuencia recibida permanece más cercana a la secuencia transmitida que cualquier otra posible secuencia de código. Sin embargo, este enunciado deja de ser válido si hay tres o más errores de transmisión espaciados a muy corta distancia en la secuencia recibida, patrones de errores triples o superiores para que sean corregibles deben estar distribuidos por un periodo más largo que una longitud de restricción.

2.8 ENTRELAZADOR.

De acuerdo a [7] y [12], los entrelazadores son estructuras que reordenan los datos, se encuentran en diversas aplicaciones, tales como la concatenación de codificadores con el fin de aumentar la diversidad temporal del código, ya que muchos codificadores son diseñados considerando únicamente errores aleatorios e independientes entre sí, pero cuando se está en presencia de un canal que genera ráfagas de error, esto no es válido. Estos errores de ráfaga son indeseables en los códigos de convolución debido a que estos pueden corregir cierta cantidad de errores en una longitud de restricción definida. Si el entrelazador dispersa estos errores, el código convolucional puede corregir la mayoría de los errores que ocurrieron en la transmisión. La mayor desventaja de los entrelazadores es que retrasan la transmisión de la señal, debido a que para decodificar una palabra de código completa se debe esperar a que todos los símbolos de la misma, esparcidos en el tiempo, lleguen al extremo receptor. Además, naturalmente agregan complejidad al sistema.

Existen varias formas de lograr el entrelazado, tales como entrelazadores convolucionales y de bloque. A continuación, se definirá brevemente el entrelazador de bloque que se usará en la implementación de este trabajo de grado.

2.8.1 Entrelazador de bloque.

En [7], esta clase de entrelazadores la permutación se realiza utilizando una matriz de A x B elementos que son guardados y leídos de distinta manera, realizando así el entrelazado. Una

estrategia usual, denominada entrelazado matricial consiste en guardar los datos provenientes del codificador de canal en columnas y luego el decodificador de canal los lee por filas.

Esta etapa de entrelazado podría ser agregada en el transmisor entre la etapa del codificador de canal y el modulador, y a su vez se debe agregar una etapa de desentrelazado en el receptor, entre el modulador y el decodificador de canal, esto con el fin de evitar errores de ráfaga que puedan ocurrir en la transmisión de información por el canal.

2.9 RASPBERRY PI.

La implementación del sistema de comunicación de este trabajo de grado se realiza con dos Raspberry Pi 1 modelo B, una como el emisor y otra como el receptor del sistema de comunicación, a pesar de que existen más modelos se usa este modelo debido a que es el que se tiene a disposición y además cumple con el alcance de este trabajo de grado. A continuación, se procederá a definir que es una Raspberry Pi y las características de la Raspberry Pi 1 modelo B.

La Raspberry Pi según [13] y [14], es una computadora de placa reducida, del tamaño de una tarjeta de crédito teniendo todo lo que se necesita en la misma, siendo de bajo costo, permitiendo estimular la enseñanza de la informática en las escuelas debido a que con este dispositivo se pueden programar diversos proyectos, utilizando lenguajes de programación como Python, C, Java, Perl, BBC BASIC, entre otros. El concepto es el de un ordenador excluyendo todos los accesorios que se pueden eliminar sin que afecte al funcionamiento básico, de igual manera la placa soporta varios componentes necesarios en un ordenador común, como una pantalla, un teclado y un mouse, ya que dicha placa tiene varios puertos.

Según [14], la Raspberry Pi no incluye memoria, debido a esto, este modelo en su parte inferior cuenta con un lector de tarjetas SD que da la posibilidad de instalar un sistema operativo en una tarjeta de memoria de 4 GB o más. Su sistema operativo oficial usa una versión adaptada de Debian, denominada Raspbian, aunque permite usar otros sistemas operativos, incluido una versión de Windows 10 [13]. La Raspberry Pi 1 modelo b, tiene un chip integrado BCM2835, y procesador ARM11 ARM1176JZF-S con frecuencia de funcionamiento de 700 MHz. Tiene un procesador gráfico VideoCore IV. Además este

modelo cuenta con 512 MB de RAM, con un puerto Ethernet, posee 26 conectores GPIO (*General Purpose Input/Output*) o pines de entrada y salida de propósito general que permiten la conexión con el exterior, una salida de vídeo y audio vía HDMI, con lo que se consigue conectar la tarjeta tanto a televisores como a monitores que cuenten con dicha conexión, una salida analógica de Video RCA, también cuenta con una salida analógica de audio estéreo a través de un conector Jack de 3.5 milímetros, dos conectores USB 2.0, un conector MicroUSB de alimentación, ya que requiere de una fuente de alimentación de 5 voltios y 2 amperios.

La tensión de trabajo de los puertos de GPIO es de 3,3 VDC para un 1 y 0 VDC para un 0 y la corriente máxima que puede suministrar es de 16 mA, pero en realidad esta fue diseñada para una carga de unos 3 mA por cada pin GPIO, es decir, cada pin ofrece una corriente hasta 3 mA, y puede consumir hasta 16 mA por pin. Además, se pueden configurar interfaces complejos de entrada/salida por lo que algunos pines pasan a ser gestionados directamente por el chip del procesador, como I^2C, UART y SPI, según [15] y [16]. De la figura 2.8, se puede observar una de las Raspberry Pi 1 modelo B que se tiene a disposición para el proyecto, con las indicaciones de la ubicación de sus puertos.

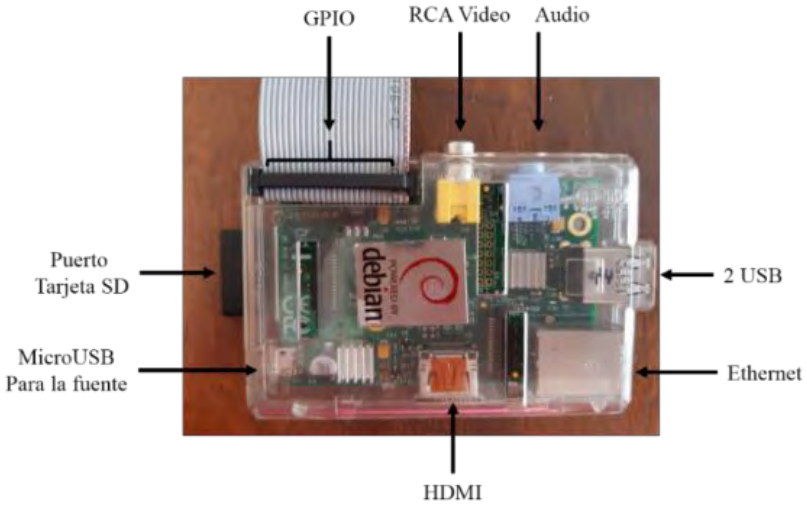


Figura 2.8. Raspberry Pi 1 modelo b.

La figura 2.9 indica la disposición de los puertos de entrada y salida de propósito general o GPIO de la Raspberry Pi 1 modelo B. El puerto cuadrado de color amarillo de la figura 2.9

acepta una alimentación de 3.3 V. EL puerto de color rojo acepta una alimentación de 5 V. Los puertos de color gris son pines reservados. El puerto de color negro es para la conexión a tierra GND. Los puertos de color verde son entradas y salidas de propósito general, pueden configurarse como entradas o salidas, teniendo presente que pueden generar y consumir hasta 3.3 V. Los puertos de color amarillo indicados en la figura 2.9 con un círculo, están destinados a conexión UART para puerto serie convencional. Y finalmente los puertos de color morado son para la comunicación mediante el protocolo SPI, para comunicarse con periféricos que siguen este protocolo [15].

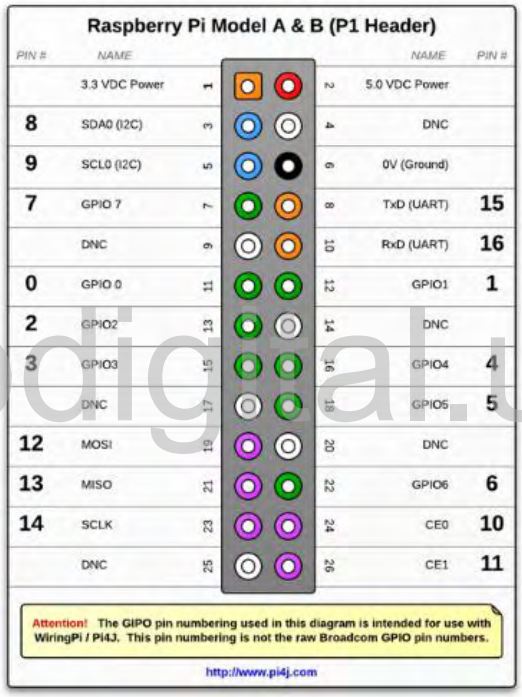


Figura 2.9. GPIO de la Raspberry Pi 1 modelo B [14].

2.10 MÓDULO RF 433 MHZ.

Según [17], [18] y [19], el módulo de radio frecuencia RF 433 MHz es un transmisor y un receptor inalámbricos usados para la comunicación entre dos microcontroladores. Está compuesto por dos placas, el transmisor FS1000A y el receptor XT-MK-5V. Para la programación no es necesario agregar ningún tipo de librería, ya que tiene un proceso

transparente, por lo que el programa sería igual a usar una comunicación serial UART entre dos microprocesadores.

La comunicación es simplex, es decir, solo se transmite la información en una dirección, del transmisor al receptor, y se realiza básicamente por modulación ASK, en donde la amplitud de la onda portadora, cambia de acuerdo a la señal de datos entrante. Por ejemplo, cuando el transmisor recibe en su pin de datos una lógica alta, el oscilador del transmisor funciona generando una corriente alterna que oscila a una frecuencia de radio de 433 MHz, siendo esta la onda portadora del sistema de comunicación, ahora cuando el transmisor recibe en su pin de datos una lógica baja, el oscilador se detiene por tanto la portadora no tiene frecuencia.

Luego, en el caso de que exista una portadora, esta se aplica a la antena, la corriente oscilante empuja los electrones en la antena hacia adelante y hacia atrás, creando campos eléctricos y magnéticos oscilantes, que irradian la energía lejos de la antena como ondas de radio, estas ondas de radio llevan la información a la ubicación del receptor. En el receptor, los campos eléctricos y magnéticos oscilantes de la onda de radio entrante empujan los electrones de la antena receptora hacia adelante y hacia atrás, creando una pequeña tensión oscilante que es una réplica más débil de la corriente en la antena transmisora, esta tensión se aplica al receptor de radio, que extrae la señal de información.

Las características del transmisor del módulo RF 433 MHz son las siguientes:

- Voltaje de operación: 3 V a 12 V.
- Corriente de trabajo: 9 mA a 40 mA.
- Modulación: ASK.
- Frecuencia de operación: 433 MHz.
- Potencia: 10 mW con voltaje de 5 V y 25 mW con voltaje de 12 V.
- Error de frecuencia: ± 150 kHz.
- Velocidad de transmisión: < 10 kbps.
- Pines de salida: GND, DATA, VCC y Antena.
    - GND, es un pin de tierra.
    - DATA, es el pin que acepta los datos digitales para ser transmitidos.

  - VCC, es el pin que suministra energía al transmisor. Puede ser un voltaje DC positivo entre 3 V y 12 V.
  - Antena, es el pin para una antena externa.
- Con la alimentación de 5 V y con la antena del módulo, el alcance difícilmente excederá de los 2 m. Alimentando a 12 V y con una antena de cobre de 17 cm el rango en exteriores puede alcanzar 50 m.

En el caso de necesitar usar una antena externa, es importante considerar que la antena más efectiva tiene la misma longitud que la longitud de onda para la que se usa, pero para fines prácticos es suficiente la mitad o un cuarto de esa longitud.

La longitud de onda de una frecuencia se calcula como:

$$Longitud\ de\ onda = \frac{Velocidad\ de\ transmision(m/s)}{Frecuencia\ de\ transmision(Hz)} \quad (2.32)$$

La velocidad de transmisión en el aire es la misma que la velocidad de la luz. Para la banda de 433 MHz, la longitud de onda es igual a 0,692 m, y para fines prácticos se puede usar una antena de un cuarto de onda de 0,173 m.

Las características del receptor del módulo RF 433 MHz son las siguientes:

- Voltaje de operación: 4,5 V a 5,5 V.
- Corriente de operación: 4 mA.
- Modulación: ASK.
- Frecuencia de operación: 433 MHz.
- Ancho de banda: 2 MHz.
- Principio de funcionamiento: Receptor Superregenerativo.
- Velocidad de transmisión: < 9,6 kbps.
- Pines: GND, 2 pines de DATA, VCC y Antena.
  - GND, es un pin de tierra.
  - DATA, estos pines emiten los datos digitales recibidos. Los dos pines de DATA están unidos internamente, por lo que se puede usar cualquiera de ellos para la salida de datos.

- VCC, es el pin que suministra energía al receptor.
- Antena, es el pin para una antena externa. En el caso de requerir un alcance mayor de 2 m, entonces, es conveniente soldar una antena de 17,3 cm al igual que en el transmisor.

En la figura 2.10, se puede observar el módulo RF 433 MHz que se usa en la implementación de este trabajo de grado, el dispositivo en la parte derecha de la figura es el transmisor FS1000A y el dispositivo en la parte izquierda de la figura es el receptor XT-MK-5V.

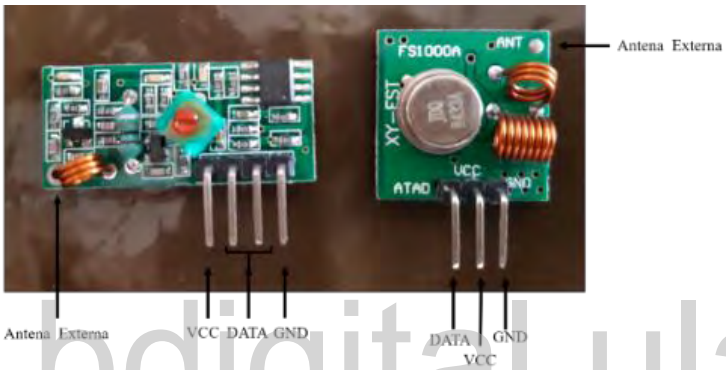


Figura 2.10. Módulo RF 433 MHz.

CAPÍTULO III
METODOLOGÍA

En el presente capítulo se encuentra la metodología aplicada en la implementación del codificador de canal concatenado.

3.1NIVEL Y DISEÑO DE INVESTIGACIÓN.

El presente proyecto está enmarcado dentro de la modalidad de proyecto de desarrollo tecnológico de carácter innovador, orientado a la implementación de un esquema de codificación de canal robusto. Así mismo, la investigación se proyecta inicialmente como un estudio de nivel de investigación exploratoria, donde se consultan diferentes monografías y trabajos de investigación referentes a la codificación de canal y todos los términos que la describen para posteriormente proceder a un nivel de investigación descriptiva.

El diseño de la investigación de este proyecto es experimental. La investigación experimental es un proceso que consiste en someter a un objeto o grupo de individuos, a determinadas condiciones, estímulos o tratamientos (variable independiente), para observar los efectos o reacciones que se producen (variable dependiente). Con este diseño de investigación, se implementó un sistema de comunicación entre dos Raspberry Pi y posteriormente, se agregaron técnicas de codificación de canal en las mismas para observar el aporte de estas técnicas al sistema de comunicación implementado, fue posible medir este aporte a partir de la obtención del valor de la BER en cada secuencia enviada.

3.2IMPLEMENTACIÓN DE UN CODIFICADOR DE CANAL ROBUSTO.

Para realizar la implementación del codificador de canal robusto, se hizo uso de una Raspberry Pi como emisor y otra Raspberry Pi como receptor, a su vez se hizo uso de un

módulo RF 433 como canal entre las dos Raspberry Pi, este último permitió el envío de información de manera inalámbrica entre ambas Raspberry Pi. La figura 3.1 señala el sistema de comunicación que se implementó en el presente trabajo de grado. El apéndice A.1 contiene una fotografía del sistema de comunicación implementado en el presente trabajo de grado. La experiencia se realizó en el entorno de una vivienda.

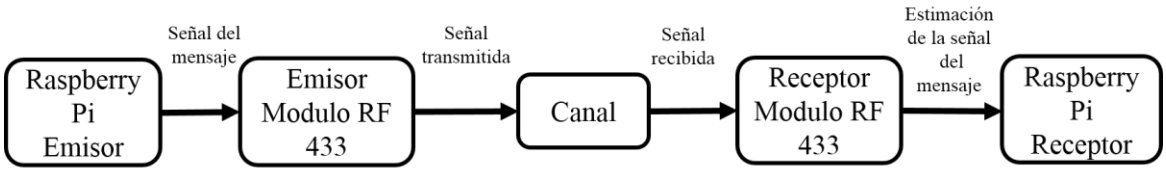


Figura 3.1. Sistema de comunicación a implementar.

Ahora, como técnica robusta de codificación de canal para corregir los errores que ocurren en la transmisión de información entre las dos Raspberry Pi, se programó en cada Raspberry Pi en lenguaje de programación C, un código donde se concatenaron en serie dos técnicas de codificación de canal, siendo estas técnicas, el código de bloques (7,4) y el código de convolución señalado en la figura 2.4. Esta técnica se seleccionó por su simplicidad de programación y porque corrige 1 error por palabra de código. Y el código convolucional, el cual se seleccionó por las conclusiones que obtuvieron los estudios recientes de los antecedentes del capítulo 2. Estos estudios indicaron que el código de convolución es muy efectivo para corregir errores, además que pueden corregir una buena cantidad de errores mientras que los errores estén espaciados a una distancia superior a la longitud de restricción.

Ahora, se dividirá la implementación en etapas, para así observar el aporte de cada una de las técnicas de codificación de canal al sistema de comunicación de la figura 3.1.

3.2.1 Primera etapa de la implementación: Sin codificación de canal.

En la primera etapa, se usó inicialmente como canal de comunicación, cables. Este canal permitió la conexión entre el puerto destinado al envío de datos de la Raspberry Pi emisor y el puerto de recepción de datos de la Raspberry Pi receptor. Posteriormente, se realizó el sistema de comunicación mostrado en la figura 3.1, usando como canal de comunicación un medio inalámbrico, es decir, se conectó a los puertos de envío y recepción de las Raspberry

Pi, el módulo RF 433 como corresponde, para así observar la influencia del mismo al sistema de comunicaciones. El diagrama de flujo que se aplicó a la Raspberry Pi emisor en la primera etapa es el señalado en la figura 3.2.

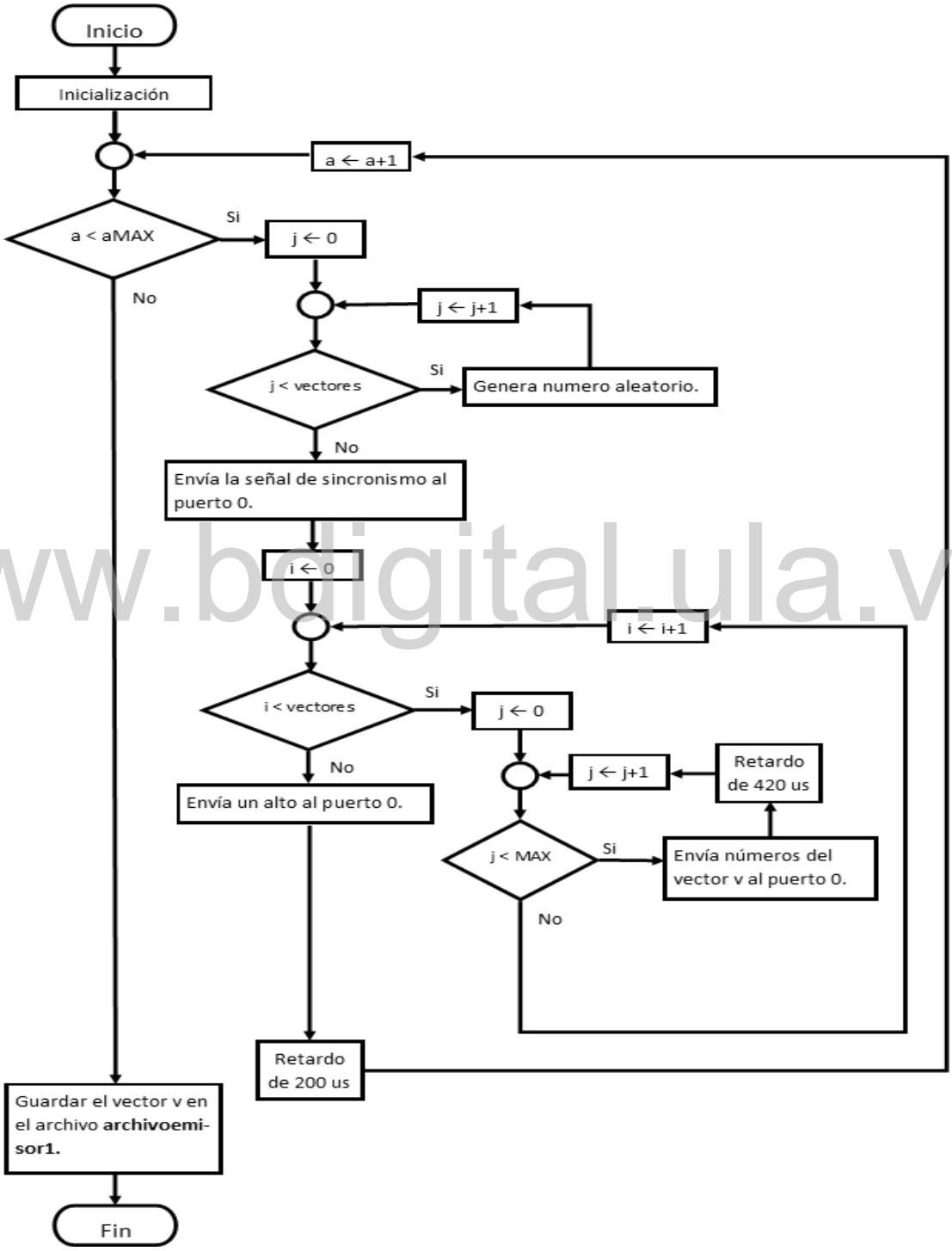


Figura 3.2. Diagrama de flujo del emisor de la etapa 1.

Ahora, respecto a la programación de las Raspberry Pi en esta primera etapa. Primero, la Raspberry Pi emisor, se le programó de manera que generara números aleatorios binarios, y los guardara en un vector. Luego, este programa escribió cada posición de este vector en un puerto del GPIO de la Raspberry Pi emisor, este mismo puerto estaba conectado al pin de DATA del emisor del módulo RF 433. Si el emisor de este módulo leía un alto proveniente de la Raspberry Pi emisor en el pin de DATA, el emisor del módulo emitía una señal analógica, inalámbricamente al receptor del módulo RF 433, y en el caso de que leía un bajo de este pin, el módulo emisor no emitía nada al módulo receptor, esto se debe a que este módulo opera con modulación ASK OOK. Para la implementación de este proyecto, la Raspberry Pi emisor debió generar una señal para que la Raspberry Pi receptor pudiese determinar cuándo se le estaba transmitiendo información realmente, esta señal era una señal de sincronismo, la cual se enviaba antes de enviar la secuencia de bits de información.

El diagrama de la figura 3.2, indica que inicialmente la Raspberry Pi emisor, realizó una inicialización de las variables que se iban a usar a lo largo del código. Luego la Raspberry Pi emisor envió un alto al puerto 0, para que el receptor se mantuviera esperando a que se le enviara la señal de sincronismo. Luego, el código generó un número aleatorio de MAX bits y lo almacenó en la posición j+a*vectores del vector **v**, y repitió esta acción vectores veces, obteniendo así, vectores números de MAX bits, guardados en el vector **v** de vectores*aMAX posiciones. Posteriormente, el programa escribió en el puerto 0 del GPIO de la Raspberry Pi emisor, una señal de sincronismo para indicarle al receptor que se le iba a enviar la información. Luego, el código escribió en este puerto el bit j del número ubicado en la posición i+a*vectores del vector **v**, empezando por la posición 0 hasta la posición vectores del vector **v**, enviando bit a bit, del bit menos significativo al más significativo de la palabra de código, cada 420 us, lo que indicó una velocidad de transmisión de 2.380 bps. El proceso de crear vectores números aleatorios y enviarlos, se repitió aMAX veces, para así enviar un total de vectores*aMAX números de MAX bits, por tanto, un total de MAX*vectores*aMAX bits. Una vez que el programa terminó de enviar toda la información, guardó el vector **v** de vectores*aMAX posiciones, en un archivo llamado archivoemisor1.

Ahora, el diagrama de flujo de la Raspberry Pi receptor es el indicado en la figura 3.3.

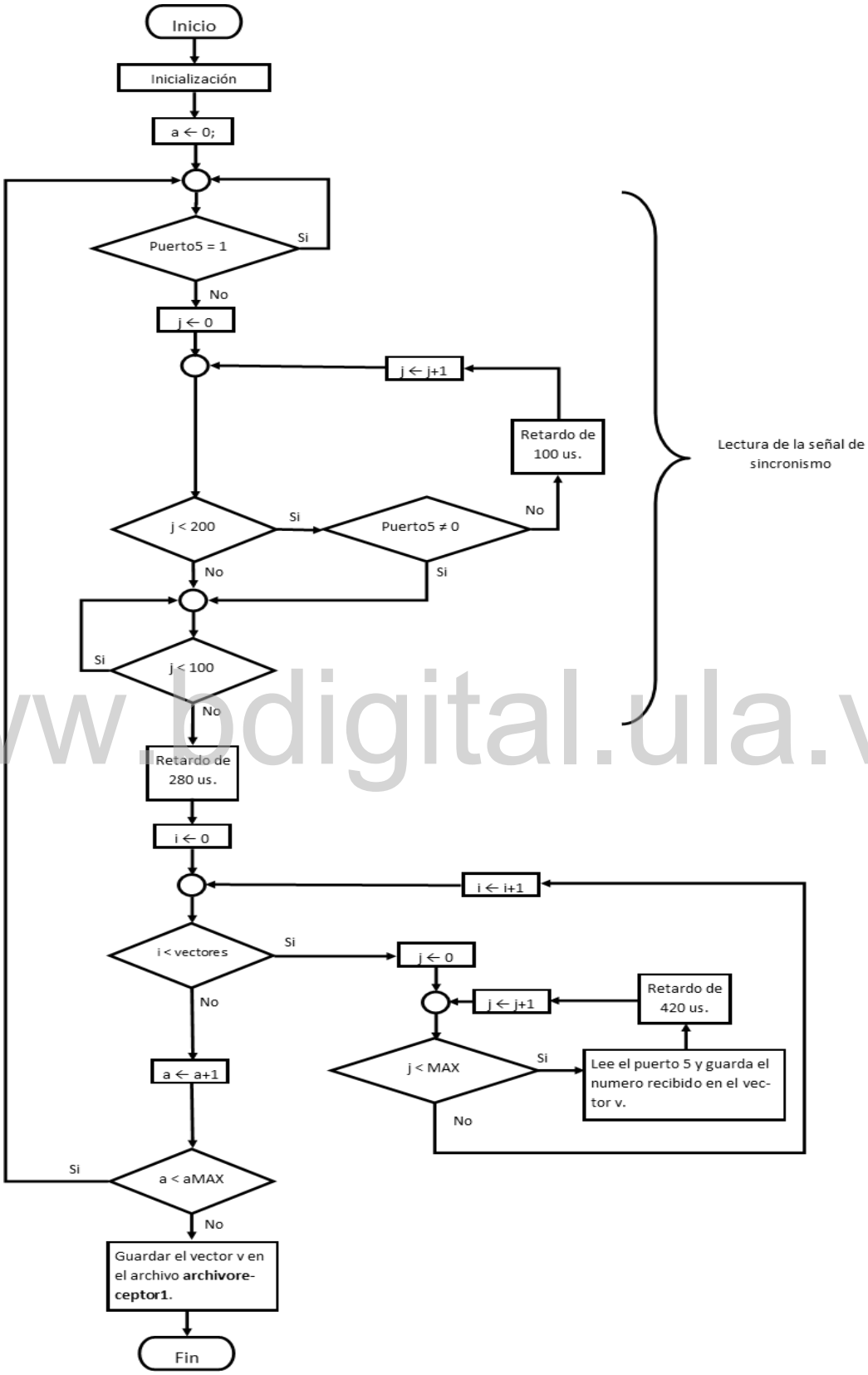


Figura 3.3. Diagrama de flujo del receptor de la etapa 1.

Este diagrama indica que inicialmente la Raspberry Pi receptor, realizó una inicialización de las variables que se iban a usar a lo largo del código. Luego, empezó a leer el puerto 5, que es el que puerto que está conectado al pin de DATA del receptor del módulo RF 433. Cuando la Raspberry Pi receptor detectó la señal de sincronismo enviada por el emisor, empezó a estar alerta para empezar a recibir y almacenar la información proveniente de la Raspberry Pi emisor. Entonces, leyó el puerto 5 cada 420 us y asignó el valor que leía de este puerto al bit ubicado en la posición j del número de MAX bits, ubicado en la posición i+a*vectores del vector **v**. Ordenó los bits recibidos del bit menos significativo al más significativo en números de MAX bits. Después, almacenó cada número de MAX bits en una posición de un vector declarado **v** de vectores*aMAX posiciones. Este proceso de lectura de bits y almacenar en números de MAX bits se repitió vectores veces.

Luego, este código volvió a esperar que llegara la señal de sincronismo para leer nuevamente vectores números de MAX bits. Finalmente, este proceso se repitió aMAX veces para recibir un total de vectores*aMAX números de MAX bits. Una vez que el programa terminó la lectura de los vectores*aMAX números de MAX bits, los guardó en un archivo llamado archivoreceptor1.

Cuando se terminó el proceso de enviar y recibir la información, se realizó un programa en la Raspberry Pi receptor que permitía saber el número de errores que hay después de la transmisión de información. El diagrama de flujo de este programa se encuentra en la figura 3.4.

De este diagrama se observa que, inicialmente la Raspberry Pi receptor, realizó una inicialización de las variables que iba a usar a lo largo del código. Luego, el programa leyó los archivos archivoemisor1 y archivoreceptor1, y guardó los datos del archivoemisor1 en el vector **v** y los datos del archivoreceptor1 en el vector **vv**. Después, comparó bit a bit ambos vectores, es decir, toda la información que se envió y se almacenó en la Raspberry Pi emisor, con toda la información que se recibió y se almacenó en la Raspberry Pi receptor. De esta manera el programa detectó cuantos bits son diferentes, lo cual indica la cantidad de errores que se cometieron en la trasmisión de información de este sistema en el que no se ha aplicado ningún tipo de codificación de canal.

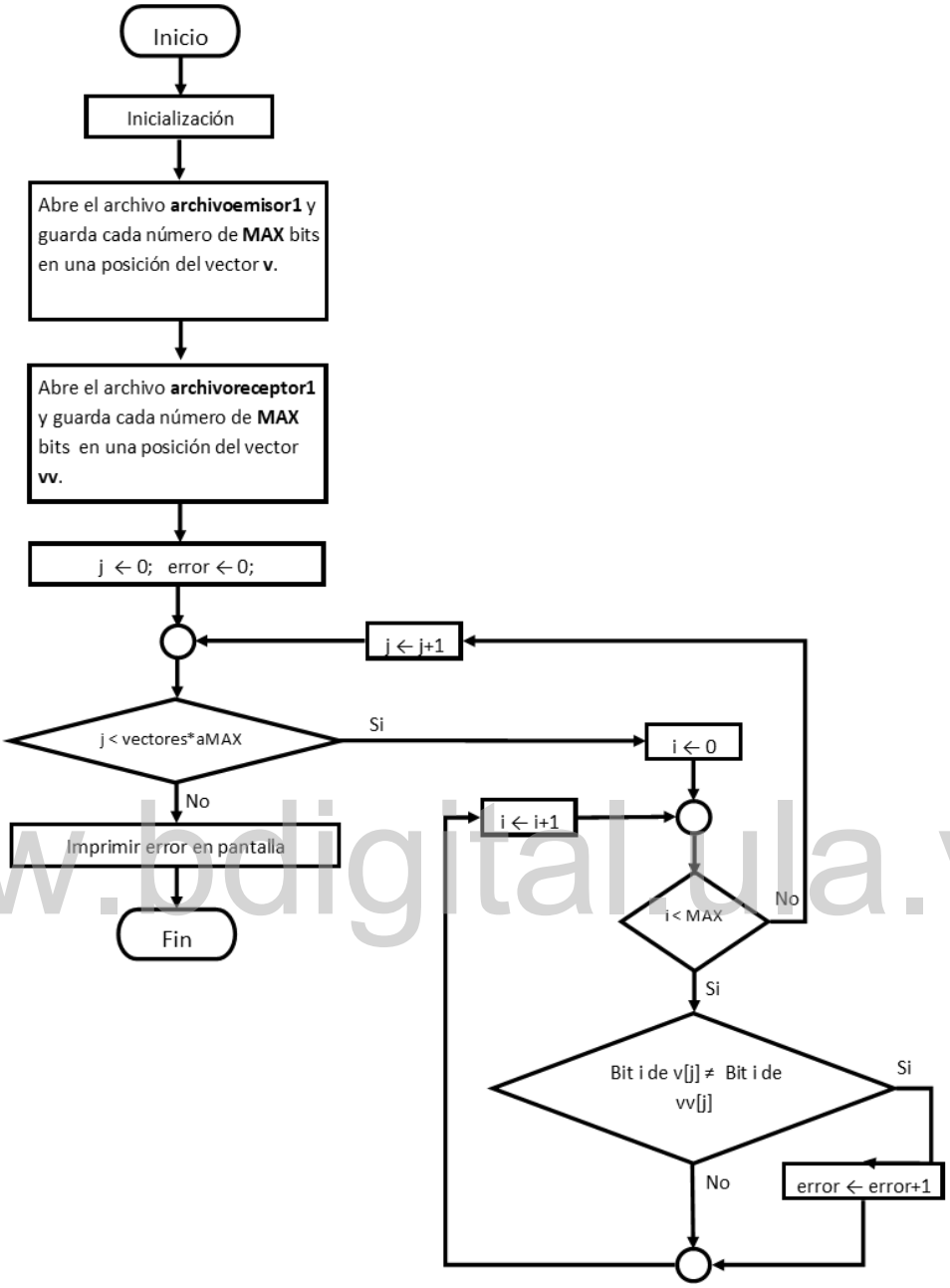


Figura 3.4. Diagrama de flujo del programa que calcula el error de la etapa 1.

3.2.2 Segunda etapa de la implementación: Código de bloques.

En esta etapa, se agregó un código de bloques a la primera etapa ya expuesta. Posteriormente se realizó la medición de la BER o tasa de error binario, este valor permitió descubrir el aporte de este codificador de canal al sistema de comunicación.

Primero, para realizar el código de bloques, se seleccionaron los siguientes parámetros del código de bloque lineal debido a que se agrega unos pocos bits de paridad y por su simplicidad de implementación.

Número de bits de mensajes (k): $k=4$

Longitud de bloque (n): $n=7$

Número de bits de paridad (m): $m=n-k=7-4=3$

Entonces, a la primera etapa se le añadió el código cíclico, el cual es una subclase de códigos de bloques lineales, de la siguiente manera. Para realizar el programa de codificación con el código cíclico en la Raspberry Pi emisor, se debía generar la palabra de código con una estructura de código como la señalada en la ecuación 2.21, una estructura igual a $b(X) + m(X)*X^{n-k}$. Con esto en consideración, el programa primero generó un mensaje aleatorio $m(X)$ de k bits, luego, desplazó 3 posiciones a la izquierda este mensaje $m(X)$, lo que simula realizar la multiplicación del mensaje $m(X)$ por X^{n-k} o específicamente por X^3, debido a los parámetros seleccionados previamente.

Ahora, de tomar en consideración los parámetros seleccionados y las ecuaciones 2.49 y 2.50, se obtuvo el polinomio generador, factorizando el polinomio X^7-1 a tres polinomios irreductibles, como se señala a continuación:

$$X^7-1=(1+X)*(1+X^2+X^3)*(1+X+X^3)$$

De este procedimiento se seleccionó como polinomio generador el polinomio X^3+X+1, por ser un polinomio irreductible de grado $m=3$, es decir, cuyo grado es igual al número de bits de paridad, en binario se representaría así, 0b00001011.

El diagrama de flujo del emisor de esta etapa se encuentra en la figura 3.5. Este diagrama indica que inicialmente la Raspberry Pi emisor, realizó una inicialización de las variables que iba a usar a lo largo del código. Luego la Raspberry Pi emisor envió un alto al puerto 0, para que el receptor se mantuviese esperando que se le enviara la señal de sincronismo.

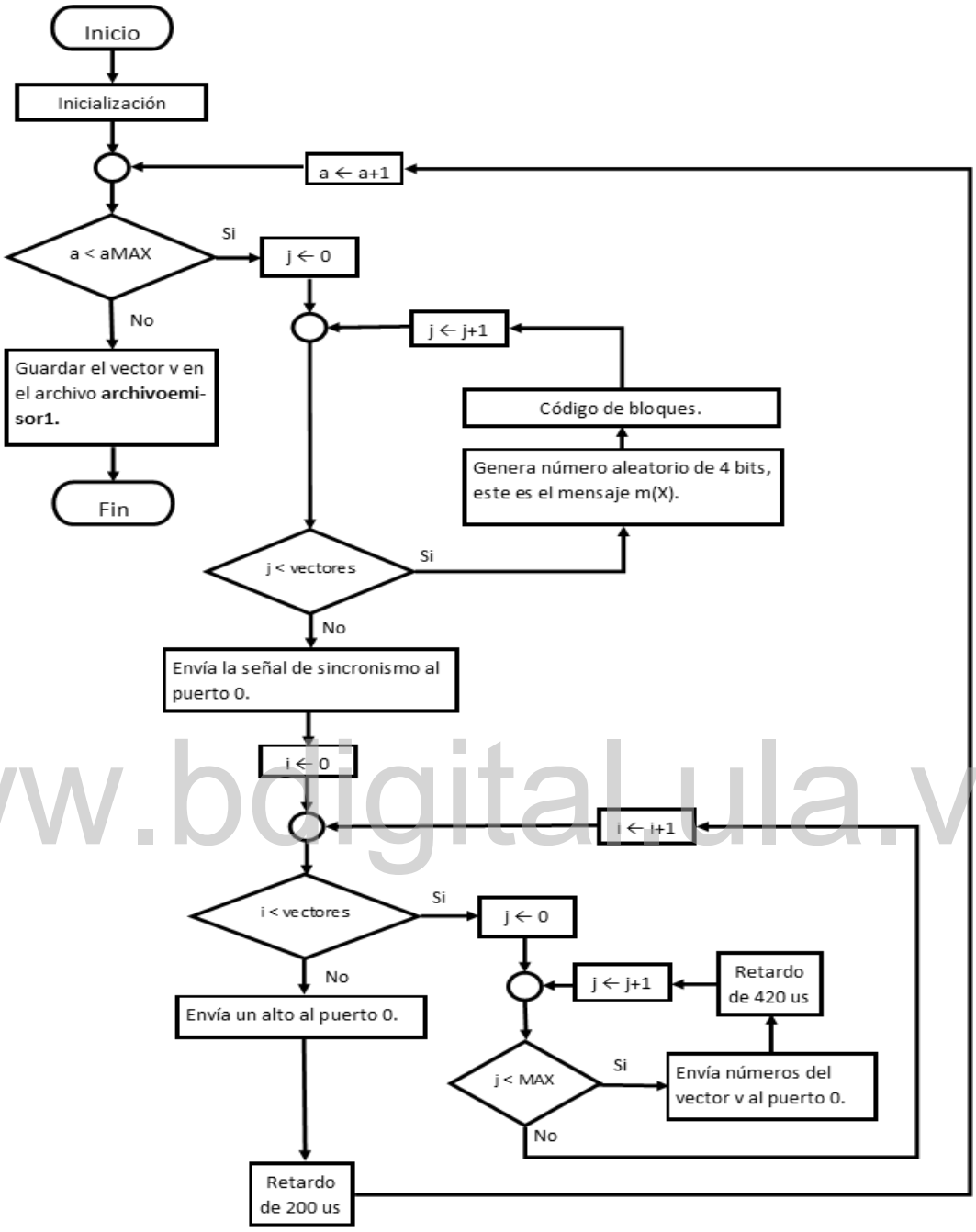


Figura 3.5. Diagrama de flujo del emisor de la etapa 2.

Después, el código generó un número aleatorio m(X) de 4 bits, luego le aplicó el código de bloques a este mensaje. Este código primero desplazó 3 posiciones a la izquierda el mensaje m(X), y resultó el nuevo mensaje $m(X)=m(X)X^3$, este se almacenó en la posición j+a*vectores del vector **v**, luego dividió $m(X)*X^3$ entre el polinomio generador pg, y el resto de esta división fue el polinomio de paridad b(X). Una vez que se obtuvo b(X), este se sumó

a m(X)*X^3, y se obtuvo la palabra de código c(X) de MAX bits correspondiente al primer mensaje aleatorio m(X) de 4 bits. Por último, se almacenó c(X) en la primera posición del vector **v** de vectores*aMAX posiciones. La generación y almacenamiento de las palabras de código, c(X), se repitió vectores veces.

Posteriormente de generar las vectores palabras de código, se envió la señal de sincronismo al receptor, la cual fue creada en la primera etapa. Luego, se escribió en el puerto 0 las palabras de código de MAX bits guardadas en el vector **v**, empezando por la posición 0 hasta la posición vectores del vector, enviando bit a bit cada 420 us, lo que indicó una velocidad de transmisión de 2.380 bps. El proceso de creación de las vectores palabras de código y el envío de las mismas se repitió aMAX veces, para transmitir un total de vectores*aMAX palabras de código de MAX bits. Posteriormente todas las posiciones del vector **v**, se almacenaron en un archivo llamado archivoemisor1. El programa funcionó correctamente debido a que generó los mensajes aleatorios de 4 bits y posteriormente los convirtió en las palabras de código válidas de 7 bits indicadas en la tabla 3.1.

Tabla 3.1. Palabras de código válidas del sistema implementado.

Mensaje de 4 bits en binario.	Palabra válida de código de bloque en binario de 7 bits.	Palabra válida de código de bloque en hexadecimal.	Secuencia de salida de código convolucional en hexadecimal.
0b0000	0b00000000	0x00	0x00000
0b0001	0b00001011	0x0B	0x00E17
0b0010	0b00010110	0x16	0x0385C
0b0011	0b00011101	0x1D	0x0364B
0b0100	0b00100111	0x27	0x0EF67
0b0101	0b00101100	0x2C	0x0E170
0b0110	0b00110001	0x31	0x0D73B
0b0111	0b00111010	0x3A	0x0D92C
0b1000	0b01000101	0x45	0x3B38B
0b1001	0b01001110	0x4E	0x3BD9C
0b1010	0b01010011	0x53	0x38BD7
0b1011	0b01011000	0x58	0x385C0
0b1100	0b01100010	0x62	0x35CEC
0b1101	0b01101001	0x69	0x352FB
0b1110	0b01110100	0x74	0x364B0
0b1111	0b01111111	0x7F	0x36AA7

El diagrama de flujo de la Raspberry Pi receptor de la etapa 2, es el indicado en la figura 3.6a y 3.6b, siendo la figura 3.6a la primera parte del diagrama de flujo del receptor de la segunda etapa, y la figura 3.6b la segunda parte del diagrama de flujo del receptor de la segunda etapa.

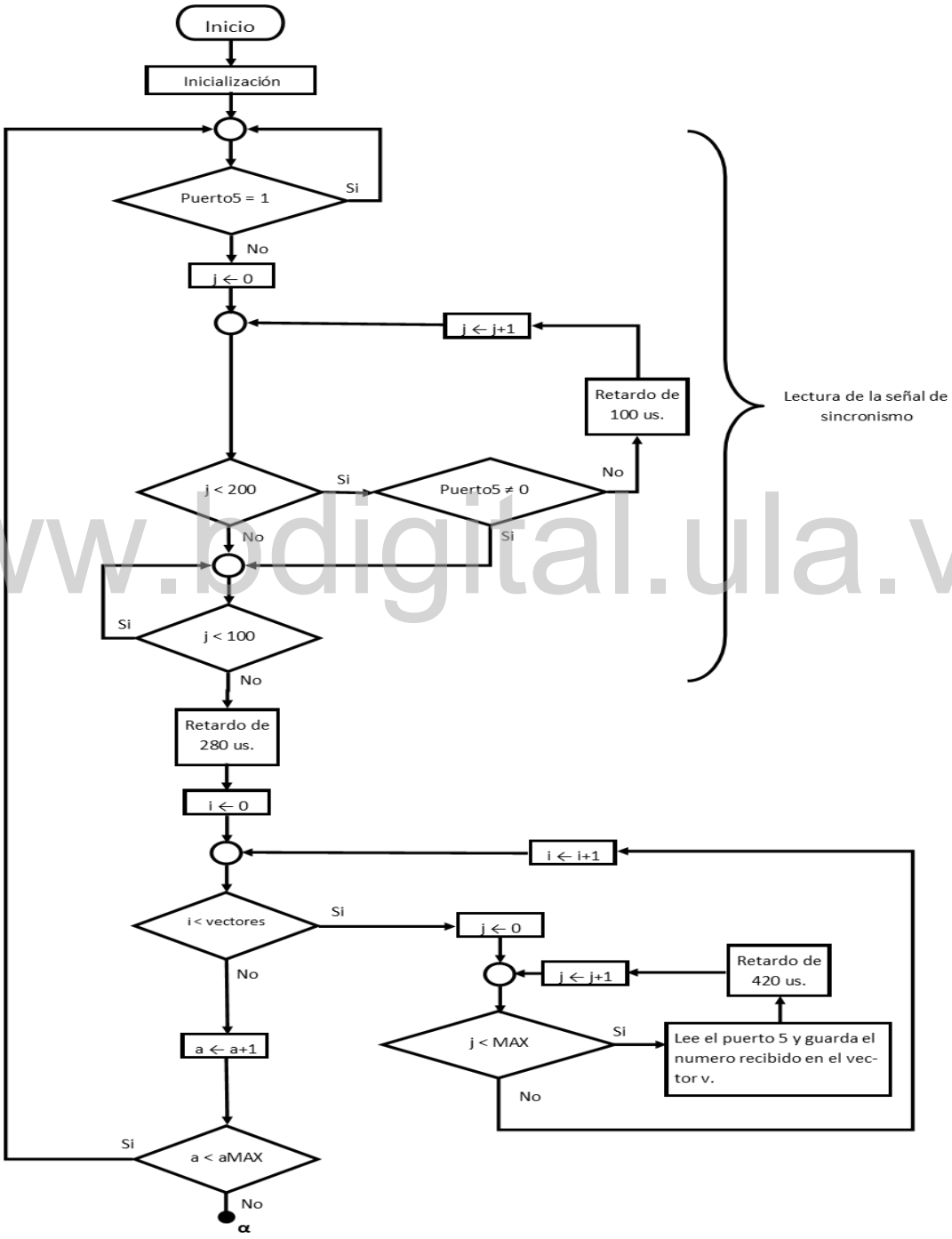


Figura 3.6a. Primera parte del diagrama de flujo del receptor de la etapa 2.

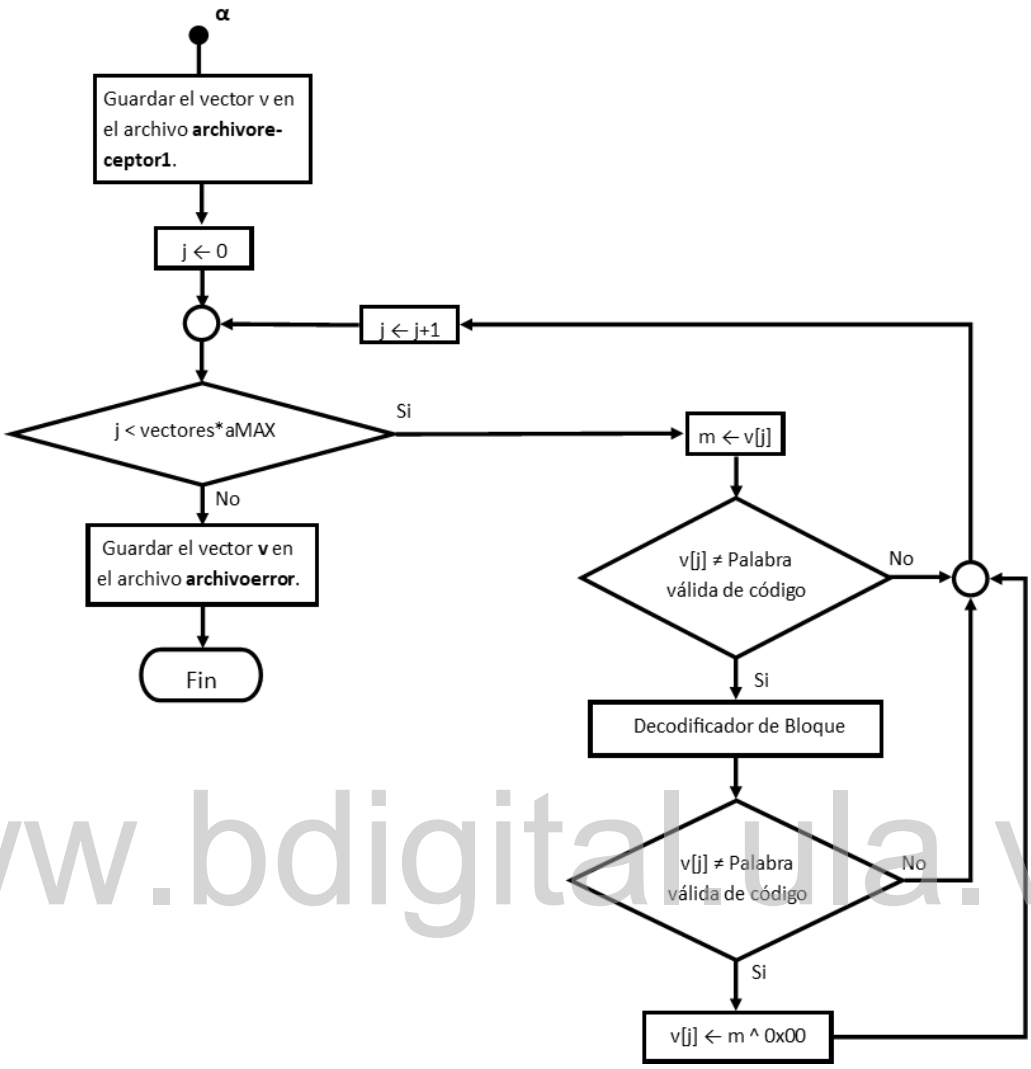


Figura 3.6b. Segunda parte del diagrama de flujo del receptor de la etapa 2.

Este diagrama indica que el programa inicialmente realizó una inicialización de las variables que iba a usar a lo largo del código. Cuando la Raspberry Pi receptor detectó la señal de sincronismo enviada por el emisor, entonces, empezó a leer cada 420 us cada bit que recibió de ese puerto. Luego, ordenó estos bits en números de MAX bits y almacenó los números de MAX bits en diferentes posiciones de un vector declarado **v**.

Este proceso de lectura de bits y almacenar en números de MAX bits se repitió vectores veces, después, volvió a esperar que llegase la señal de sincronismo para leer nuevamente vectores números de MAX bits. Este proceso se repitió aMAX veces para recibir un total de vectores*aMAX números de MAX bits. Cuando terminó de recibir toda la información,

guardó en un archivo llamado archivoreceptor1 toda la información almacenada en las vectores*aMAX posiciones del vector **v**. Luego, empezó a comparar cada una de las vectores*aMAX palabras de código recibidas, empezando con la primera palabra de código recibida ubicada en la posición 0 del vector **v**, con las palabras válidas de código indicadas en la tabla 3.1.

Luego, según los parámetros que se asignaron en el codificador, el total de palabras válidas de código son 2^4 de 2^7 palabras de código, debido a que cada mensaje es de 4 bits, si la palabra de código recibida no coincidía con ninguna palabra válida de código válida de la tabla 3.1, el programa procedía a decodificar la palabra de código recibida. En este decodificador primero se dividió esta palabra con el mismo polinomio generador pg del emisor, 0b00001011, la división se realizó de la misma manera que se realizó la división del mensaje entre el polinomio generador en el emisor. El resto de esta división fue el síndrome, este síndrome se comparó con los síndromes de los patrones de error que podía corregir el código, y los patrones de error para corregir 1 solo error por palabra de código, como es el caso de este decodificador, son, 0b00000001, 0b00000010, 0b00000100, 0b00001000, 0b00010000, 0b00100000, 0b01000000 y 0b00000000, y sus síndromes son, 0b001, 0b010, 0b100, 0b011, 0b110, 0b111, 0b101 y 0b000, respectivamente. Los síndromes de los patrones de error se obtuvieron del resto de la división del patrón de error entre el polinomio generador, pg, 0b00001011.

Entonces, se terminó el proceso de decodificación cuando el programa encontró el síndrome de la palabra de código recibida que coincidía con algún síndrome de los patrones de error posibles, sumó en modulo 2 ese patrón de error con la palabra de código recibida, con el fin de eliminar el error existente en la misma. Finalmente, esta palabra de código corregida se comparó con las palabras válidas de código de la tabla 3.1 nuevamente, ya que, si no coincidía con ninguna, se conservaría la palabra de código recibida sin la corrección. Esto es debido a que probablemente se había realizado el código de bloque a una palabra de código con más de 1 error. Luego, la palabra corregida, o no corregida, se almacenó en la posición j del vector **v**. Este proceso de corrección de la palabra de código recibida se repitió con las vectores*aMAX palabras de código, con la finalidad de corregir el error que tenía cada una de estas palabras. Después, estas palabras de código corregidas guardadas en el

vector **v** fueron almacenadas en un archivo llamado archivoerror para posteriormente analizar el aporte de esta etapa de codificación al sistema de comunicación.

El diagrama de flujo del programa que calcula el error de esta etapa se encuentra en la figura 3.7.

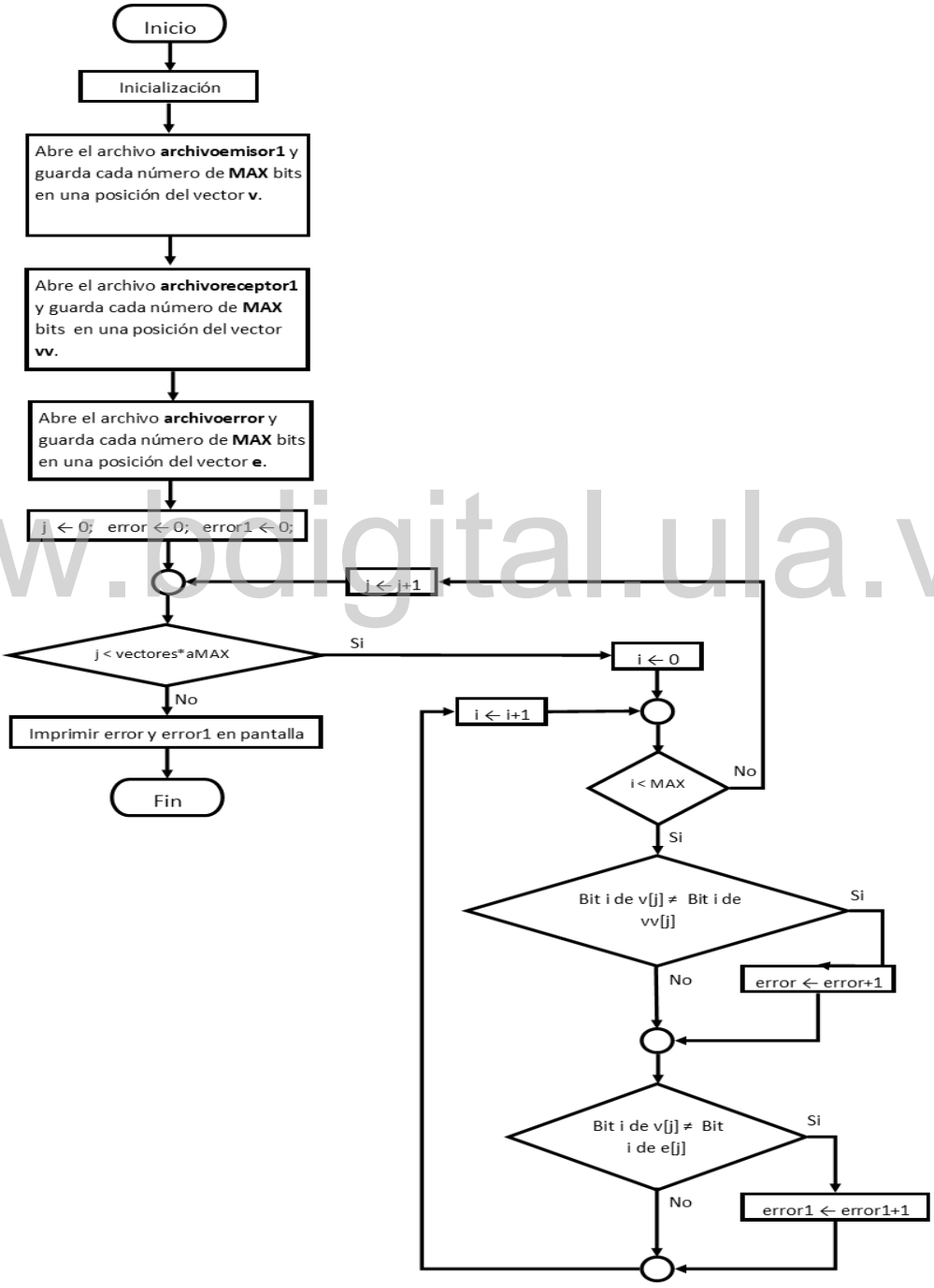


Figura 3.7. Diagrama de flujo del programa que calcula el error de la etapa 2.

Cuando se terminó el proceso de enviar y recibir la información, se realizó un programa en la Raspberry Pi receptor que permitió saber el número de errores que ocurrieron después de la transmisión de información. En el diagrama de la figura 3.7, se observa que inicialmente la Raspberry Pi receptor, realizó una inicialización de las variables que se iban a usar a lo largo del código. Luego, el programa leyó los archivos archivoemisor1, archivoreceptor1, y archivoerror, y guardó los datos del archivoemisor1 en el vector **v**, los datos del archivoreceptor1 en el vector **vv**, y los datos del archivoerror en el vector **e**. Luego, el programa comparó bit a bit los vectores **v** y **vv**, es decir, toda la información que se envió y se almacenó en la Raspberry Pi emisor, con toda la información que se recibió y se almacenó en la Raspberry Pi receptor. De igual manera, comparó bit a bit los vectores **v** y **e**, es decir, toda la información que se envió y se almacenó de la Raspberry Pi emisor, con toda la información que se recibió, se decodificó con el código cíclico y se almacenó de la Raspberry Pi receptor.

3.2.3 Tercera etapa de la implementación: Código de convolución.

En esta etapa, se agregó el codificador convolucional a la segunda etapa de la implementación. Se seleccionó el código de convolución como segunda técnica de codificación, a pesar de que es más difícil de implementar que el código de bloques. Esta técnica puede corregir un número superior de errores por secuencia de bits, además en los antecedentes señalan que esta tiene una alta efectividad. De esta manera, se concatenó en serie el código de bloques que se realizó en la segunda etapa con el código convolucional que se realizó en esta etapa, todo esto con el fin de alcanzar una codificación de canal robusta. Ahora, para cada palabra de código válida de L bits, que genere el código de bloques, el código de convolución generó una secuencia de salida codificada de longitud n*(L+M) bits. Siendo n el número de sumadores que tenga el codificador convolucional y M el número de elementos de memoria que tenga el codificador convolucional. Posteriormente se realizó la medición de la BER para determinar el rendimiento de esta nueva etapa.

El codificador convolucional que se implementó fue el de la figura 2.4, este tiene dos elementos de memoria, M=2, y dos sumadores, n=2. La entrada a este codificador convolucional fue la palabra de código generada por el código de bloques de la etapa dos, el cual generó palabras de código de 7 bits de longitud, por tanto, el codificador convolucional

de esta etapa para entradas de longitud de 7 bits, generó una secuencia de salida codificada de 2*(7+2) bits, es decir, de 18 bits.

Para realizar el código de convolución en la Raspberry Emisor, se tomó como referencia el diagrama de estados de la figura 2.7, ya que es el diagrama de estados del codificador convolucional de la figura 2.4. A su vez, como es un código de convolución con dos elementos de memoria, tiene 2^2 estados, como está reflejado en la figura 2.7, estos estados son 00 representado por la letra a, 10 representado por b, 01 representado por la letra c y finalmente el estado 11 representado por la letra d. Una transición de un estado a otro en respuesta a la entrada 0 se representa mediante una rama continua, en tanto que una transición en respuesta a la entrada 1 se ilustra mediante una rama punteada. La marca binaria sobre cada rama representa la salida del codificador cuando esta se mueve de un estado a otro. Para obtener la secuencia de salida, se empieza con el estado inicial de puros ceros que es el estado a, y se continua a través del diagrama de estado de acuerdo con la secuencia de mensaje. Se sigue una rama continua si la entrada es cero y una rama punteada si esta es uno. A medida que se recorre cada rama, se genera la marca binaria correspondiente sobre ellas.

El diagrama de flujo del emisor de esta etapa se encuentra en la figura 3.8. Este diagrama indica que inicialmente la Raspberry Pi emisor, realizó una inicialización de las variables, MAX, esta variable definía el tamaño de la palabra de código de bloque, que según los parámetros selecciones en la etapa 2 era igual a 7, la variable vectores, esta variable indicó el número de palabras de código que se debían generar, la variable aMAX, indicó cuantas veces se debían generar y enviar los vectores números de 2*(MAX+2) bits, la variable a igual a 0, la variable x igual a 0, esta variable indicó el estado en el que se encontraba el código de convolución, inicialmente estaba en el estado a, y finalmente la variable pg igual a 0b00001011, esta variable era el polinomio generador seleccionado para el código de bloques. También en la inicialización se declaró el vector **v** y el vector **vv,** con vectores*aMAX posiciones, para que se pudiesen guardar un total de vectores*aMAX números de MAX bits en estos vectores.

Luego, la Raspberry Pi emisor envió un alto al puerto 0, para que el receptor se mantuviese esperando a que se le enviase la señal de sincronismo. Posteriormente, el código generó un número aleatorio m(X) de 4 bits, y le aplicó el código de bloques a este mensaje como se

indicó en la etapa 2, donde se obtuvo la palabra de código de 7 bits, c(X), que fue guardada en la posición j del vector **v**.

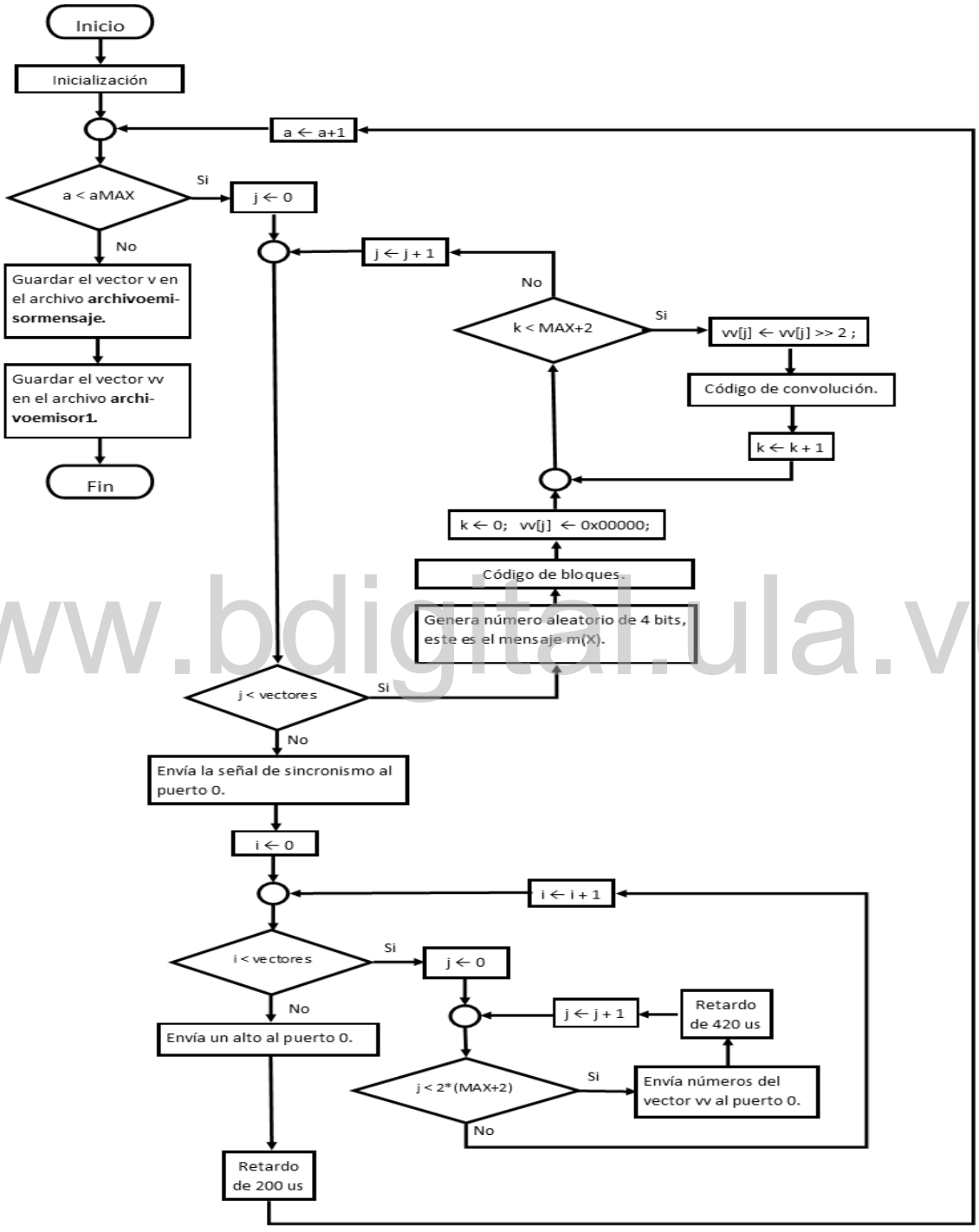


Figura 3.8. Diagrama de flujo del emisor de la etapa 3.

Después a esta palabra de código de 7 bits se le agregaron 2 ceros en los bits más

significados para que el codificador convolucional pudiese retornar al estado inicial cuando terminase de leer la secuencia de entrada. Esta palabra de código de ahora 9 bits la leyó el codificador convolucional bit a bit, leyendo del bit menos significativo al más significativo. Se realizó la codificación de convolución a la palabra de código c(x), ubicada en la posición j del vector **v**, de la siguiente manera, primero se inicializó la variable k igual a 0, esta variable era para indicar que bit se estaba evaluando de la palabra de código c(X), como inicialmente estaba en 0, el código de convolución empezó con el bit menos significativo de c(X), ahora dependiendo del estado x en el que se encontraba el código de convolución, y del valor que tenía el k bit de la palabra de código, el programa determinó que transición seguiría el código de convolución de un estado a otro, y por tanto definió los dos bits a la salida y los ubicó en los bits más significativos de la posición j del vector **vv**.

Finalmente, el código de convolución determinó el nuevo estado x del que debía partir el siguiente k bit a la entrada, e incrementó en 1 la variable k. Después de aplicar el código de convolución, desplazó el número ubicado en la posición j del vector **vv,** dos posiciones a la derecha. Así continuó el programa leyendo bit a bit la palabra de código **v[j],** y correspondientemente asignó bits a la salida del codificador de convolución, hasta cumplir con la secuencia de 9 bits de la palabra de código **v[j]**. Y de esta manera se obtuvo una secuencia de 18 bits a la salida del codificador convolucional. Esta secuencia de 18 bits fue guardada en la posición j del vector **vv**. La generación de las palabras de código y las secuencias de 18 bits correspondientes, se repitió vectores veces. Una vez que el programa generó y guardó las vectores palabras de código de 7 bits y los vectores números de 18 bits, se envió a la Raspberry Pi receptor la señal de sincronismo creada en la primera etapa, con el objetivo de que esta empezase a leer y guardar la información que le enviará la Raspberry Pi emisor.

Luego de enviar la señal de sincronismo se procedío a enviar los vectores números de 18 bits almacenados en el vector **vv**, enviando bit a bit cada 420 us, por tanto, se envió la información a una velocidad de 2.380 bps. Una vez que se enviaron los vectores números de 18 bits, se repitió todo el proceso descrito anteriormente aMAX veces, con el fin de generar y enviar un total de vectores*(2*(MAX+2))*aMAX bits. Luego de enviar toda la información, todas las palabras de código de 7 bits guardadas en el vector **v** fueron

almacenadas en un archivo denominado archivoemisormensaje, y todas las secuencias de salida de 18 bits generadas por el codificador convolucional almacenadas en el vector **vv** fueron almacenadas en un archivo denominado archivoemisor1. El programa generó las secuencias de salida codificadas, de 18 bits de longitud, indicadas en la tabla 3.1, correspondientes a cada palabra de código de bloque válida de 7 bits de la etapa 2.

El diagrama de flujo de la Raspberry Pi receptor de la etapa 3 es el indicado en la figura 3.9a y 3.9b, siendo la figura 3.9a la primera parte del diagrama de flujo del receptor de la segunda etapa, y la figura 3.9b la segunda parte del diagrama de flujo del receptor de la segunda etapa. Este diagrama indica que inicialmente el programa realizó una inicialización de las variables que iba a usar a lo largo del código. Luego, la Raspberry Pi receptor se mantuvo leyendo el puerto 5 que estaba conectado al pin de DATA del receptor del módulo RF 433, cuando la Raspberry Pi receptor detectó la señal de sincronismo enviada por el emisor, entonces, empezó a leer cada 420 us los bits que recibía de ese puerto.

Después, ordenó estos bits en números de 2*(MAX+2) bits y los almacenó en diferentes posiciones de un vector declarado **v**. Este proceso de lectura de bits y almacenar en números de 2*(MAX+2) bits se repitió vectores veces. Posteriormente, volvió a esperar a que llegara la señal de sincronismo para leer nuevamente vectores números de MAX bits. Este proceso se repitió aMAX veces para recibir un total de vectores*aMAX números de MAX bits. Cuando terminó de recibir toda la información, guardó en un archivo llamado archivoreceptor1 toda la información almacenada en las vectores*aMAX posiciones del vector **v**.

Después, el programa ejecutó la decodificación de convolución, para realizar dicha decodificación se tomó como referencia el algoritmo de Viterbi de la figura 2.5. El programa tomó el número de 18 bits guardado en la posición j del vector **v** de vectores*aMAX, para compararlo con las transiciones del enramado de la figura 2.5. El programa realizó esta comparación, tomando este número ubicado en la posición j del vector **v**. Luego, almacenó los dos bits menos significativos de este número en la variable y, y comparó esta variable con el valor que corresponde a cada transición del primer nivel del enramado de la figura 2.5, dependiendo de cuantos bits eran diferentes entre estos dos valores, el programa definió la

métrica para cada trayectoria. La trayectoria que tenía menor métrica, fue la que se guardó en el nodo a la que llegó esa trayectoria.

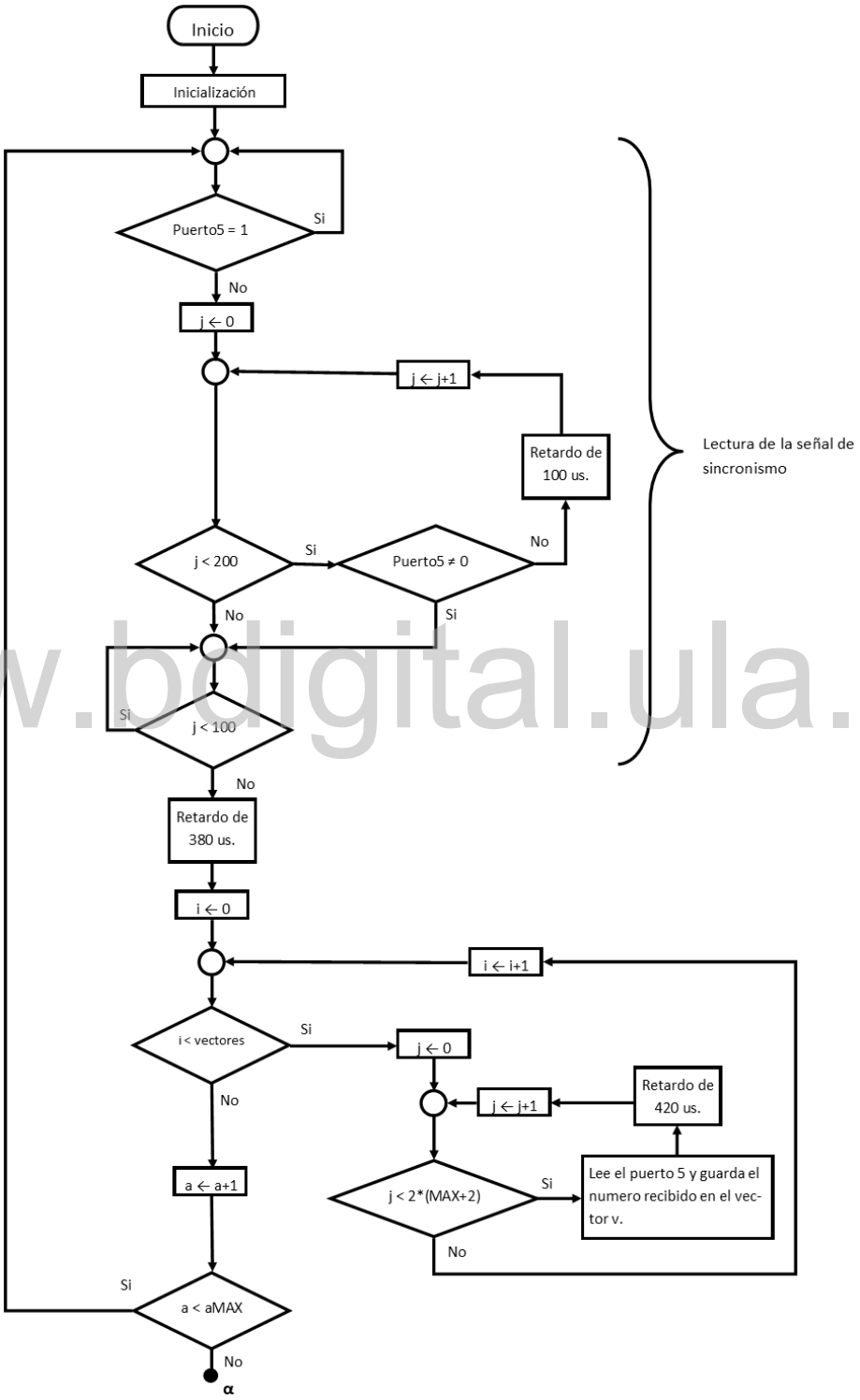


Figura 3.9a. Primera parte del diagrama de flujo del receptor de la etapa 3.

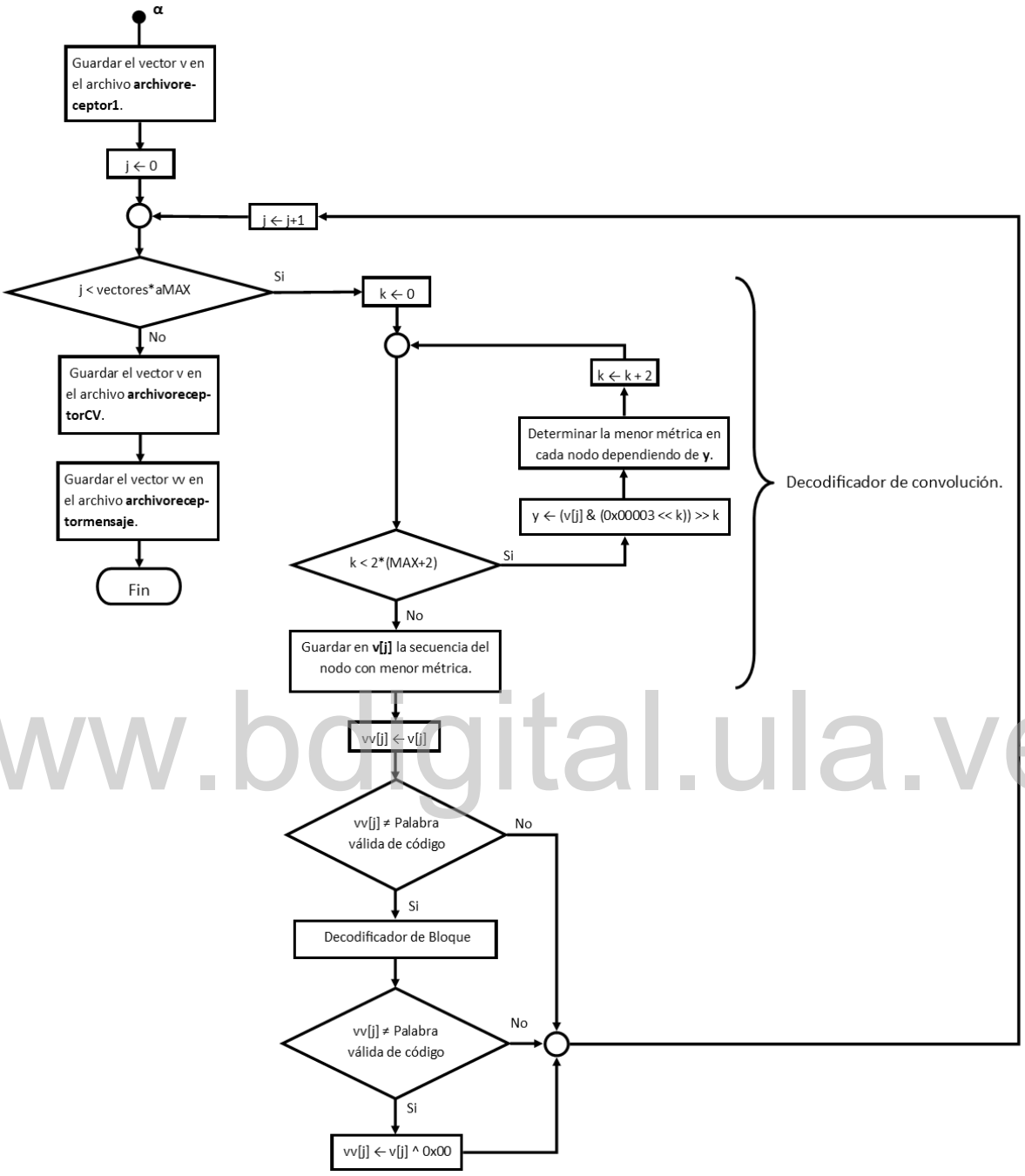


Figura 3.9b. Segunda parte del diagrama de flujo del receptor de la etapa 3.

Fue importante tomar en consideración que cada nodo generaba dos transiciones, y a su vez a partir del tercer nivel del enramado de la figura 2.5 en cada nodo entraban dos transiciones, por tanto, se debía determinar cuál de estas dos transiciones diferían menos del par de bits y que se están analizando del número recibido **v[j]**. Cuando se seleccionaba una

transición, la otra se descartaba y así se seguía evaluando cada nodo hasta que se analizaron los 18 bits recibidos. En el caso de que las dos transiciones que entraban a un nodo generaban la misma métrica, el programa seleccionaba la trayectoria de la siguiente manera, en el caso del nodo a, donde entran la trayectoria que viene del nodo a y la trayectoria que viene del nodo c, el programa seleccionaba la trayectoria que viene del nodo a, en el caso del nodo b, el programa seleccionaba la trayectoria que viene del nodo a, en el caso del nodo c, el programa seleccionaba la trayectoria que proviene del nodo b, y finalmente en el caso del nodo d, el programa seleccionaba la trayectoria que proviene del nodo b en vez de la trayectoria que proviene del nodo d.

Cada vez que el programa seleccionó la métrica más pequeña en cada nodo, guardó el número binario que correspondía a esa transición, si la transición era punteada se guardaba un 1 y si era una transición continua se guardaba un 0. Cada número se guardó de forma ordenada del bit menos significativo al bit más significativo, uno tras otro, para así generar una palabra de código de 7 bits en cada nodo, correspondiente a la secuencia de 18 bits recibida. En el caso de que varias coincidan el programa supuso una de las dos, en el caso de este programa, si dos nodos tienen la misma métrica, el programa tuvo el siguiente orden de elección, primero el nodo a, luego el nodo b, luego el nodo c y por último el nodo d.

Posteriormente de que el programa seleccionó la trayectoria del nodo con menor métrica guardó la palabra de código generada por esta trayectoria en la posición j del vector **v** y esta posteriormente se envió al decodificador de bloques. Luego, el programa guardó esta palabra de código en la posición j del vector **vv**. Comparó esta palabra de código generada por el decodificador de convolución, con las palabras válidas de código indicadas en la tabla 3.1. Si no coincidía con ninguna palabra válida de código, el programa procedió a decodificar la palabra de código proveniente del decodificador de convolución, pero ahora con el decodificador de bloque de la etapa 2. La palabra de código generada ahora por el código de bloques fue guardada en la posición j del vector **vv**. Luego, esta palabra de código fue comparada con las palabras válidas de código indicadas en la tabla 3.1, si no coincidía con ninguna palabra válida de código válida, el programa entonces guardaba en la posición j del vector **vv**, la palabra de código de convolución guardada en la posición j del vector **v**. Este proceso de corrección de la secuencia recibida por ambos decodificadores de canal se repitió

con las vectores*aMAX secuencias de 18 bits recibidas. Después, las palabras de código corregidas por el código de convolución guardadas en el vector **v** fueron almacenadas en un archivo llamado archivoreceptorCV para posteriormente analizar el aporte de esta etapa de codificación al sistema de comunicación, y las palabras de código corregidas por el código de bloques guardadas en el vector **vv** fueron almacenadas en un archivo llamado archivoreceptormensaje, para posteriormente analizar el aporte al sistema de comunicación de estas dos etapas de codificación concatenadas en serie.

Cuando se terminó el proceso de enviar y recibir la información, se realizó un programa en la Raspberry Pi receptor que permitía saber el número de errores que había después de la transmisión de información. La primera parte del diagrama de flujo de este programa se encuentra en la figura 3.10a y la segunda parte en la figura 3.10b.

El programa leyó los datos del archivoemisor1 y los guardó en un vector llamado **v**, este archivo correspondía al archivo que almacenaba las palabras de código de 18 bits, generadas en el emisor por el código de bloques concatenado con el código de convolución, el programa también leyó y guardó los datos del archivoreceptor1 en un vector llamado **vv**, este archivo almacenó las palabras de código de 18 bits recibidas en el receptor. Después, se obtuvo la cantidad de errores que se cometieron en la trasmisión de información de este sistema en el que no se ha aplicado ningún tipo de decodificación, comparando bit a bit los vectores **v** y **vv**, es decir, toda la información que se envió y se almacenó de la Raspberry Pi emisor, con toda la información que se recibió y se almacenó de la Raspberry Pi receptor.

Para determinar cuántos errores corrigió solo el decodificador de convolución, este programa leyó y guardó los datos del archivo archivoemisormensaje en un vector denominado **v**, este archivo almacenó las palabras de código de 7 bits generadas en el emisor por el codificador de bloques, y además leyó y guardó los datos del archivo archivoreceptorCV en un vector denominado **vv**, este archivo almacenó las palabras de código de 7 bits decodificadas por el decodificador de convolución. Luego comparó bit a bit los vectores **v** y **vv**, es decir, todas las palabras de código de 7 bits generadas por el codificador de bloques en el emisor con todas las palabras de código de 7 bits decodificadas por el decodificador de convolución del receptor.

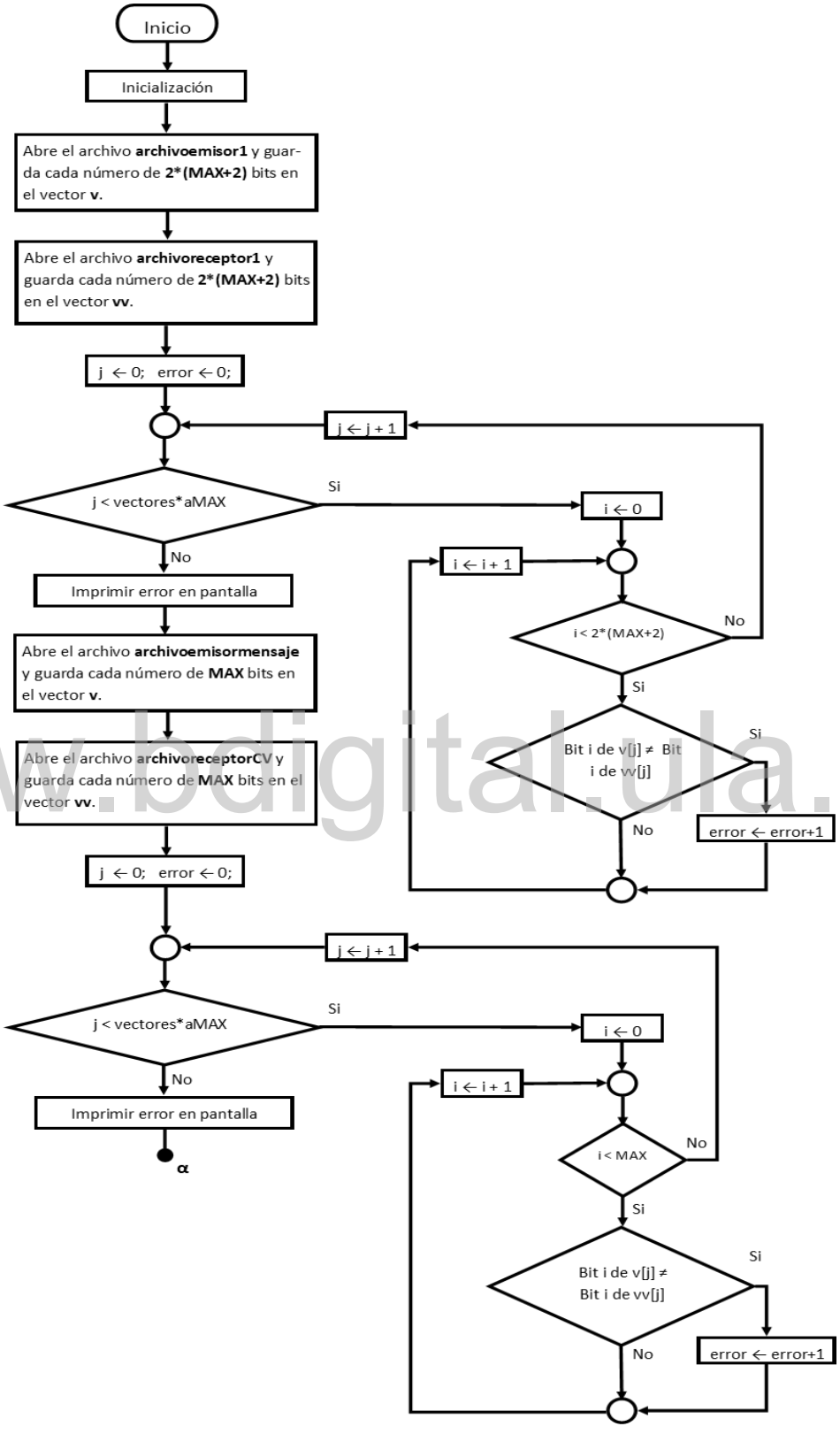


Figura 3.10a. Primera parte del diagrama de flujo para calcular el error de la etapa 3.

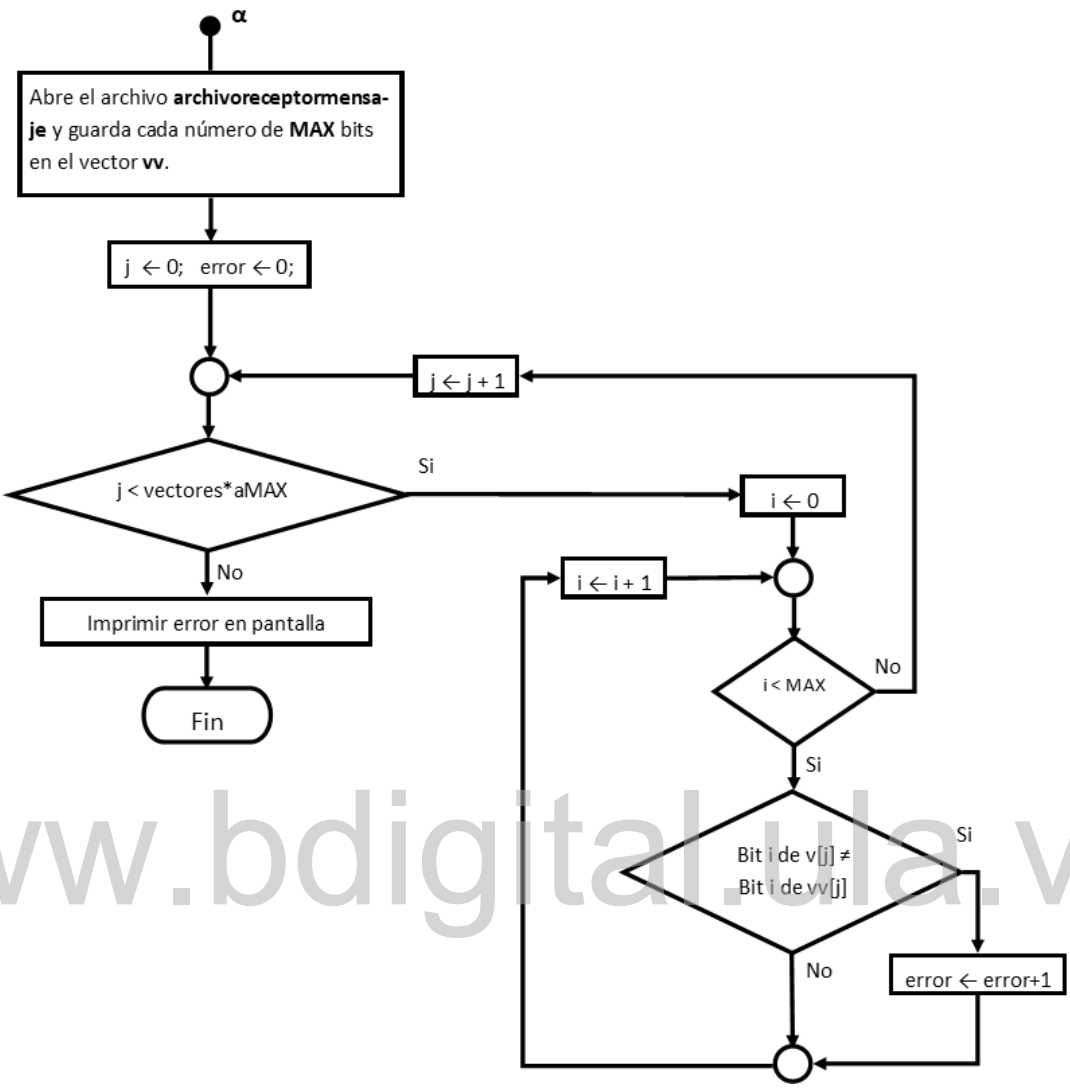


Figura 3.10b. Segunda parte del diagrama de flujo del programa que calcula el error de la etapa 3.

Y finalmente, para determinar cuántos errores se corrigieron con ambos decodificadores de canal concatenados, el programa leyó y guardó los datos del archivo archivoreceptormensaje en el vector **vv**, este archivo almacenó las palabras de código de 7 bits decodificadas por el decodificador de convolución concatenado con el decodificador de bloque. Luego, comparó bit a bit los vectores **v** y **vv**, es decir, todas las palabras de código de 7 bits generadas por el codificador de bloques en el emisor con todas las palabras de código de 7 bits decodificadas por el decodificador de convolución concatenado con el decodificador de bloque.

3.2.4 Cuarta etapa de la implementación: Entrelazado.

El código de convolución implementado en la tercera etapa como se ha mencionado antes, tenía la posibilidad de corregir 3 o más errores siempre que los bits incorrectos estuviesen distribuidos por un periodo más largo que una longitud de restricción, es decir que, si llegaban a ocurrir más de 2 errores espaciados a una distancia menor que la longitud de restricción, el decodificador de convolución no eliminaba los errores en esa secuencia. Y en el caso del código de bloques implementado en las etapas 2 y 3, debido a que solo podía corregir 1 error por palabra de código, si llegaba a ocurrir más de 1 error por palabra de código, este código no podía corregir los errores que tenía esa palabra de código.

Tomando esto en consideración, se destacó la importancia de que los errores que ocurriesen en la transmisión de información de la Raspberry Pi emisor a la Raspberry Pi receptor estuviesen espaciados en cada secuencia de MAXX bits, en un periodo más largo que una longitud de restricción. Con la ayuda de un entrelazador se logró dispersar los errores, lo cual permitió que un menor número de errores ocurriesen por cada secuencia de información.

En la etapa 3 se generaba y guardaba la información codificada en un vector declarado **v**, y posteriormente se enviaba cada posición de este vector **v** al receptor en el mismo orden que fue generada. En esta etapa, se agregó un entrelazador al código de la tercera etapa, como técnica para dispersar las ráfagas de errores, por tanto, en esta nueva etapa se generó y guardó la información en diferentes posiciones de un vector declarado **v** como está indicado en la figura 3.11.

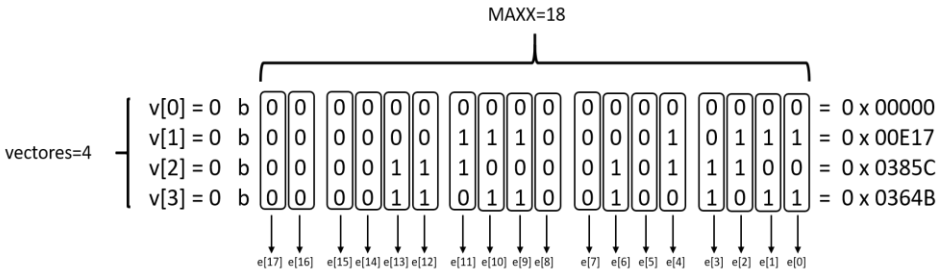


Figura 3.11. Entrelazador del vector v el cual tiene 4 números de 18 bits.

Supongamos que esta información se guardó en filas, luego, se guardó la misma información del vector **v** en diferentes posiciones de un vector declarado **e,** pero ahora se guardó en columnas, como lo indica la figura 3.12, y este vector **e**, es el que finalmente se envía al receptor.

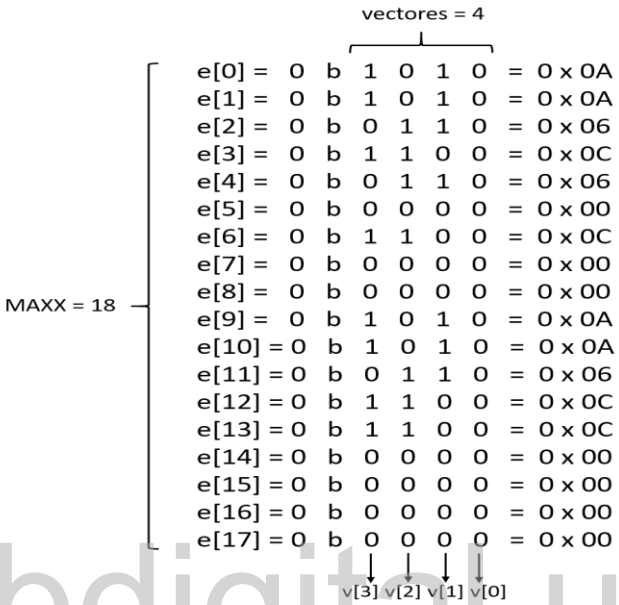


Figura 3.12. Los 18 números de 4 bits del vector e.

Luego de que el receptor recibió esta información, la reordenó antes de ser decodificada, es decir, cambió la información de columnas a filas, y después de esto aplicó la decodificación de canal a esta información. De acuerdo a las figuras 3.11 y 3.12, se observó que el entrelazador generó MAXX números de vectores bits, a partir de las vectores palabras de código de MAXX bits generadas por el codificador de canal concatenado.

Para esta etapa, se hicieron dos pruebas, primero se explicará la primera prueba. El diagrama de flujo de esta primera prueba del emisor se encuentra en la figura 3.13. La primera prueba en el emisor, es prácticamente el mismo programa de la Raspberry Pi emisor de la etapa 3, pero se agregó un entrelazador después de la etapa de generación de los vectores números de MAXX bits, para que mezclara y generara números de vectores bits, y luego los guardara en el vector **e**. De esta manera, el programa solo mezcló vectores números, los guardó en el vector **e** y los envió.

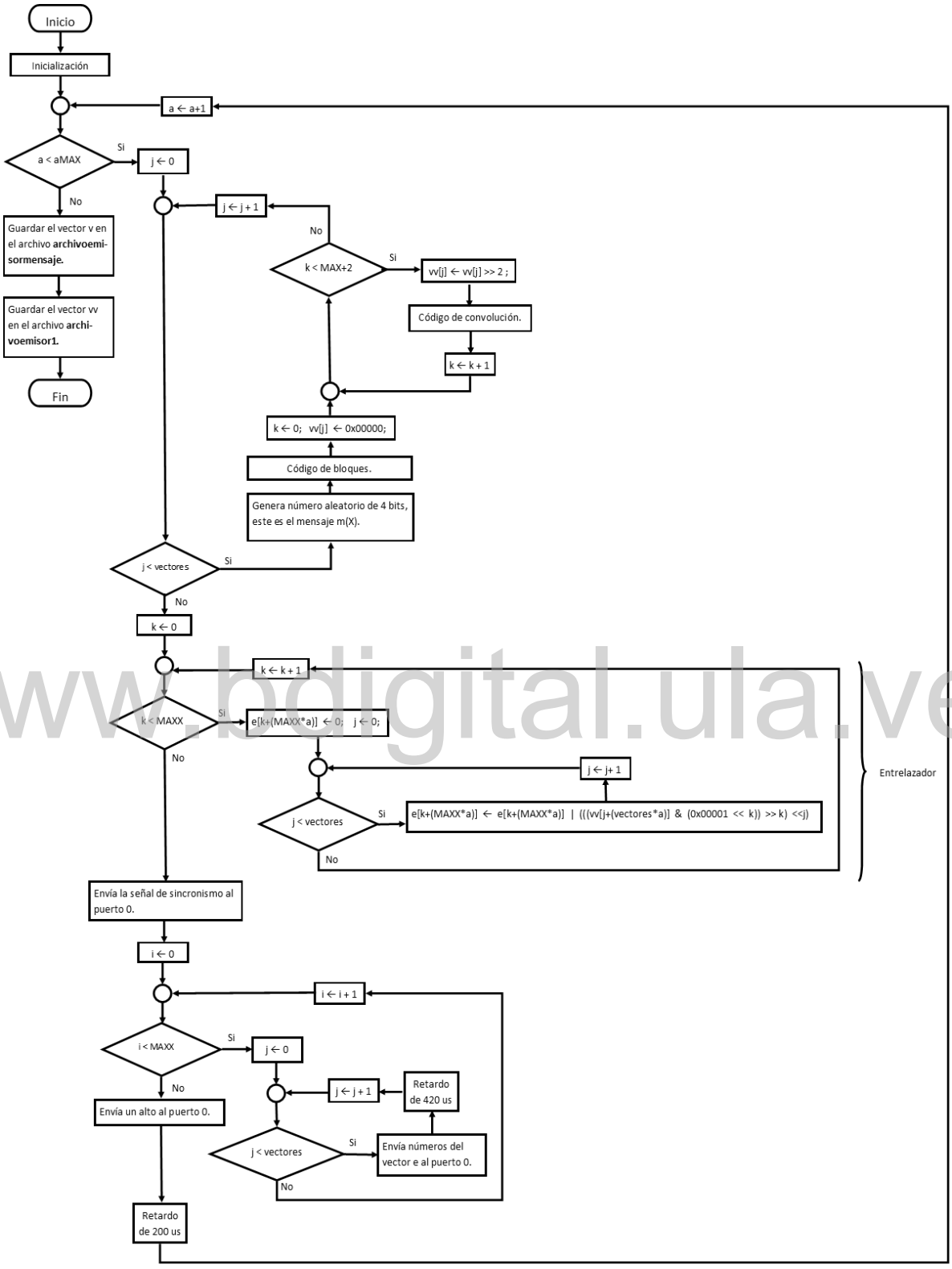


Figura 3.13. Diagrama de flujo del emisor de la primera prueba de la etapa 4.

Después el programa pasó a la siguiente trama de las aMAX tramas que se iban a generarar y luego a enviar, por tanto, se estuvieron entrelazando solo vectores números de MAXX bits en cada trama que se envió. Ahora, para la segunda prueba el emisor tiene el diagrama de flujo de las figuras 3.14a y 3.14b.

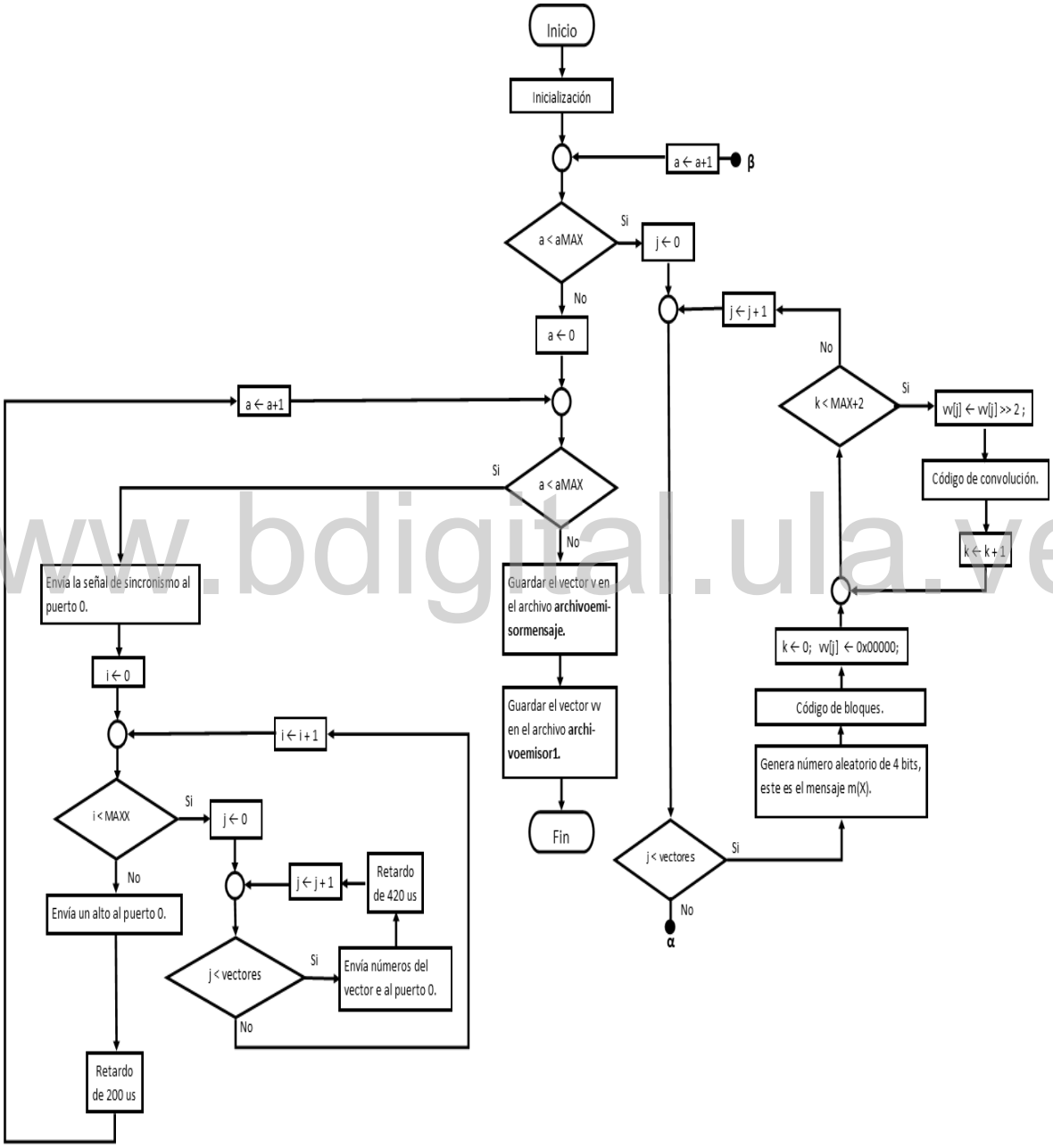


Figura 3.14a. Primera parte del diagrama de flujo del emisor de la segunda prueba de la etapa 4.

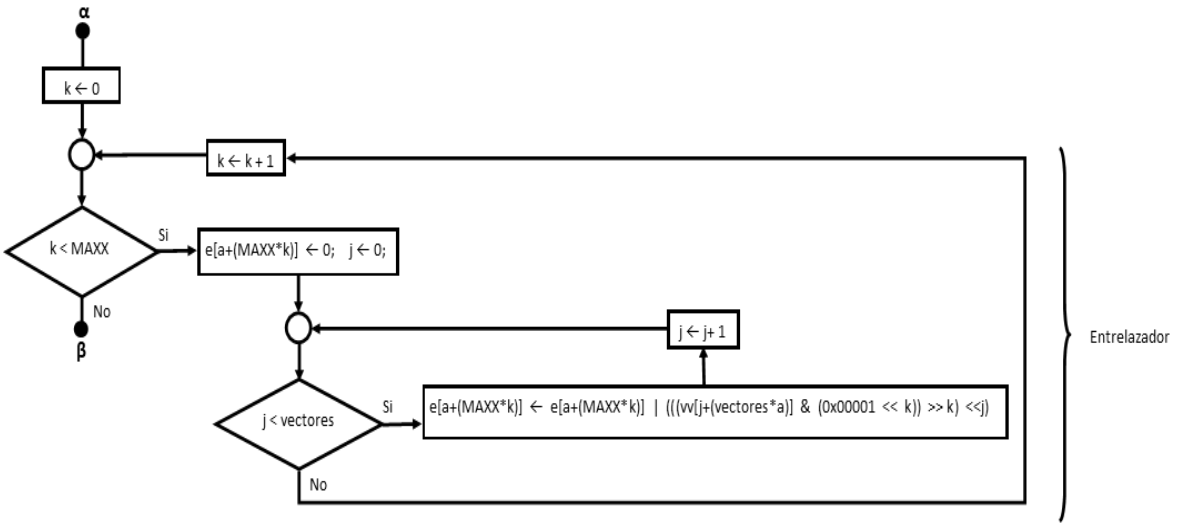


Figura 3.14b. Segunda parte del diagrama de flujo del emisor de la segunda prueba de la etapa 4.

El emisor de esta segunda prueba, tuvo el objetivo de entrelazar más bits, y dispersar aún más los errores. En la segunda prueba del entrelazador de la Raspberry Pi emisor, el programa fue el mismo de la Raspberry Pi emisor de la etapa 3, pero se le agregó en la inicialización, la declaración del vector **e** con MAXX*aMAX posiciones, para guardar los MAXX*aMAX números entrelazados de vectores bits. Además, en esta prueba el programa generó primero los vectores*aMAX números de MAXX bits que se deseaban enviar, luego los entrelazó y los guardó en el vector **e** de forma más separada, después transmitió este vector al receptor.

En el caso de la Raspberry Pi receptor de la cuarta etapa, se le agregó al código del receptor de la tercera etapa un entrelazador, de manera que la información que recibía fuese guardada en columnas. Una vez que terminó de recibir y guardar toda la información, el entrelazador reordenó la información en filas, y de esta manera ordenó la información a como estaba antes de que el entrelazador del emisor la mezclara. Posteriormente el código decodificó toda la información ordenada, con el decodificador de canal concatenado realizado en la etapa 3. El diagrama de flujo de la primera prueba de la cuarta etapa de la Raspberry Pi receptor es el indicado en las figuras 3.15a y 3.15b.

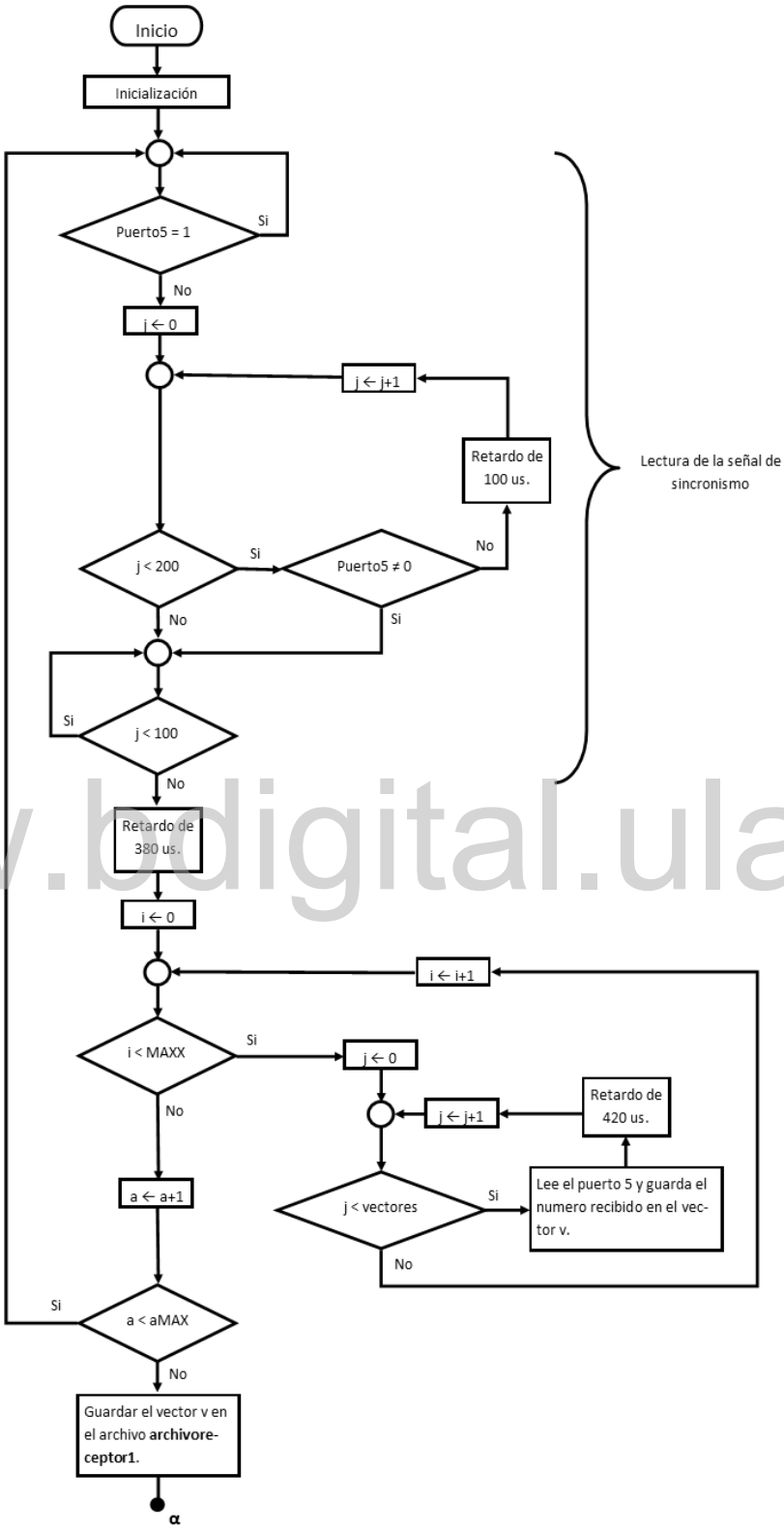


Figura 3.15a. Primera etapa del diagrama de flujo del receptor de la primera prueba y de la segunda prueba de la etapa 4.

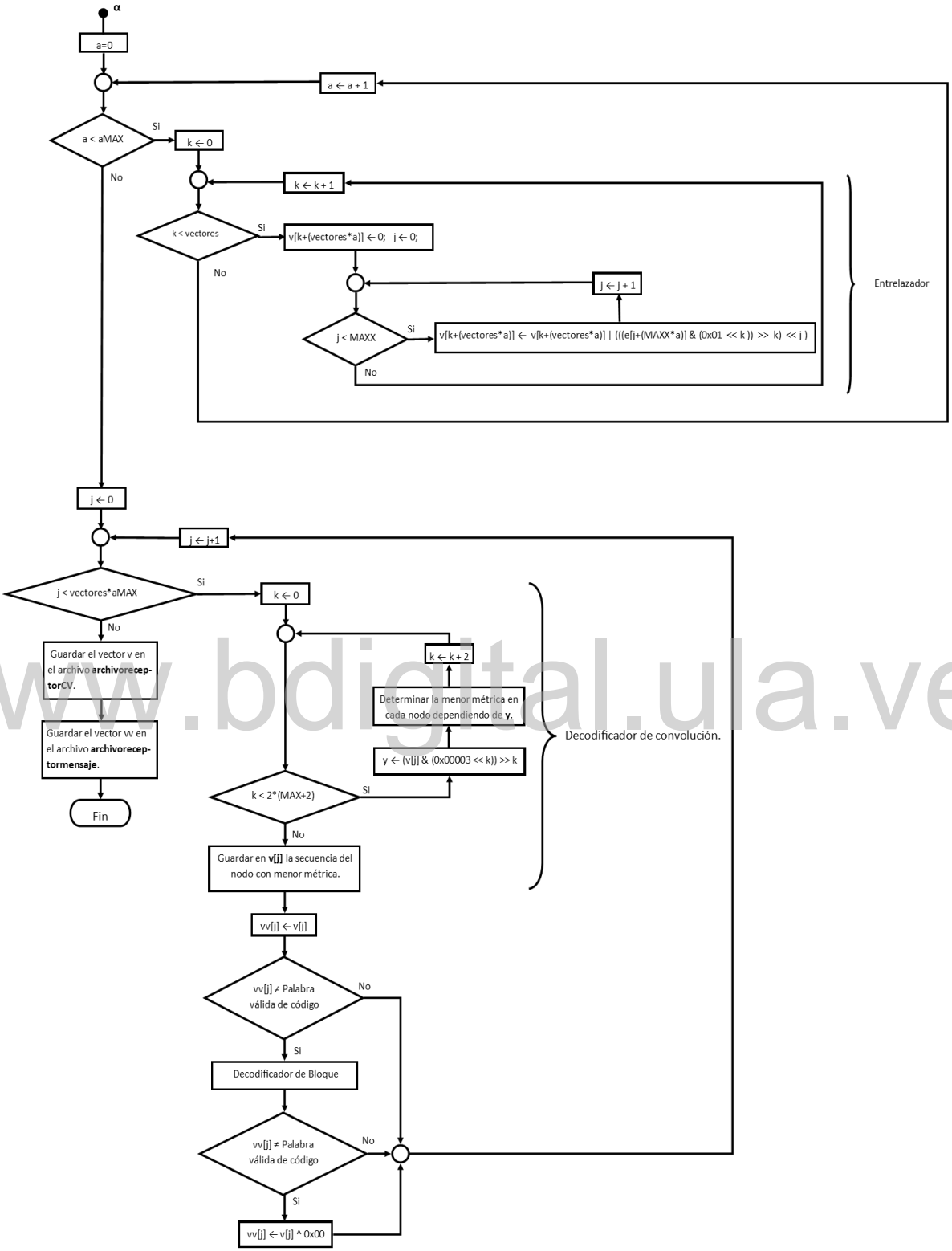


Figura 3.15b. Segunda etapa del diagrama de flujo del receptor de la primera prueba de la etapa 4.

Y el diagrama de flujo de la segunda prueba de la cuarta etapa de la Raspberry Pi receptor es el indicado en las figuras 3.15a y 3.16.

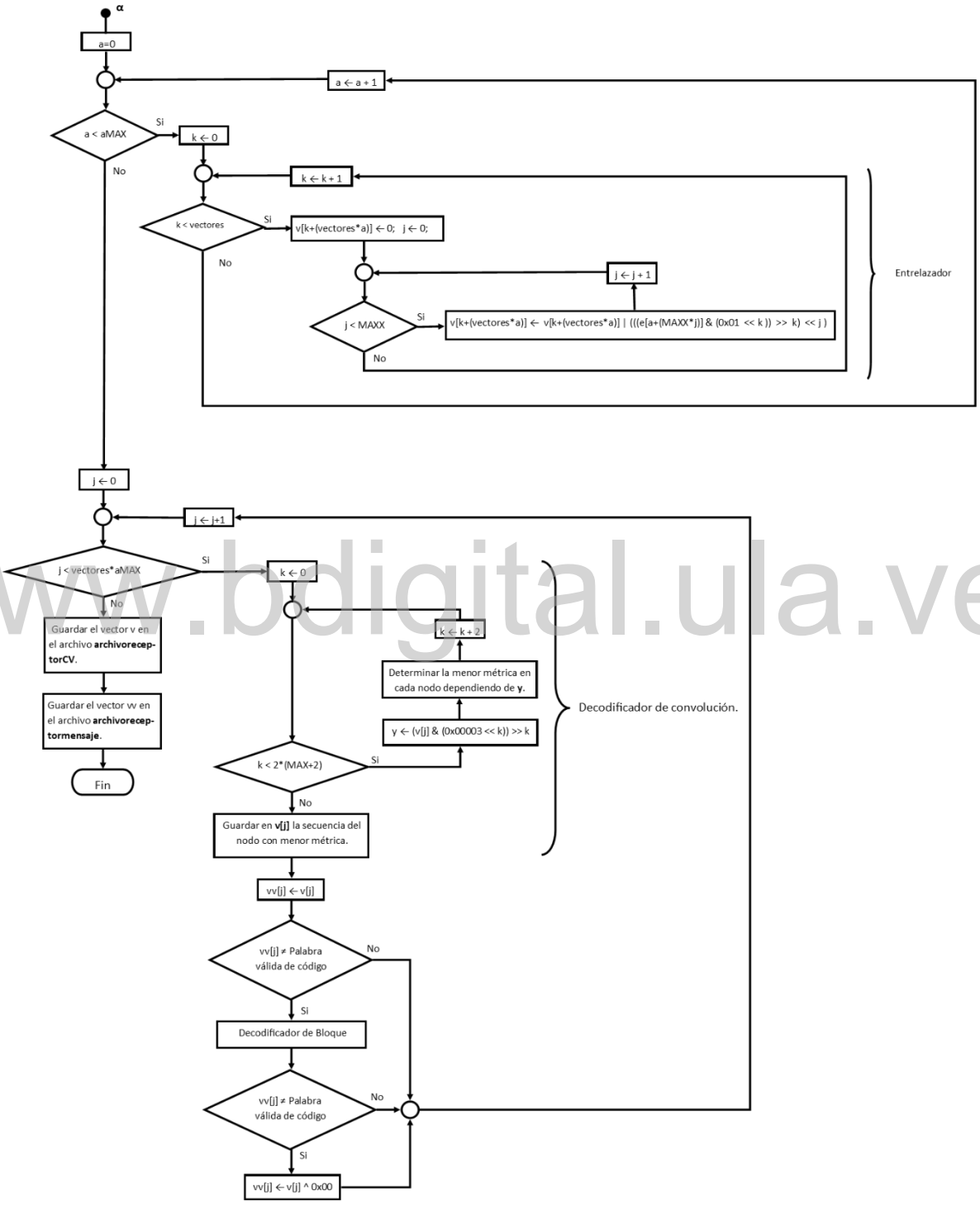


Figura 3.16. Segunda etapa del diagrama de flujo del receptor de la segunda prueba de la etapa 4.

Una muestra del correcto funcionamiento del código implementado en el emisor y el receptor de la segunda prueba en esta etapa se encuentra en el apéndice A.2, donde se encuentra una captura de pantalla del computador que se conecta remotamente con la Raspberry Pi emisor y la Raspberry Pi receptor. En la parte izquierda del apéndice se encuentra la pantalla de la Raspberry Pi emisor y en la parte derecha se encuentra la pantalla de la Raspberry Pi receptor, cada una muestra el resultado en pantalla de aplicar el código de esta etapa al emisor y al receptor respectivamente en cada Raspberry Pi. Donde la variable MAX es igual a 7, la variable MAXX es igual a 18, la variable vectores es igual a 4 y la variable aMAX es igual a 1, y los mensajes de 4 bits que se enviaron en la ejecución de este programa son los primeros 4 mensajes que están en la tabla 3.1, en el mismo orden de aparición que señala esta tabla. Y además son los mismos 4 mensajes que generan las palabras de código de la figura 3.11, de esta manera también se puede corroborar el funcionamiento del entrelazador. Si se observa el vector **e** del emisor en el apéndice A.2, se aprecia que el entrelazador funciona correctamente, ya que da el mismo resultado que el señalado en la figura 3.12.

Para determinar el número de errores cometidos y corregidos en el sistema de comunicación implementado en esta cuarta etapa, para ambas pruebas, se utilizó el programa indicado en el diagrama de flujo de las figuras 3.10a y 3.10b.

CAPITULO IV
ANALISIS DE RESULTADOS

A continuación, se presentan los análisis de resultados de las diferentes etapas que se llevaron a cabo en la implementación de la codificación de canal concatenada.

4.1 ANÁLISIS DE RESULTADOS DE LA PRIMERA ETAPA DE LA IMPLEMENTACIÓN.

Para verificar el correcto funcionamiento de la primera etapa, primero se usó como canal de comunicación cables entre los puertos de comunicación de la Raspberry Pi emisor y de la Raspberry Pi receptor. Una vez que se realizó la transmisión de información entre las dos Raspberry Pi, se calcularon los errores que se cometieron en la comunicación. Con este canal en diferentes ejecuciones del programa, con diferentes cantidades de bits a trasmitir y con diferentes velocidades de transmisión se obtuvieron valores de error igual a 0. Esto es debido a la corta distancia que hubo entre las dos Raspberry Pi, además este canal es menos propenso al ruido que un canal no guiado. Luego, se procedió a realizar la transmisión de información de la primera etapa usando como canal de comunicación el módulo RF 433. En este caso, se reflejó la alta sensibilidad del receptor de este módulo al ruido, ya que el receptor detectaba constantemente señales aun cuando no se está transmitiendo nada.

En esta etapa para una velocidad de transmisión de 1 bit cada 420 us, es decir, 2.380 bps, se obtuvieron diversos resultados dependiendo de los valores que se les dieron a las variables, MAX, que representa el número de bits que tendrá cada número binario, la variable vectores, que representa la cantidad de números binarios de MAX bits que se iban a generar, y la variable aMAX que representa la cantidad de veces que se repetiría la generación y envío de los vectores números binarios de MAX bits. Para alcanzar las velocidades de transmisión,

de 2.500 bps y de 9.090 bps, se modificaron los retardos de los diagramas de flujo del emisor y del receptor de esta etapa, ubicados en las figuras 3.2 y 3.3.

La tabla 4.1 señala los diferentes casos que fueron aplicados en las pruebas de la primera etapa. Cada caso indica los valores que tienen las variables MAX, aMAX y vectores, y también indica el total de bits que se transmiten en el mismo.

Tabla 4.1. Casos aplicados en la implementación de la etapa 1.

Caso	MAX	vectores	aMAX	Cantidad de bits transmitidos
1	8	20	640	102.400
2	16	10	640	102.400
3	16	64	100	102.400

La tabla 4.2 señala las tasas de error binario, o BER, que se obtuvieron en la implementación de la etapa 1 a partir el código indicado en el diagrama de flujo de la figura 3.4, para cada caso de la tabla 4.1, con velocidades de transmisión de 2.380 bps, 2.500 bps y 9.090 bps.

Tabla 4.2. Número de errores en la implementación de la etapa 1 para los diferentes casos de la tabla 4.1 con diferentes velocidades de transmisión.

	Velocidad de transmisión (bps)		
	2.380	**2.500**	**9.090**
BER Caso 1	0,142	0,163	0,307
BER Caso 2	0,160	0,170	0,320
BER Caso 3	0,465	0,425	0,466

Para obtener los datos de las tablas 4.2, 4.4, 4.6, 4.8, 4.10, se realizaron para cada caso evaluado, por cada velocidad de transmisión, 5 simulaciones del código, por tanto, los datos señalados en dichas tablas, son un valor promedio que se obtuvo de promediar los resultados de las distintas ejecuciones del código para cada situación, esto con el fin de que los resultados ofrecidos por la implementación sean confiables. De la tabla 4.2, si se observa el caso 1, donde se transmitieron un total de 102.400 bits, para la velocidad de transmisión de

2.380 bps, se obtuvo un promedio de 0,142 de la BER, para la velocidad de transmisión de 2.500 bps se obtuvo un promedio de 0,163 de la BER, y para la velocidad de transmisión de 9.090 bps, se obtuvo una tasa de error binario de 0,307. Se puede apreciar que, a medida que la velocidad de transmisión aumentó, la tasa de error binario también aumentó, es decir, la cantidad de errores en la transmisión aumentó.

Los casos 1, 2 y 3, transmitían la misma cantidad de bits, un total de 102.400 bits, pero si se observa la tabla 4.1, cada caso tiene valores diferentes para las variables MAX, vectores y aMAX. A pesar de que enviaban la misma cantidad de bits, el promedio de errores en cada caso es diferente, por ejemplo, en el caso 1 para una velocidad de transmisión de 2.380 bps se obtuvo una tasa de error binario de 0,142, para el caso 2, para la misma velocidad señaló una tasa de error binario de 0,160, y en el caso 3, para la misma velocidad de transmisión se obtuvo una tasa de error binario de 0,465. Concluyendo de esta comparación, que el caso 1 tuvo una menor tasa de error binario, en todas las velocidades, que el caso 2, y el caso 2 tuvo a su vez, una menor tasa de error binario, en todas las velocidades, que el caso 3. A partir de los casos 1 y 2, se puede concluir que se generaron menos errores en la transmisión de información cuando la variable MAX era más pequeña.

Y de los casos 2 y 3, la tasa de error era menor cuando la variable vectores era más pequeña. Esto sucedió porque las variables MAX y vectores, definían el tamaño de la trama a enviar, mientras más pequeñas eran las tramas, menor era la tasa de error binario. Por tanto, si se desea enviar un número alto de bits, es conveniente enviar tramas de bits más pequeñas y repetir este proceso más veces, al contrario de, enviar tramas muy largas y repetir este proceso menos veces, esto se logra, disminuyendo las variables MAX y vectores, y aumentando la variable aMAX.

4.2 ANÁLISIS DE RESULTADOS DE LA SEGUNDA ETAPA DE LA IMPLEMENTACIÓN.

Para la segunda etapa, la tabla 4.3 señala los casos evaluados en el sistema de comunicación donde se implementó como técnica de codificación de canal el código cíclico (7,4), siendo MAX la variable que definía el tamaño de la palabra de código de bloque, vectores la variable que definía cuantas palabras de código de MAX bits se enviarían por cada señal de

sincronismo, y aMAX la variable que definía cuantas tramas se iban a enviar o cuantas veces se iban a enviar las vectores palabras de código de MAX bits.

Tabla 4.3. Casos aplicados en la implementación de la etapa 2.

Caso	MAX	vectores	aMAX	Cantidad de bits transmitidos
1	7	10	200	14.000
2	7	10	500	35.000
3	7	20	500	70.000
4	7	20	725	101.500

La tabla 4.4 señala la BER o tasa de error binario para cada caso de la tabla 4.3, con velocidades de transmisión de 2.380 bps, 2.500 bps y 9.090 bps. Estos son los resultados de implementar el código indicado en el diagrama de flujo de la figura 3.7.

Tabla 4.4. Número de errores en la implementación de la etapa 2 para los diferentes casos de la tabla 4.3 con diferentes velocidades de transmisión.

		Velocidad de transmisión (bps)		
		2380	2500	9090
BER sin Codificación de canal.	**BER Caso 1**	0,102	0,104	0,252
	BER Caso 2	0,096	0,119	0,257
	BER Caso 3	0,131	0,155	0,305
	BER Caso 4	0,143	0,143	0,296
BER con Codificación de canal.	**BER Caso 1**	0,099	0,103	0,248
	BER Caso 2	0,093	0,112	0,252
	BER Caso 3	0,127	0,143	0,304
	BER Caso 4	0,141	0,141	0,294
BER con Codificación de canal, sin contar unos casos.	**BER Caso 1**	0,092	0,095	0,223
	BER Caso 2	0,086	0,113	0,229
	BER Caso 3	0,117	0,139	0,272
	BER Caso 4	0,128	0,129	0,264

Para alcanzar las velocidades de transmisión, de 2.500 bps y de 9.090 bps, se modificaron los retardos de los diagramas de flujo del emisor y del receptor de esta etapa, ubicados en las figuras 3.5 y 3.6a.

A partir de la tabla 4.4, si se observan y se comparan las filas denominadas **BER sin Codificación de canal** y **BER con Codificación de canal,** para los diferentes casos y velocidades de información, se puede apreciar que la tasa de error binario del sistema de comunicación disminuyó cuando se agregó al sistema de comunicación el código de bloques. A su vez, se pudo apreciar que la tasa de error binario disminuyó tan solo un poco, por ejemplo, si se observa la fila denominada **BER sin Codificación de canal**, el caso 1, para la velocidad de 2.380 bps, se obtuvo una BER de 0,102, y para el caso de la fila denominada **BER con Codificación de canal**, para el caso 1, y con la misma velocidad de 2.380 bps, se obtuvo una BER de 0,099. Esto se debe a que los parámetros que se seleccionaron para definir el código de bloques, permiten corregir solo 1 error por cada palabra de código de 7 bits. Es decir, que si ocurría más de 1 error, el código de bloques no corregía los errores o en algunos casos, provocaba que esta palabra de código tuviese aún más errores, como se puede observar en algunos de los casos de la tabla, más específicamente, en la fila denominada **BER con Codificación de canal, sin contar unos casos**, esta fila muestra los resultados de añadir al programa de detección de errores del código de bloques, el de la figuras 3.7a y 3.7b, un programa que permitía eliminar todos los casos en los que se realizó la decodificación a palabras de código que no era necesario aplicarle el mismo, pues dicha palabra tenía más de 1 error, con esto se filtraban todos los casos que no se les podía decodificar por la cantidad de errores, y además ayudaba a detectar, cuanta era la tasa de error binario para el caso ideal, donde el código de bloques corregía solo los casos donde había 1 solo error en la palabra de código recibida.

Se observa de los valores obtenidos en esta fila, que el código de bloques en algunos casos, generó más errores y en otros, si realizó la corrección de errores correctamente. Por eso esta fila denominada **BER con Codificación de canal, sin contar unos casos**, para los diferentes casos, señala tasas de error binario inferiores a las tasas de error obtenidas de la fila denominada **BER con Codificación de canal**. Igualmente, descartando los casos en los que el código añadió más errores, la tasa de error binario disminuyó tan solo un poco con este código de bloques, por ejemplo, si se observa la fila denominada **BER con Codificación de canal**, para el caso 1, con la velocidad de 2.380 bps, se obtuvo una BER de 0,099, y para el mismo caso y la misma velocidad, de la fila denominada **BER con Codificación de canal, sin contar unos casos,** la BER fue de 0,092. Lo que señala que existían muchos errores en

ráfaga en el sistema de transmisión implementado, impidiéndole al código de bloques corregir un mayor número de errores.

4.3ANÁLISIS DE RESULTADOS DE LA TERCERA ETAPA DE LA IMPLEMENTACIÓN.

Para la tercera etapa, la tabla número 4.5 señala los casos evaluados en el sistema de comunicación donde se aplicó como técnica robusta de codificación de canal, el código cíclico concatenado al código de convolución, siendo MAX la variable que definía el tamaño de la palabra de código de bloque, vectores la variable que definía cuantas secuencias de (2*(MAX+2)) bits se enviarían por cada señal de sincronismo, y aMAX la variable que definía cuantas tramas se enviarían o cuantas veces se iban a enviar las vectores palabras de código de MAX bits.

Tabla 4.5. Casos aplicados en la implementación de la etapa 3.

Caso	MAX	vectores	aMAX	Total de bits transmitidos
1	7	6	130	14.040
2	7	18	60	19.440
3	7	18	100	32.400
4	7	18	320	103.680

La tabla 4.6 señala la tasa de error binario, o la BER, que se obtuvo de la implementación de la etapa 3 a partir del código indicado en el diagrama de flujo de las figuras 3.10a y 3.10b, para cada caso de la tabla 4.5, con velocidades de transmisión de 2.380 bps, 2.500 bps y 9.090 bps. Para alcanzar las velocidades de transmisión, de 2.500 bps y de 9.090 bps, se modificaron los retardos de los diagramas de flujo del emisor y del receptor de esta etapa, ubicados en las figuras 3.8 y 3.9a.

A partir de la tabla 4.6 se puede apreciar que a medida que aumentaba la cantidad de bits a transmitir, el porcentaje de error era mayor. Si se aprecian los resultados obtenidos para todos los casos, comparando la fila llamada **BER sin Codificación de canal**, con las filas denominadas **BER con Código de convolución** y **BER con Código de bloques**, que sin la

codificación de canal se obtuvo una menor tasa de error binario, es decir, la codificación de canal, añadió más errores al sistema de transmisión. También, de apreciar las filas denominadas **BER con Código de convolución** y **BER con Código de bloques,** de la tabla 4.6, se observa que el código de bloques no aportó nada al sistema de comunicación, debido a que refleja la misma tasa de error binario que las indicadas en la fila **BER con Código de convolución**.

Tabla 4.6. Número de errores en la implementación de la etapa 3 para los diferentes casos de la tabla 4.5 con diferentes velocidades de transmisión.

		Velocidad de transmisión (bps)		
		2380	**2500**	**9090**
BER sin Codificación de canal.	**BER Caso 1**	0,108	0,101	0,283
	BER Caso 2	0,155	0,155	0,351
	BER Caso 3	0,181	0,195	0,366
	BER Caso 4	0,221	0,236	0,372
BER con Código de convolución.	**BER Caso 1**	0,109	0,115	0,298
	BER Caso 2	0,172	0,177	0,374
	BER Caso 3	0,200	0,205	0,385
	BER Caso 4	0,238	0,253	0,389
BER con Código de bloques.	**BER Caso 1**	0,109	0,114	0,298
	BER Caso 2	0,172	0,177	0,374
	BER Caso 3	0,200	0,205	0,385
	BER Caso 4	0,238	0,253	0,389
BER de solo el mensaje, después del código de concatenado.	**BER Caso 1**	0,111	0,119	0,302
	BER Caso 2	0,156	0,160	0,342
	BER Caso 3	0,200	0,210	0,385
	BER Caso 4	0,240	0,258	0,389

Teniendo en cuenta que el código de bloques implementado corregía solo 1 error por palabra de código de 7 bits, y el código de convolución implementado corregía 2 errores por secuencia de 18 bits, incluso, podía corregir en algunos casos, más de 2 errores si estos estaban espaciados a una longitud superior de la longitud de restricción, que era igual a 3, para el caso implementado en esta etapa. Se concluye que en la transmisión de información estaban ocurriendo errores en ráfaga, uno tras otro, a una medida que dificultaba demasiado, que tanto el código de convolución como el código de bloques pudiesen funcionar correctamente. Por tanto, el código de convolución se estaba enfrentando a casos donde existían más de 2 errores en las secuencias de 18 bits, espaciados en longitudes menores que

la longitud de restricción. Y luego este código, pasó una palabra de código de 7 bits al decodificador de bloque, la cual probablemente tenía más de 1 error, lo cual impedía que el decodificador de bloque operase correctamente.

De la etapa 1 se concluyó que mientras mayor era la trama que se enviaba, más errores se cometían en la transmisión de información, por tanto, era conveniente asignar valores pequeños a las variables, MAX y vectores, y esto se compensaba aumentando la variable aMAX. Se puede corroborar que se cometieron más errores cuando las variables MAX y vectores eran de valores más grandes, si se compara el caso 4 de la fila denominada **BER sin Codificación de canal** de la tabla 4.4 y el caso 4 de la final denominada **BER sin Codificación de canal** de tabla 4.6, en los cuales se enviaron alrededor de 100.000 bits, la diferencia estaba en que, en el caso de la etapa 3, se enviaron 60 veces 18 palabras de código de 18 bits, es decir, se enviaron 60 veces tramas de 324 bits, en vez de enviar 725 veces tramas de 140 bits, como fue en el caso de la etapa 2, teniendo este último, una menor tasa de error binario. En el caso de la tabla 4.6 se enviaron tramas de 18*18, porque se deseaba agregar en la próxima etapa un entrelazador al sistema de comunicación, como técnica para dispersar las ráfagas de errores que impedían que tanto el código de convolución como el código cíclico (7,4) corrigiesen una mayor cantidad de errores.

4.4ANÁLISIS DE RESULTADOS DE LA CUARTA ETAPA DE LA IMPLEMENTACIÓN.

Para la cuarta etapa, la tabla número 4.7 señala los casos evaluados en el sistema de comunicación de la primera prueba de esta etapa, donde se agregó un entrelazador al esquema de codificación robusto de la etapa 3. En esta primera prueba, se generaron vectores números de MAXX bits, después se entrelazaron y se enviaron al receptor, esta operación se repitió aMAX veces.

La tabla 4.8 señala la cantidad de bits incorrectos que se obtuvieron en la implementación de la primera prueba de la etapa 4 a partir del código señalado en el diagrama de flujo de las figuras 3.10a y 3.10b, para cada caso de la tabla 4.7, con velocidades de transmisión de 2.380 bps, 2.500 bps y 9.090 bps.

Tabla 4.7. Casos aplicados en la implementación de la primera prueba de la cuarta etapa.

Caso	MAX	MAXX	vectores	aMAX	Total de bits transmitidos
1	7	18	6	130	14.040
2	7	18	18	60	19.440
3	7	18	18	100	32.400
4	7	18	4	200	14.400

Para alcanzar las velocidades de transmisión, de 2.500 bps y de 9.090 bps, se modificaron los retardos de los diagramas de flujo del emisor y del receptor de esta etapa, ubicados en las figuras 3.13 y 3.15a.

Tabla 4.8. Número de errores de la implementación de la primera prueba de la etapa 4 para los diferentes casos de la tabla 4.7 con diferentes velocidades de transmisión.

		Velocidad de transmisión (bps)		
		2380	2500	9090
BER sin Codificación de canal.	BER Caso 1	0,112	0,115	0,284
	BER Caso 2	0,177	0,170	0,364
	BER Caso 3	0,194	0,222	0,382
	BER Caso 4	0,070	0,058	0,266
BER con Código de convolución.	BER Caso 1	0,098	0,101	0,283
	BER Caso 2	0,155	0,160	0,369
	BER Caso 3	0,185	0,212	0,397
	BER Caso 4	0,059	0,048	0,263
BER con Código de bloques.	BER Caso 1	0,095	0,098	0,273
	BER Caso 2	0,154	0,160	0,368
	BER Caso 3	0,183	0,213	0,401
	BER Caso 4	0,047	0,058	0,262
BER con solo el mensaje.	BER Caso 1	0,102	0,109	0,320
	BER Caso 2	0,173	0,188	0,392
	BER Caso 3	0,216	0,246	0,431
	BER Caso 4	0,054	0,070	0,317

De este sistema de comunicación donde se encuentra concatenado un entrelazador al código de canal robusto de la etapa 3, se puede apreciar de la tabla 4.8, que la tasa de error binario de la fila denominada **BER sin Codificación de canal**, para todos los casos, era mayor que la fila denominada **BER con Código de convolución**, y a su vez todos los casos

de esta última fila, eran mayores que los señalados por la fila denominada **BER con Código de bloques.** Estos resultados indican que tanto el código de convolución como el código de bloques, disminuyeron la tasa de error binario. La tasa de error a la que se llegó luego de aplicar la codificación de canal robusta era pequeña, pero se debía a que el entrelazador solo estaba mezclando pequeñas tramas, luego las enviaba, y continuaba hasta cumplir con el total de bits a transmitir.

De igual forma se evalúo el aporte del entrelazador al codificador de canal robusto, si se comparan los casos 1, 2 y 3 de la tabla 4.6, con los casos 1, 2 y 3 de la tabla 4.8, respectivamente. A partir de comparar estos casos, observando más específicamente las primeras tres filas de cada tabla, se puede apreciar que en la tabla 4.6, donde no existía en el sistema un entrelazador, el codificador de canal no aportó nada al sistema, y que en la tabla 4.8, donde si existía un entrelazador, el codificador de canal robusto aportó algo al sistema de comunicación. A partir de estos resultados, se concluye que el entrelazador le permitió a la codificación de canal robusta de la etapa 3, mejorar la tasa de error binaria del sistema de comunicación implementado, es decir, que le ofreció a cada codificador de canal la posibilidad de operar bajo sus limitaciones.

Además, también se puede observar de la tabla 4.8 que, para cada caso, la tasa de error bianria era menor, si la velocidad de información era menor. Y además si se observa de todas las filas de la tabla 4.8, y se comparan los casos 1 y 4, ambos casos envían alrededor de 14.000 bits, se aprecia que la BER se redujo mucho cuando la variable vectores era igual a 4, es decir, era pequeña, como lo era en el caso 4. Es muy importante tomar en consideración el tamaño de las tramas a enviar, ya que el caso 4 tenía una menor tasa de error binario, que la del caso 1, y en lo único que diferían era en el tamaño de las tramas y la cantidad de veces que estas se enviaban.

Para la cuarta etapa, la tabla número 4.8 señala los casos evaluados en el sistema de comunicación de la segunda prueba de esta etapa, donde se agregó un entrelazador al esquema de codificación robusto de la etapa 3. A diferencia de la primera prueba, en esta segunda prueba, se generaron y entrelazaron todas las vectores*aMAX secuencias que se deseaban enviar, y luego si se enviaron aMAX veces tramas de MAX*vectores bits al receptor, todo esto con el fin de separar aún más los bits de cada secuencia al momento de la

transmisión de la información. Siendo MAX la variable que definía el tamaño de la palabra de código de bloque, MAXX la variable que definía el tamaño de la secuencia generada por el código de convolución, vectores la variable que definía cuantas secuencias de MAXX bits se enviarían por cada señal de sincronismo, y aMAX la variable que definía cuantas tramas se enviarían o cuantas veces se iban a enviar las vectores palabras de código de MAX bits.

Tabla 4.9. Casos aplicados en la implementación de la segunda prueba de la cuarta etapa.

Caso	MAX	MAXX	vectores	aMAX	Total de bits transmitidos
1	7	18	18	60	19.440
2	7	18	4	200	14.400
3	7	18	4	270	19.440
4	7	18	4	450	32.400

La tabla 4.10 señala la cantidad de bits incorrectos que se obtuvieron en la implementación de la segunda prueba de la etapa 4 a partir del código señalado en el diagrama de flujo de las figuras 3.10a y 3.10b, para cada caso de la tabla 4.9, con velocidades de transmisión de 2.380 bps, 2.500 bps y 9.090 bps. Para alcanzar las velocidades de transmisión, de 2.500 bps y de 9.090 bps, se modificaron los retardos de los diagramas de flujo del emisor y del receptor de esta etapa, ubicados en las figuras 3.14a y 3.16a.

En los casos que se transmitieron más de 14.000 bits, en la Raspberry Pi receptor se le añadió un delay de 400 us, justo antes de que empezara a leer la señal de sincronismo, con el objetivo de darle tiempo a la Raspberry Pi emisor de generar y entrelazar todos los bits para posteriormente enviarlos a la Raspberry Pi receptor.

Ahora a partir de la tabla 4.10, si se observan los casos 1 y 3, los cuales enviaban la misma cantidad de bits, pero en el caso 1, las tramas eran más grandes que las del caso 3, y se envían menos veces que las del caso 3, se aprecia que para el caso 3 se obtuvieron menores tasas de error binario, para las diferentes filas de esta tabla, y a su vez para cada velocidad. Concluyendo que el sistema funcionó mejor, cuando la información que se iba a transmitir se fraccionaba en tramas pequeñas y luego si se enviaba una tras otra.

Tabla 4.10. Número de errores de la implementación de la segunda prueba de la etapa 4 para los diferentes casos de la tabla 4.9 con diferentes velocidades de transmisión.

		Velocidad de transmisión (bps)		
		2380	**2500**	**9090**
BER sin Codificación de canal.	**BER Caso 1**	0,214	0,215	0,379
	BER Caso 2	0,055	0,089	0,221
	BER Caso 3	0,067	0,085	0,230
	BER Caso 4	0,076	0,082	0,243
BER con Código de convolución.	**BER Caso 1**	0,209	0,211	0,422
	BER Caso 2	0,006	0,020	0,218
	BER Caso 3	0,018	0,035	0,227
	BER Caso 4	0,022	0,032	0,243
BER con Código de bloques.	**BER Caso 1**	0,201	0,210	0,425
	BER Caso 2	0,006	0,019	0,215
	BER Caso 3	0,020	0,036	0,224
	BER Caso 4	0,022	0,033	0,241
BER con solo el mensaje.	**BER Caso 1**	0,206	0,221	0,447
	BER Caso 2	0,005	0,025	0,229
	BER Caso 3	0,022	0,038	0,230
	BER Caso 4	0,024	0,035	0,245

Si se comparan las tablas 4.8 y 4.10, en donde se usó el entrelazador, si se comparan el caso 4 de la tabla 4.8 y el caso 2 de la tabla 4.10, para todas las filas, el caso 2 de la tabla 4.10, tuvo menores tasas de error binario, señalando que el entrelazador de la segunda prueba era más efectivo, ya que dispersaba más los errores que el de la primera prueba. Lo mismo se puede observar del caso 3 de la tabla 4.8 comparado con el caso 4 de la tabla 4.10, señalando menores tasas de errores en las diferentes filas, el entrelazador de la tabla 4.10.

La conclusión de esta etapa es que el aporte del entrelazador era aún mayor cuando se entrelazaban primero todos los bits que se iban a transmitir, los MAXX*vectores*aMAX bits, y una vez que ya estaban todos entrelazados, se enviaban al receptor por tramas, de esta manera quedaban más separados los errores, a diferencia del entrelazador de la primera prueba de la cuarta etapa, en la que se mezclaban vectores números de MAXX bits, se enviaban y luego se volvían a mezclar vectores números de MAXX bits, hasta enviar el total de MAXX*vectores*aMAX bits. Si los bits estaban más dispersos, los errores que ocurrían en la transmisión de información también lo estaban, y mientras más dispersos estaban los errores, más fácil era para el codificador de canal concatenado implementado en este proyecto de grado, corregir los errores.

CONCLUSIONES

Con el objetivo de descubrir qué tipo de aporte puede generar la codificación de canal como técnica para mejorar la calidad de la transmisión de información en un sistema de comunicación, se realizó la implementación del codificador de canal robusto usando el código de bloques concatenado en serie con el código de convolución en dos Raspberry Pi, a partir de la cual se alcanzaron las siguientes conclusiones. En la primera etapa de la implementación, donde no se aplicó ningún tipo de codificación de canal, se destacó la importancia de que el tamaño de las tramas fuese pequeño para que el número de errores fuese pequeño, es decir, si era mucha la información que se deseaba transmitir, era mejor separarla en tramas pequeñas y mandarlas por separado hasta cumplir con el total de bits a enviar. Además, también los resultados señalaron que mientras más bits se enviaban, sin importar si se dividía la información en tramas, el número de errores se incrementaba.

Ahora, en la implementación de la segunda etapa, se agregó al sistema de comunicación el código de bloques (7,4) como técnica de corrección de errores. Los resultados de esta etapa reflejaron que la tasa de error binario o BER, después de la implementación del código de bloques (7,4), era bastante pequeña, e incluso en algunos casos aumentaba, en comparación con la BER obtenida antes de aplicar el código de bloques. De acuerdo a estos resultados, se puede concluir que el código de bloques implementado, se estaba enfrentando a un número de errores superior al que podía corregir por palabra de código, se pudo concluir esto debido a que el código de bloques implementado en esta etapa, podía corregir tan solo 1 error por palabra de código de 7 bits, y si ocurrían casos en los que existan 2 errores o más por palabra de código recibida, esta palabra recibida se parecía más a otra palabra de código válida, que a la palabra de código válida que fue enviada, por tanto este código trataba de arreglarla, trayendo como consecuencia que añadía más errores al sistema. Concluyendo que este código de bloques sirve para sistemas de comunicación donde los errores estén muy espaciados, específicamente para este caso, donde exista solo un error por palabra de código de 7 bits. Este código de bloques (7,4), tiene la posibilidad de funcionar mejor, si se concatena a otra codificación de canal previa, de manera que el primer codificador de canal corrija una mayor cantidad de errores, y solo le queden unos pocos errores a este código de bloques.

En la tercera etapa de la implementación, se agregó a la segunda etapa un código de convolución, el cual podía corregir hasta 2 errores por secuencia de 18 bits. Se observó de los resultados de esta etapa que, en este codificador de canal concatenado, la BER del código de convolución era la misma que la del código de bloques, por tanto, solo el código de convolución estaba generando resultados. Además, los resultados de la tabla 4.6 señalan que la BER del código de convolución concatenado con el código de bloques, era mayor que la BER donde no se aplicó ninguna codificación de canal, por consiguiente, el codificador de canal concatenado, empeoró la tasa de error del sistema. Esto es debido a que los códigos de codificación de canal, funcionan correctamente siempre que los errores ocurran bajo sus limitaciones. Se puede deducir de los resultados no favorables de esta etapa que, este codificador de canal robusto se enfrentó a una transmisión en la que ocurrían una gran cantidad de errores en ráfaga. En definitiva, es muy importante tomar en cuenta las limitaciones que ofrecen estos codificadores de canal en cuanto al número de errores que pueden corregir por secuencia de bits. Finalmente, se buscó como solución al problema a la ráfaga de errores, un entrelazador, ya que este último dispersa los errores ocurridos en la transmisión, permitiéndole a los códigos de convolución y de bloque (7,4), corregir un mayor número de errores.

La cuarta y última etapa se dividió en dos pruebas. En la primera prueba de esta etapa, se agregó un entrelazador a la etapa tres, que mezcló un número pequeño de bits por trama a enviar. En el caso de la segunda prueba de la cuarta etapa, se tenía un entrelazador que mezclaba un número superior de bits que la primera prueba de la cuarta etapa. De acuerdo a los resultados en ambas pruebas, se puede afirmar que el entrelazador permitió que el codificador de canal concatenado de la etapa tres, tuviese una menor BER. Además, si se observan los resultados obtenidos en esta etapa, la segunda prueba obtuvo mejores resultados que la primera prueba, esto es debido a que los bits a transmitir fueron entrelazados a mayor escala en la segunda prueba, y esto produjo que los bits erróneos estuviesen más dispersos. Tomando esto en consideración se puede concluir que concatenar en serie el codificador de convolución con el código de bloques (7,4), puede servir como técnica robusta para corregir un número grande de errores, siempre y cuando, se concatene a ellos un entrelazador, que permita dispersar los errores lo mayor posible, para cumplir con las limitaciones de la cantidad de errores que cada código puede corregir por secuencia de bits.

RECOMENDACIONES

Con el objetivo de corregir un mayor número de errores por palabra de código, se recomienda concatenar al código de convolución del sistema de comunicación implementado en el presente trabajo, un código de bloques como el Reed Solomon en vez del código de bloques (7,4) implementado en este trabajo de grado.

Se recomienda llevar a cabo este estudio empleando otras herramientas computacionales y otras tarjetas de desarrollo a modo de comparar los resultados obtenidos.

REFERENCIAS

[1] Briceño M, J. E. (1990). "*Principios de las Comunicaciones"*. Universidad de Los Andes. Mérida. Disponible: https://www.academia.edu/11803894/Principio_de_las_Comunicaciones_Jose_Bric e%C3%B1o_ULA-Electrica_1_1_

[2] Briceño M, J. E. (1991). "*Transmisión de datos"*. Universidad de Los Andes. Mérida. Disponible : https://www.academia.edu/37113086/Transmision_de_Datos

[3] Haykin, S. (2001). "*Communication Systems"*. Jhon Wiley & Sons, Inc. Nueva York.

[4] Shannon, C. E. (1948). "*A Mathematical Theory of Communication"*. The Bell System Technical Journal, Vol. 27, pp. 379–423, 623–656, Julio, Octubre, 1948. Disponible: http://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

[5] Balderas S, H. F. (2008). "*La modulación codificada entramada (TCM) en los nuevos estándares de comunicación"*. Tesis. Instituto Politécnico Nacional. Escuela Superior de Ingeniería Mecánica y Eléctrica. México, D.F. Disponible: https://tesis.ipn.mx/jspui/bitstream/123456789/2041/1/LA%20MODULACION%20 CODIFICADA%20ENTRAMADA.pdf

[6] Victorino V, C. G. (2008). "*Concatenación de Códigos Convolucionales de Canal con Códigos Espacio-Tiempo y Espacio-Frecuencia"*. Tesis. Centro de Investigación Científica y de Educación Superior de Ensenada. México, Baja California. Disponible: https://cicese.repositorioinstitucional.mx/jspui/handle/1007/182

[7] Derteano H, S. A. (2012). "*Aplicación y Evaluación de Estrategias para Control de Errores en Canales Satelitales mediante BICM"*. Memoria para optar al título de ingeniero civil electricista. Universidad de Chile. Departamento de Ingeniería Eléctrica. Santiago de Chile. Disponible: http://repositorio.uchile.cl/handle/2250/111912

[8] Ortega M, J. A. (2017). "*Situación del Código LDCP del Estándar AOS en Sistemas de Transmisión de Datos Satelital"*. Trabajo Especial de Grado. Universidad Central de Venezuela. Caracas.

[9] Serway, R. A. y Jewett J, J. W. (2009). "*Física para ciencias e ingeniería con Física Moderna*". Cengage Learning. Vol 2, pp. 952-968. México D.F. Disponible: https://www.academia.edu/27915502/Serway_7_Edicion_2_Volumen

[10] Troncoso H, F. I. (2012). "*Evaluación y Aplicación de Estrategias para Control de Errores en Canales Satelitales mediante Codificación Algebraica"*. Memoria para optar al título de ingeniero civil electricista. Universidad de chile. Santiago de Chile. Departamento de Ingeniería Eléctrica. Disponible: http://repositorio.uchile.cl/handle/2250/112615

[11] Mackay, S., Park, J. y Wright, E. (2003). "*Practical Data Communications for Instrumentation and Control"*. Elsevier. Gran Bretaña. Disponible: https://www.elsevier.com/books/practical-data-communications-for-instrumentation-and-control/mackay/978-0-7506-5797-6

[12] Sklar, B. (1988). "*Digital communications: fundamentals and applications"*. Prentice-Hall, Inc.

[13] Raspberry Pi. [Página Web en Línea]. Disponible: https://www.raspberrypi.org/

[14] Contreras, L. (2020). Raspberry Pi. [Página Web en Línea]. Disponible: https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

[15] Entradas y salidas digitales de la Raspberry Pi. [Página Web en Línea]. Disponible: https://franciscomoya.gitbooks.io/taller-de-raspberry-pi/content/es/elems/gpio.html

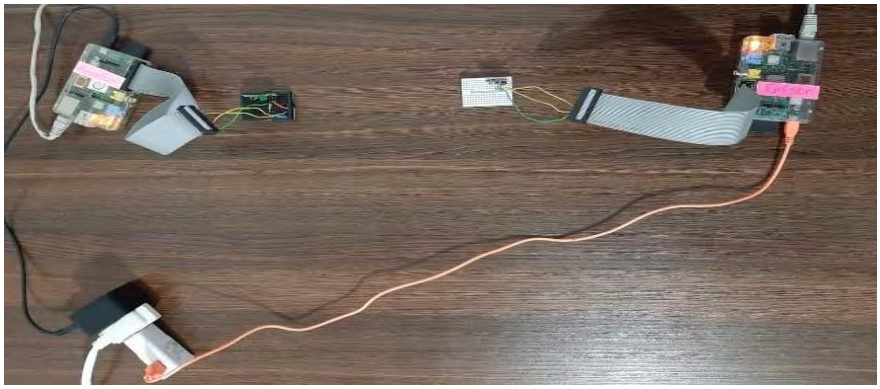
[16] Control de GPIO con Python en Raspberry Pi. [Página Web en Línea]. Disponible: https://www.programoergosum.com/cursos-online/raspberry-pi/238-control-de-gpio-con-python-en-raspberry-pi/que-es-gpio

[17] Modulo RF 433 MHz TX y RX. Naylamp Mechatronics SAC. [Página Web en Línea]. Disponible: https://naylampmechatronics.com/inalambrico/13-modulo-rf-433mhz.html

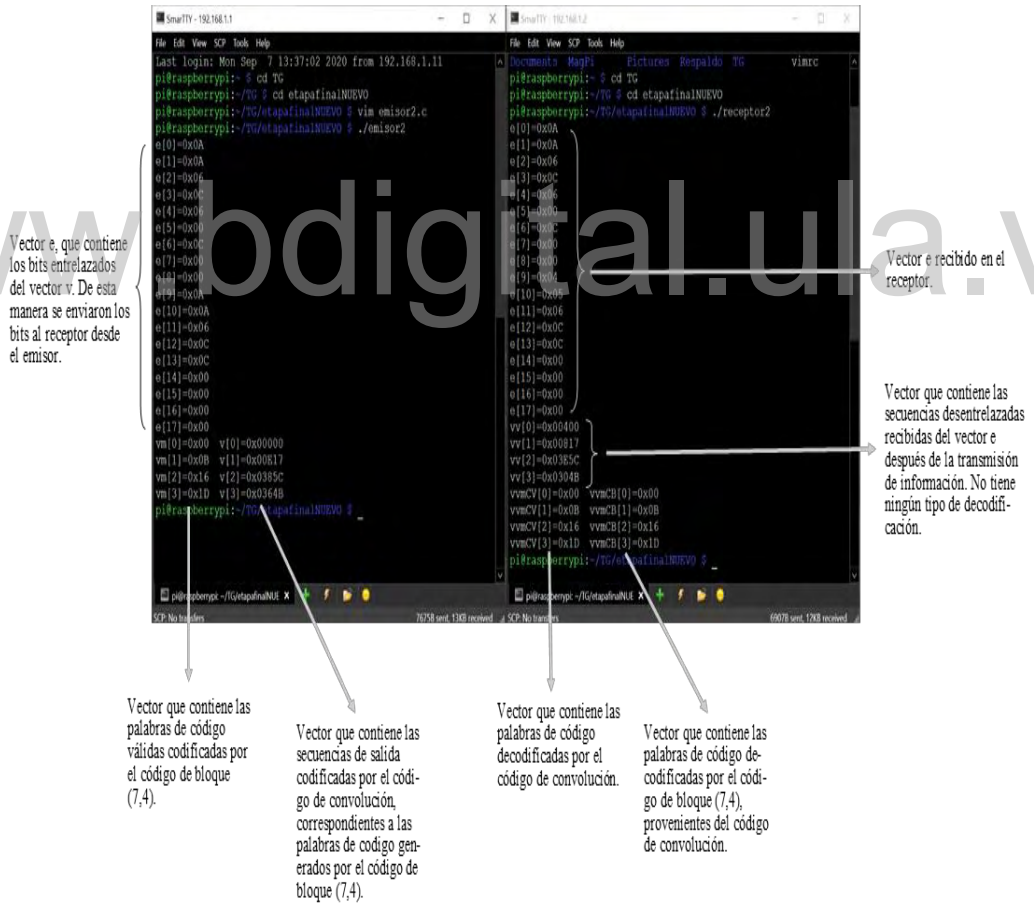
[18] Modulo RF 433 MHz -Emisor Y Receptor. Patagoniatec blog. [Página Web en Línea]. Disponible: https://saber.patagoniatec.com/2015/04/modulos-emisor-y-receptor-rf-433mhz/

[19] Ondas de radio. Wikipedia. [Página Web en Linea]. Disponible: https://es.wikipedia.org/wiki/Ondas_de_radio

APÉNDICES



A.1 Sistema de comunicación implementado en el presente trabajo de grado.



A.2 Captura de la pantalla del computador, esta imagen refleja las pantallas de las Raspberry Pi del sistema de comunicación implementado, la pantalla a la izquierda es la de la Raspberry Pi emisor, y la pantalla a la derecha es la de la Raspberry Pi receptor.