Universidad del Azuay

Departamento de Posgrados

Maestría en Telemática

TEMA: ANÁLISIS, MODELADO Y SIMULACIÓN DE ALGORITMOS SOBRE TÉCNICAS DE COMUNICACIÓN TOLERANTE A FALLAS APLICADAS EN SISTEMAS INDUSTRIALES

HUGO MARCELO TORRES SALAMEA

Director: Master Francisco Vásquez

Trabajo de tesis para optar al grado de Magister en Telemática

Cuenca - Ecuador

2014

DEDICATORIA

Este presente trabajo de tesis deseo dedicar a mis hijos GEOVANNY y ESTEBAN, que son la razón de mi vida y que continuamente me estuvieron apoyando con su comprensión y cariño en los momentos más difíciles, además deseo dejarles a mis hijos una nueva enseñanza que nunca es tarde para alcanzar los objetivos y metas que uno se propone a pesar de las diversas dificultades que se puede encontrar en el camino.

A mi esposa FANNY por su apoyo incondicional y su comprensión durante el transcurso de mis estudios y desarrollo del presente trabajo.

A Mis padres HUMBERTO Y ALBA por su confianza y sus buenos consejos.

AGRADECIMIENTO

Por medio de estas palabras deseo agradecer a Dios y a la Virgen Auxiliadora por estar siempre conmigo y haberme dado nuevas oportunidades de seguir adelante en mi formación científica.

A los miembros del tribunal Master Francisco Vásquez, Master Edgar Pauta y Master Freddy Pesántez por su guía y apoyo en el desarrollo de la tesis.

Deseo agradecer al Doctor Claudio Urrea y al Doctor John Kern Catedráticos de la Universidad de Santiago de Chile por su apoyo incondicional en este trabajo como parte de mi formación doctoral en Ciencias de la Ingeniería.

Un sincero agradecimiento a los directivos de la Universidad del Azuay por el apoyo y la facilidad que me brindaron para desarrollar esta maestría.

CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El objetivo de este trabajo es analizar, modelar y simular algoritmos para las diferentes técnicas en el ámbito de las comunicaciones industriales de tal manera de conseguir que los sistemas de comunicación continúen funcionando correctamente a pesar de diferentes fallos. A estos tipos de sistemas se denominan sistemas tolerantes a fallas

Primero se realizará una descripción de las diferentes redes de comunicación industrial y las fallas en los procesos de comunicaciones industriales respectivamente, el mismo que se utiliza para introducirnos en uno de los protocolos fundamentales y de mayor utilidad en el sector industrial.

En el tercer capítulo se basa principalmente en explicar la especificación ModBus de forma general, sin entrar en mucho detalle en algunas de sus particularidades, no obstante su contenido es de gran utilidad para comprender el funcionamiento general de este estándar.

En el cuarto capítulo como una aplicación del protocolo ModBus se describe diferentes comunicaciones entre un PC que hace las funciones de maestro y una unidad de terminal remota que hace las funciones de esclavo, los enlaces que se describe en este documento tienen el objetivo de verificar las códigos de envío y respuesta de cada una de las tramas como los códigos de errores, para lo cual se utilizó las funciones principales que tienen ModBus. Las RTU que se utilizaron para las aplicaciones son las siguientes:

- Un PLC Modicon *micro* 61200.
- Un PLC Schneider con CPU 113 02.
- Un Medidor de Energía PM 710.

Además en este capítulo se visualizan diferentes formas de onda de envío y respuesta que proporciona las funciones principales con que cuenta la comunicación ModBus, con el objetivo de analizar los tiempos que utiliza cada una de las tramas.

Para analizar, modelar y simular algoritmos para las diferentes técnicas en el ámbito de las comunicaciones industriales es importante abordar el tema relacionado con

las técnicas de detección de error para luego tratar de corregirlo por medio de dos métodos: el primero trata de corregirlo en el sitio y el segundo solicita la retransmisión del bloque.

En el capítulo seis se estudia las técnicas (*Backwards Error Correction* BEC) o Corrección de Errores hacia Atrás (también conocido como petición automática de repetición, ARQ) que se basa en la utilización de la retroalimentación desde el receptor al transmisor, en este capítulo se presentan las configuraciones que ofrece la técnica ARQ como son:

- Stop and Wait ARQ (Pare y Espere).
- Go-Back-N.
- Selective Repeat ARQ

En el capítulo siete se aborda la "Corrección Directa de Error (*Forward Error Correction,* FEC)" y la "Solicitud de Repetición Automática (*Automatic Repeat Request*, ARQ)". En la técnica FEC se estudia diferentes códigos para detectar y corregir los errores en el receptor como lo son: los códigos *Hamming*, BCH, campo de *Galois*, Reed – Solomon y los códigos Convolucionales; mientras que en la técnica ARQ los códigos solamente detectan la presencia de errores en los datos recibidos y se solicita en alguna forma la repetición de los bloques que vienen con error; dadas las características de este sistema.

Para todas las técnicas descritas en el capítulo seis y siete he realizado su respectiva simulación con la ayuda del *Simulink* de *MatLab* que tiene las herramientas necesarias para analizar cada uno de los códigos como también para detectar y corregir los errores, obteniendo en cada una de las simulaciones la respectiva Tasa de Error de Bit (BER).

Como conclusión de este trabajo he llegado a obtener algunos comportamientos del código BCH para diferentes valores de trama como también he realizado una comparación entre tres técnicas de corrección de errores: BCH, *Hamming* y Convolucional.

Palabras Claves: ModBus, Maestro, Esclavo, Códigos: BEC, FEC, Técnicas: BCH, Hamming, Convolucional

ABSTRACT

The aim of this paper is to analyze, model, and simulate algorithms for the different techniques in the field of industrial communications so as to get the communication systems running well despite various faults

First a description of the different industrial communication networks and the faults that occur was done. Next we described, developed and displayed some ModBus Master/Slave applications protocol, which is the most used in the industrial area.

By means of MatLab the different techniques in the field of industrial communication were analyzed, model, and simulated. These techniques are Backwards Error Correction (BEC) and Forward Error Correction (FEC).

As a conclusion of this work some BCH code behaviors for different frame values were obtained. Also a comparison between three error correction techniques: BACH, Hamming and Convolutional were performed.

UNIVERSIDAD DEL AZUAY
OPTO. IDIOMAS

Translated by,
Lic. Lourdes Crespo

CAPÍTULO 1

REDES DE COMUNICACIÓN INDUSTRIAL

1.1 INTRODUCCIÓN

La proliferación de los usos de dispositivos inteligentes en el control de procesos industriales ha provocado resultados espectaculares en los índices de producción de la compañía, dando lugar a la fabricación de productos cada vez más homogéneos, de mayor calidad y en mayor cantidad. Las características y prestaciones de estos equipos han permitido definir lo que hoy en día se conoce como Fabricación Integrada por Computador (CIM) y Proceso integrado por Computador (CIP), donde las fábricas están formadas por amplios conjuntos de máquinas y dispositivos de control comunicados entre sí y funcionando de forma automatizada. La introducción de tecnologías de la comunicación permitió pasar de los clásicos sistemas centralizados, típicos de la década de los setenta y normalmente basados en equipos de alta prestaciones y elevado costo, a los actuales sistemas distribuidos de control basados en equipos más sencillos conectados a través de redes de comunicación. Por todo ello, las comunicaciones constituyen un elemento fundamental en los nuevos entornos de fabricación, constituidos en general por una cantidad de dispositivos de control inteligente (sensores, actuadores, transmisores, controladores, PLC's, etc.) que deben trabajar de forma coordinada a través de una red de comunicaciones [1].

1.2 ESTRUCTURA JERÁRQUICA DE LAS COMUNICACIONES INDUSTRIALES

La figura 1 muestra una estructura jerárquica para una Red LAN que se puede dar en cualquier industria con el único objetivo de mejorar su sistema de comunicación.

Figura 1: Estructura jerárquica de una red de comunicación industrial [2]

Nivel de Campo: en este nivel se realiza el control directo de las máquinas y sistemas de producción. Los dispositivos conectados son sensores, actuadores, instrumentos de medida, máquinas de control numérico, etc. Se suele utilizar cableado tradicional o buses de campo: AS-i.

Nivel de Célula: se realiza el control individual de cada recurso. Los dispositivos conectados son autómatas de gama baja y media, sistemas de control numérico, transporte automatizado. Se utilizan las medidas proporcionadas por el nivel 1 y se dan las consignas a los actuadores y máquinas de dicho nivel. Se usan buses de campo del tipo: AS-i, Device Net, ProfiBus DP *o* InterBus. Los niveles de proceso y campo utilizan paquetes de información del orden de los *bits* o *bytes*.

Nivel de Proceso: incluye los sistemas que controlan la secuencia de fabricación y/o producción (dan las consignas al nivel de campo). Se emplean autómatas de gama media y alta, ordenadores industriales, etc. Se usan buses de campo y redes LAN* **(***Local Area Network)* del tipo*:* ProfiBus FMS, ProfiBus PA, Ethernet, CAN.

Nivel de Planta: corresponde al órgano de diseño y gestión en el que se estudian las órdenes de fabricación y/o producción que seguirán los niveles inferiores y su supervisión. Suele coincidir con los recursos destinados a la producción de uno o varios productos

similares (secciones). Se emplean autómatas, estaciones de trabajo, servidores de bases de datos y *backup*. Se usan redes LAN del tipo Ethernet TCP/IP.

Los niveles de proceso y planta utilizan paquetes de información del orden de los *Kbytes*.

Nivel de Corporación: Gestiona la producción completa de la empresa, comunica las distintas plantas, mantiene las relaciones con los proveedores y clientes y proporciona las consignas básicas para el diseño y la producción de la empresa. Se emplean ordenadores, estaciones de trabajo y servidores de distinta índole. Se usan redes del tipo LAN o WAN usando Ethernet TCP/IP, ModBus plus. La transferencia de datos que realiza son programas completos.

El flujo de información existente en la pirámide debe ser:

- Vertical: incluye las órdenes enviadas por el nivel superior al inferior (descendente) y los informes sobre la ejecución de las órdenes recibidas (ascendente).
- Horizontal: debe existir un intercambio de información entre entidades de un mismo nivel.

1.3 BUSES DE CAMPO

IMS *Research* estima que los protocolos de buses de campo representaron en 2011 un 75% de nuevas conexiones de red de componentes de automatización industrial, considerando que esta cifra bajará al 69% en 2016. Hoy por hoy, las nuevas conexiones de red de buses de campo puestas a punto van aún muy por delante de Ethernet, pero el crecimiento de conexiones Ethernet deberá ser considerablemente más elevado en el 2016.

La mayoría de suministradores de componentes de automatización ofrecen actualmente en versión estándar una interface Ethernet en sus equipos y son numerosos los que ponen el acento en las prestaciones de la tecnología Ethernet. Los constructores de máquinas y los utilizadores finales van, pues, progresivamente pasando a las tecnologías Ethernet.

El estudio de IMS señala que estas conexiones basadas en buses de campo continúan aumentando debido al largo ciclo de vida de los productos y también al conservadurismo industrial; sin embargo, acabarán siendo relegadas a un papel de soporte. Su menor utilización repercutirá en su costo, lo que acelerará la transición a Ethernet [3].

1.4 VENTAJA DE LOS BUSE DE CAMPO

En las redes de campo se pueden destacar las siguientes características principales:

- Sustitución de la señal de 4-20mA por señales digitales.
- Aplicación a sistemas de control distribuido.
- Utilización de diferentes topologías.
- Interoperabilidad de dispositivos.
- Sistemas abiertos.
- Ahorro.
- Fiabilidad.
- Flexibilidad.
- Comunicación bidireccional.
- Capas Físicas.

1.5 CLASIFICACIÓN DE LOS BUSES DE CAMPO

Debido a la falta de estándares, diferentes compañías han desarrollado diferentes soluciones, cada una de ellas con diferentes prestaciones y campos de aplicación. En una primera clasificación tenemos los siguientes grupos [4]:

1.5.1 Buses de Alta Velocidad y Baja Funcionalidad

Están diseñados para integrar dispositivos simples como finales de carrera, fotocélulas, relés y actuadores simples, funcionando en aplicaciones de tiempo real, y agrupados en una pequeña zona de la planta, típicamente una máquina. Básicamente comprenden las capas física y de enlace del modelo OSI, es decir, señales físicas y patrones de *bits* de las tramas. Algunos ejemplos son:

- CAN: Diseñado originalmente para su aplicación en vehículos.
- SDS: Bus para la integración de sensores y actuadores, basado en CAN.
- ASI: Bus serie diseñado por Siemens para la integración de sensores y actuadores.

1.5.2 Buses de Alta Velocidad y Funcionalidad Media

Se basan en el diseño de una capa de enlace para el envío eficiente de bloques de datos de tamaño medio. Estos mensajes permiten que el dispositivo tenga mayor funcionalidad de modo que permite incluir aspectos como la configuración, calibración o programación del dispositivo.

- DeviceNet: Desarrollado por Allen-Bradley, utiliza como base el bus CAN, e incorpora una capa de aplicación orientada a objetos.
- LONWorks: Red desarrollada por Echelon.
- BitBus: Red desarrollada por INTEL.
- DIN MessBus: Estándar alemán de bus de instrumentación, basado en comunicación RS-232.
- InterBus-S: Bus de campo alemán de uso común en aplicaciones medias.

1.5.3 Buses de Altas Prestaciones

Son capaces de soportar comunicaciones a nivel de todos los niveles de la producción Aunque se basan en buses de alta velocidad, algunos presentan problemas debido a la sobrecarga necesaria para alcanzar las características funcionales y de seguridad que se les exigen. La capa de aplicación tiene un gran número de servicios a la capa de usuario.

- ProfiBus.
- WorldFIP.
- FieldBus Foundation.

1.5.4 Buses para Áreas de Seguridad Intrínseca

Incluyen modificaciones en la capa física para cumplir con los requisitos específicos de seguridad intrínseca en ambientes con atmósferas explosivas. (Dentro del cual figuran las condiciones de operación normal y de fallo específicas). Algunos ejemplos son:

- HART.
- ProfiBus PAo WorldFIP

1.6 DESCRIPCIÓN DE LOS BUSES DE CAMPO

1.6.1 ProfiBus: Es de estándar Alemán y está normalizado por DIN 19245 y en Europa por EN 50170, este bus de campo es una de las tecnologías más populares y es el único bus a campo estándar GB/T20540-2006 aprobado por China, sus productos son ampliamente utilizados en la industria, la electricidad, la energía, transporte y otros campos automatizados. De acuerdo a las características de sus aplicaciones, ProfiBus se divide en tres tipos: ProfiBus-DP, ProfiBus FMS y ProfiBus PA. ProfiBus-DP es adecuado para comunicación de alta velocidad entre los periféricos descentralizados, lo cual es cada vez más popular entre los dispositivos de automatización e pequeñas industrias de instrumentación [5].

ProfiBus es un bus de campo estándar que acoge un amplio rango de aplicaciones en fabricación, procesado y automatización. Con ProfiBus los componentes de distintos fabricantes pueden comunicarse sin necesidad de ajustes especiales de interfaces. ProfiBus puede ser usado para transmisión crítica en el tiempo de datos a alta velocidad y para tareas de comunicación extensas y complejas, ver figura 2.

Puede decirse sin lugar a dudas que ProfiBus ha conseguido definir toda una red de comunicación industrial, desde el nivel físico hasta el de aplicación, integrando al máximo las técnicas de comunicación

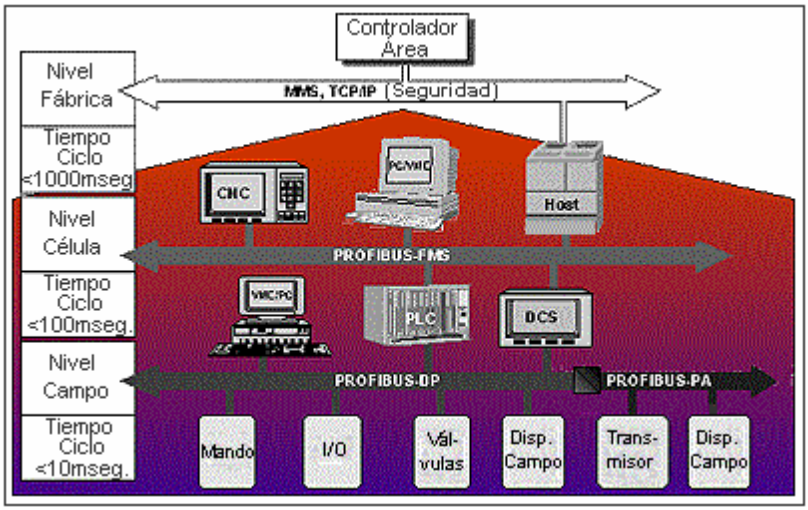


Figura 2: Áreas de aplicación [6]

1.6.1.1 Familia de ProfiBus

La familia de ProfiBus viene dada por las tres versiones compatibles que componen la familia ProfiBus, ver figura 3.

ProfiBus PA:

- Diseñado para automatización de procesos.
- Permite la conexión de sensores y actuadores a una línea de bus común incluso en áreas especialmente protegidas.
- Permite la comunicación de datos y energía en el bus mediante el uso de 2 tecnologías (norma IEC 1158-2).

ProfiBus DP:

- Optimizado para alta velocidad.
- Conexiones sencillas y baratas.
- Diseñada especialmente para la comunicación entre los sistemas de control de automatismos y las entradas/salidas distribuidas.

ProfiBus FMS:

- Solución general para tareas de comunicación a nivel de célula.
- Gran rango de aplicaciones y flexibilidad.
- Posibilidad de uso en tareas de comunicaciones complejas y extensas.

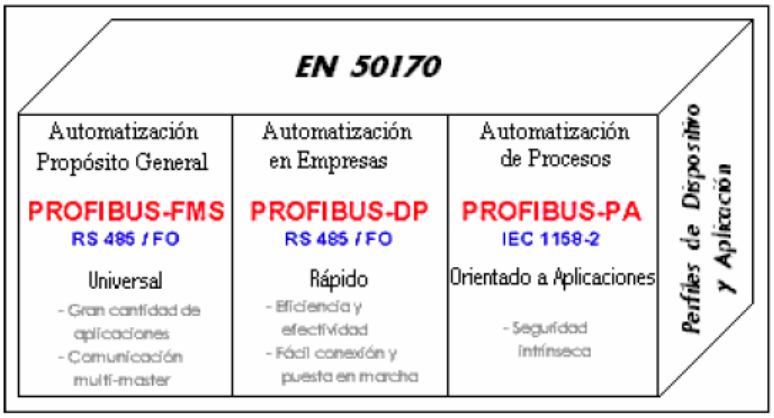


Figura 3: Familia de ProfiBus [6]

Las distancias potenciales de la red ProfiBus van de 100 m a 24 Km (con repetidores y fibra óptica). La velocidad de comunicación puede ir de 9600 bps a 12 Mbps. Utiliza mensajes de hasta 244 *bytes* de datos [5].

ProfiBus se ha difundido ampliamente en Europa y también tiene un mercado importante en América y Asia. El conjunto ProfiBus DP- ProfiBus PA cubre la automatización de plantas de proceso discontinuo y proceso continuo cubriendo normas de seguridad intrínseca [5].

1.6.2 InterBus

El bus de campo InterBus fue desarrollado por Phoenix Contact (Alemania) en 1984. Desde entonces, se convierte en norma DIN19258 Alemania, la norma europea EN50254 y la norma internacional IEC61158. En mayo de 2005, fue aceptado por China como estándar industrial JB/TIO308.8. La figura 4 muestra la estructura de una red InterBus. El sistema InterBus durante la instalación se comporta como una línea o estructura de árbol, y trabaja como un sistema de anillo. Para transferir datos correctamente desde el maestro InterBus a los dispositivos esclavos [7].

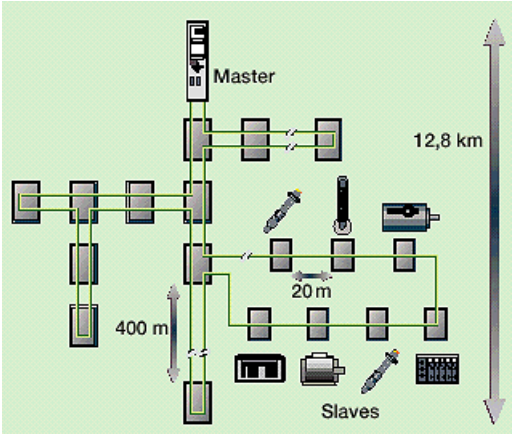


Figura 4: Estructura de una red InterBus [8]

Con sus características especiales y una amplia gama de productos, InterBus se ha estabilizado exitosamente en el sector de la industria. Su campo tradicional de aplicación es la industria de automóviles, pero InterBus está incrementando su uso como una solución en otras áreas tales como manejo y transportación de materiales, imprentas e industria del papel, industria de alimentos y bebidas, automatización de edificios, industria del procesamiento de madera, ensamblaje y aplicaciones robóticas, ingeniería mecánica en general y, más recientemente, en ingeniería de procesos. Además de las aplicaciones estándar para la conexión de grandes cantidades de sensores/actuadores en el campo al más alto nivel del sistema de control a través de un sistema de bus serial, InterBus puede también ser usado para cumplir una variedad de requerimientos de aplicaciones especiales.

Capa física basada en RS-485. Cada dispositivo actúa como repetidor. Así se puede alcanzar una distancia entre nodos de 400 m para 500Kbps y una distancia total de 12 KM. Es posible utilizar también enlaces de fibra óptica.

1.6.3 DeviceNet

DeviceNet aborda IEC 61508 SIL3, AK6 según DIN 19250 y EN 954-1 categoría 4.

DeviceNet es un bus de campo de bajo costo basado en CAN (*Controller Area Network*). Esta red sirve para conectar dispositivos industriales, eliminando así el cableado el costo y tiempo. CAN es una tecnología que se utilizan en muchas industrias, lo que minimiza el costo de los componentes.

Los dispositivos conectados directamente en DeviceNet ofrecer una mejor comunicación, así como los diagnósticos que no están disponibles con cableado de I / O [9].

DeviceNet fue desarrollado por Allen-Bradley a mediados de los noventa, mediante DeviceNet es posible la conexión de hasta 64 nodos con velocidades de 125 Kbps a 500 Kbps en distancias de 100 a 500 m [4] [9].

Este protocolo es de Red de bajo nivel adecuada para conectar dispositivos simples como sensores fotoeléctricos, sensores magnéticos, pulsadores, etc. y dispositivos de alto nivel (PLC, controladores, computadores, HMI, entre otros), ver figura 5. Provee información adicional sobre el estado de la red, cuyos datos serán desplegados en la interfaz del usuario.

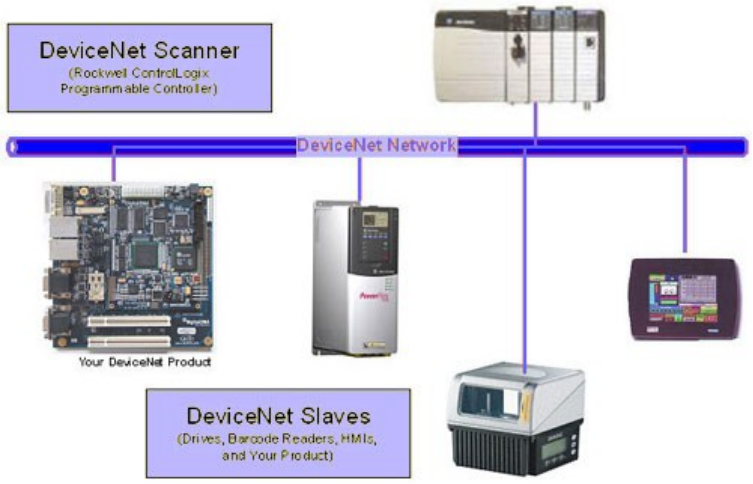


Figura 5: Red DeviceNet [10]

DeviceNet permite que los dispositivos industriales puedan ser fácilmente interconectados en una red y ser manejados remotamente.

Por tratarse de una red estándar, DeviceNet es ideal para aplicaciones que se benefician de una estrecha integración entre dispositivos [11].

1.6.4 Foundation FieldBus

Este bus de campo se caracteriza por la interoperabilidad e intercambiabilidad de los sistemas de bus de campo. Actualmente estandarizado internacionalmente por IEC 61804-2 para la conocida tecnología de bus de campo ProfiBus, Hart y Foundation FieldBus (FF) [12].

Foundation FieldBus Es una red de tipo industrial que permite realizar la comunicación entre un Sistema de Control y los dispositivos de campo. Soporta tanto señales análogas como discretas. Como otros buses, remplaza a nuestro viejo estándar de señales análogas de 4-20 mA, llevando por un sólo cable varios, ver figura 6 [13].

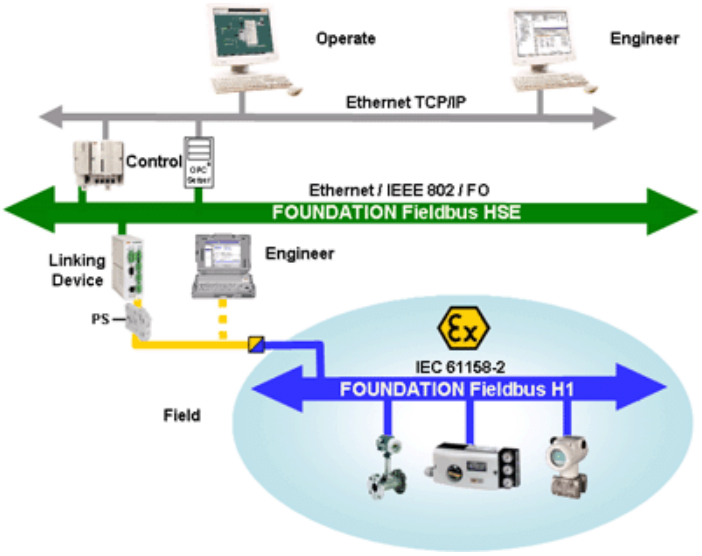


Figura 6: Red Foundation FieldBus [13]

En términos de características físicas, la implementación de Foundation FieldBus tiene dos estándares: H1 y HSE. El estándar H1 tiene una velocidad de 31,25 Kbps y el HSE una velocidad de 100Mbps:

Los aspectos anteriores pueden compararse a otros buses de campo. Pero lo que hace diferente a Foundation FieldBus es lo siguiente.

- Desde el inicio fue conceptualizado no sólo como bus de comunicaciones, sino como un sistema que permitiera interoperabilidad entre equipos de diferentes marcas.

- Cada dispositivo que cumple el estándar, es capaz de ejecutar un número de funciones, y pueden comunicarse entre sí, traspasándose información de control. Ello permite en la práctica, que los lazos de control puedan ser ejecutados sin necesidad de un sistema de control central.

1.6.5 FIP- WorldFIP

Es un Protocolo de Instrumentación de Fábrica (FIP) fue originalmente creado en Francia y por usuarios italianos al comienzo de la década de 1980. De hecho, fue la reacción a las necesidades del mercado nacional en los países que llevan el desarrollo del protocolo FIP. Desde el principio, FIP se dedicó a trabajar en tiempo real y satisfacer requisitos de control y supervisar tareas de instrumentación.

Este bus de campo fue adaptado por muchas empresas como las generadoras de generación de energía eléctrica, vialidad, y el tren de alta velocidad (TGV), junto con otras aplicaciones.

El WorldFIP es la estandarización a la antigua bus de campo FIP (Factory Protocolo Interoperable).

En marzo de 1993, el WorldFIP (Protocolo de Fabricación de Instrumental) fue conjuntamente creado por Honeywell, AB (Allen-Bradley), CECELEC, Telemechanique y varias otras empresas.

La figura 7 nos indica un red WorldFIP –Ethernet donde un PC es la puerta entre los dos buses de campo WorldFIP y la red Ethernet [14].

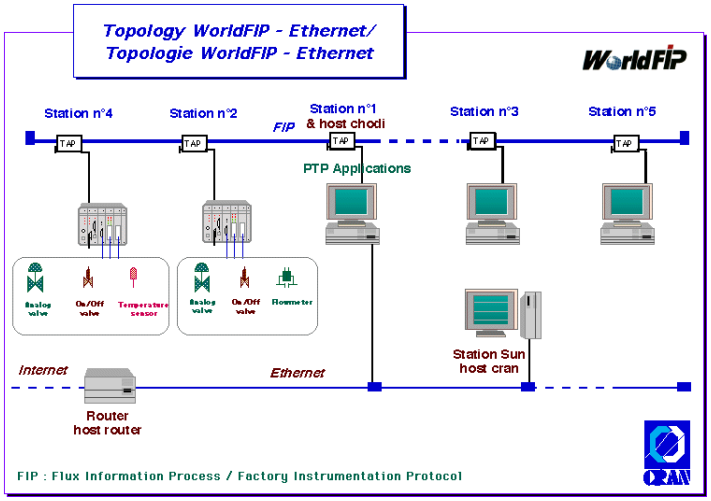


Figura 7: Red WorldFIP – Ethernet [14]

1.6.6 LonWorks

LonWorks es un tipo de tecnología de bus de campo que fue puesta en marcha por la empresa Echelon de los Estados Unidos en los años 90. Se trata de una plataforma completa y una técnica utilizada para desarrollar el sistema de redes de vigilancia, y tiene todas las características de la tecnología de bus de campo. El sistema de red LonWorks está compuesta por nodo inteligente, cada nodo inteligente tiene muchos tipos de funciones I / O, los nodos se pueden comunicar a través de diferentes medios de transmisión como se indica en la figura 8, que también se rige por la norma ISO / OSI de siete capas de acuerdo a cada capa [15] [16].

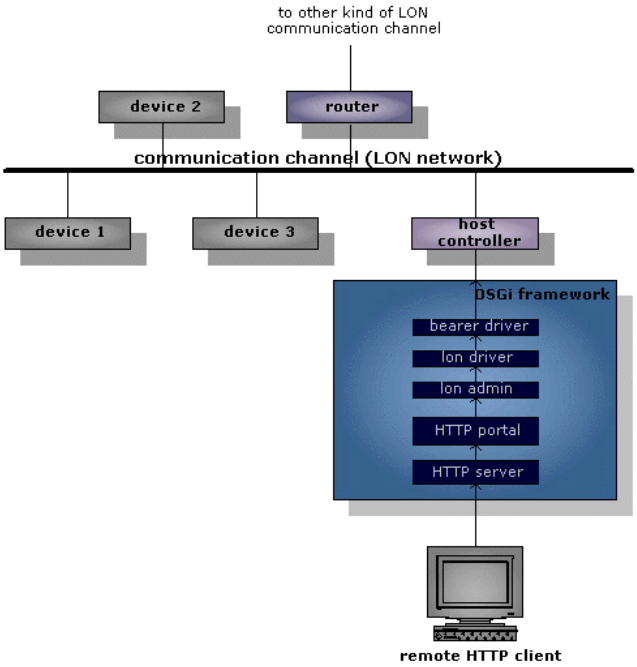


Figura 8: Red WorldFIP – Ethernet [16]

La red Lonworks ofrece una variada selección de medios físicos y topologías de red: par trenzado en bus, anillo y topología libre, fibra óptica, radio, transmisión sobre red eléctrica etc. El soporte más usual es par trenzado a 38 o 78 Kbps. Se ofrece una amplia gama de servicios de red que permiten la construcción de extensas arquitecturas con multitud de nodos, dominios y grupos, típicas de grandes edificios inteligentes [4].

1.6.7 Smart Distributed System (SDS)

El Smart Distributed System o Sistema Inteligente Distribuido (SDS) proporciona un medio para conectar equipos de automatización y dispositivos en una sola red, lo cual elimina el elevado costo del cableado. Este medio de comunicación estándar proporciona un método de bajo costo para los controladores y dispositivos para comunicar datos de bajo nivel a altas velocidades, ver figura 9. El estándar SDS proporciona especificaciones para el intercambio de información entre los nodos, así como a nivel de dispositivos de diagnóstico que normalmente no se encuentran en otros sistemas I/O [17].

El Sistema Inteligente Distribuido ofrece:

- Rentabilidad: SDS ofrece control económico.
- Fácil conectividad: SDS es de bajo costo, es fácil de implementar y mantener el sistema de cableado.
- Diagnóstico: SDS ofrece diagnósticos avanzados de errores que no se encuentran comúnmente en los sistemas tradicionales.
- Altas velocidades de transferencia: Velocidad de transmisión que llevan tiempo de respuesta de hasta 0,10 ms por dispositivo.

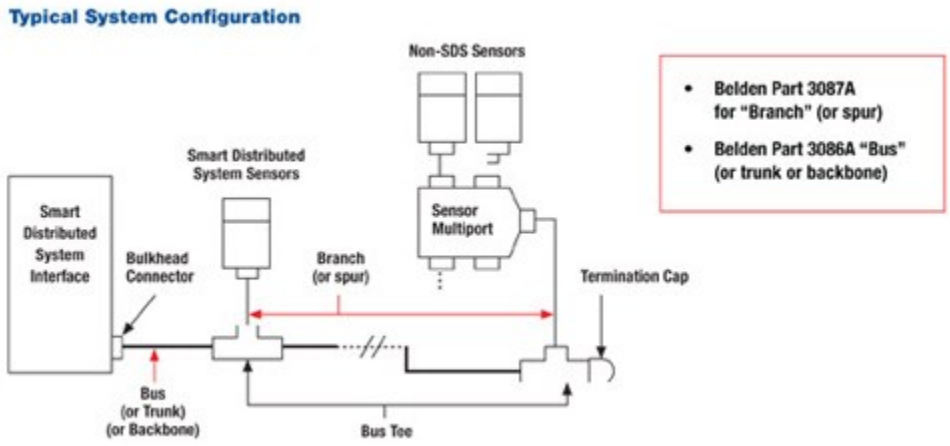


Figura 9: Smart Distributed System (SDS) [18]

1.6.8 CANOpen

CANOpen es un protocolo realmente abierto que no ha sido desarrollada por una sola empresa, varios grupos de trabajo formados por diversos fabricantes de dispositivos y usuarios finales, han colaborado para producir el CANOpen estándares, ahora bajo la supervisión del protocolo.

CANOpen es un protocolo de estructura de siete capas que se define en el modelo OSI. Este bus de campo proporciona mecanismos que hacen posible trabajar con diferentes dispositivos de diferente tipo de protocolo de comunicación [19].

1.6.9 ModBus

ModBus es un protocolo de comunicación industrial que apareció en 1979 como un bus para transmitir y recibir datos de control entre controladores e instrumentos de campo en forma serial, mediante una topología de maestro/esclavo.

Las especificaciones del protocolo ModBus se encuentran disponibles al público y está reconocido por la lEC como una especificación públicamente disponible (*Public Available Specification*) bajo la designación lEC PAS6203. Actualmente es soportado por la organización independiente ModBus-IDA.

La designación ModBus no corresponde propiamente a un estándar de red que incluye todos los aspectos desde el nivel físico hasta el de aplicación, sino a un protocolo de mensajes, posicionado en la capa de aplicación o nivel 7 del modelo OSI (*Open System Interconnectio*), tal como se muestra en la figura 10.

ModBus es un protocolo de comunicación maestro/esclavo entre dispositivos conectados sobre una red tipo bus, con transmisión serial asíncrona sobre uno de los siguientes medios: par trenzado, fibra óptica y radio [20].

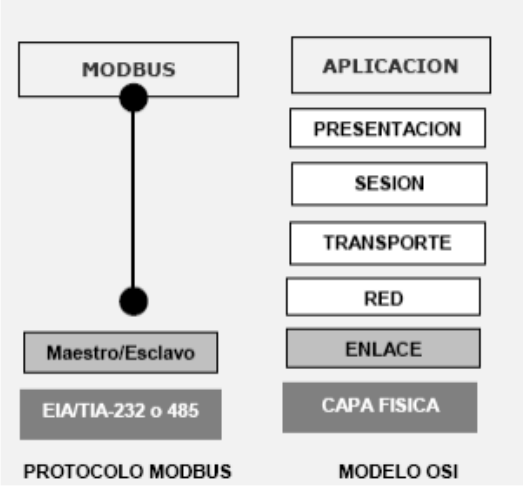


Figura 10. Capas del Protocolo ModBus [20]

El ModBus es un protocolo de transmisión para sistemas de control y supervisión de procesos (SCADA) con control centralizado, puede comunicarse con una o varias Estaciones Remotas (RTU) con la finalidad de obtener datos de campo para la supervisión y control de un proceso. La Interfaces de Capa Física puede estar configurada en: RS-232, RS-422, RS-485.

En ModBus los datos pueden intercambiarse en dos modos de transmisión:

- Modo RTU (Unidad de Terminal Remota).
- Modo ASCII(Código Estándar Estadounidense para el Intercambio de Información).

1.7 LA GUERRA DE LOS BUSES

Ante la variedad de opciones existente, parece razonable pensar que fabricantes y usuarios hicieran un esfuerzo en la búsqueda de normativas comunes para la interconexión de sistemas industriales.

Lo que ha venido llamándose "la guerra de los buses" tiene que ver con la permanente confusión reinante en los entornos normalizadores en los que se debate la especificación del supuesto "bus de campo universal". Desde mediados de los años ‘80 la Comisión Electrotécnica Internacional (IEC-CEI) y la Sociedad de Instrumentación Americana (ISA) ha sido escenario del supuesto esfuerzo de los fabricantes para lograr el establecimiento de una norma única de bus de campo de uso general [4].

CAPÍTULO 2
FALLAS EN LOS PROCESOS DE SISTEMAS DE COMUNICACIÓN INDUSTRIALES

2.1 INTRODUCCIÓN

Dentro de las comunicaciones industriales continuamente se habla sobre las redes de computadores que deben ser capaces de transmitir datos de un dispositivo a otro con un alto grado de precisión. Para muchas aplicaciones industriales como en el campo de la robótica los sistemas de comunicación deben garantizar que los datos recibidos son iguales a los trasmitidos. Pero en el proceso de transmisión de los datos pueden existir diferentes interferencias debidos al calor, temperatura, o campos electromagnéticos producidos en el canal de transmisión (cable multipar, par trenzado, coaxial, etc.), que interfiere en la señal y sobre todo si los datos a transmitir son de carácter binarios codificados, tales cambios pueden alterar su significado, ver figura11.

Las comunicaciones en los robots requieren entonces un mecanismo que permita detectar y corregir los posibles errores ocurridos durante la transmisión debido a que necesita un alto nivel de precisión ya que continuamente se transmiten diferentes tipos de archivos.

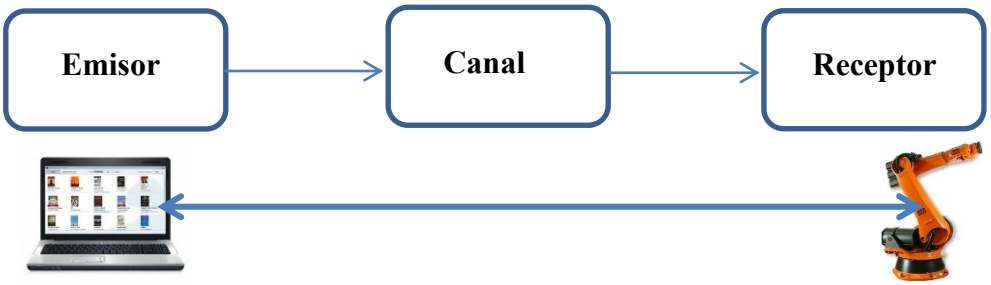


Figura11: Comunicación industrial

2.2 TRANSMISIÓN DE DATOS

Las características que distinguen a las posibles configuraciones del enlace de datos son el modo de transmisión, el modo de funcionamiento en *simplex*, *semi-duplex* o *full-duplex*, topología (tipos de conexión).

2.2.1 Modos de Transmisión

- **Punto a Punto.-** Son aquellas donde hay muchas conexiones entre parejas individuales de máquinas [21].
- **Multipunto**.- Son las redes donde lo datos llegan a todas las máquinas de la red, un sólo canal de comunicación [21].

2.2.2 Modos de Funcionamiento.- Los distintos tipos de transmisión de un canal de comunicaciones pueden ser de tres clases diferentes: *Simplex, Dúplex* o *Half-dúplex* o *Semi-dúplex* y *Full-Dúplex* o *dúplex* completo [21].

- **Simplex.-** En este caso el transmisor y el receptor están perfectamente definidos y la comunicación es unidireccional, ver figura 12.

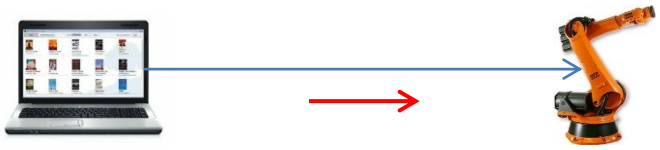


Figura 12: Transmisión simplex

- ***Dúplex* o Semi-*dúplex***.- En este caso ambos extremos del sistema de comunicación cumplen funciones de transmisor y receptor y los datos se desplazan en ambos sentidos pero no simultáneamente. En la transmisión semi-*dúplex* cada vez sólo una de las dos estaciones del enlace punto a punto puede transmitir.
- **Full Dúplex**: En la transmisión Full-Dúplex las dos estaciones pueden simultáneamente enviar y recibir datos. Este modo se denomina dos sentidos simultáneos figura 13.

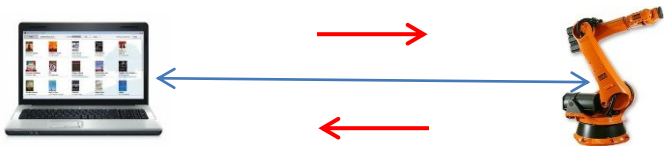


Figura 13: Transmisión full dúplex

2.2.3 Topologías

- **Bus.-** Todas las máquinas están conectadas a un único cable por donde pasa toda la información, esta llega a todas las máquinas. Tienen un *Hub* o un *switch* al final [21].

- **Anillo.-** Es un anillo cerrado donde cada nodo o PC está conectado con sus nodos adyacentes formando un anillo. La información se transmite de nodo en nodo.

- **Estrella.-** Es un nodo central que normalmente es un *hub* y a él están conectados todos los PC, la información pasa por el *hub* para luego ir a su destino.

- **Árbol.-** Tiene un nodo troncal que suele ser un *hub* desde el que se ramifican los demás nodos.

- **Malla.-** Todos los nodos se comunican directamente entre sí figura 14.

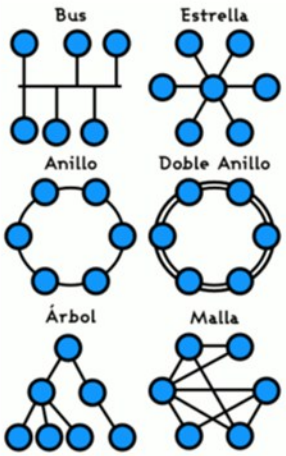


Figura 14: Topologías de red

2.3 PROTOCOLOS DE COMUNICACIÓN

Un protocolo de comunicación es un conjunto de reglas que permiten la transferencia e intercambio de datos entre los distintos dispositivos para que puedan comunicarse unos con otros e intercambiar información de manera eficiente y eficaz [22].

Teniendo en cuenta la diversidad de protocolos, estándares y tecnologías utilizadas en el mercado se clasifican en:

2.3.1 Protocolo o Sistema Abierto.- Es aquel que no está sujeto a pago de licencias de uso para su utilización, sus componentes están de acuerdo con normas que garantizan su compatibilidad, aunque procedan de diferentes fuentes de suministro. Permite sustituir cualquiera de los dispositivos por uno similar de otro fabricante, que siga cumpliendo la

funcionalidad y requisitos impuestos por el proyecto. Un sistema abierto no significa que sea un estándar reconocido por un organismo internacional [23].

2.3.2 Protocolo o Sistema Propietario.- Es un producto o sistema desarrollado por una empresa para sólo poder operar con sus propios dispositivos o con otros de terceros especificados anticipadamente. No es posible intercambiar dispositivos con diferentes tecnologías o de otros fabricantes. Los protocolos propietarios poseen ventaja frente a los estándar en cuanto a la economía y costo de los equipos pero resulta un riesgo emplear un solo tipo de tecnología, pues si la empresa desaparece entonces no se puede seguir obteniendo soporte técnico ni posibilidades para ampliaciones futuras y existe una dependencia a una marca en particular.

2.3.3 Protocolo Estándar.- Un protocolo estándar es aquel que ha sido reconocido por uno o varios organismos internacionales de normalización como IEEE, CENELEC, ETSIT etc. y que, por lo tanto, está siendo utilizada por muchas empresas en sus productos. La ventaja principal que proporcionan estos protocolos es la capacidad para implementar o configurar una instalación y su posible ampliación, debido a la compatibilidad en el estándar que pueden poseer diversos equipos de diferentes fabricantes, pero resultan ser más costosos que los equipos de tecnología propietaria.

2.4 BUS DE DATOS

Los buses de datos permiten la integración de diferentes equipos para la medición y control de variables de procesos. Un bus de campo es un sistema de transmisión de información (datos) que simplifica enormemente la instalación y operación de máquinas, robots y equipamiento utilizados en los procesos industriales.

El objetivo de un bus de campo es sustituir las conexiones punto a punto entre los elementos de campo y el equipo de control. Generalmente el bus de campo son redes digitales bidireccionales montadas sobre un bus serie que conectan diferentes dispositivos de campo como transductores, actuadores, sensores y equipos de supervisión.

2.5 TRAMA Y ELEMENTOS DE UNA TRAMA

Se entiende como trama informática a la secuencia de caracteres o bits (unidad de datos) que se transmiten por el nivel de enlace y tiene los siguientes elementos, ver figura15 [24]:

- **Mensaje.-** Información que se requiere transmitir que puede ser una secuencia de caracteres o bits que representan la información que se pretende enviar.
- **Bloque**.- Secuencia de caracteres o bits que se agrupan para su transmisión debido a razones técnicas.
- **Iniciación.-** Envío de tramas de control entre las estaciones enlazadas para averiguar la dsponibilidad para transmitir o recibir.
- **Identificación.-** Procesos para identificar la estación u origen de la información. Se usa en enlaces conmutados punto a punto.
- **Terminación.-** Comunicación de que todos los elementos se han recibido correctaente y desconexión para dejar libre los recursos ocupados.
- **Sincronización de Trama.-** Diferenciar la trama del conjunto de información transmitida. Se consigue añadiendo a la trama información de control que indique donde empieza y donde termina.

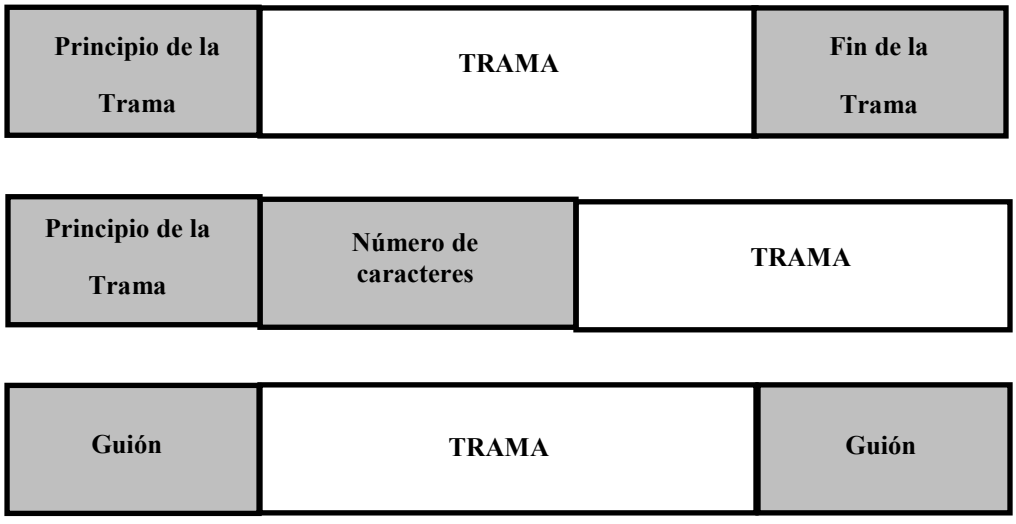


Figura 15: Conformación de una trama

2.6 TASA DE ERROR

Todo canal de transmisión de datos introduce errores en la información transmitida y su relación está dado entre el número de bits erróneos recibidos y el número de bits transmitidos:

$$BER = \frac{N_R}{N_E}$$

2.7 TIPOS DE ERRORES EN LA TRANSMISIÓN

Es importante primeramente entender los diferentes tipos de errores que pueden suceder en la transmisión de datos, para luego estudiar los mecanismos que permitan la detección y/o corrección de errores.

2.7.1 Error de Bit.- El error de bit es cuando cambia un bit de un byte, carácter o un paquete, es decir cambia de 0 a 1 o de 1 a 0, por ejemplo si se transmite un carácter en códigos ASCII con 8 bits agregando un 0 como el bit más significativo, puede ocurrir un error por influencias del canal y transformarse en otro carácter diferente al transmitido, ver figura 16.

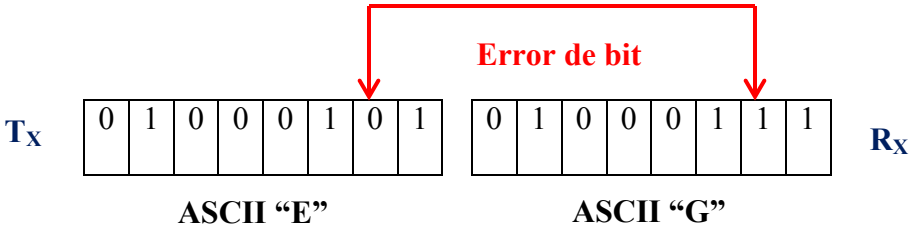


Figura 16: Error de bit

El error de bit más es más probable en la transmisión en paralelo que la transmisión en serie debido a que si el emisor envía datos a una velocidad de 1Mbps, quiere decir que cada bit se transmitirá cada 1µs, por tal motivo el ruido que puede introducirse para cambiar un bit debe ser menor, lo que resulta casi imposible.

2.7.2 Error de Ráfaga.- El error de ráfaga es cuando al receptor llega cambiados dos o más bits que pueden ser o no consecutivos. La longitud de la ráfaga se mide desde el primer bit afectado hasta el último bit afectado. Dentro de esta ráfaga pueden existir bits afectados

y otro que no son afectados, ver figura17. La presencia de errores en la ráfaga es más probable en las transmisiones serie debido a que la duración del ruido es normalmente mayor a la velocidad de transmisión de cada uno de los bits de datos, lo que implica que el ruido afecta a un conjunto de datos. Por tal motivo la cantidad de bits afectados depende del tiempo de ruido y de la tasa de datos.

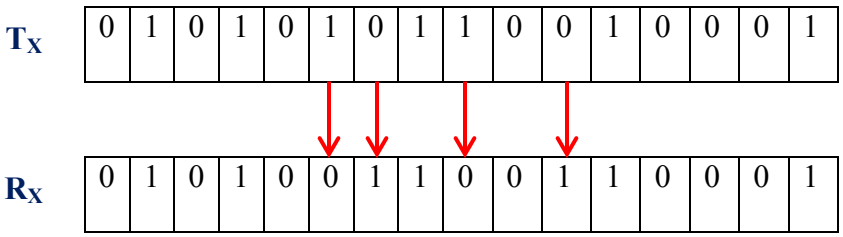


Figura 17: Error de ráfaga de longitud 6

2.8 FALLAS EN LOS PROCESOS INDUSTRIALES

Falla.- La falla es un error causado por un problema de diseño, construcción, programación, físico o lógico tanto en el *hardware* como en el *software* de un sistema, debidos a diferentes factores humanos, ambientales y propios de los diferentes componentes o elementos dentro de una determinada programación, por este motivo la falla de un componente del sistema no conduce directamente a la falla del sistema, pero puede ser el comienzo de una serie de fallas que quizás si terminen con la falla del sistema [25].

Error.- El error es considerado como el resultado de una falla; un error es la consecuencia de una causa mecánica o algorítmica desde el punto de vista de la entrega y recepción de la información [26].

Avería.- Una avería es una desviación del comportamiento de un sistema respecto de su especificación [26]. Las averías se manifiestan en el comportamiento externo del sistema pero son el resultado de errores internos, ver figura 18.

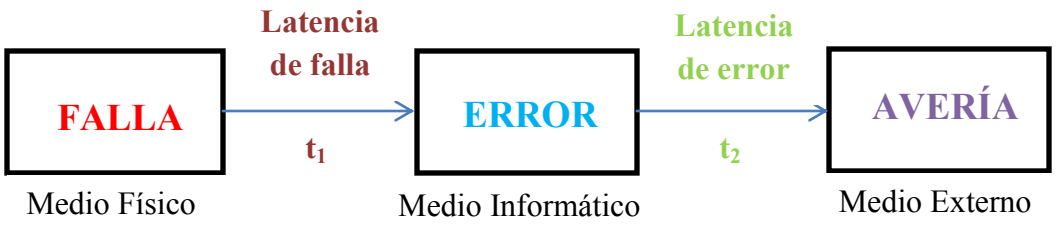


Figura 18: Relación entre falla, error y avería

Latencia de una Falla.- Es el tiempo t_1 que transcurre desde que ocurre el falla hasta que se manifiesta el error.

Latencia de un Error.- Es el tiempo t_2 que transcurre desde que ocurre la aparición del error y la manifestacion de ese error en el medio externo.

2.8.1 Causas de las Fallas

Para determinar las causas en las fallas se debe analizar todo el proceso desde el diseño, implementación y ejecución, en todo este proceso puede dar origen a diferentes fallas, por ejemplo si se desea construir un robot manipulador tipo industrial, debe tener en cuenta las posibles fallas que puede tener desde el momento de su diseño hasta cuando está funcionando ya que el mismo puede tener fallas tanto en el *hardware, software*, comunicación, ver figura 19.

Las causas de las fallas pueden clasificarse de la siguiente manera [27]:

Fallas de Especificaciones.- que son las fallas ocasionadas en el momento del diseño que pueden ser de *hardware* y *de software*.

Fallas en la Implementación.- que se en el proceso de transformar la parte del diseño del *hardware* y del *software* en un producto terminado.

Fallas de Componentes.- estas fallas pueden darse debido al mal dimensionamiento de los mismos que pueden ocasionar grandes averías en el medio externo.

Fallas por Perturbaciones Externas.- dentro de estas fallas se encuentran todas las variaciones de las condiciones ambientales, tales como presión, temperatura, humedad y

dentro de las comunicaciones se puede decir que una perturbación externa son los campos electromagnéticos, los ruidos, que son elementos que pueden modificar la transmisión de datos.

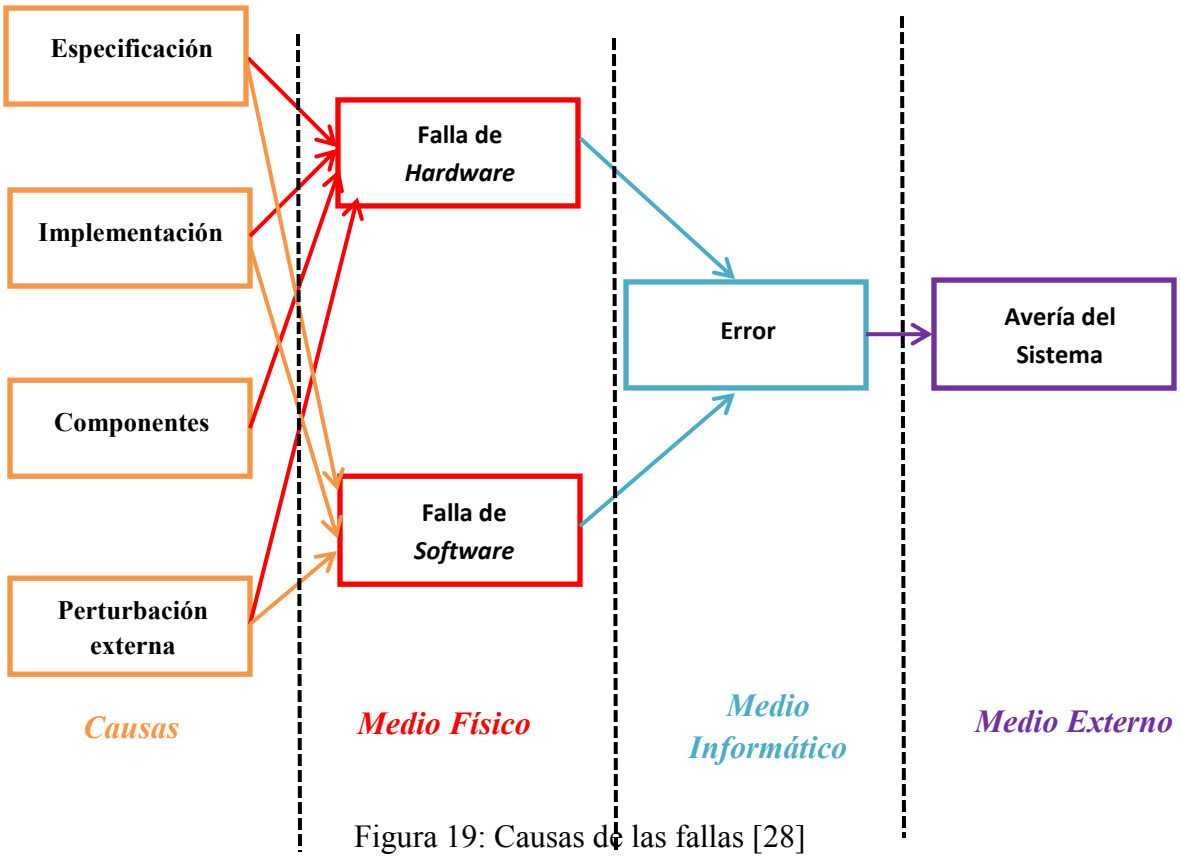


Figura 19: Causas de las fallas [28]

2.8.2 Clasificación de las Fallas

Las fallas se clasifican de acuerdo a los siguientes parámetros ver figura 20 [27]:

- Causa
- Naturaleza
- Duración
- Extensión
- Viabilidad

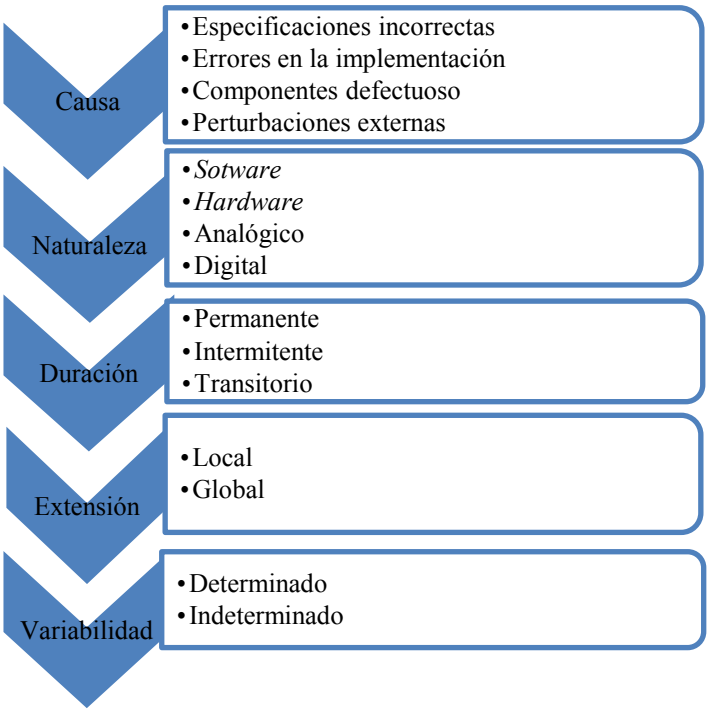


Figura 20: Clasificación de las fallas

2.8.3 Técnicas para Combatir las Fallas

En la figura 21 se observa tres técnicas básicas para mantener un correcto funcionamiento de un sistema digital ante posibles fallas del mismo.

Prevención de Fallas.- Esta técnica es capaz de evitar las fallas antes de que se produzcan. Esta técnica incluye la revisión de los diseños, análisis de los componentes, y sobre todo el análisis de ruido de factores externos.

Enmascaramiento de las Fallas.- Dentro de esta categoría se encuentra las técnicas que encubren las consecuencias de una falla. Como ejemplo de esta técnica podemos poner la utilización de bits redundantes en una memoria con el fin de corregir posibles errores en la memoria de esta forma corregimos el dato erróneo antes de que el sistema lo use.

Tolerancia a Fallas.- Es la capacidad de un sistema para continuar funcionando normalmente después de producir una falla [25]. En general cuando ocurre una falla es necesaria la reconfiguración del sistema, es decir, la puesta en funcionamiento de elementos redundantes para sustituir al componente causante de la falla.

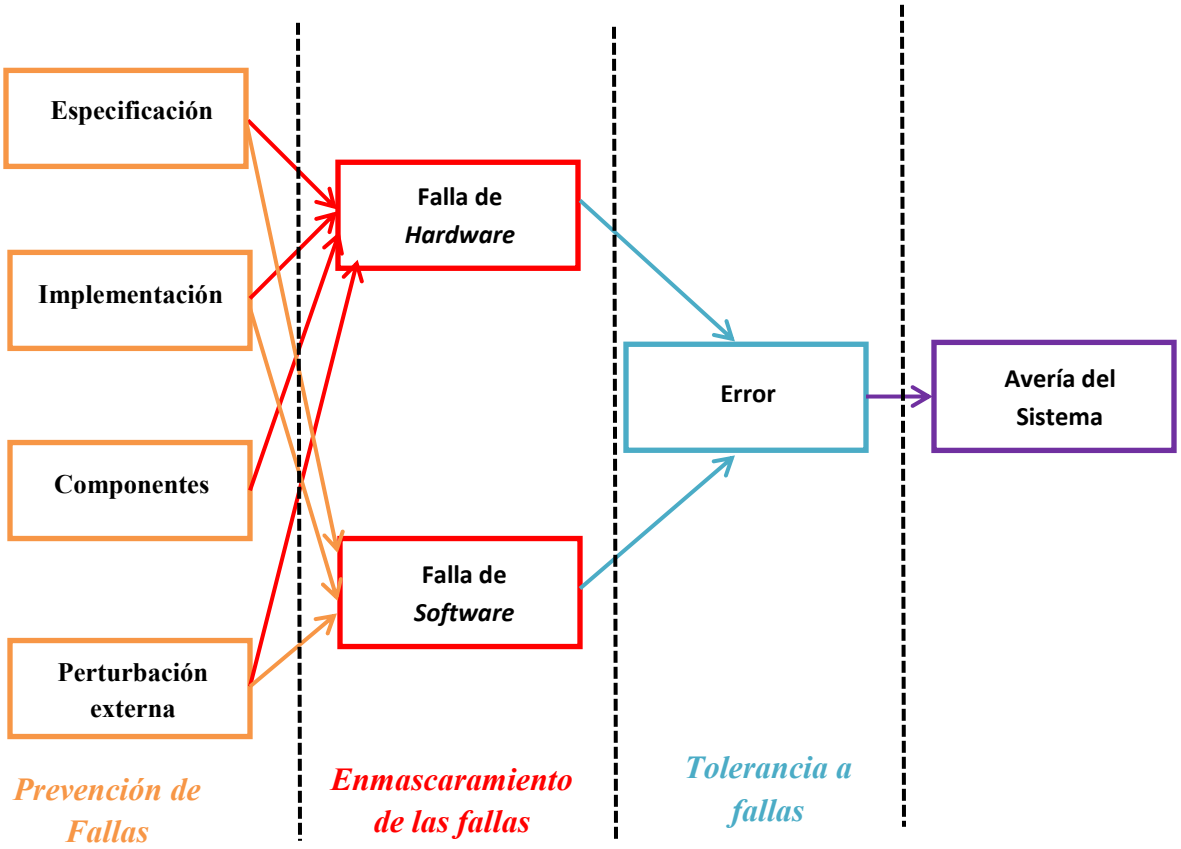


Figura 21: Técnicas para combatir las fallas [28]

2.8.4 Redundancia

La tolerancia de fallas se basa en la redundancia donde se utilizan componentes adicionales para detectar las fallas y recuperar el comportamiento correcto, esto aumenta la complejidad del sistema y puede introducir fallas adicionales, por tal motivo es mejor separar los componentes tolerantes del resto del sistema. La redundancia podemos dividir en cuatro grandes grupos:

Redundancia Física o de *Hardware*.- Se agrega equipo adicional para permitir que el sistema tolere la pérdida o mal funcionamiento de algunos componentes, por este motivo es el encargado de adicionar *hardware* al sistema, con el objetivo de detectar fallas y conseguir la tolerancia del sistema [25].

Redundancia de S*oftware*.- Esta redundancia es la encargada de adicionar códigos redundate en los programas para evitar errores [25].

Redundancia Informacional.- Esta redundanca es la encargada de manejar la información suplementaria con el objetivo de detectar o corregir errores potenciales por ejemplo se puede agregar código de *Hamming* para transmitir los datos y recuperarse del ruido en la línea.

Redundancia Temporal.- Esta redundancia es la encargada de realizar una acción, y de ser necesario, se vuelve a realizar. Es de particular utilidad cuando las fallas son transitorias o intermitentes, por este motivo es necesario emplear un tiempo adicional de proceso para detectar posibles fallas o en su caso, corregirlos.

Esto da lugar a dos formas de organizar los equipos redundantes: la activa y el respaldo primario. Para el primer caso, todos los equipos funcionan en paralelo para ocultar la falla de alguno(s) de ellos. Por su parte, el otro esquema utiliza el equipo redundante de respaldo, sólo cuando el equipo principal falla.

Las redundancias de *hardware* y *software* pueden trabajar juntos, por ejemplo la redundancia de *software* implica redundancia temporal, salvo si se emplea un procesador suplementario para ejecutar las instrucciones adicionales, en este caso existirá una redundancia de *hardware*, es decir la redundancia de *software* necesitará una memoria *(hardware*) para almacenar los datos del sistema [25].

2.9 VENTAJA DE LOS SISTEMAS DIGITALES EN LAS COMUNICACIONES INDUSTRIALES

Los sistemas digitales tiene características especiales que determinan cómo estos sistemas fallan y que mecanismos de tolerancia de fallas son apropiados. Primero, los sistemas digitales de comunicación son discretos. A diferencia de sistemas continuos, tales como sistemas de control analógicos, operan en pasos discontinuos. Segundo, los sistemas digitales codifican la información. A diferencia de sistemas continuos, los valores se representan por series de símbolos codificados. Tercero, los sistemas digitales pueden modificar su comportamiento basados en información que ellos procesan, ver figura 22.

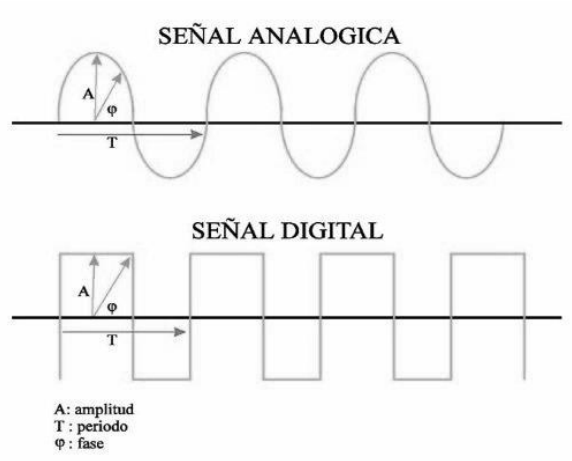


Figura 22: Sistemas digitales y sistemas analógicos [29]

Ya que los sistemas digitales son sistemas discretos, los resultados pueden probarse o compararse antes de que sean liberados al mundo exterior. Mientras los sistemas analógicos deben aplicar continuamente valores límites o redundantes, un sistema digital puede sustituir un resultado alternativo antes de enviar un valor de salida, en la práctica las comunicaciones digitales se secuencia a partir de una señal de reloj. Esta dependencia a un reloj hace que una fuente de reloj exacta sea tan importante como una fuente de alimentación, pero también significa que secuencias idénticas de instrucciones llevan esencialmente la misma cantidad de tiempo. Una de los mecanismos de tolerancia de fallas más común es el tiempo de espera, usa esta propiedad para medir la actividad (o falta de actividad) de un programa.

El hecho de que los sistemas digitales codifiquen la información es extremadamente importante. La implicación más importante de la codificación de información es que los sistemas digitales pueden guardar información en forma segura por un largo período de tiempo, lo que no sucede con sistemas analógicos que se necesita una gran capacidad de almacenamiento. Esto significa también que los sistemas digitales pueden guardar copias idénticas de información y esperar que las copias guardadas sean idénticas todavía después de un período de tiempo substancial. Esto hace posible el uso de las técnicas de comparación.

La codificación de información en sistemas digitales puede ser redundante, con varios códigos representando el mismo valor. La codificación redundante es la herramienta

más poderosa disponible para asegurar que la información en un sistema digital no ha sido cambiada durante el almacenamiento o transmisión. La codificación redundante puede implementarse en varios niveles de la estructura jerárquica de los sistemas de comunicación industrial. En los niveles inferiores, patrones de código cuidadosamente diseñados anexados a bloques de información digital pueden permitir que *hardware* de propósito general corrija un número de fallas diferentes de comunicación o almacenamiento, incluyendo cambios a *bits* únicos o cambios a varios *bits* adyacentes. La paridad en los sistemas de transmisión y recepción de datos es un ejemplo común del uso de esta codificación. Ya que un solo *bit* de información puede tener consecuencias significativas en niveles superiores.

2.10 REDUNDANCIA EN LA TRANSMISIÓN DE DATOS

Una vez que se conocen los tipos de errores que pueden existir, es necesario identificarlos. En un entorno de comunicación de datos no se tendrá una copia de los datos originales que permita comparar los datos recibidos para detectar si hubo errores en la transmisión. En este caso, no habrá forma de detectar si ocurrió un error hasta que se haya decodificado la transmisión y se vea que no tienen sentido los datos recibidos. Si los computadores comprobaran errores de esta forma, sería un proceso muy lento y costoso. Es necesario un mecanismo que sea sencillo y completamente efectivo [30].

El concepto clave para detectar o corregir errores es la redundancia. Para esto es necesario enviar bits extra junto con los datos. Estos bits son añadidos por el emisor y eliminados por el receptor, permitiendo detectar y posiblemente corregir los bits afectados.

Un mecanismo de detección de errores que podría satisfacer los requisitos antes expuestos sería enviar dos veces cada unidad de datos. El dispositivo receptor podría entonces comparar ambas copias bit a bit. Cualquier discrepancia indicaría un error y se podría corregir mediante un mecanismo apropiado. Este sistema sería extremadamente lento. No solamente se doblaría el tiempo de transmisión, sino que además habría que añadir el tiempo necesario para comparar cada unidad bit a bit.

El concepto de incluir información extra en la transmisión con el único propósito de comparar es bueno. Pero en lugar de repetir todo el flujo de datos, se puede añadir un grupo

más pequeño de bits al final de cada unidad. Esta técnica se denomina redundancia porque los bits extra son redundantes a la información, descartándose tan pronto como se ha comprobado la exactitud de la transmisión.

2.11 DIFERENCIA ENTRE DETECCIÓN Y CORRECCIÓN DE ERRORES

En los sistemas de comunicación es más fácil corregir un error que llegar a detectarlo. La detección en los sistemas digitales es únicamente verificar las dos única posibilidades 0 ó 1, por tal motivo la corrección se vuelve sencilla ya que únicamente se invierte los valores de los bits en caso de que exista el error, pero primeramente se debe determinar la cantidad de bits erróneos y la ubicación dentro de la trama de bits.

Para realizar la corrección de los errores se puede utilizar dos formas:

2.11.1 El *Backwards Error Correction*.- Es también conocida como una "solicitud de repetición automática" [31] es una técnica de corrección de error en el que un dispositivo receptor envía una solicitud al dispositivo fuente para volver a enviar la información. Esta técnica se utiliza en situaciones en las que algunos de los datos transmitidos se han perdido o dañado durante el tránsito y el dispositivo de transmisión debe volver a enviar la información a fin de que el dispositivo receptor pueda comprender la transmisión.

2.11.2 *Forward Error Correction*.- Es una técnica de corrección de errores que implica la codificación de un mensaje de una manera redundante, que permite al receptor reconstruir los bits perdidos sin la necesidad de retransmisión [31].

El "*Forward Error Correction*" es lo contrario de "Forward Error Correction", en el que un dispositivo de transmisión simplemente envía la información redundante para compensar los posibles errores.

Teóricamente es posible corregir cualquier error automáticamente en un código binario. Sin embargo, los códigos correctores son más sofisticados que los códigos detectores y necesitan más bits de redundancia. El número de bits necesarios para corregir un error de ráfaga es tan alto que en la mayoría de los casos su uso no resulta eficiente.

CAPÍTULO 3

PROTOCOLO DE COMUNICACIÓN MODBUS

3.1.- CARACTERÍSTICAS

Protocolo MODBUS fue desarrollado por MODICON en 1979 es un protocolo de comunicación situado en el nivel 7 del modelo OSI basado en la arquitectura maestro-esclavo (*master/slave*) o cliente servidor (*client/server*) los mismo que se utilizaron para interconectar inicialmente sus Controladores Programables [32].

ModBus es ampliamente utilizado en la automatización industrial y se ha convertido en el estándar industrial. Los dispositivos de control o los instrumentos de medida de diferentes fabricantes pueden vincularse a una red de seguimiento de la industria mediante el protocolo ModBus [33].

Los dispositivos que utilizan el protocolo ModBus para comunicarse, normalmente tienen como medio físico una interfaz RS485, para lograr una conexión multipunto o multinodo, aunque en algunas ocasiones, se pueden encontrar dispositivos estableciendo comunicación bajo el protocolo ModBus; usando como medio físico una interfaz RS232 [33] [34].

Además existe ModBus Plus que es un sistema de red de área local para aplicaciones de control industrial. Los dispositivos conectados a la red pueden intercambiar mensajes para el control y la supervisión de los procesos en las ubicaciones remotas de la planta industrial [35].

La figura 23 nos indica los dispositivos que utiliza el protocolo ModBus para comunicarse [32].

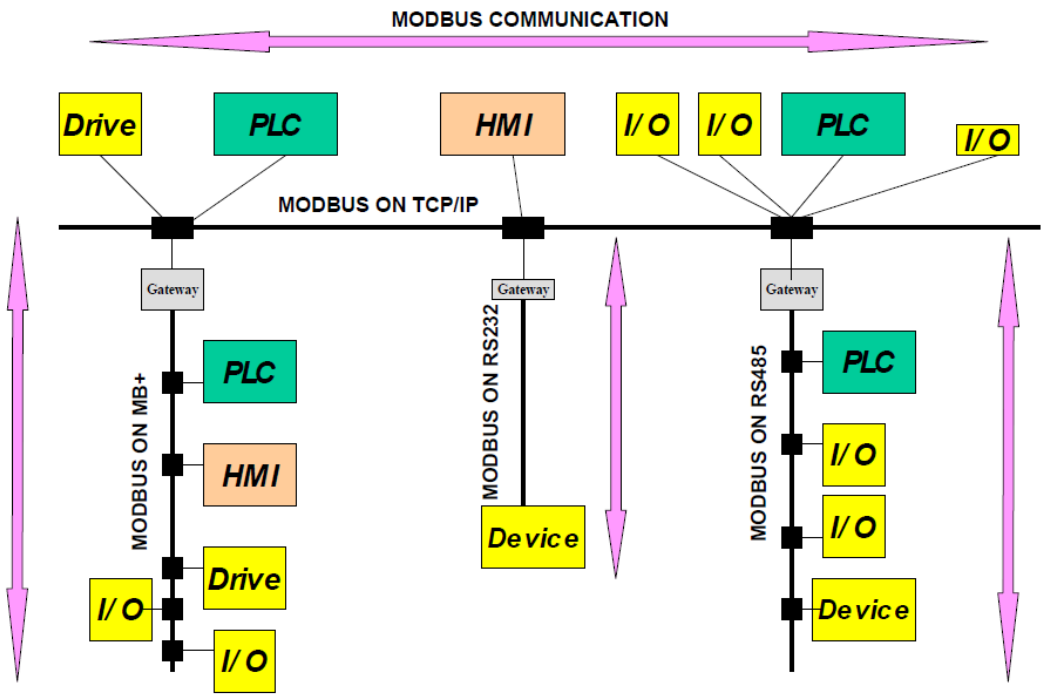


Figura 23: Ejemplo de arquitectura de una red ModBus [32]

Los dispositivos se comunican usando una técnica Maestro – Esclavo (Master - Slave), en la cual sólo un dispositivo (el maestro) puede iniciar transacciones (llamadas 'peticiones'). Los otros dispositivos (los esclavos) responden suministrando al maestro el dato solicitado, o realizando la acción solicitada en la petición. Entre los dispositivos maestros típicos se incluyen los procesadores centrales, los paneles de programación, PC (Computadores Personales) y PLC (Controladores Lógicos Programables). Algunos de los esclavos típicos son los PLC (Controladores Lógicos Programables), controladores, analizadores y tarjetas de adquisición de datos.

ModBus también se usa para la conexión de un ordenador de supervisión con una unidad remota (RTU) en sistemas de supervisión adquisición de datos (SCADA). Existen versiones de protocolo ModBus para puertos serie y Ethernet (ModBus/TCP) [36].

El protocolo ModBus tiene las siguientes características:

- Soportado por: Compañías internacionales de equipos de control de procesos
- Capa Física: RS-232, RS-422, RS-485, TTY.

- Longitud: 15m a 1200 m.
- Velocidad de transmisión: Máximo 19.2 *Kbits*/s.
- Método de acceso al bus: central Maestro/esclavo.
- Capacidad: Un maestro y 247 esclavos.

3.2.-FUNCIONAMIENTO DEL CICLO PETICIÓN RESPUESTA DEL PROTOCOLO MODBUS

3.2.1 Pregunta o Petición: El código de función en la petición indica la dirección del dispositivo esclavo como la acción o función que debe realizar. Los *bytes* de datos contienen cualquier información adicional que el esclavo necesita para llevar a cabo la función. El campo de comprobación de error proporciona un método para que el esclavo valide la integridad del contenido del mensaje recibido.

3.2.2 Respuesta: Una vez que el dispositivo esclavo direccionado ha recibido la pregunta o petición, este realiza la tarea indicada por el código de función y arma una trama de respuesta. Esta respuesta puede tener dos formatos.

El primero que se elabora como respuesta normal a la petición realizada por el maestro, la cual contiene una réplica del código de función de la pregunta, la confirmación o los datos solicitados y obviamente el campo de chequeo de errores.

El segundo posible formato sería que el esclavo genere una respuesta tipo error, en el caso que este dispositivo no pueda cumplir con la petición realizada o haya ocurrido un fallo; en este caso la trama de respuesta estará conformada por un código de función diferente al de la pregunta o petición, en el campo de datos se especificará el tipo de error ocurrido y, por último se adiciona el campo de comprobación de errores para demostrar la integridad de la trama durante el envío.

En la figura 24 se hace una representación de la forma en que se realizan las peticiones y respuestas en el protocolo ModBus.

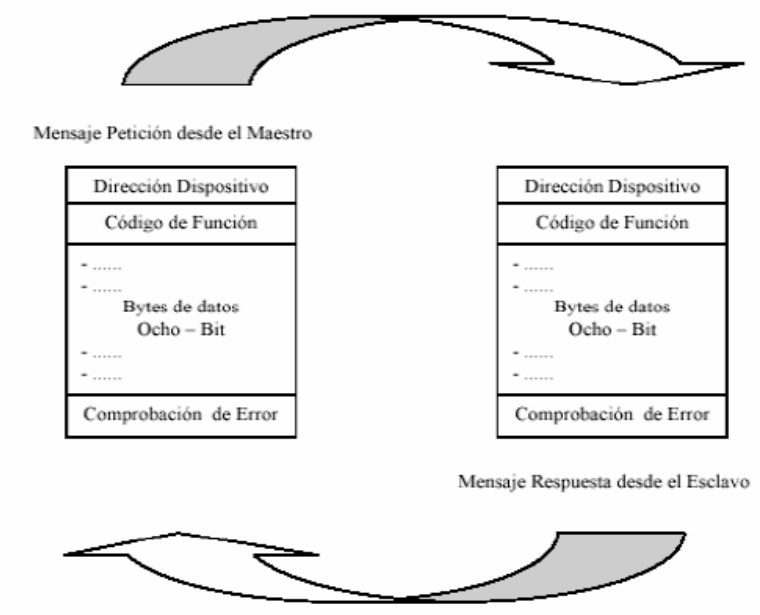


Figura 24: Ciclo petición-respuesta maestro-esclavo [34]

3.3.- MODOS DE TRANSMISIÓN SERIE DEL PROTOCOLO MODBUS

El protocolo ModBus posee dos modos esenciales de funcionamiento:

3.3.1 Modo ASCII (*American Standard Code for Information Interchange*): Cada *byte* en un mensaje, se envía como dos caracteres ASCII. La principal ventaja de este modo es que permite intervalos de tiempo de hasta un segundo entre caracteres sin dar lugar a error.

El formato para cada *byte* en modo ASCII es:

Sistema de codificación:

- Hexadecimal, caracteres ASCII 0-9, A-F.
- Un caracter hexadecimal contenido en cada caracter ASCII del mensaje.

***Bits* por *byte*:**

- 1 *bit* de arranque.
- 7 *bits* de datos, el menos significativo se envía primero.
- 1 *bit* para paridad Par o Impar; ningún *bit* para No paridad.
- 1 *bit* de paro si se usa paridad; 2 *bits* si no se usa paridad.

Campo de Comprobación de Error:

- Comprobación de Redundancia Longitudinal (LRC).

En la figura 25 se representa la forma en que se arma o se encapsula un *byte* en el modo ASCII.

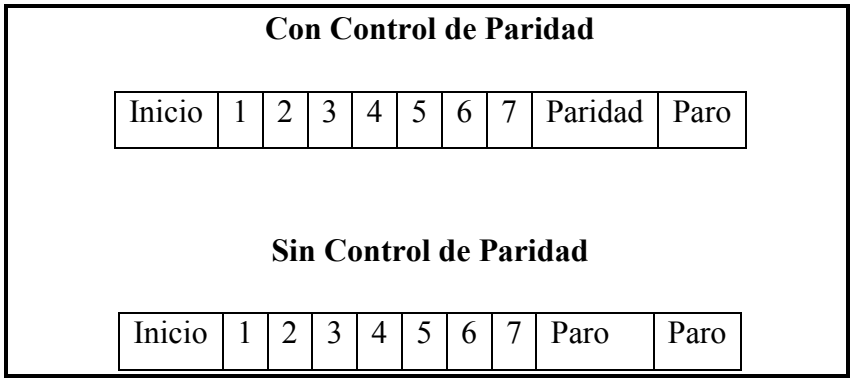


Figura 25: Orden de *bits* en una trama ASCII

3.3.2 Modo RTU (*Remote Terminal Unit*): Cuando los controladores son configurados para comunicar en una red ModBus usando el modo RTU, cada *byte* en un mensaje contiene dos dígitos hexadecimales de 4 *bits*. La principal ventaja de este modo es que su mayor densidad de caracter permite un mejor rendimiento que el modo ASCII para la misma velocidad. Cada mensaje debe ser transmitido en un flujo continuo.

El formato para cada *byte* en modo RTU es:

Sistema de codificación:

- Binario 8-*bits*, hexadecimal 0-9, A-F.
- Dos dígitos hexadecimales contenidos en cada campo de 8 *bits* del mensaje.

Bits* por *byte:

- 1 *bit* de arranque.
- 8 *bits* de datos, el menos significativo se envía primero.
- 1 *bit* para paridad Par o Impar; ningún *bit* para No paridad.
- 1 *bit* de paro si se usa paridad; 2 *bits* si no se usa paridad.

Campo de Comprobación de Error:

Comprobación de Redundancia Cíclica (CRC).

En la figura 26 se representa la forma en que se arma o se encapsula un *byte* en el modo RTU.

Figura 26: Orden de *bits* en una trama RTU

3.4.- TRAMA DEL MENSAJE MODBUS

En cualquiera de los modos de transmisión serie (ASCII o RTU), un mensaje ModBus es situado por el dispositivo que transmite, en una trama que tiene un comienzo y un final conocidos. Esto permite a los dispositivos receptores comenzar en el inicio del mensaje, leer la parte de la dirección y determinar qué dispositivo es solicitado (o todos los dispositivos si es una difusión 'dirección = 0') y conocer cuándo se ha completado el mensaje.

3.4.1 Trama ModBus ASCII: En modo ASCII, los mensajes comienzan con el caracter (:) 'dos puntos' (ASCII 3A hex) y terminan con el par de caracteres (CRLF) 'Retorno de Carro + Avance de Línea) (ASCII 0D hex y 0A hex).

Los caracteres a transmitir permitidos para todos los demás campos son 0-A, A-F hexadecimal; pueden haber intervalos de hasta un segundo entre caracteres dentro del

mensaje. Si transcurre más tiempo entre caracteres, el dispositivo receptor asume que ha ocurrido un error.

Los dispositivos conectados a la red ModBus, monitorizan el bus de red continuamente hasta que detectan un carácter 'dos puntos'. Cuando se recibe cada dispositivo decodifica el próximo campo (el campo de direcciones) para así saber si esta es la dirección que tiene asignada, tal como se muestra en la figura 27.

INICIO	DIRECCIÓN	FUNCIÓN	DATOS	CONTROL LRC	FIN
1 Caracter (:)	2 caracteres	2 caracteres	N caracteres	2 caracteres	2 caracteres CRLF

Figura 27: Trama ModBus ASCII

3.4.2 Trama ModBus RTU: En modo RTU, los mensajes comienzan con un intervalo silencioso de al menos 3.5 veces el tiempo de un de caracter. Esto es más fácilmente implementado como un múltiplo de tiempos de caracter a la velocidad de transmisión configurada en la red (mostrado como T1-T2-T3-T4 en la Figura 28). El primer campo transmitido es entonces la dirección del dispositivo destinatario.

Los caracteres a transmitir permitidos para todos los campos son 0-9, A-F hexadecimal. Los dispositivos conectados en red revisan el bus de red continuamente incluso durante los intervalos 'silenciosos'. Cuando el primer campo (el campo de dirección) es recibido, cada dispositivo lo decodifica para enterarse si es el dispositivo requerido.

Siguiendo al último caracter transmitido, un intervalo de al menos 3.5 veces el tiempo de un caracter, señala el final del mensaje. Un nuevo mensaje puede comenzar después de transcurrido este intervalo.

La trama completa del mensaje debe ser transmitida como un flujo continuo, si un intervalo silencioso de más de 1.5 veces el tiempo de un caracter tiene lugar antes de completar la trama, el dispositivo receptor desecha el mensaje incompleto y asume que el próximo *byte* será el campo de dirección de un nuevo mensaje.

De forma similar, si un nuevo mensaje comienza antes de que transcurran 3.5 veces el tiempo de un caracter después de un mensaje previo, el dispositivo receptor lo considerará una continuación del mensaje previo. Esto dará lugar a un error, ya que el valor en el campo final CRC no será válido para el mensaje combinado. La figura 28 muestra una trama de mensaje ModBus RTU típica [37].

INICIO	DIRECCIÓN	FUNCIÓN	DATOS	CONTROL CRC	FIN
T1-T2-T3-T4	8 *bits*	8 *bits*	N· 8 *bits*	16 *bits*	T1=T2=T3-=T4

Figura 28: Trama ModBus RTU

3.5.- MANIPULACIÓN DE LOS CAMPOS DE LAS TRAMAS DEL PROTOCOLO MODBUS

3.5.1 Campo de Dirección: El campo de dirección de un mensaje contiene ocho *bits*. Las direcciones de esclavo válidas están en el rango de 0 – 247 decimal. Los dispositivos esclavos individuales tienen direcciones asignadas en el rango 1–247. Un maestro se comunica con un esclavo situando la dirección del mismo en el campo dirección del mensaje. Cuando el esclavo envía su respuesta, sitúa su propia dirección en el campo dirección de la respuesta para dar a conocer al maestro qué esclavo está respondiendo [37].

La dirección 0 es utilizada como dirección de difusión, la cual todos los dispositivos esclavos reconocen. Cuando el protocolo ModBus es usado en redes de nivel más alto, las difusiones pueden no estar permitidas o pueden ser reemplazadas por otros métodos.

3.5.2 Campo de Función: El campo de código de función de una trama de mensaje contiene ocho *bits*. Los códigos válidos están en el rango de 1–255 decimal. De los cuales, solo algunos códigos pueden ser utilizados, en la actualidad solo pueden ser utilizados 24 códigos de función.

Cuando un mensaje es enviado desde un maestro a un dispositivo esclavo, el campo de código de función indica al esclavo qué tipo de acción ha de ejecutar. Por ejemplo: Lectura de los estados *ON/OFF* de un grupo de bobinas o entradas discretas; lectura de los datos contenidos en un grupo de registros; lectura del estado de diagnóstico de un esclavo; escribir en determinadas bobinas o registros; o permitir cargar, salvar o verificar el programa dentro del esclavo.

Cuando el esclavo responde al maestro, utiliza el campo de código de función para indicar, ya sea, una respuesta normal (libre de error) o que algún tipo de error ha tenido lugar (denominado respuesta de excepción). Para una respuesta normal, el esclavo simplemente replica el código de función original. Para una respuesta de excepción, el esclavo devuelve un código que es equivalente al código de función original con su *bit* más significativo (MSB) puesto en 1.

Por ejemplo, un mensaje desde un maestro a un esclavo para leer un grupo de registros sostenidos tendría el siguiente código de función:

0000 0011 (03 Hexadecimal)

Si el dispositivo esclavo ejecuta la acción solicitada, sin error, devuelve el mismo código en su respuesta. Si ocurre una excepción. Devuelve:

1000 0011 (83 Hexadecimal)

Además de la modificación del código de función para una respuesta de excepción, el esclavo sitúa un único código en el campo de datos del mensaje respuesta. Esto indica al maestro qué tipo de error ha tenido lugar, o la razón para la excepción.

El programa de aplicación del maestro tiene la responsabilidad de manejar las respuestas de excepción. Los procedimientos típicos son: enviar subsiguientes reintentos de mensaje, intentar mensajes de diagnóstico al esclavo y notificar operadores.

3.5.3 Campo de Datos: El campo de datos se construye utilizando conjuntos de 2 dígitos hexadecimales, en el rango de 00 a FF hexadecimal. Puede formarse a partir de un caracter RTU.

El campo de datos de los mensajes enviados desde un maestro a un esclavo, contiene información adicional que el esclavo debe usar para tomar la acción definida por el código de función. Esto puede incluir partes como direcciones discretas y de registros, la cantidad de partes que han de ser manipuladas y la cantidad de *bytes* de datos contenidos en el campo.

Por ejemplo, si el maestro solicita a un esclavo leer un grupo de registros sostenidos (código de función 03), el campo de datos especifica el registro de inicio y cuántos registros han de ser leídos. Si el maestro escribe sobre un grupo de registros en el esclavo (código de función 10 hexadecimal), el campo de datos especifica el registro de inicio, cuántos registros se van a escribir, la cantidad de *bytes* de datos que siguen en el campo de datos y los datos que se deben escribir en los registros.

Si no ocurre ningún error, el campo de datos en la respuesta del esclavo al maestro contiene los datos requeridos. Si ocurre un error, él campo de datos contiene un código de excepción que puede ser usada por el maestro para determinar la próxima acción a tomar.

El campo de datos puede ser inexistente (de longitud cero) en ciertos tipos de mensajes. Por ejemplo, en una petición de un dispositivo maestro a un esclavo para que responda con su Anotación de Eventos de Comunicación (código de función 0B hexadecimal), el esclavo no requiere ninguna información adicional. En este caso, el código de función por sí solo especifica la acción.

3.5.4 Campo Comprobación de Error: Dos tipos de métodos de comprobación de error son utilizados para las redes ModBus Estándar. El contenido del campo de comprobación de error depende del método que esté siendo utilizado.

- **ASCII**: Cuando el modo ASCII es usado, el campo de Comprobación de Error contiene dos caracteres ASCII. Los caracteres de comprobación de error son el resultado de un cálculo de Comprobación de Redundancia Longitudinal (LRC), que es realizado sobre el contenido del mensaje; excluyendo los 'dos puntos' del comienzo y los caracteres 'CRLF' de finalización. Los caracteres LRC son añadidos al mensaje como el último campo que precede a los caracteres CRLF.

- **RTU:** Cuando el modo RTU es usado, el campo de Comprobación de Error contiene un valor de 16 *bits* implementado como dos *bytes* de 8 *bits*. El valor de comprobación de error es el resultado de un cálculo de Comprobación de Redundancia Cíclica (CRC), realizado sobre el contenido del mensaje. El campo CRC es añadido al mensaje como último campo del mensaje. La forma de hacerlo

es, añadir primero el *byte* de orden bajo del campo, seguido del *byte* de orden alto. El *byte* de orden alto del CRC es el último *byte* a enviar en el mensaje.

3.6.- MÉTODO DE COMPROBACIÓN DE ERRORES

Las redes serie ModBus Estándar utilizan dos tipos de comprobación de error. La comprobación de paridad (Par o Impar) puede ser aplicada opcionalmente a cada caracter. La comprobación de la trama (LRC-Comprobación de Redundancia Longitudinal o CRC-Comprobación de Redundancia Cíclica) es aplicada al mensaje completo. Ambas comprobaciones, de caracter y de trama de mensaje son generadas en el dispositivo maestro y aplicadas a los contenidos del mensaje antes de la transmisión.

El dispositivo esclavo comprueba cada caracter y la trama del mensaje completo durante la recepción.

El maestro es configurado por el usuario para aguardar durante un tiempo de espera predeterminado antes de abortar la transacción. Este intervalo es establecido para ser lo suficientemente largo para que cualquier esclavo responda normalmente. Si el esclavo detecta un error de transmisión, el mensaje no será tenido en cuenta. El esclavo no construirá una respuesta para el maestro. Así el tiempo de espera expirará y permite al programa del maestro tratar el error. Observe que un mensaje enviado a un dispositivo esclavo inexistente también causará un error de tiempo excedido *time out.*

En esta sección se explicará únicamente el método de comprobación de error CRC, puesto que tanto el Control de Comunicación ModBus (en este caso un control COM *Object* que contiene las propiedades, procedimientos y funciones necesarias para establecer y mantener una comunicación ModBus), como el dispositivo Esclavo ModBus que se implementarán, sólo han soportar el modo de transmisión ModBus RTU.

3.6.1 Comprobación CRC (Comprobación de Redundancia Cíclica) En modo RTU, los mensajes incluyen un campo de comprobación de error que está basado en un método Comprobación de Redundancia Cíclica (CRC). El campo CRC controla el contenido del mensaje completo. Se aplica con independencia de cualquier método de control de paridad utilizado para los caracteres individuales del mensaje.

El campo CRC es de dos *bytes*, conteniendo un valor binario de 16 *bits*. El valor CRC es calculado por el dispositivo emisor, que añade el CRC al mensaje. El dispositivo receptor calcula el CRC durante la recepción del mensaje y compara el valor calculado con el valor recibido en el campo CRC. Si los dos valores no son iguales, resulta un error.

Para calcular el valor CRC ModBus se precarga un registro de 16 *bits*, con cada uno de los *bits* puestos en 1. Luego comienza un proceso que toma los sucesivos *bytes* del mensaje y los opera con el contenido del registro y actualiza éste con el resultado obtenido. Sólo los 8 *bits* de dato de cada caracter son utilizados para generar el CRC.

Los *bits* de arranque y paro y el *bit* de paridad, no se tienen en cuenta para el CRC. Durante la generación del CRC, se efectúa una operación booleana OR exclusivo (XOR) a cada caracter de 8 *bits* con el contenido del registro. Entonces al resultado se le aplica un desplazamiento de *bit* en la dirección de *bit* menos significativo (LSB), rellenando la posición del *bit* más significativo (MSB) con un cero. El LSB es extraído y examinado. Si el LSB extraído fuese un 1, se realiza un XOR entre el registro y un valor fijo preestablecido[1]. Si el LSB fuese un 0, no se efectúa un el XOR.

Este proceso es repetido hasta haber cumplido 8 desplazamientos. Después del último desplazamiento (el octavo), el próximo *byte* es operado XOR con el valor actual del registro y el proceso se repite con ocho desplazamientos más, como se ha descrito más arriba y así con todos los *bytes* del mensaje. El contenido final del registro, después de que todos los *bytes* del mensaje han sido procesados, es el valor del CRC.

Cuando el CRC es añadido al mensaje, primero se añade el *byte* de orden bajo seguido del *byte* de orden alto.

Un procedimiento para generar un CRC es:

1. Cargar un registro de 16 *bits* que denominaremos registro CRC, con FFFF (todos 1).
2. XOR del primer *byte* - 8 *bits* - del mensaje con el *byte* de orden bajo del registro CRC de 16 *bits*, colocando el resultado en el registro CRC.

[1] El valor preestablecido es A001 hex, correspondiente al polinomio generador CRC16 'Inverso', que es el que se aplica al CRC *Modbus*. *Modicon Modbus* Reference Guide. PI-MBUS-300 Rev. J.

3. Desplazar el registro CRC un *bit* a la derecha (hacia el LSB "*bit* menos significativo") rellenando con un cero el MSB (*bit* más significativo). Extraer y examinar el LSB.

4. (Si el LSB era 0): Repetir paso 3 (otro desplazamiento). (Si el LSB era 1): Hacer XOR entre el registro CRC y el valor polinómico A001hex (1010 0000 0000 0001).

5. Repetir los pasos 3 y 4 hasta que se hayan efectuado 8 desplazamientos. Una vez hecho esto, se habrá procesado un *byte* completo – 8 *bits*.

6. Repetir los pasos 2 al 5 para el próximo *byte* – 8 *bits* – del mensaje. Continuar haciendo esto hasta que todos los *bytes* hayan sido procesados.

7. El contenido final del registro CRC es el valor CRC.

8. Cuando el CRC es situado en el mensaje, sus *bytes* de orden alto y bajo han de ser permutados.

3.7 DATOS Y FUNCIONES DE CONTROL DEL PROTOCOLO MODBUS

3.7.1 Formato de las Funciones ModBus

3.7.1.1 Expresión de los Valores Numéricos

Los valores numéricos de datos, direcciones, y códigos se expresan como valores como valores hexadecimales en los campos de los mensajes de las tramas los mismos que pueden ser transformados a códigos decimales.

3.7.1.2 Direcciones en los Mensajes ModBus

Todas las direcciones en los mensajes ModBus son referenciadas desde cero en código hexadecimal, el mismo, que corresponde a la primera unidad en código decimal. Por ejemplo:

La bobina conocida como "bobina 1", en un PLC tiene la dirección de bobina 0000 en el campo de dirección de un mensaje ModBus.

La bobina 127 decimal es direccionada como bobina 007E hex (126 decimal).

El registro sostenido 40001 tiene la dirección de registro 0000 en el campo de dirección de un mensaje ModBus. El campo de código de función ya especifica una operación sobre un 'registro sostenido'. Por lo tanto la referencia '4XXXX' está implícita.

El registro sostenido 40108 es direccionado como registro 006B hex (107 decimal).

3.7.1.3 Ejemplo de una Petición y una Respuesta en ModBus

La tabla 1 y 2 muestran un ejemplo de una petición y una respuesta respectivamente en ModBus. Ambas tablas muestran el contenido del campo en hexadecimal, en modo ASCII y en RTU de la siguiente manera:

Tabla 1: Petición en código ASCII y RTU

Nombre del Campo	Ejemplo (Hexadecimal)	Caracteres ASCII	RTU Campo de 8 *bits*
Cabecera		: (dos puntos)	Ninguno
Dirección del Esclavo	06	0 6	0000 0110
Función	03	0 3	0000 0011
Dirección de Comienzo Alto	00	0 0	0000 0000
Dirección de Comienzo bajo	0B	0 B	0000 1011
Número de Registros Altos	00	0 0	0000 0000
Número de Registros Bajos	03	0 3	0000 0011
Comprobación de Errores		LRC (2 caract)	CRC (16*bits*)
Terminación		CR LF	Ninguno
	Total de *Bytes*	17	8

Tabla 2: Respuesta en código ASCII y RTU

Nombre del Campo	Ejemplo (Hexadecimal)	Caracteres ASCII	RTU Campo de 8 *bits*
Cabecera		: (dos puntos)	Ninguno
Dirección del Esclavo	06	0 6	0000 0110
Función	03	0 3	0000 0011
Cómputo de *Bytes*	06	0 6	0000 0110
Dato Alto	02	0 2	0000 0010
Dato Bajo	2B	2 B	0010 1011
Dato Alto	00	0 0	0000 0000
Dato Bajo	00	0 0	0000 0000
Dato Alto	00	0 0	0000 0000
Dato Bajo	63	6 3	0110 0011
Comprobación de Errores		LRC (2 caract)	CRC (16*bits*)
Terminación		CR LF	Ninguno
	Total de *Bytes*	23	11

El maestro realiza una solicitud de Lectura de Registros Sostenidos, al dispositivo esclavo con dirección 06. El mensaje solicita datos numéricos de tres registros desde el 40108 al 40110. Observe que el mensaje especifica la dirección de comienzo como 0107 (006B hex).

La respuesta del esclavo replica el código de función, indicando que esto es una respuesta normal.

El campo 'Cómputo de *Bytes'* especifica cuántas unidades de datos de 8 *bits* se devuelven. Muestra la cantidad de *bytes* de datos que vienen a continuación, ya sea ASCII o RTU.

Si se trabaja en modo ASCII debe haber dos caracteres ASCII para contener cada unidad de dato de 8 *bits*. Por ejemplo, el dato: 63 hexadecimal se envía como un *byte* de ocho *bits* en modo RTU (0110 0011). El mismo valor, enviado en modo ASCII requiere dos caracteres ASCII, el ASCII '6' (011 0110) y el ASCII '3' (011 0011). El campo 'Cómputo de *Bytes'* contabiliza este dato como un solo dato de 8 *bits*, con independencia del método de trama de carácter (ASCII o RTU).

3.7.1.4 Código de Funciones

La tabla 3 indica los códigos de Funciones que utiliza el protocolo ModBus

Tabla 3: Código de funciones del protocolo ModBus

Código	**Nombre**
01	Leer Estados de Bobinas
02	Leer Estados de Entradas
03	Leer Registros Mantenidos
04	Leer Registros de Entradas
05	Forzar una única Bobina
06	Preestablecer un único Registro
07	Leer Status de Excepción
08	Diagnósticos
09	Programar 484
10	Selección 484
11	Buscar Contador de Eventos de Comunicación
12	Buscar Anotación de Eventos de Comunicación
13	Programar Controlador
14	Selección Controlador
15	Forzar Múltiples Bobinas
16	Preestablecer Múltiples Registros
17	Reportar Identificación de Esclavo
18	Programar 884/M84
19	Resetear Enlace de Comunicaciones
20	Leer Referencia General
21	Escribir Referencia General
22	Escribir con máscara en Registros 4X
23	Leer/Escribir Registros
24	Leer Cola FIFO

A pesar de la gran cantidad de funciones disponibles en el protocolo, la gran mayoría de ellos son utilizados por los controladores propios de Modicon, por lo que es

muy poco común encontrar aplicaciones o componentes de *Software* que las implementen todas las funciones, siendo las más utilizadas las que se indica en la tabla 4:

Tabla 4: Funciones más utilizadas del protocolo ModBus

Código	Nombre
01	Leer Estados de Bobinas
02	Leer Estados de Entradas
03	Leer Registros Mantenidos
04	Leer Registros de Entradas
05	Forzar una única Bobina
06	Preestablecer un único Registro
15	Forzar Múltiples Bobinas
16	Preestablecer Múltiples Registros

3.7.1.5 Errores de Petición para Funciones:

Cuando un "Esclavo" MODBUS detecta un problema al tratar de ejecutar la tarea indicada en la petición del Master MODBUS, responde una trama en la que indica que se produjo un error en el desarrollo de la petición y el tipo de error que se ha originado. Esta información se da a conocer al Master MODBUS modificando el Código de Función en la respuesta e indicando un valor en un nuevo campo denominado Código de Excepción.

El *byte* que da a conocer de la generación de un error es el que se envía después de MODBUS, incrementado en un valor de 80h. Luego se envía el Código de Excepción, el cual toma valores entre 1 y 4 dependiendo del tipo de error generado.

3.7.2 Funciones más Utilizados en los Protocolos ModBus

3.7.2.1 Función 01: Leer Estados de Bobinas

Lee el estado ENCENDIDO/APAGADO de las salidas discretas (referencias 0x, bobinas) en el esclavo. *Broadcast*[2] no es soportado.

Pregunta:

[2] *Broadcast*, difusión en español, es una forma de transmisión de información donde un nodo emisor envía información a una multitud de nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo.

El mensaje de Pregunta especifica la bobina de inicio y la cantidad de bobinas a ser leídas. Las bobinas son direccionadas de la siguiente manera: las bobinas 1 ... 16 son direccionadas en las localidades 0 ... 15.

En la tabla 5 se indica los valores en hexadecimal y decimal para el ejemplo de una pregunta para leer las bobinas 20 ... 56 del dispositivo esclavo 17 (11h).

Tabla 5: Petición función 01

Nombre del Campo	Hexadecimal	Decimal
Dirección del esclavo	11	17
Función	01	01
Dirección de inicio alto	00	00
Dirección de inicio bajo	13	20-1 (dirección)
Número de registros alto	00	00
Número de registro bajo	25	56-20+1=37 (cantidad de bobinas)
Comprobación de errores (LRC o CRC)	…	……

Respuesta:

El estado de la bobina en el mensaje de respuesta es empaquetado como una bobina por *bit* del campo de datos. El estado es indicado como: 1 = ENCENDIDO; 0=APAGADO. El *bit* menos significante del primer *byte* de datos contiene la bobina direccionada en la Pregunta. Las otras bobinas se van ubicando hasta la de mayor orden al final de este *byte*, y desde la de menor orden hasta la de mayor orden en los *bytes* subsiguientes.

Si la cantidad de bobinas devueltas no es un múltiplo de ocho, los *bits* restantes en el último *byte* de datos serán llenados con ceros (hasta el final de mayor orden del *byte*). El campo de Cuenta de *Bytes* especifica la cantidad de *bytes* de datos completos.

En la tabla 6 se indica un ejemplo de una respuesta para la pregunta anterior:

Tabla 6: Respuesta de la función 01

Nombre del campo	Hexadecimal
Dirección del esclavo	11
Función	01
Cuenta en *bytes*	05
Datos bobinas (27…….20)	CD
Datos bobinas (35…….28)	6B
Datos bobinas (43…….36)	B2
Datos bobinas (51…….44)	0E
Datos bobinas (56…….52)	1B
Comprobación de error (LRC o CRC)	------

El estado de las bobinas 27 ... 20 es mostrado con el *byte* de valor CD en hexadecimal, o el binario 1100 1101. La bobina 27 es el *bit* más significante de este *byte*, y la bobina 20 es el *bit* menos significante. De izquierda a derecha, el estado de las bobinas 27 ... 20 es ENCENDIDO – ENCENDIDO – APAGADO – APAGADO – ENCENDIDO – ENCENDIDO – APAGADO – ENCENDIDO.

Por convenio, los *bits* dentro de un *byte* son mostrados con el *bit* más significante a la izquierda, y el menos significante a la derecha. De esta forma las bobinas en el primer *byte* son 27 ... 20, de izquierda a derecha. El siguiente *byte* tiene las bobinas 35 ... 28, de izquierda a derecha. Como los *bits* son transmitidos en forma serial, éstos fluyen desde el *bit* menos significante al más significante así: 20 ... 27, 28 ... 35, y así sucesivamente.

En el último *byte* de datos, el estado de las bobinas 56 ... 52 es mostrado con el *byte* de valor 1B en hexadecimal, o el binario 0001 1011. La bobina 56 está en el *bit* en la cuarta posición desde la izquierda, y la bobina 52 es el *bit* menos significante de este *byte*. El estado de las bobinas 56 ... 52 es: ENCENDIDO – ENCENDIDO – APAGADO – ENCENDIDO – ENCENDIDO. Los tres *bits* restantes (hacia el final de mayor orden) son llenados con ceros [37].

Códigos de Excepción para Errores en la Función 01:

Para la Función 01 los Códigos de Excepción se muestran en la tabla 7 [32]:

Tabla 7: Códigos de excepción para errores en la función 01

Código	Error
01	Función 01 (Leer estados de bobinas) no soportada
02	Error de valores en cuanto a dirección de inicio para lectura y cantidad de salidas digitales a leer.
03	Error de cantidad de salidas digitales a leer.
04	Error al intentar leer las salidas digitales.

La tabla 8 indica la respuesta en el caso que exista un error para la función 01:

Tabla 8: Respuesta de la función 01 cuando tiene error

Nombre del campo	Valor en hexadecimal
Dirección de esclavo	11
Código de función	81(01+80h)
Código de excepción	01 ó 02 ó 03 ó 04
Comprobación de error (LRC o CRC)	……

3.7.2.2 Función 02: Leer Estados de Entrada

Lee los estados ENCENDIDO/APAGADO de las entradas discretas (referencias 1x) en el esclavo. *Broadcast* no es soportado.

Pregunta:

El mensaje de Pregunta especifica la entrada de inicio y la cantidad de entradas a ser leídas. Las entradas son direccionadas de la siguiente manera: las entradas 1...16 son direccionadas en las localidades 0 ... 15.

En la tabla 9 se indica los valores en hexadecimal y decimal para el ejemplo de una pregunta para leer las entradas 10197 ... 10218 del dispositivo esclavo 17 (11h).

Tabla 9: Petición función 02

Nombre del Campo	Hexadecimal	Decimal
Dirección del esclavo	11	17
Función	02	02
Dirección de inicio alto	00	00
Dirección de inicio bajo	C4	197-1 (dirección)
Número de registros alto	00	00
Número de registro bajo	16	218-197+1=22 (cantidad de entradas)
Comprobación de errores (LRC o CRC)	…	……

Respuesta:

El estado de la entrada en el mensaje de respuesta es empaquetado como una entrada por *bit* del campo de datos. El estado es indicado como: 1 = ENCENDIDO; 0=APAGADO. El *bit* menos significante del primer *byte* de datos contiene la entrada direccionada en la pregunta. Las otras entradas se van ubicando hasta la de mayor orden al final de este *byte*, y desde la de menor orden hasta la de mayor orden en los *bytes* subsiguientes.

Si la cantidad de entradas devueltas no es un múltiplo de ocho, los *bits* restantes en el último *byte* de datos serán llenados con ceros (hasta el final de mayor orden del *byte*). El campo de Cuenta de *Bytes* especifica la cantidad de *bytes* de datos completos.

En la tabla 10 se indica un ejemplo de una respuesta para la pregunta anterior:

Tabla 10: Respuesta de la función 02

Nombre del campo	Hexadecimal
Dirección del esclavo	11
Función	02
Cuenta en *bytes*	03
Datos (Entrada 10204…….10197)	AC
Datos (Entrada 10212…….10205)	DB
Datos (Entrada 10218…….10213)	35
Comprobación de error (LRC o CRC)	------

El estado de las entradas 10204 ... 10197 es mostrado con el *byte* de valor AC en hexadecimal o el binario 1010 1100. La entrada 10204 es el *bit* más significante de este *byte*, y la entrada 10197 es el *bit* menos significante. De izquierda a derecha, el estado de las entradas 10204 ... 10197 es ENCENDIDO – APAGADO – ENCENDIDO – APAGADO – ENCENDIDO – ENCENDIDO – APAGADO – APAGADO.

El estado de las entradas 10218 ... 10213 es mostrado con el *byte* de valor 35 en hexadecimal, o el binario 0011 0101. La entrada 10218 está en el *bit* de la tercera posición desde la izquierda, y la entrada 10213 es el *bit* menos significante. El estado de las entradas 10218 ... 10213 es: ENCENDIDO – ENCENDIDO – APAGADO – ENCENDIDO – APAGADO – ENCENDIDO. Los dos *bits* restantes (hacia el final de mayor orden) son llenados con ceros [37].

Códigos de Excepción para Errores en la Función 02:

Para la Función 02 los Códigos de Excepción se muestran en la tabla 11 [32]:

Tabla 11: Códigos de excepción para errores en la función 02

Código	Error
01	Función 02 (Leer estados discretos de entrada) no soportada
02	Error de valores en cuanto a dirección de inicio para lectura y cantidad de entradas digitales a leer.
03	Error de cantidad de entradas digitales a leer.
04	Error al intentar leer las entradas digitales.

La tabla 12 indica la respuesta en el caso que exista un error para la función 02:

Tabla 12: Respuesta de la función 02 cuando tiene error

Nombre del campo	Valor en hexadecimal
Dirección de esclavo	11
Código de función	82 (02+80h)
Código de excepción	01 ó 02 ó 03 ó 04
Comprobación de error (LRC o CRC)	……

CAPÍTULO 4

PRUEBAS Y RESULTADOS DE UNA COMUNICACIÓN INDUSTRIAL MEDIANTE PROTOCOLO MODBUS

En el desarrollo de este capítulo se verificará el correcto funcionamiento del protocolo ModBus para lo cual se utulizará un PC como maestro y un PLC Schneider con CPU 113 02 como esclavo, los mismos, que serán sometidos a pruebas y registrndo los datos obtenidos.

El programa Modscan32 permite trabajar al computador como un dispositivo Maestro MODBUS. Cuenta con la capacidad de mostrar tanto los valores de las variable que están siendo transportadas por el protocolo, así como las tramas que se envía y reciben por medio del puerto de comunicación RS-232, ver figura 29.

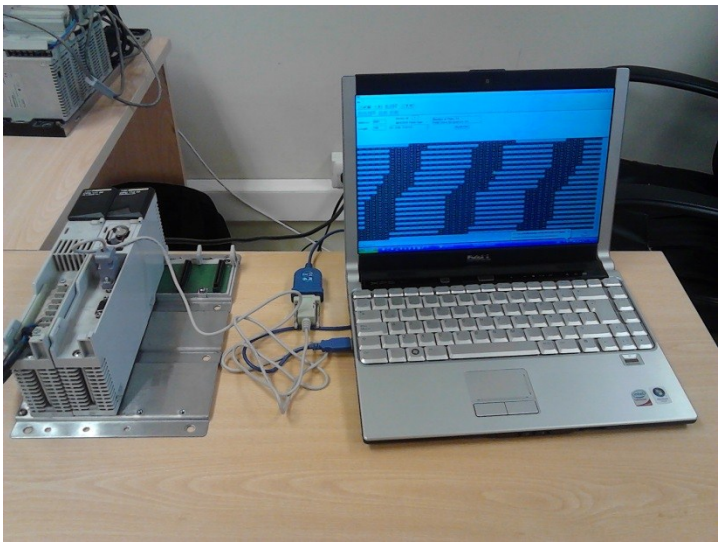


Figura 29: Comunicación entre un PC y un PLC Schneider con CPU 113 02

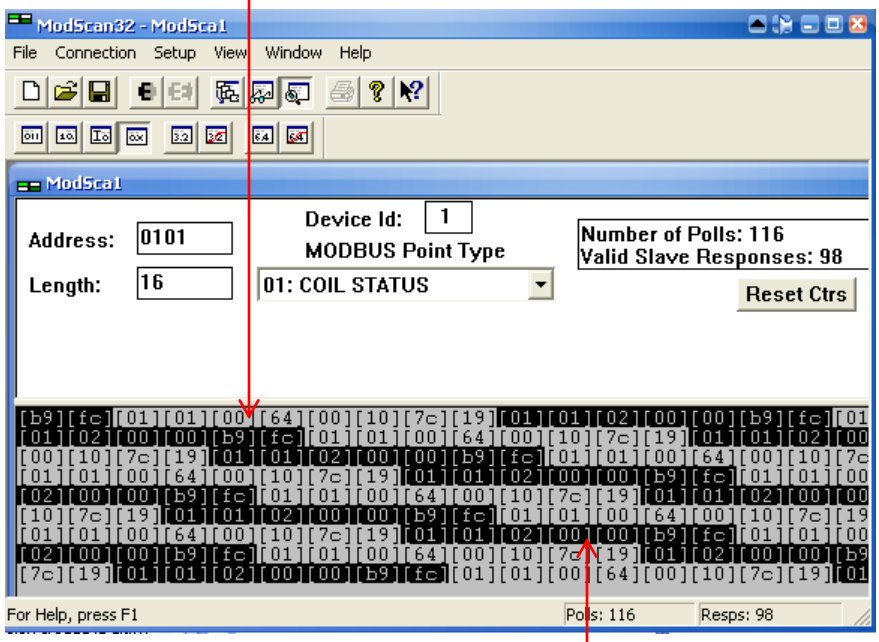
4.1 COMPROBACION DE LA FUNCIONES DEL PROTOCOLO MODBUS USANDO EL PROGRAMA MODSCAN 32 DE WINTECH

4.1.1 Comprobación de la Función 01: Leer Estados de Bobinas

Esta función puede ser comprobada usando directamente la interfaz principal del programa Modscan32. En la figura 30 se muestra el tráfico generado en el puerto RS-232

del computador, en el cual se aprecia la trama de petición y respuesta correspondiente al protocolo ModBus.

Solicitud enviada por el "Maestro" ModBus
01 Dirección del esclavo
01 Código de función
00 64 Dirección inicial de la bobina a leer
00 10 Cantidad de bobinas a leer
7C 19 CRC



Respuesta enviada por el "Esclavo" ModBus
01 Dirección del esclavo
01 Código de función
02 Cantidad de bytes con datos
00 00 Estado de las bobinas
B9 FC CRC

Figura 30: Visualización de tráfico de datos en Modscan32 para la función 01

4.1.2 Comprobación de la Función 02: Leer Estados de Entrada

Esta función puede ser comprobada usando directamente la interfaz principal del programa Modscan32. En la figura 31 se muestra el tráfico generado en el puerto RS-232 del computador, en el cual se aprecia la trama de petición y respuesta correspondiente al protocolo ModBus.

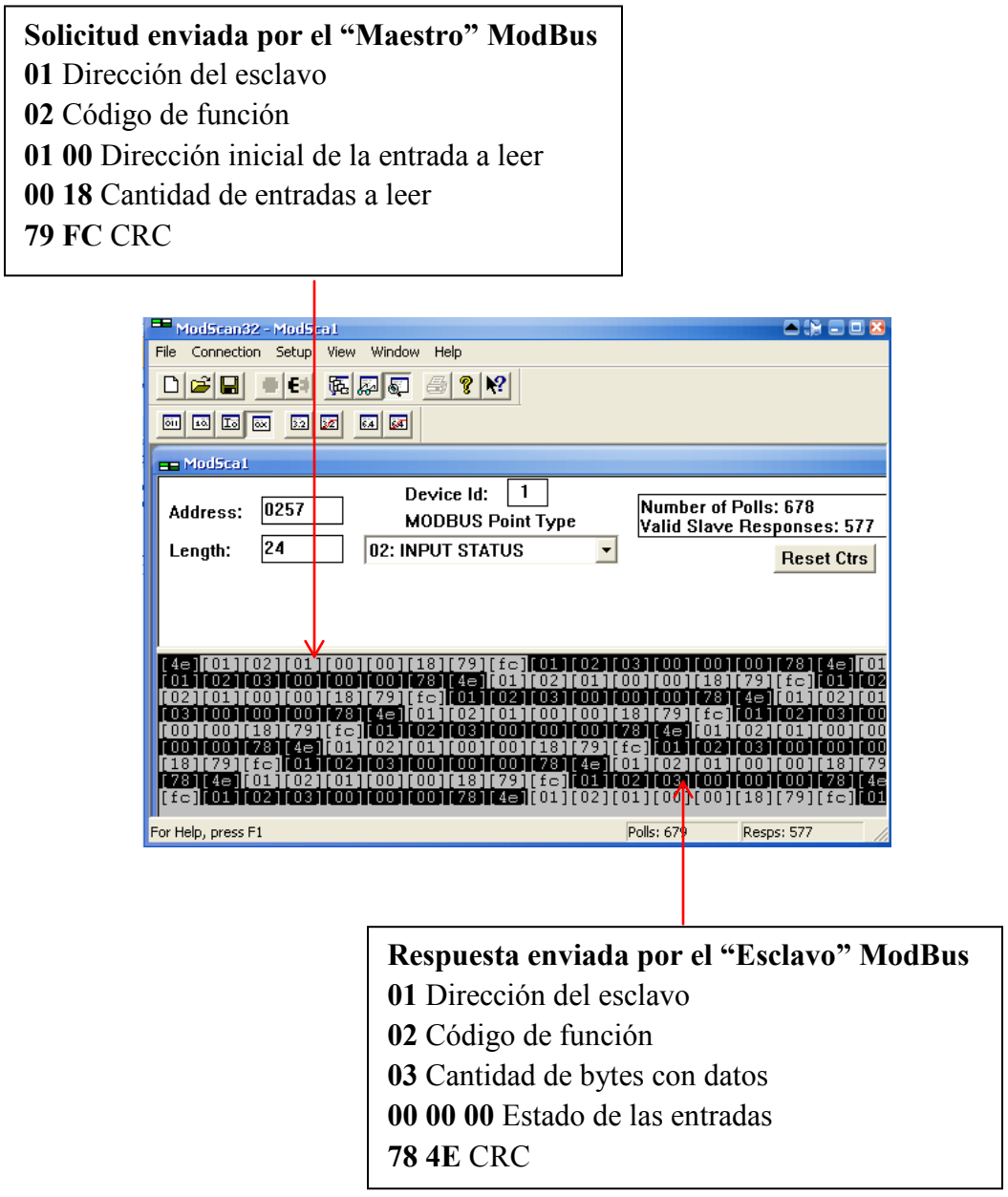


Figura 31: Visualización de tráfico de datos en Modscan32 para la función 02

4.1.3 Comprobación de la Función 04: Leer Registros de Entrada

Esta función puede ser comprobada usando directamente la interfaz principal del programa Modscan32. En la figura 32 se muestra el tráfico generado en el puerto RS-232 del computador, en el cual se aprecia la trama de petición y respuesta correspondiente al protocolo ModBus.

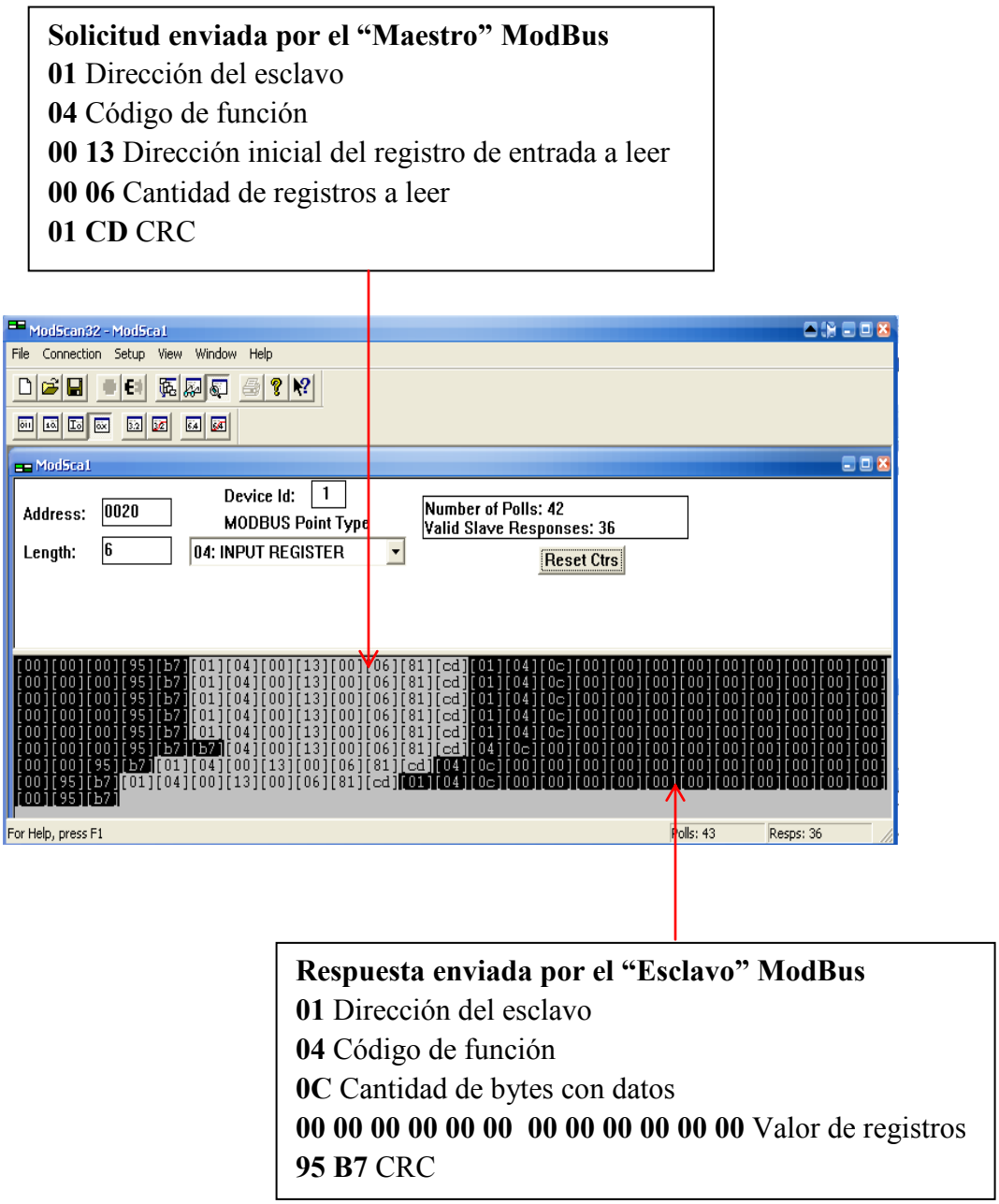
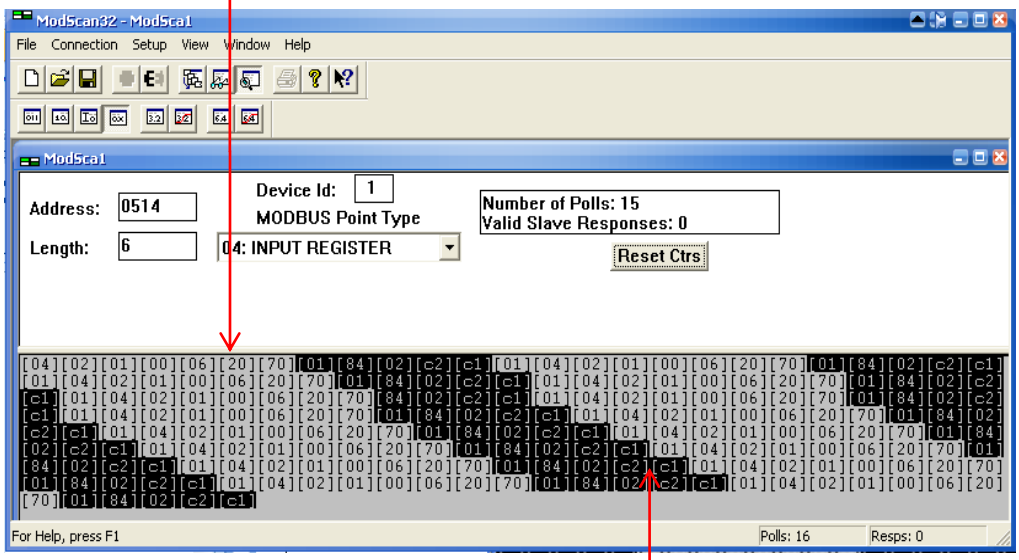


Figura 32: Visualización de tráfico de datos en Modscan32 para la función 04

4.1.4 Comprobación de la Función 04 con Error: Leer Registros de Entrada

Esta función puede ser comprobada usando directamente la interfaz principal del programa Modscan32. En la figura 33 se muestra el tráfico generado en el puerto RS-232 del computador, en el cual se aprecia la trama de petición y respuesta correspondiente al protocolo ModBus.

Solicitud enviada por el "Maestro" ModBus
01 Dirección del esclavo
04 Código de función
02 01 Dirección inicial del registro de entrada a leer
00 06 Cantidad de registros a leer
20 70 CRC



Respuesta enviada por el "Esclavo" ModBus
01 Dirección del esclavo
84 Código de función (04 + 80)
02 Código de error (Error de valores en cuanto a dirección de inicio para lectura y cantidad de registros de entrada a leer).
C2 C1 CRC

Figura 33: Visualización de tráfico de datos en Modscan32 para la función 04 con error

4.1.5 Comprobación de la Función 15: Forzar Múltiples Bobinas

Para verificar el comportamiento del módulo 15, se debe usar las operaciones extendidas del programa Modscan32.

Para utilizar esta función del programa, se selecciona en el menú "*Setup*", "*Extended*" y se elige la opción "*Foce Coils*", la cual utiliza la función 15 para escribir las bobinas de salida.

Para el ejemplo se escoge la posibilidad de modificar las 16 salidas digitales de módulo como se ve en la figura 34.

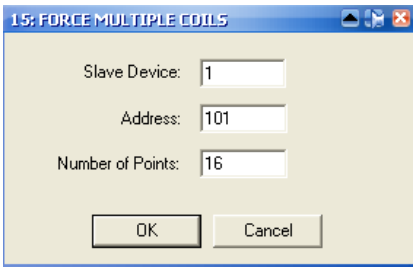


Figura 34: Configuración para usar la función 15 en el programa Modscan32

La figura 35 muestra como establecer los datos a cargar en las bobinas

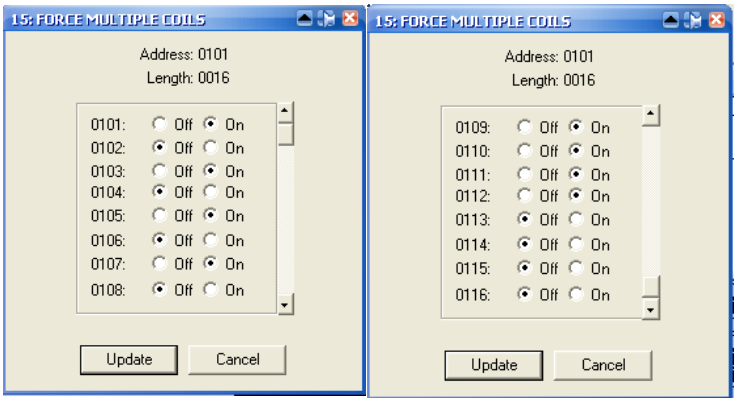
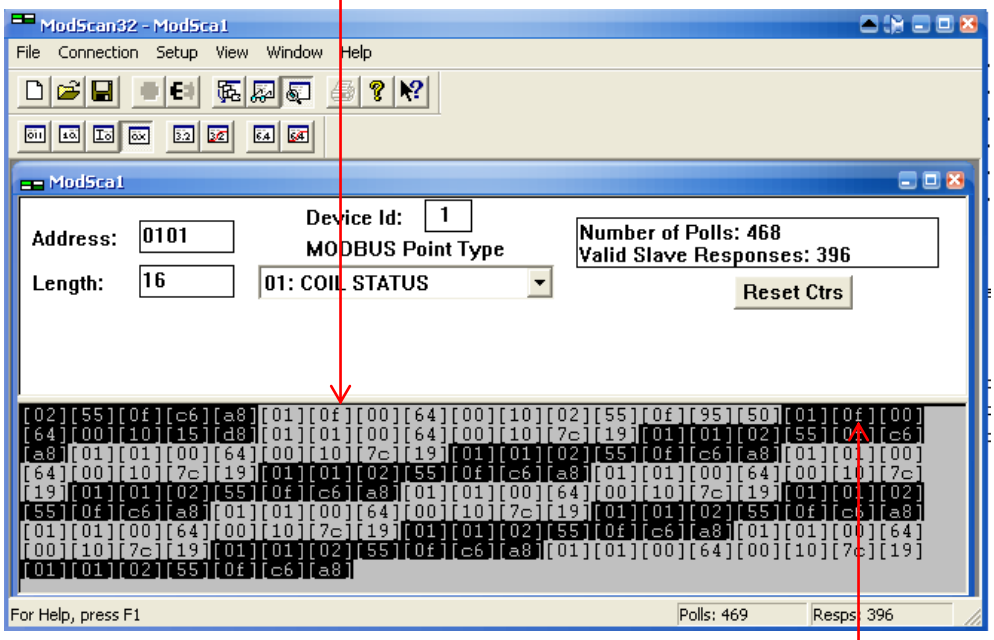


Figura 35: Usando la función 15 en el programa Moscan32

Después de hacer *clic* en "*Update*" se puede ver el tráfico de datos que se ha generado en el puerto de comunicaciones, como se puede ver en la figura 36.

Solicitud enviada por el "Maestro" ModBus
01 Dirección del esclavo
0F Código de función
00 64 Dirección inicial de las bobinas a escribir
00 10 Cantidad de bobinas a escribir
02 Cantidad de bytes que contienen datos
55 0F Datos a escribir en la bobinas
95 50 CRC



Respuesta enviada por el "Esclavo" ModBus
01 Dirección del esclavo
0F Código de función
00 64 Dirección inicial de las bobinas escritas
00 10 Cantidad de bobinas escritas
15 D8 CRC

Figura 36: Visualización de tráfico de datos en Modscan32 para la función 15

4.2 VISUALIZACIÓN DE LA ESTRUCTURA DE LAS TRAMAS MODBUS

4.2.1 Interface RS-232.

Los niveles de voltaje que se usa en el estándar RS-232 se muestran en la tabla 13 y la figura 37:

Tabla 13: Niveles de voltaje para RS-232

Voltaje	Nivel Lógico	Control	Terminología
+3[v] a +25[v]	0	Activo	Espacio
-3[v] a - 25[v]	1	Inactivo	Marca

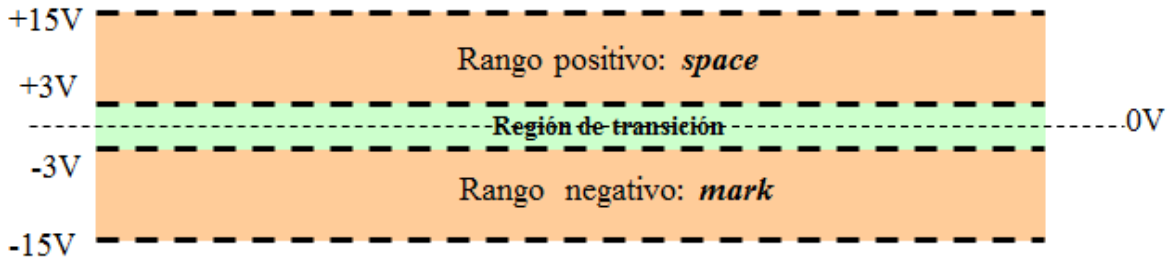


Figura 37: Niveles de voltaje para RS-232 [38]

Se observa que un 1 lógico equivale a un voltaje negativo (-3v a -25 v), y un 0 lógico equivale a un voltaje positivo (+3v a +25v). Ha esta característica se la llama "lógica invertida". Un voltaje que está entre +3[v] y -3[v] es tomado como indeterminado, ver figura 38.

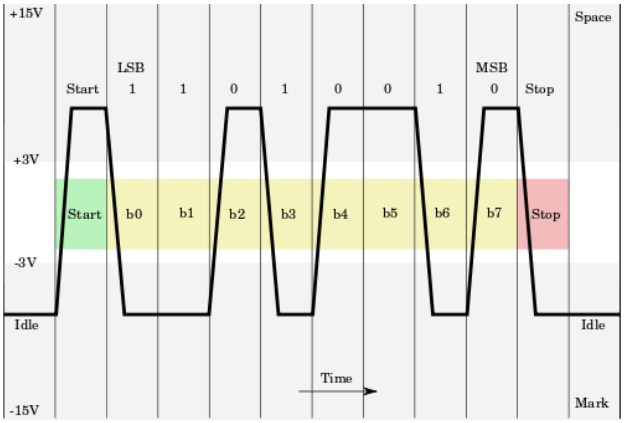


Figura 38: Trama con lógica invertida para RS-232 [39]

4.2.2 Características del Interface RS-232

La velocidad a la que se envían datos en forma serial a través de una línea de comunicación, se denomina velocidad de transmisión en baudios. La velocidad de baudios es expresada en unidades de *bits* (para este caso) por segundo. Una conexión RS-232 ModBus conectado a 9600 baudios tiene la capacidad de enviar 9600 *bits* de datos en 1 segundo.

Si se pueden enviar 9600 *bits* en un segundo, como máximo, el inverso de 9600 da un resultado el tiempo de *bit* (periodo de un *bit*).

$$\frac{1}{(Velocidad\ de\ baudios)} = tiempo\ de\ bits = \frac{1}{9600} = 104\mu s$$

Si un receptor y transmisor se conectan a 9600 baudios, el transmisor enviara *bits* de datos cada 104us, y el receptor tomara lectura de los *bits* de datos cada 104us.

Si la línea de transmisión es inactiva entonces el estado de marca (1 lógico). La transmisión de cada carácter en una línea de comunicación asíncrona va precedida de un *bit* de inicio. El *bit* de inicio es un espacio (0 lógico) con duración igual al tiempo de *bit*. En el receptor, cuando la línea cambia de marca a espacio se indica la presencia de un *bit* de inicio y después se envían los *bits* de datos con un tiempo de *bit* igual a 104us, si la transmisión es a 9600 baudios.

Después de que el último *bit* de datos ha sido enviado, el transmisor pasa al nivel de marca durante un tiempo de *bit*. Este *bit* es llamado *bit* de paro. El *bit* de paro indica que todos los *bits* de datos han sido enviados y la transmisión del carácter se ha completado. Si el receptor detecta un *bit* de inicio y el apropiado número de *bits*, pero no detecta el nivel de marca al final, esto indica un error en la transmisión.

Los *bits* de parada pueden ser uno o dos *bits* de parada (en esto también deben ponerse de acuerdo el transmisor y el receptor). Algunas implementaciones cortan la transmisión del segundo *bit* de parada a la mitad, se dice entonces que utiliza uno y medio *bits* de parada. Los *bits* de parada se transmiten como unos lógicos (*mark*).

4.2.3 Estructura de la Trama ModBus RTU

MODBUS RTU (*Remote Terminal Unit*) se caracteriza por que los *bytes* se envían en su codificación binaria plana, sin ningún tipo de conversión. Está inicialmente pensado para comunicaciones en bus serie. Como ventaja principal tiene el buen aprovechamiento del canal de comunicación, mejorando la velocidad de la transmisión de los datos. El inconveniente es que requiere una gestión de tiempos entre *bytes* recibidos para saber cuando empiezan y terminan las tramas (ver capítulo 5).

Con la trama MODBUS RTU, la delimitación de la misma se realiza por intervalos de tiempo de caracteres de silencio, como muestra la figura 39. Un carácter de silencio tiene la duración de un *byte* de datos enviado por el medio, pero no transporta datos, y su duración (T) depende de la velocidad (Vt) y del número *bits* que se usen para su codificación (N) según la siguiente ecuación:

$$T = \frac{N}{Vt}$$

Según el estándar de MODBUS, para velocidades de hasta 19.200 bps, el tiempo entre tramas debe ser como mínimo 3,5 veces la duración de un carácter, y para velocidades superiores se recomienda un tiempo fijo de 1,75ms.

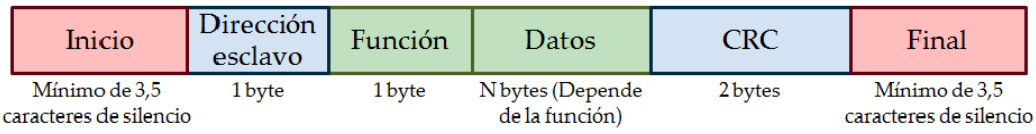


Figura 39: Estructura de la trama en modo RTU [40]

En modo RTU, las tramas son separadas por un intervalo de silencio de al menos el tiempo de 3.5 caracteres, ver figura 40 [41].

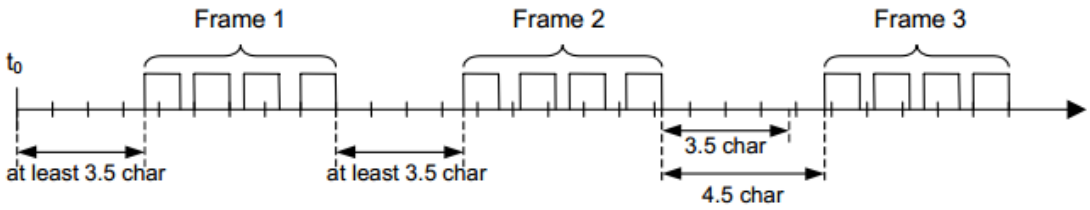


Figura 40: Intervalos de silencio entre tramas en modo RTU [41]

La Trama entera debe ser transmitida como un flujo continuo de caracteres. Si un intervalo de silencio mayor al tiempo de 1.5 caracteres ocurre entre dos caracteres, la trama es declarada incompleta y debe ser descartada por el receptor, ver figura 41 [41].

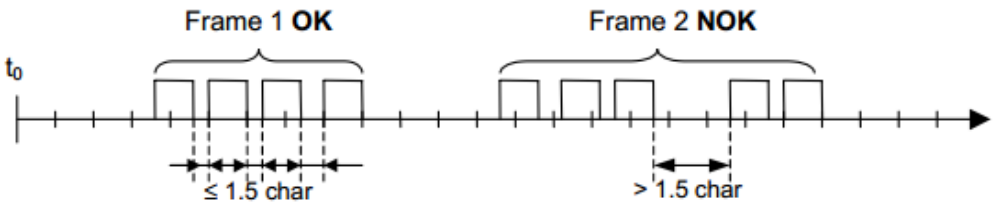


Figura 41: Distancia mínima entre caracteres [41]

La trama MODBUS RTU incorpora un código *Cyclical Redundancy Check* (CRC) de 16 *bits* para poder detectar errores, que debe ser calculado por el emisor a partir de todos los *bytes* de la trama enviados antes del CRC, exceptuando los delimitadores. Para ello se usa un algoritmo específico, bien definido en la especificación de ModBus serie. El receptor debe volver a calcular el código de igual forma que el emisor, y comprobar que el valor obtenido del cálculo es igual al valor presente en la trama para poder validar los datos.

4.2.4 Visualización de los Tiempos de la Trama ModBus RTU

Para visualizar los tiempos de las tramas se procedió a utilizar un osciloscopio digita para obtener las respectivas tramas de envío y respuesta de una comunicación ModBus entre un PC (Maestro) y una central de medida de energía PM 710 (Esclavo) como se indica en la figura 42.

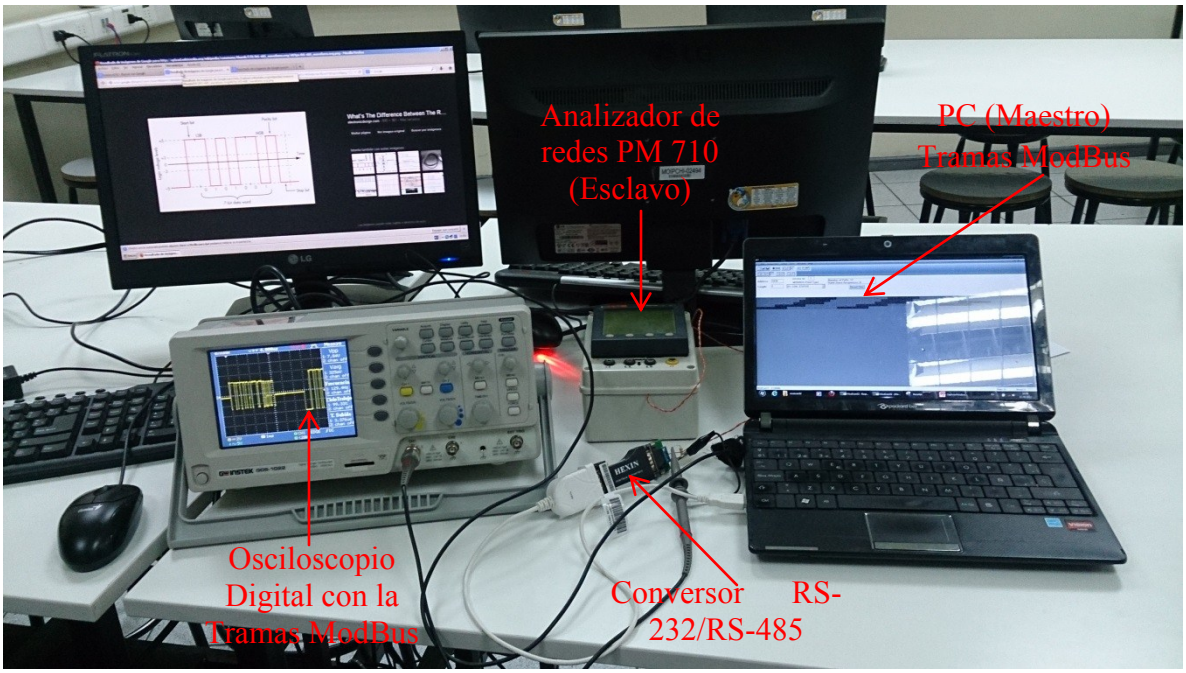


Figura 42 Visualización de tramas de comunicación ModBus

4.2.4.1 Central de Medida de Energía PM 710

El medidor de energía es un instrumento que puede ser aplicado en un sistema trifásico, de tres o cuatro hilos así como también en sistemas monofásicos, además puede trabajar sin modificaciones a frecuencia nominal de 45 a 65 Hz, sobre todo es un medidor que se comunicará vía RS-485 con protocolo ModBus o Jbus con una conexión 2-hilos a una velocidad de hasta 19.2 *kBaudios*, ver figura 43.

Figura 43: Medidor de energía PM 710 [42]

4.2.4.2 Medición y Visualización de las Tramas

Para visualizar las tramas en el osciloscopio se procedió a utilizar el manual del medidor de energía PM 710 [42], donde se obtuvo la información de los registros a ser utilizados en la simulación, donde el PC cumplió las funciones de maestro y el medidor de energía de esclavo, ver figura 44.

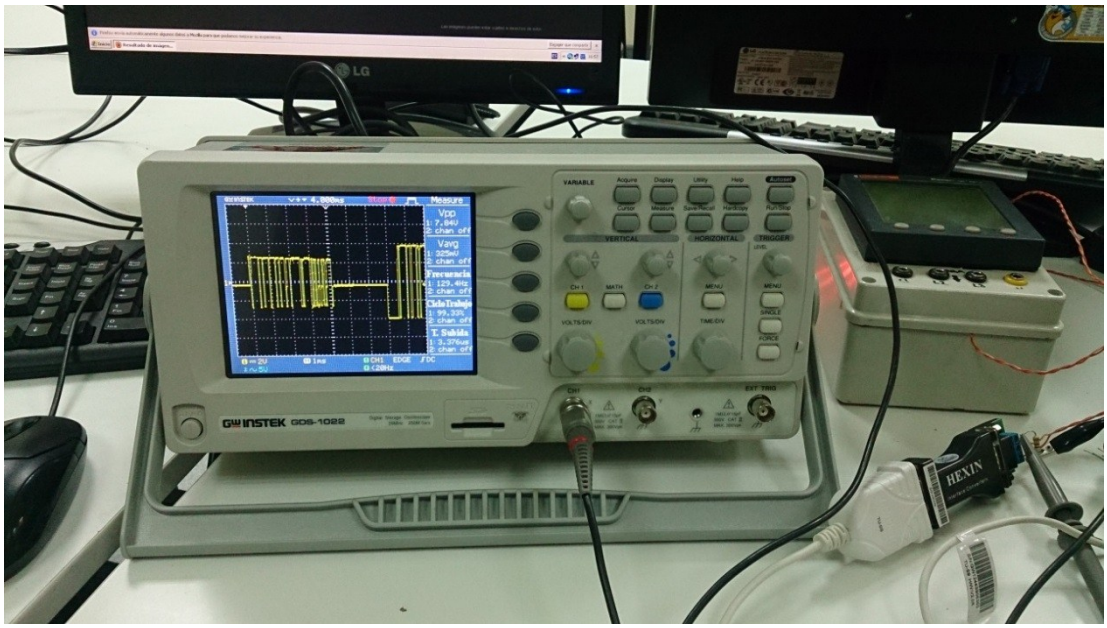


Figura 44: Visualización de las tramas de envío y respuesta

4.2.4.3 Función 01: Leer Estados de Bobinas

La trama de petición es la que se indica en la tabla 14.

Tabla 14: Petición función 01

Nombre del Campo	Hexadecimal
Dirección del esclavo	01
Función	01
Dirección de inicio alto	00
Dirección de inicio bajo	00
Número de registros alto	00
Número de registro bajo	64
Comprobación de errores CRC	3D E1

La trama de respuesta es como se indica en la tabla 15 que tiene un error debido a la dirección de inicio para la lectura y cantidad de salidas digitales a leer.

Tabla 15: Respuesta de la función 01 cuando tiene error

Nombre del campo	Valor en hexadecimal
Dirección de esclavo	01
Código de función	81(01+80h)
Código de excepción: (Error en cuanto a dirección de y cantidad de salidas a leer)	02
Comprobación de error CRC	C1 91

La visualización de la trama ModBus de envío y respuesta se observa en la figura 45 y 46.

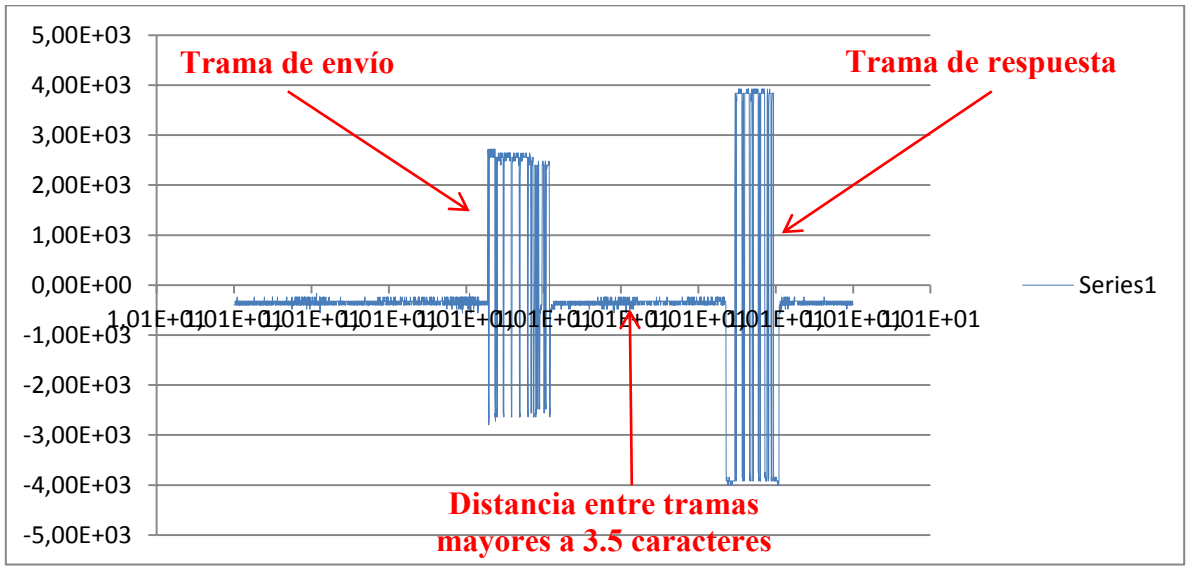


Figura 45: Visualización de las tramas ModBus de envío y respuesta de la función 01

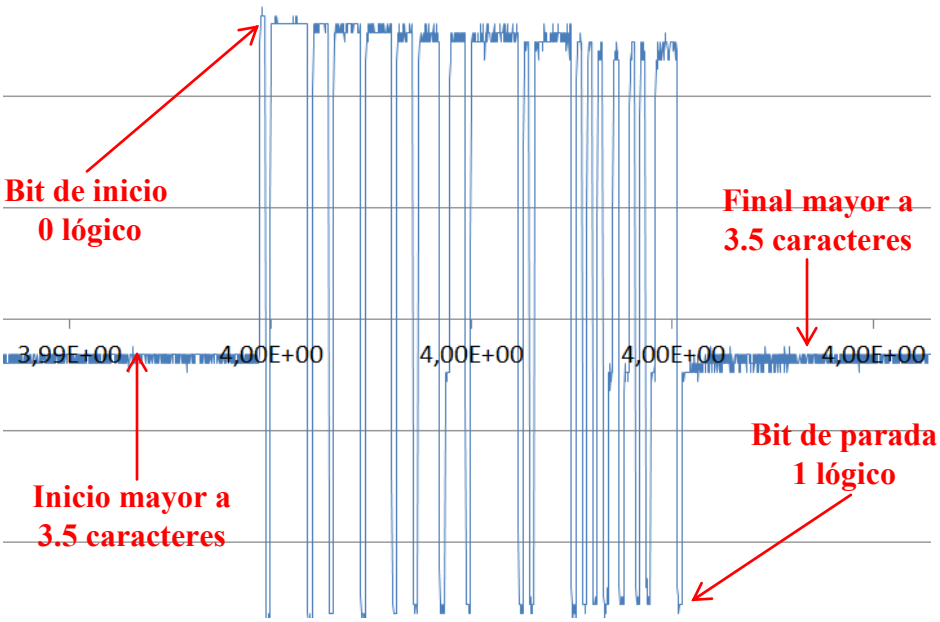


Figura 46: característica de la trama de envío

4.2.4.4 Función 03: Leer Registros Mantenidos

La trama de petición es la que se indica en la tabla 16.

Tabla 16: Petición función 03

Nombre del Campo	Hexadecimal
Dirección del esclavo	01
Función	03
Dirección de inicio alto	04
Dirección de inicio bajo	05
Número de registros alto	00
Número de registro bajo	01
Comprobación de errores CRC	95 3b

La trama de respuesta es como se indica en la tabla 17.

Tabla 17: Respuesta de la función 03

Nombre del campo	Hexadecimal
Dirección del esclavo	01
Función	03
Cuenta en *bytes*	02
Datos altos del Registro	41
Datos bajos del Registro	C7
Comprobación de error CRC	C9 86

La visualización de la trama de envío y respuesta se observa en la figura 47.

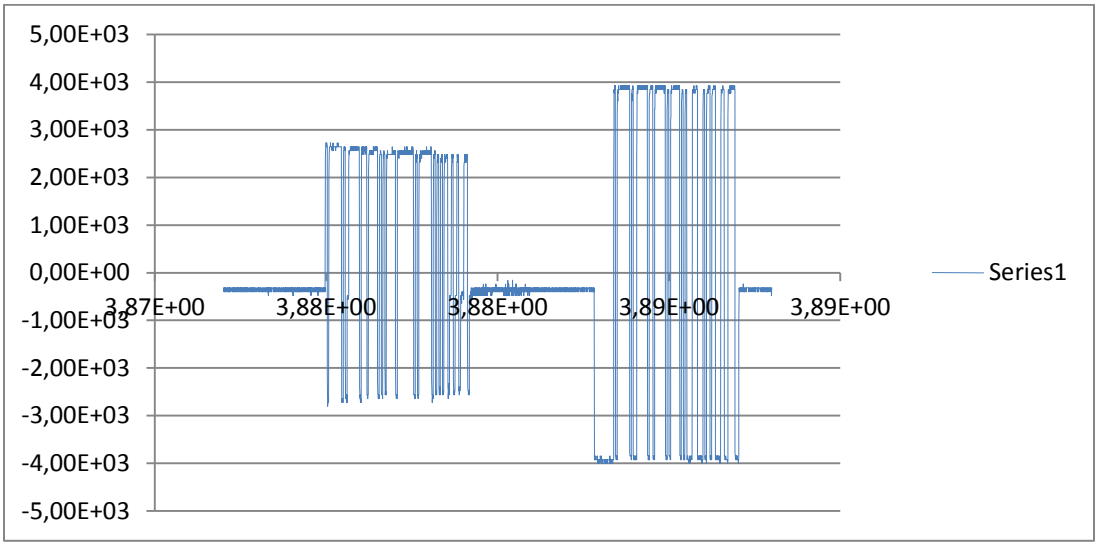


Figura 47: Visualización de las tramas ModBus de envío y respuesta de la función 03

4.2.4.5 Función 03: Leer Registros Mantenidos

La trama de petición es la que se indica en la tabla 18.

Tabla 18: Petición función 03

Nombre del Campo	Hexadecimal
Dirección del esclavo	01
Función	03
Dirección de inicio alto	0F
Dirección de inicio bajo	A8
Número de registros alto	00
Número de registro bajo	03
Comprobación de errores CRC	87 3F

La trama de respuesta es como se indica en la tabla 19.

Tabla 19: Respuesta de la función 03

Nombre del campo	Hexadecimal
Dirección del esclavo	11
Función	03
Cuenta en *bytes*	06
Datos altos (Registro 4008)	00
Datos bajos (Registro 4008)	00
Datos altos (Registro 4009)	00
Datos bajos (Registro 4009)	00
Datos altos (Registro 4010)	00
Datos bajos (Registro 4010)	00
Comprobación de error CRC	21 75

La visualización de la trama de envío y respuesta se observa en la figura 48.

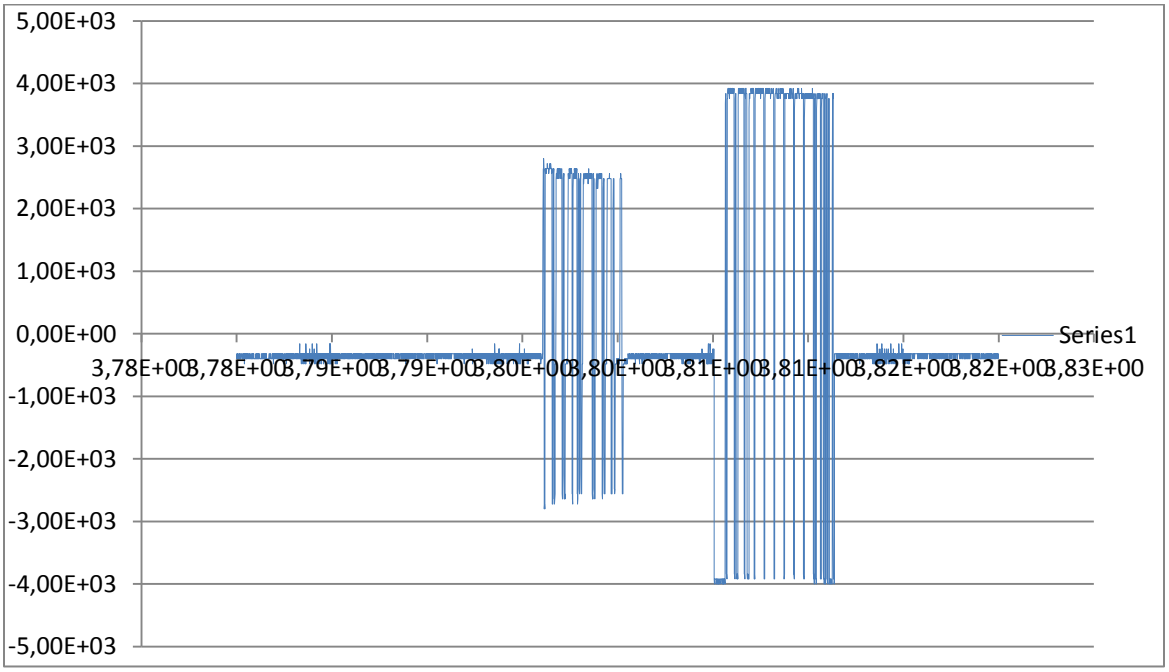


Figura 48: Visualización de las tramas ModBus de envío y respuesta de la función 03

4.2.4.6 Función 04: Lectura de Registros de Entrada

La trama de petición es la que se indica en la tabla 20.

Tabla 20: Petición función 04

Nombre del Campo	Hexadecimal
Dirección del esclavo	11
Función	04
Dirección de inicio alto	10
Dirección de inicio bajo	18
Número de registros alto	00
Número de registro bajo	01
Comprobación de errores CRC	B5 0D

La trama de respuesta es como se indica en la tabla 21.

Tabla 21: Respuesta de la función 04

Nombre del campo	Hexadecimal
Dirección del esclavo	01
Función	04
Cuenta en *bytes*	02
Datos altos (Registro 4120)	00
Datos altos (Registro 4120)	01
Comprobación de error CRC	78 F0

La visualización de la trama de envío y respuesta se observa en la figura 49.

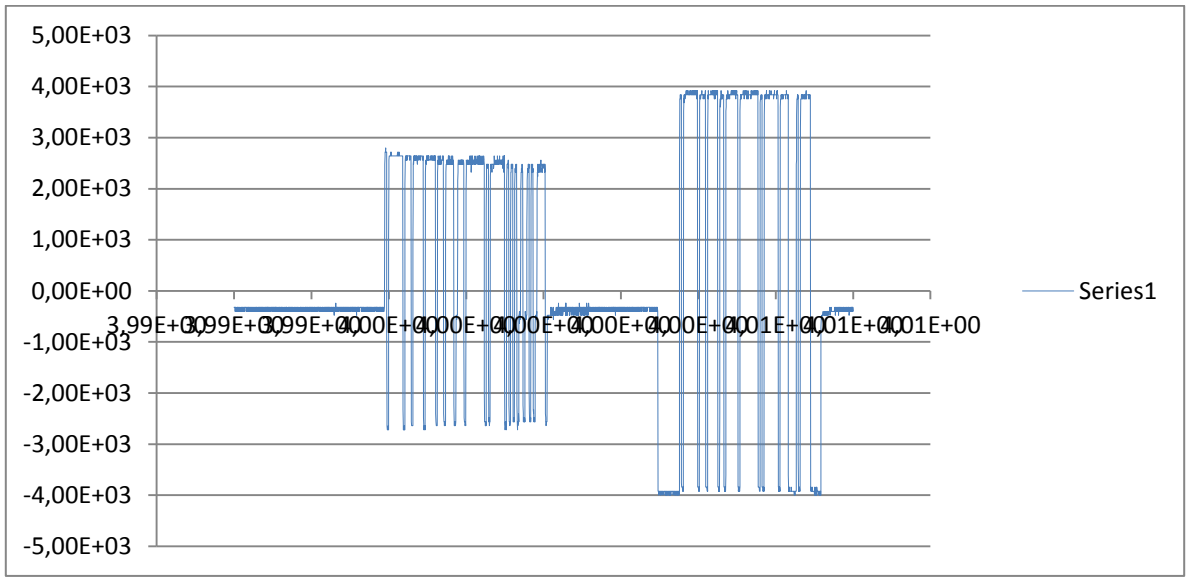


Figura 49: Visualización de las tramas ModBus de envío y respuesta de la función 04

CAPÍTULO 5

TÉCNICAS DE DETECCIÓN DE ERRORES

La detección de errores consiste en monitorear la información recibida y a través de técnicas implementadas en el codificador del canal se determina si se presenta algún problema en la transmisión.

Las técnicas más comunes son:

- Chequeo de Paridad Vertical (VRC).
- Chequeo de Paridad Vertical y Longitudinal (VRC/LRC) o chequeo por Bloques.
- *Checksum*.
- Chequeo de Redundancia Cíclica (CRC).

5.1 CHEQUEO DE PARIDAD VERTICAL.- Este método también es conocido como el método de paridad simple o por caracter es un método capaz de detectar errores en la transmisión de una cadena binaria, pero sin especificar el lugar donde se produce ese error.

Por este motivo, se dice que el método es detector pero no corrector, ya que solo se limita a pedir retransmisión del mensaje si llega distorsionado.

Este método, como todos los que siguen, hace uso del agregado de bits de control.

Se trata de la técnica más simple usada en los sistemas de comunicación digitales (Redes Digitales, Comunicaciones de Datos) y es aplicable a nivel de byte ya que su uso está directamente relacionado con el código ASCII.

Como se recordará, el código ASCII utiliza 7 bits para representar los datos, lo que da lugar a 128 combinaciones distintas. Si definimos un carácter con 8 bits (un byte) quedará un bit libre para control, ese bit se denomina bit de paridad y se puede escoger de dos formas: [43].

- Paridad par
- Paridad impar

La paridad por carácter trabaja agregando redundancia a los datos a transmitir, es decir que añade más información al mensaje que se va a enviar para que el receptor pueda realizar el control y saber si lo que llegó fue correcto o no. En este método tal redundancia es mínima comparada con los otros métodos pero también trae aparejado ciertos inconvenientes. Es un método muy simple, pero presenta algunas desventajas. Una de ellas es que puede fallar ante ciertas circunstancias, como por ejemplo cuando por un error en la transmisión el bit que se ve afectado es precisamente el que realizaba el control, o cuando se modifica de manera uniforme un número par de bits. Otra desventaja, es que en un medio ruidoso, una transmisión correcta podría tardar mucho debido a las sucesivas retransmisiones, o en el peor de los casos, no darse nunca.

Este método es utilizado por otros métodos detectores y/o correctores de errores como *Hamming* o Paridad por Bloque.

Este método añade un bit, denominado bit de paridad que indica si el número de bits de valor 1 en los datos precedentes es par o impar, compensando la cadena que se desea transmitir. Si un solo bit cambiara por error en la transmisión, el mensaje cambiará de paridad y el error se puede detectar [44]. Cuando se habla de paridad par, se refiere a la cantidad par de 1s de una cadena binaria a la cual se indica con un 0. En cambio, cuando se habla de paridad impar, se refiere a la cantidad par de 1s de una cadena binaria la misma que se indica con 1, ver figura 50.

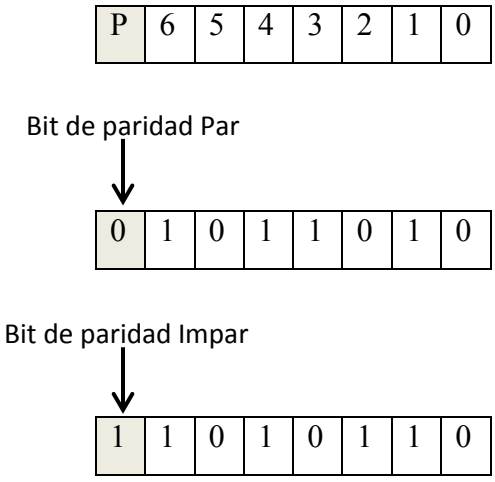


Figura 50: Bit de paridad par e impar

En el extremo de transmisión el codificador de canal calcula el bit de paridad y lo adosa a los 7 bits de datos. El decodificador de canal recibe los 8 bits de datos calcula la paridad y la compara con el criterio utilizado, tal como describe la figura 51.

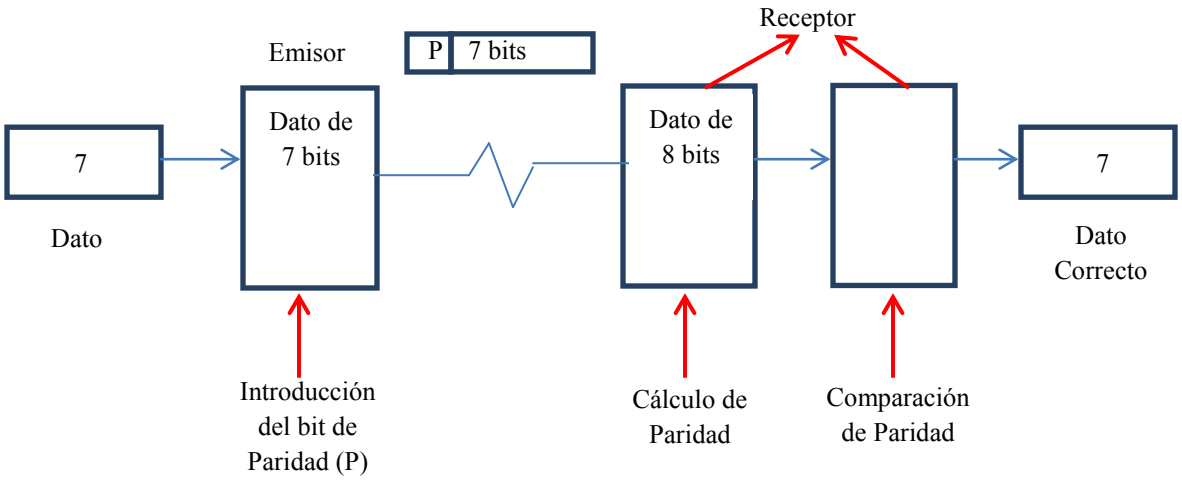


Figura 51: Proceso de codificación y decodificación del bit de paridad

5.2 CONTROL DE PARIDAD MATRICIAL.- Esta técnica de control de errores también se conoce como control de paridad a Bloques o control de paridad tipo vertical y longitudinal (VRC\LRC), éste método es una extensión del método de paridad simple o vertical [43]. Consiste en aplicar el control de paridad en forma horizontal y en forma vertical, es decir por filas y columnas por tal motivo se le conoce como matricial o por bloques, ver figura 52.

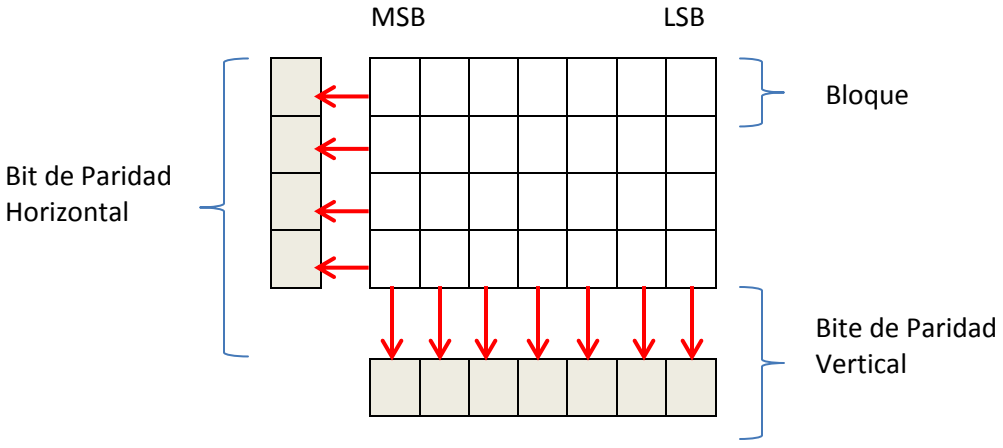


Figura 52: Control de paridad matricial [43]

Para determinar el error por medio de esta técnica se realiza mediante la intersección de filas y columnas.

La utilidad de la técnica a bloques permite determinar con precisión cual es el bit que llegó al receptor con error ya que se compara los bits de paridad tanto de la intersección de la fila como de la columna correspondiente a ese bit.

Cuando el error es de un solo bit permite detectarle con precisión el bit de error por lo tanto corregirlo, ver figura 53, pero puede existir el caso de que exista un error en un número par de bits que ocupa iguales posiciones, en este caso el error no va a ser detectado, ver figura 54.

En los siguientes ejemplos se puede identificar de mejor manera el procedimiento de la técnica de control de paridad matricial, la misma que se trabajará con paridad par.

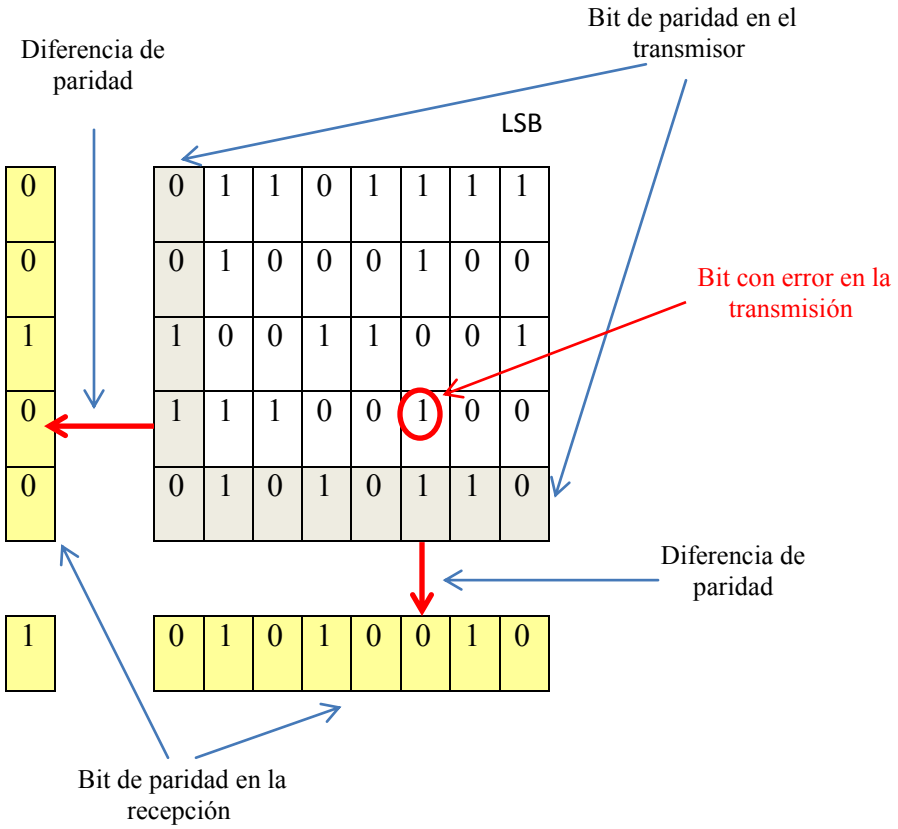


Figura 53: Determinación del error de un bit [43]

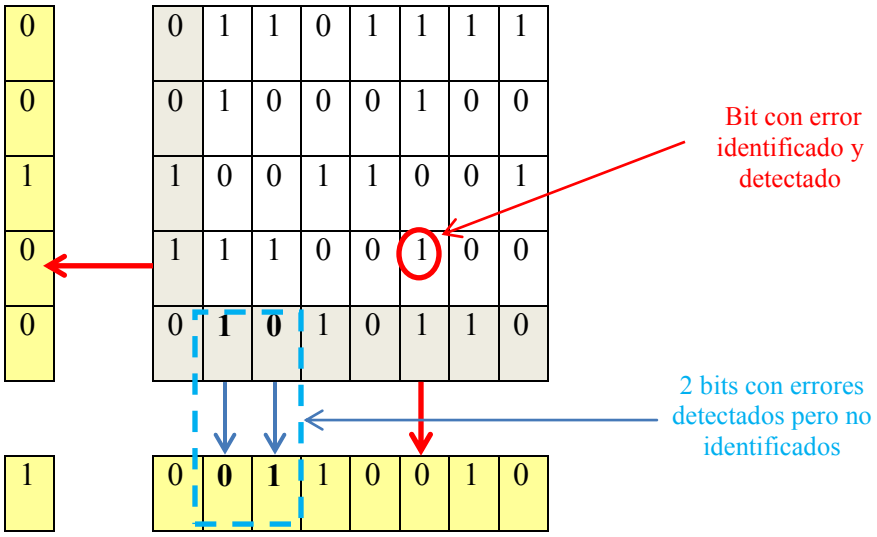


Figura 54: Mensaje con tres errores, uno detectado e identificado y 2 bits detectados pero no identificados [43]

5.3 *CHEKSUM*.- El *cheksum* es otra técnica de detección de errores que se utiliza frecuentemente cuando se transmite o se almacenan bloques de datos. El *cheksum* son bits redundantes obtenidos de la suma de todos los bits del bloque, ver figura 55 [22]. El *cheksum* es un conjunto de bits que se calcula del bloque a ser transmitido, el mismo que debe ser igual al *cheksum* de los bits recibidos, si los dos *cheksums* no coinciden existirá un error en los bits trasmitidos. Existen diferentes técnicas de *cheksum*, de acuerdo a la forma en que se realice el cálculo de los bits.

- *Cheksum* de simple precisión
- *Cheksum* de doble precisión
- *Cheksum Honeywell*
- *Cheksum* de residuo

Es necesario señalar que los *cheksums* sólo pueden detectar errores, pero no localiza ni corrige.

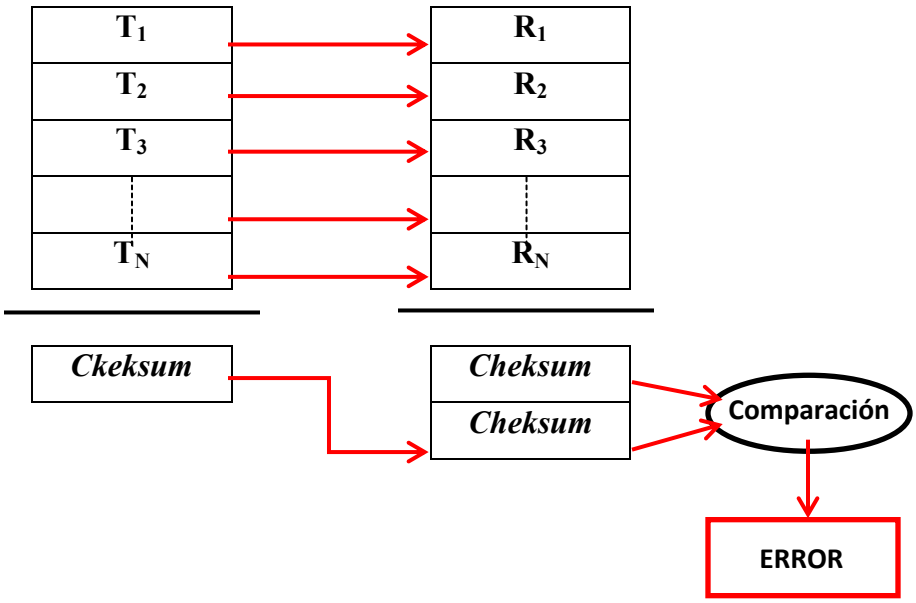


Figura 55: Proceso del *cheksum*

5.3.1 *Cheksum* de Simple Precisión.- Esta técnica consiste en realizar la suma de todos los datos del bloque y se ignoran los bits que superan la longitud del dato, la desventaja del *cheksums* de simple precisión es que no determinan algunos tipos de errores como se puede observar en la figura 56 los *cheksum* transmitidos son iguales a los recibidos a pesar de que existen dos errores esto se da por la capacidad de la longitud del bloque.

5.3.2 Cheksum de Doble Precisión.- Con el *cheksum* de doble precisión se evita el error debido a que el bloque del *ckeksum* tendrá 2N bits, de esta manera es imposible obtener errores siempre que el bloque tenga un tamaño menor a la 2N, caso contrario el error comenzará aumentar, ver figura 57.

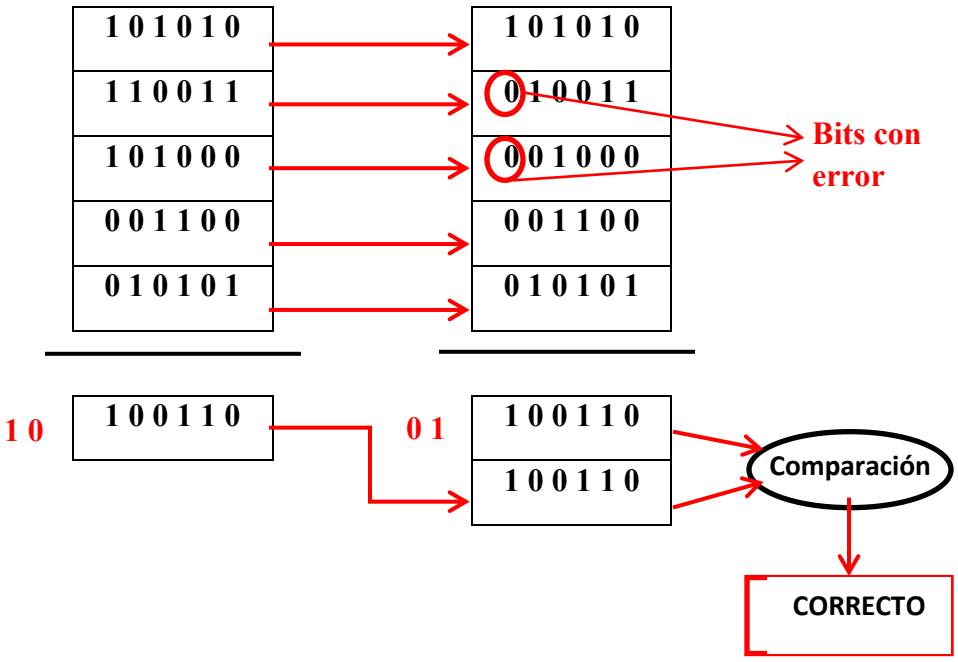


Figura 56: Imposibilidad del *ckeksum* simple para encontrar errores

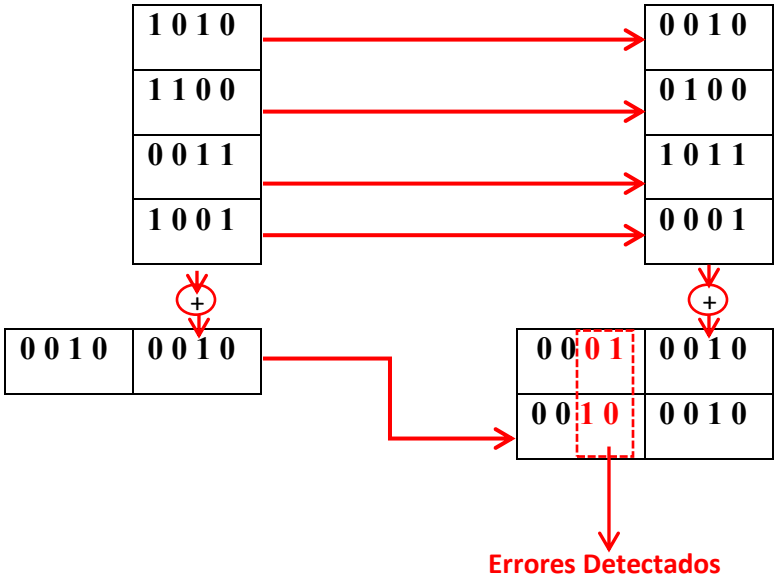


Figura 57: Ckeksum de doble precisión

5.3.3 *Cheksum* de *Honeywell*.- El *Cheksum* de *Honeywell* es una alternativa al *cheksum* de doble precisión. Los datos luego de realizar el *cheksum* tienen una longitud de 2N bits, siendo N el número de bits.

Está técnica concatena pareja de datos consecutivos para formar palabras de doble longitud, luego se calcula el *cheksum* de simple precisión correspondiente a esa palabra de doble longitud, así el *cheksum* también tendrá doble longitud, con este método es muy poco probable que una falla en la comunicación no quede detectado, ver figura 58.

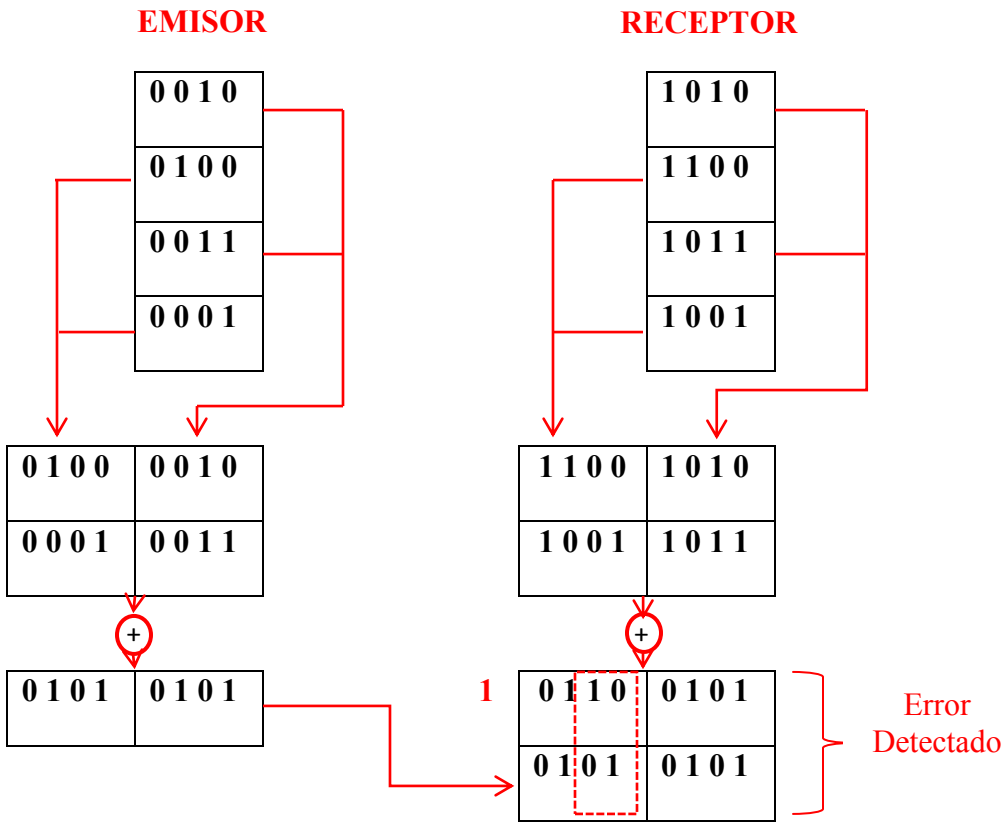


Figura 58: *Ckeksum* de Honeywell

5.3.4 *Cheksum* de Residuo.- Esta técnica es una variación del *cheksum* de simple precisión con la única diferencia que los bits que están fuera de la longitud del bloque se suman con l parte baja figura 59.

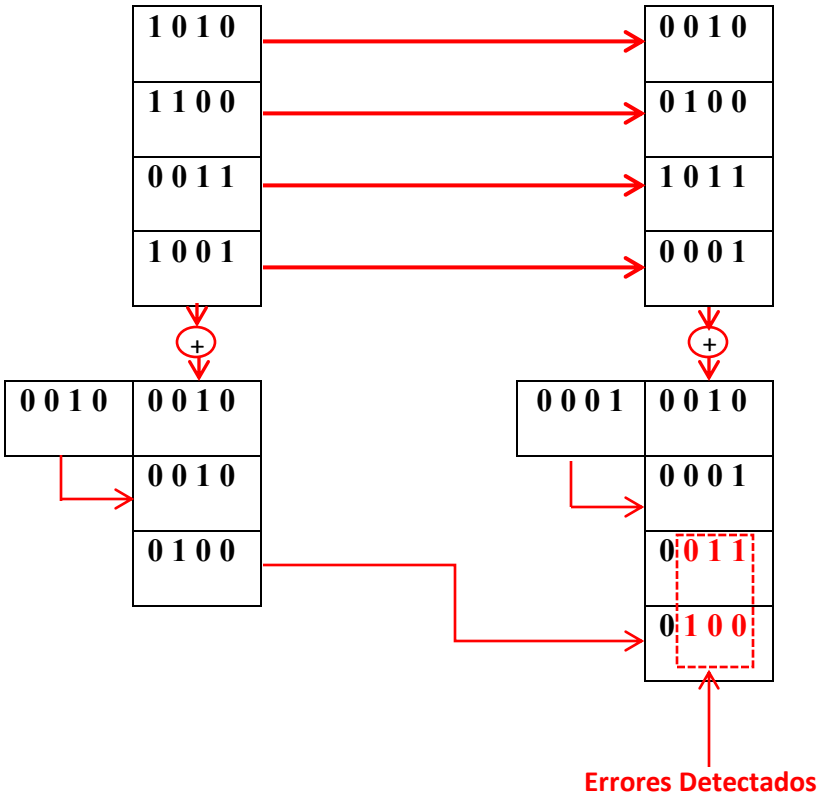


Figura 59: *Ckeksum* de residuo

5.4 CÓDIGO DE REDUNDANCIA CÍCLICA.- Está técnica es utilizada para detectar errores en bloques grandes de datos y sobre todo tiene una confiabilidad alta y una detección de error mayor al 99.9% [25].

Se llaman códigos cíclicos debido a que cumplen la propiedad matemática de que sus elementos rotados son parte del código. La importancia de esto radica en que la implementación de un código cíclico con circuitos es muy simple. Los CRC pertenecen a la familia de los códigos de bloque lineales cuyas principales características es que son fáciles de generar y verificar.

5.4.1 Procedimiento Para el Cálculo del Código de Redundancia Cíclica.- Los códigos de redundancia cíclica se calcula dividiendo el bloque de mensaje para un polinomio generador. El cociente es rechazado y el residuo se añade al final de bloque del mensaje para su transmisión [44].

Para comprobar si han existido errores se realiza el procedimiento inverso, es decir los datos que llegaron al receptor se divide para el mismo polinomio generador calcula

ahora con los datos que han llegado al receptor, los mismos que se dividen para el mismo polinomio generador, el cociente se rechaza y el resto sirve para comprobar si existe error o no en la transmisión, si el resto es cero indica que no ha existido error, caso contrario nos indica que ha existido error en la transmisión.

Los códigos de redundancia cíclica sirven solo para detectar si existen errores, pero no me indica la posición del mismo. Estos códigos son utilizados en los protocolos que utilizan solicitud de repetición automática (ARQ).

El funcionamiento de todos los códigos CRC son idénticos. Para explicar el procedimiento de generación de CRC, se parte de un bloque de mensaje corto y simple y un pequeño polinomio generador CRC que se utilizará. El CRC está compuesto por datos binarios que se ha representado como un polinomio. Cada posición de bit del bloque de mensaje se representa como un coeficiente de un polinomio. La forma general del polinomio es:

$$M(x) = b_n + b_{n-1}X_{n-1} + b_{n-2}X_{n-2} + \ldots . + b_2X_2 + b_1X_1 + b_0 1$$

Donde:

bn = el valor del bloque del mensaje cuyo valor puede ser 0 ó 1.

El grado del polinomio será una unidad menos que el número total de bits del bloque del mensaje.

Si el bloque del mensaje es “101001101”cuenta con 9 bits (k=9). La representación polinómica para este bloque de mensaje tendrá un grado 8. Con el bit LSB a la derecha, el polinomio que representa el bloque del mensaje es:

$$M(x) = 1X^8 + 0X^7 + 1X^6 + 0X^5 + 0X^4 + 1X^3 + 1X^2 + 0X^1 + 1X^0$$

Simplificando El polinomio quedará de la siguiente manera:

$$M(x) = X^8 + X^6 + X^3 + X^2 + 1$$

Para el polinomio generador se usará:

$$G(x) = X^3 + X + 1$$

1011

Puesto que el grado del polinomio generador es 3, el número de bits generado en el CRC será 3. El número total de bits transmitidos por lo tanto será 12 (n=12).

5.4.1.1 Cálculo del Bloque de Transmisión.- El cálculo del CRC de transmisión se obtendrá de la siguiente manera:

1.- Se multiplicará el polinomio del mensaje por X^k, donde k es el exponente más alto del polinomio generador G(x). En este ejemplo, el valor será 3.

$$M(x) * X^k = (X^8 + X^6 + X^3 + X^2 + 1)X^3$$
$$= X^{11} + X^9 + X^6 + X^5 + X^3$$
$$= 1\,0\,1\,0\,0\,1\,1\,0\,1\,0\,0\,0$$

2.- Se divide $M(x) * x^k$ para el polinomio generador G(x).

```
1 0 1 0 0 1 1 0 1 0 0 0  | 1 0 1 1
1 0 1 1                  ----------
-------                  1 0 0 1 0 0 0 0 1
  0 0 1 0
  0 0 0 0
  -------
    0 1 0 1
    0 0 0 0
    -------
      1 0 1 1
      1 0 1 1
      -------
        0 0 0 0
        0 0 0 0
        -------
          0 0 0 1
          0 0 0 0
          -------
            0 0 1 0
            0 0 0 0
            -------
              0 1 0 0
              0 0 0 0
              -------
                1 0 0 0
                1 0 1 1
                -------
                0 1 1   <-- CRC
```

3.- Se suma el código CRC al bloque del mensaje M(x) para crear el mensaje transmitido.

$$CRC = B(x)$$

$$T(x) = [M(x) * X^k] + B(x)$$

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

\+ 0 1 1

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

5.4.1.2 Comprobación del Bloque de Transmitido en el Extremo del Receptor.- Para determinar si el mensaje llegó correctamente se procede de la siguiente manera:

Se divide el mensaje que llegó al receptor para el generador polinomio G(x) y se desecha el cociente. Un residuo igual a cero indica que el mensaje se ha recibido sin error; para este análisis suponemos que llegó el mismo mensaje es decir T(x).

```
1 0 1 0 0 1 1 0 1 0 1 1  | 1 0 1 1
1 0 1 1                  | 1 0 0 1 0 0 0 0
  0 0 1 0
  0 0 0 0
    0 1 0 1
    0 0 0 0
      1 0 1 1
      1 0 1 1
        0 0 0 0
        0 0 0 0
          0 0 0 1
          0 0 0 0
            0 0 1 0
            0 0 0 0
              0 1 0 1
              0 0 0 0
                1 0 1 1
                1 0 1 1
                0 0 0 0   <-- El resto es igual a cero no existe error
```

5.5 ALGORITMO CRC

La generación del CRC, se realiza por medio de una operación booleana OR exclusivo (XOR) a cada carácter de 8 bits con el dato que se encuentra en registro. Entonces al resultado se le aplica un desplazamiento de bit en la dirección de bit menos significativo (LSB), rellenando la posición del bit más significativo (MSB) con un cero. El LSB es extraído y examinado. Si el LSB extraído fuese un 1, se realiza un XOR entre el registro y un valor fijo preestablecido. Si el LSB fuese un 0, no se efectúa la función XOR (OR Exclusiva).

Este proceso se realiza para los 8 bits. Después del último desplazamiento (el octavo), el próximo byte es operado XOR con el valor actual del registro y el proceso se repite con ocho desplazamientos más, y así con todos los bytes del mensaje. El contenido final del registro, después de que todos los bytes del mensaje han sido procesados, es el valor del CRC, ver figura 60.

Cuando el CRC es añadido al mensaje, primero se añade el byte de orden bajo seguido del byte de orden alto.

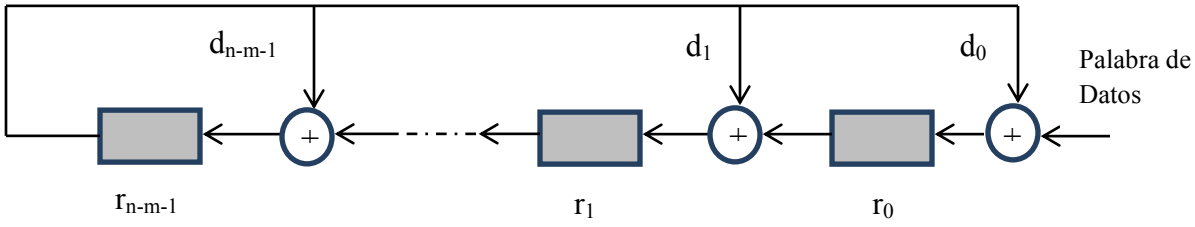


Figura 60: Algoritmo CRC

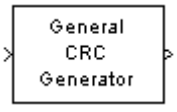
5.6 SIMULACIÓN DEL CÓDIGO DE REDUNDANCIA CÍCLICA (CRC)

Para la simulación se utilizó *Simulink* de *Matlab*.

5.6.1 Librería de CRC: La librería de bloques de CRC de *simulink* contiene cuatro bloques que se pueden utilizar para implementar el algoritmo:

- Generador General CRC
- Detector de Síndrome General
- Generador CRC –N
- Detector de Síndrome CRC-N

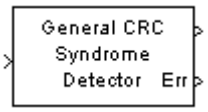
Generador General CRC



Este bloque genera el código de redundancia cíclica (CRC) para cada trama de datos de entrada a la misma que le suma el CRC. Este bloque acepta un vector binario columna

de señal de entrada. Este bloque es general en el sentido de que el grado del polinomio puede ser cualquiera.

Detector de Síndrome General

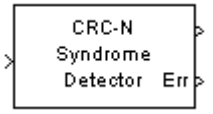


Este bloque recibe una palabra transmitida y calcula su suma de comprobación. El bloque tiene dos salidas. La primera es la palabra de mensaje sin la suma de comprobación transmitida. El segundo producto es un indicador de error binario, que es 0 si la suma de comprobación calculada para la palabra recibida es cero, y 1 en caso contrario.

Generador CRC –N

El bloque generador de CRC-N genera código de redundancia cíclica (CRC) para cada trama de datos de entrada y anexa al bastidor. La entrada debe ser un vector columna binaria. El bloque CRC-N Generador es una versión simplificada del bloque generador general CRC. Con el bloque generador de CRC-N, se puede seleccionar el polinomio generador para el algoritmo CRC de una lista de polinomios de uso común. N es el grado del polinomio generador.

Detector de Síndrome CRC-N



El Bloque Detector de Síndrome CRC-N es una versión simplificada del bloque Detector de Síndrome General CRC. El mismo que puede seleccionar el polinomio generador para el algoritmo CRC de una lista de polinomios de uso común. N es el grado del polinomio generador.

5.6.2 Elementos Utilizados en la Simulación:

Fuente del Bloque de Mensaje

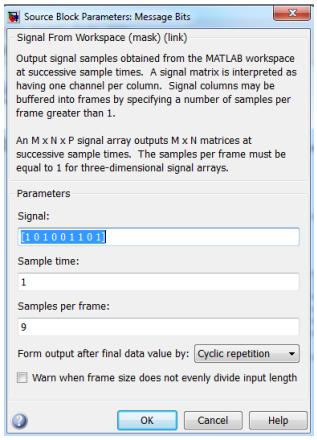


Figura 61: Fuente del bloque de mensajes

Señal: Mensaje a ser transmitido en forma binaria $M(x) = X^8 + X^6 + X^3 + X^2 + 1$.

Tiempo de Muestreo $t(s) = 1$.

Muestra por Trama: Elementos que forman el polinomio a ser transmitido en forma binaria en este caso = 9 figura 61.

Generador General CRC

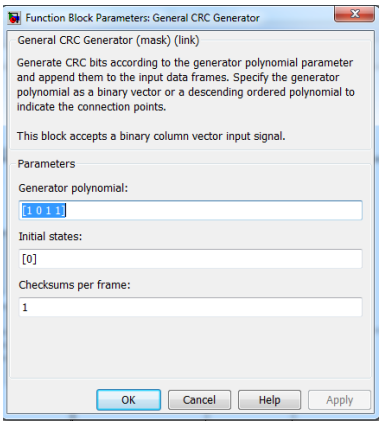


Figura 62: Generador general CRC

Polinomio Generador: En este recuadro se coloca los valores binarios en forma de fila que especifica el polinomio generador en forma de polinomio descendente figura 62.

Estados Inicial: En este cuadro se coloca un vector fila de longitud igual al grado del polinomio, que especifica el estado inicial del registro de desplazamiento.

Suma de Comprobación por Tramas: Número entero positivo que especifica el número de sumas de control. El bloque calcula para cada trama de entrada.

Selector:

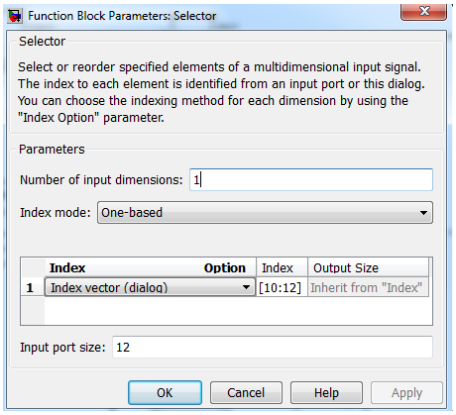


Figura 63: Selector

Dimensión del Número de Ingreso: Número de filas que ingresan figura 63.

Modo de Indexación: Indica el orden como ingresa el mensaje: basado en uno indica que 1 será el primer elemento del vector, 2 el segundo elemento y así sucesivamente, en caso de ser basado en cero, 0 será el primer elemento de vector, 1 el segundo elemento y así sucesivamente.

Opinión de Indexado: Valores que se desea visualizar del total de la trama por ejemplo [12:10] me dejará solamente visualizar los 3 últimos bits del total.

Tamaño del Puerto de Entrada: especifica el ancho del bloque de la señal de entrada en este caso es 12 que es el número de bits que compone el polinomio resultante de la multiplicación de $M(x) * X^k$.

Display

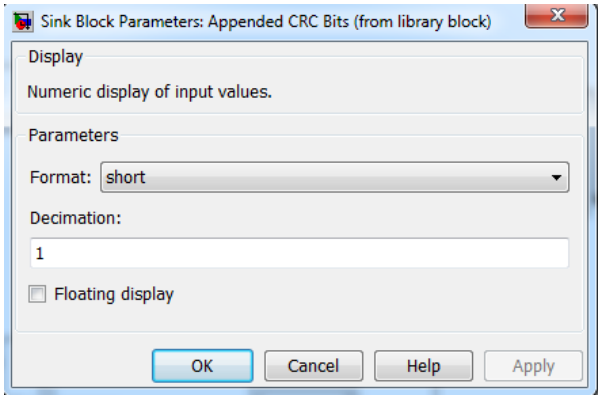


Figura 64: *Display*

Formato: Es el formato como desea que se visualice en el *display*; el valor predeterminado es corto figura 64.

Ejecución: Indica la frecuencia para mostrar los datos; el valor predeterminado es 1

5.6.3 Simulación del Código de Redundancia Cíclica CRC

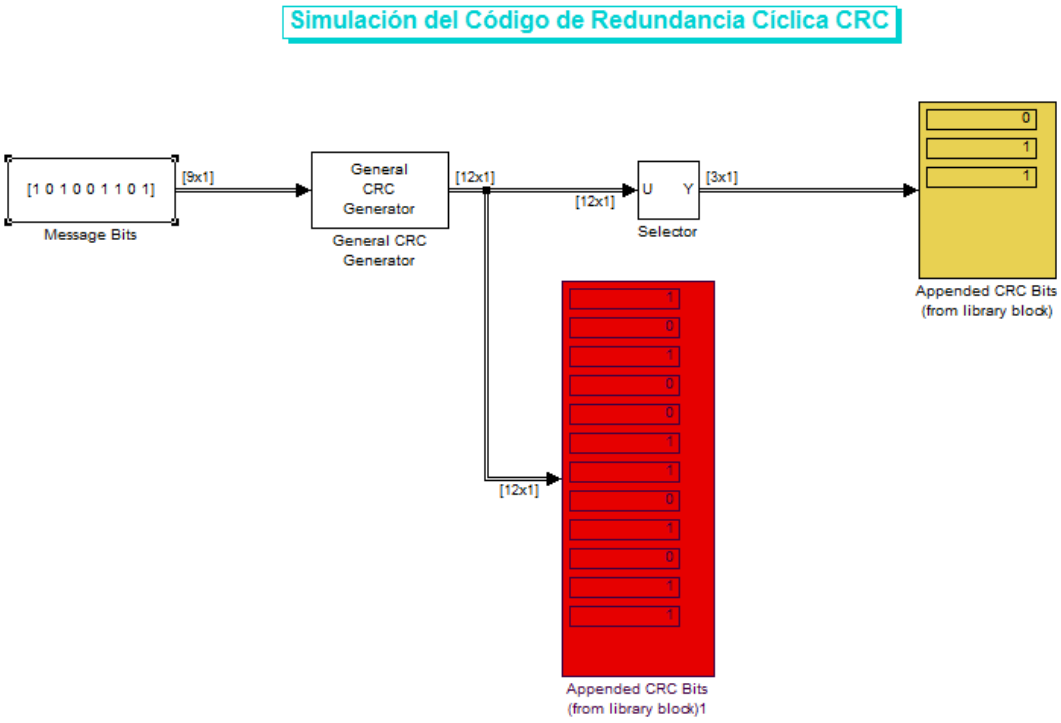


Figura 65: Simulación del código de redundancia cíclica CRC

Display Amarillo = Código Redundante figura 65.

Display Rojo = Código transmitido más el código redundante.

5.6.4 Simulación de Transmitido en el Extremo del Receptor (Sin Error).

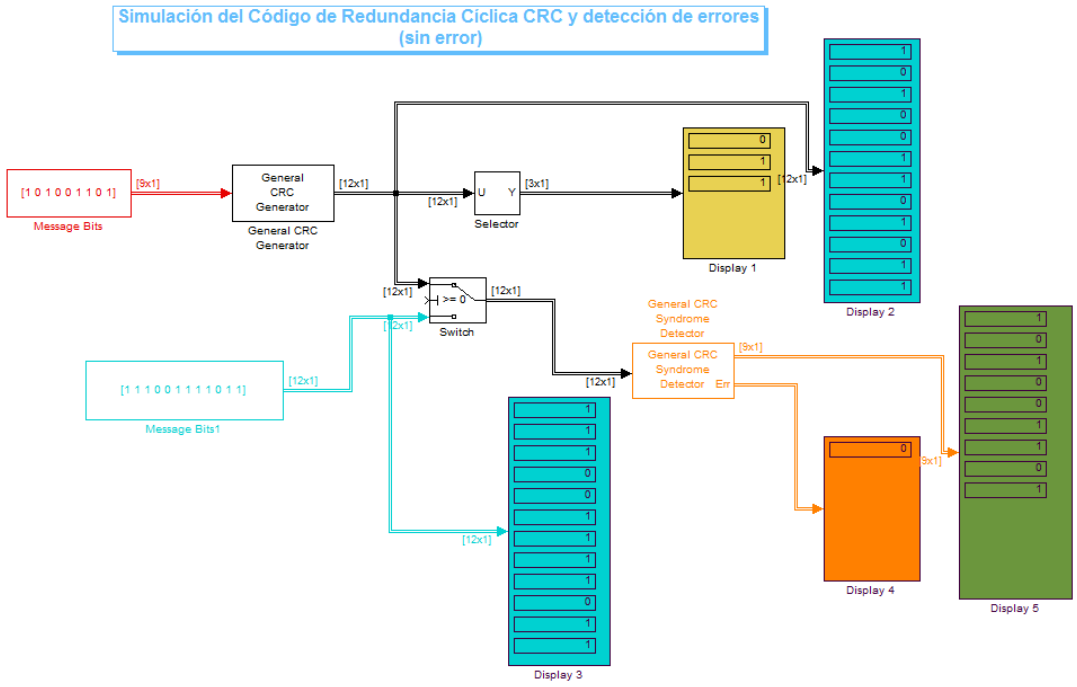


Figura 66: Simulación del código de redundancia cíclica CRC y detección de errores (Sin error)

Display amarillo 1 = Código redundante de la trama que llega al receptor figura 66.

Display azul 2 = Trama de bit llega al recepto más el código redundante.

Display azul 3 = Código que llega al receptor con error.

Display tomate = error.

Display verde = Código que llega al transmisor.

5.6.5 Simulación de Transmitido en el Extremo del Receptor (Con Error)

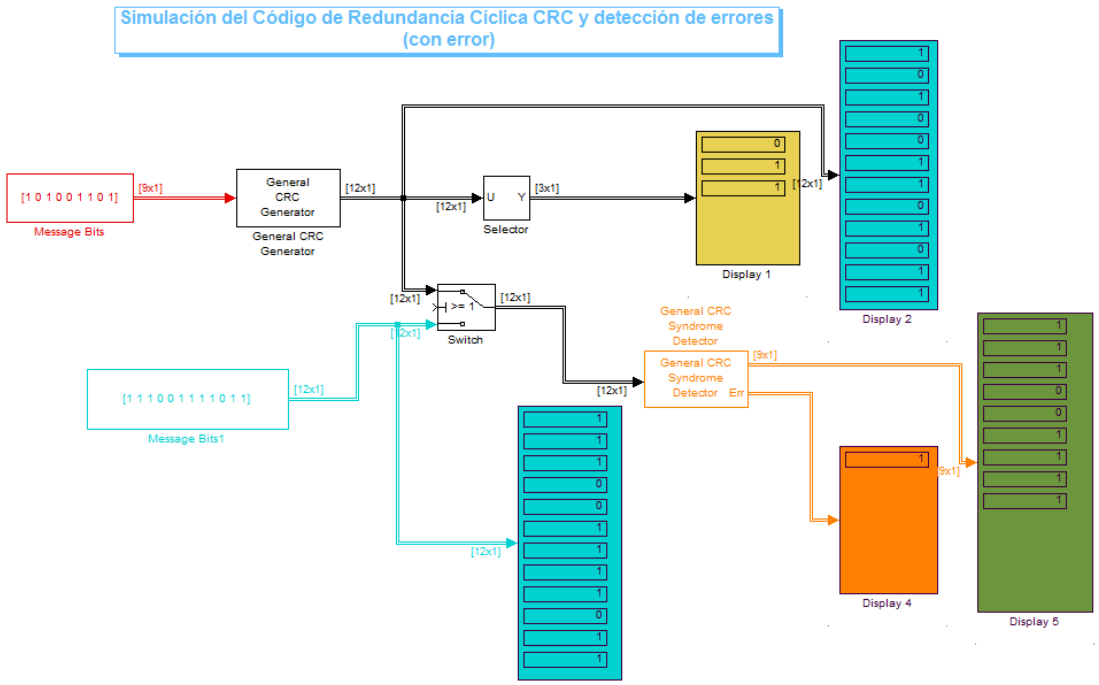


Figura 67: Simulación del código de redundancia cíclica CRC y detección de errores (Con error)

Display amarillo 1 = Código redundante de la trama que llega al receptor figura 67.

Display azul 2 = Trama de bit llega al recepto más el código redundante.

Display azul 3 = Código que llega al receptor con error.

Display tomate = error.

Display verde = Código que llega al transmisor.

5.6.6 Simulación de Detección de Error Utilizando la Técnica de *Cheksum*

5.6.6.1 Algoritmo de Funcionamiento para 2 *Cheksums*: En la figura 68 se indica el algoritmo para detectar el error por medio de 2 *cheksums.*

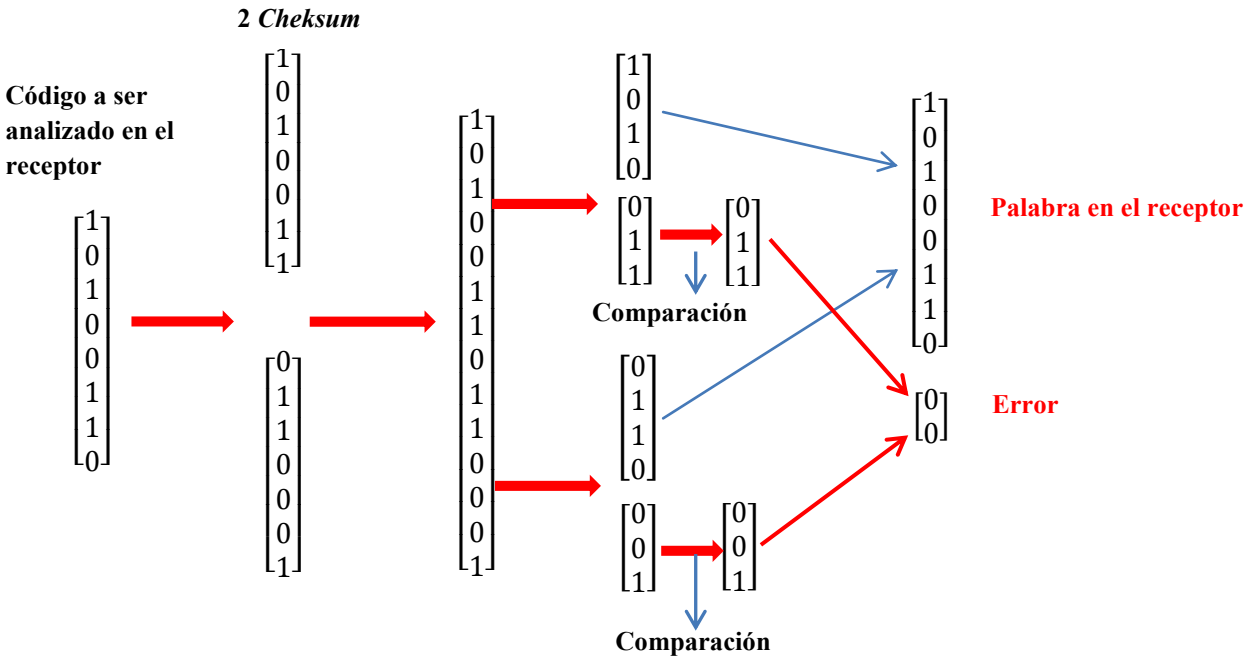


Figura 68: Algoritmo de funcionamiento para 2 *cheksums*

5.6.6.2 Simulación de la Trama en el Receptor sin Bits de Error

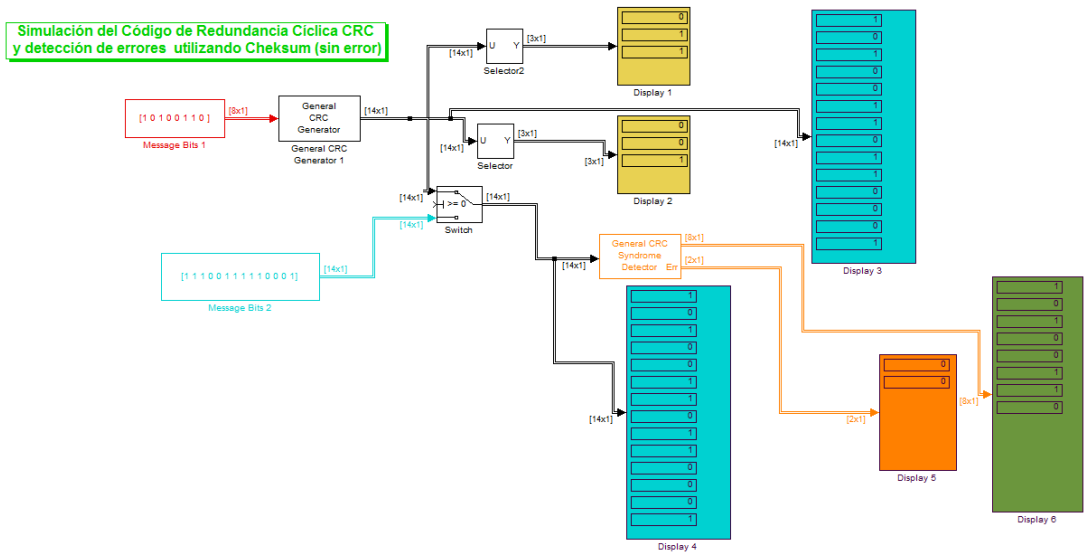


Figura 69: Simulación y detección de errores en el receptor utilizando 2 *cheksums* (Sin errores)

Display Amarillo1 = Código redundante del primer *ckehsum* de los 4 bits más significativos de la primera trama de datos figura 69).

Display Amarillo 2 = Código redundante del segundo *cheksum* de los 4 bits menos significativos de la primera trama de datos.

Display Azul 3 = Primera trama de datos más códigos redundantes que llegan en el receptor.

Display Azul 4 = Segunda trama de datos con error más código redundante a ser trasmitido.

Display Naranja 5 = Indicador de error producido en cada *ckeksum*.

Display Verde 6 = Bits de llegada en el receptor.

5.6.6.3 Simulación de la Trama en el Receptor con Bits de Error

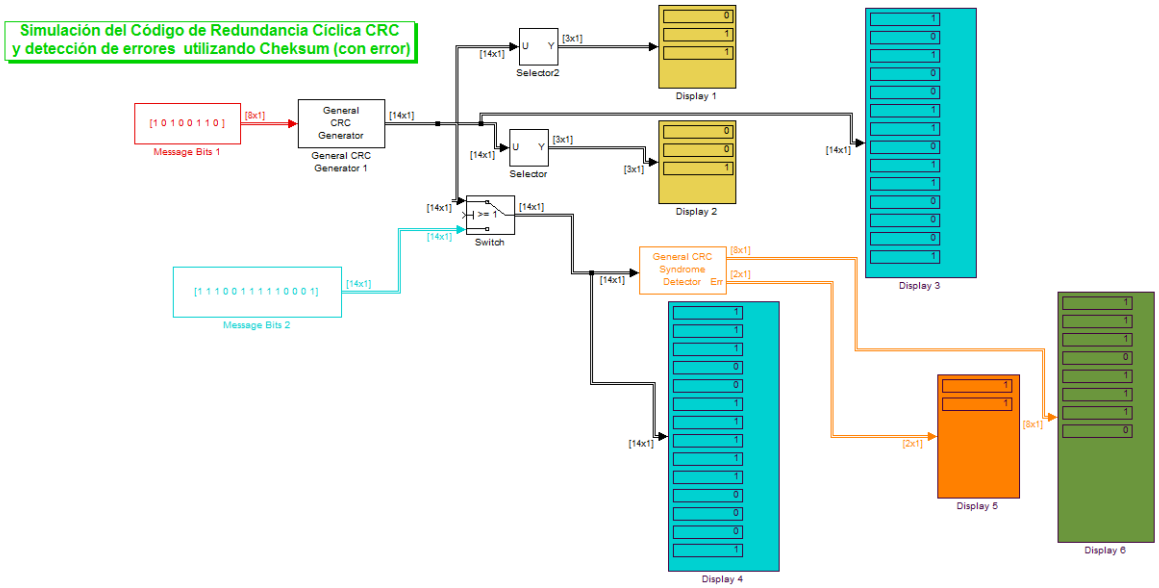


Figura 70: Simulación y detección de errores en el receptor utilizando 2 cheksums (Con errores)

Display Amarillo1 = Código redundante del primer *ckehsum* de los 4 bits más significativos de la primera trama de datos figura 70.

Display Amarillo 2 = Código redundante del segundo *cheksum* de los 4 bits menos significativos de la primera trama de datos.

Display Azul 3 = Primera trama de datos más códigos redundantes que llegan en el receptor.

Display Azul 4 = Segunda trama de datos con error más código redundante a ser trasmitido.

Display Naranja 5 = Indicador de error producido en cada *ckeksum*. En este caso existe. error en las dos tramas de 4 bits.

Display Verde 6 = Bits de llegada en el receptor.

CAPÍTULO 6

TÉCNICAS DE CORRECCIÓN DE ERRORES BACKWARDS ERROR CORRECTION

El *Backwards Error Correction*.- Conocida como "solicitud de repetición automática" es una técnica de corrección de error, donde el receptor envía una solicitud al dispositivo fuente para volver a enviar la información, dentro de esta técnica se encuentra los Requerimiento automático de repetición: *Automatic Request for Repeat* [31].

6.1.- *AUTOMATIC REPEAT-REQUEST*.- En español Solicitud Automática de Reenvío, es un protocolo de control de errores, que automáticamente realiza una solicitud de reenvío cuando existe un paquete de datos erróneos o incorrectos [45]. Cuando el dispositivo transmisor no recibe una señal de confirmación para confirmar que los datos han sido recibidos, por lo general transmite los datos después de un tiempo de espera predefinido y repite el proceso un número predeterminado de veces hasta que el dispositivo de transmisión recibe una respuesta. Existen tres formas de ARQ [46].

- *Stop and Wait* ARQ (Pare y Espere).
- *Go-Back-N.*
- *Selective Repeat* ARQ (Envío Continuo).

La operación de los métodos de ARQ se basa en mensajes de reconocimiento. Para controlar la correcta recepción de un paquete de datos se utiliza ACK (*Acknowledgment)* y (*Nonacknowledgment*) NACK, de forma que cuando el receptor recibe un paquete correctamente el receptor envía un ACK y si no es correcto responde con un NACK. Durante el protocolo que controla la recepción de paquetes puede surgir múltiples problemas (pérdida de ACK, recibir un incorrecto, etc.).

6.2 PARE Y ESPERE

Esta forma de transmisión consiste en que el transmisor deja de transmitir después de una trama completa que ha sido enviado y espera una respuesta desde el receptor para confirmar la correcta recepción de la trama. Este método ARQ puede ser descrito de la siguiente manera [47]:

1. El equipo de origen comienza a transmitir una nueva trama.
2. Después de transmitir la trama, un contador de tiempo que expira después del tiempo de espera de la llegada del ACK se inicia (si la máquina transmisora espera una respuesta ACK después de x segundos, un temporizador establecido con t segundos se activa).
3. Una vez que la trama llega a la máquina de destino, la máquina de destino responde con un mensaje ACK que indica la recepción de una trama sin errores.
4. Una vez que el mensaje de ACK es recibido por el equipo de origen, Etapa 1 se repite el proceso.

6.2.1 Análisis

Hay varias cuestiones que se debe considerar para el método Pare y Espere ARQ. Estos incluyen la posibilidad de tramas con retraso o pérdida, y la posibilidad de mensajes ACK con retraso o pérdida. Se considera por ejemplo el enlace de comunicación, donde tenemos la máquina A la misma que desea enviar una serie de tramas a la máquina B. Y se supone que los datos se transmiten de la máquina A para la máquina B sólo en esta dirección y tramas ACK en la otra dirección, y se asume que el algoritmo de detección de errores utilizado es suficientemente fuerte como para detectar todos los errores.

A) Información de la trama que generalmente son varios miles de bits de longitud y que contiene los siguientes componentes:

1. Cabecera que contiene información de direccionamiento y otros.
2. Paquete de datos.
3. Bits CRC para detección de errores (calculados sobre la base del Encabezado y del Paquete de Información).

B) Control de Tramas (trama ACK), que contiene los siguientes componentes:

1. Cabecera que contiene información de direccionamiento y otros.
2. Bits CRC calculados sobre la base de la cabecera. Un CRC en una trama de control es importante debido a que es donde se toma la decisión si los datos fueros recibidos correctamente o tienen algún error.

En base a estas características para su análisis se considerará los siguientes escenarios.

a) Trama Perdida

1. La máquina A transmite la trama 0 e inicia el temporizador 0, que expira después del tiempo previsto para la llegada del ACK.
2. La trama 0 llega a la máquina B dentro del tiempo 0 que fue establecido, (se asume que la trama es analizada si llegó correctamente o si tiene errores).
3. La máquina B transmite el ACK.
4. La trama ACK es recibida por la máquina en el tiempo establecido (antes del tiempo t0).
5. La máquina A transmite la trama 1 e inicia el temporizador t1 que expira después del tiempo previsto para la llegada del ACK.
6. La trama 1 llega a la máquina B dentro del tiempo 1 que fue establecida, (se asume que la trama es analizada si llegó correctamente o si tiene errores). La máquina B transmite el ACK.
7. Se supone que el ACK se pierde o se recibe con errores.
8. El tiempo t1 expira y la máquina A decide retransmitir la trama 1.
9. Esta vez la máquina B recibe correctamente la trama 1 y en vía el ACK dentro del tiempo establecido t1.
10. La máquina A recibe el ACK antes que el temporizador t1 expire y decide transmitir la trama 2, y así sucesivamente, ver figura 71.

Conclusión: La transmisión recupera la trama 1 perdida.

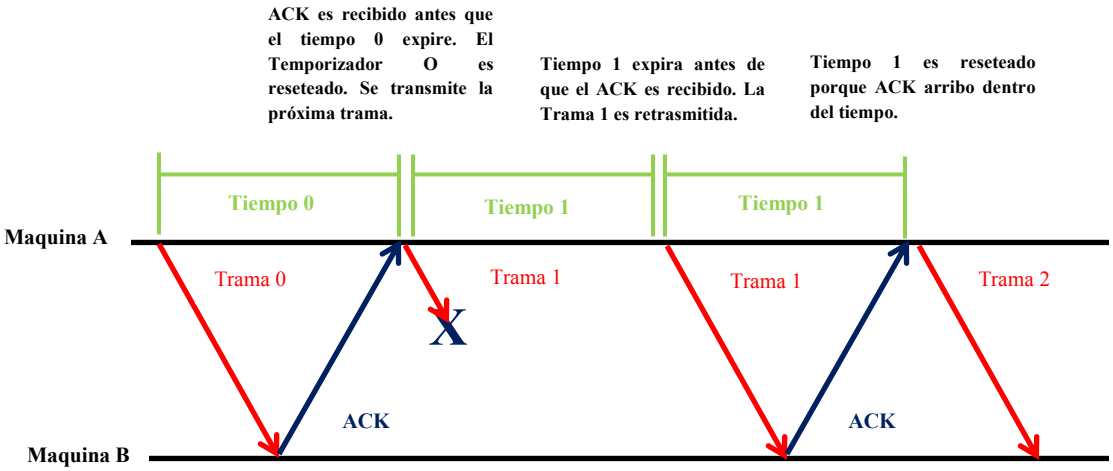


Figura 71: Trama perdida

b) Pérdida de ACK

Se considera el siguiente escenario:

1. La máquina A transmite la trama 0 e inicia el temporizador 0 que expira después del tiempo previsto para la llegada del ACK.
2. La trama 0 llega a la máquina B dentro del tiempo 0 que fue establecido, (se asume que la trama es analizada si llegó correctamente o si tiene errores).
3. La máquina B transmite el ACK.
4. La trama ACK es recibida por la máquina en el tiempo establecido (antes del tiempo t0).
5. La máquina A transmite la trama 1 e inicia el temporizador t1 que expira después del tiempo previsto para la llegada del ACK.
6. Se supone que la trama 1 se pierde o se recibe con errores.
7. La máquina B no envía el ACK debido a que no recibió la trama 1 correctamente.
8. El tiempo t1 expira y la máquina A decide retransmitir la trama 1. Se supone que el ACK se pierde o se recibe con errores.
9. Esta vez la máquina B recibe la trama 1 correctamente y envía el ACK.
10. La máquina A recibe el ACK antes que el temporizador t1 expire y decide transmitir la trama 2, y así sucesivamente, ver figura 72.

Conclusión: La transmisión recupera la trama 1 perdida.

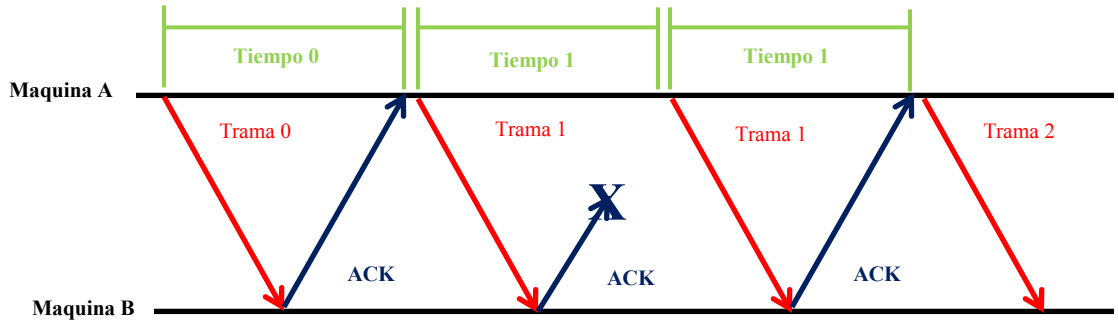


Figura 72: Pérdida de ACK

c) ACK Fuera del Tiempo Establecido

Se considera el siguiente escenario:

1. La máquina A transmite la trama 0 e inicia el temporizador 0 que expira después del tiempo previsto para la llegada del ACK.
2. La trama 0 llega a la máquina B dentro del tiempo 0 que fue establecido, (se asume que la trama es analizada si llegó correctamente o si tiene errores).
3. La máquina B transmite el ACK.
4. La trama ACK es recibida por la máquina en el tiempo establecido (antes del tiempo t0).
5. La máquina A transmite la trama 1 e inicia el temporizador t1 que expira después del tiempo previsto para la llegada del ACK.
6. La trama 1 es recibido por la máquina dentro del tiempo t1.
7. La máquina B transmite el ACK para confirmar que se recibió la trama 1, pero el ACK se retrasa, por lo que llega a la máquina A después del tiempo t1 establecido.
8. El temporizador 1 expira, y la máquina A decide volver a transmitir la trama 1.
9. Después de la retransmisión de la trama 1, el ACK es recibido por la máquina A fuera del tiempo establecido, la máquina A asume que es el ACK de la reciente trama 1 transmitida.
10. La máquina A transmitirá la trama 2 e iniciará el temporizador 2.
11. Se asume que la trama 2 se pierde o tiene algún error.
12. El ACK de retransmisión de la trama 1 es recibido en el período que se espera el ACK de la trama 2, por tal motivo la máquina A asume que es el ACK de la trama 2.
13. La máquina A decide transmitir la trama 3 y así sucesivamente.
14. La trama 2 se pierde y nunca fue retransmitida, figura 73.

Conclusión: La transmisión no recupera la trama perdida.

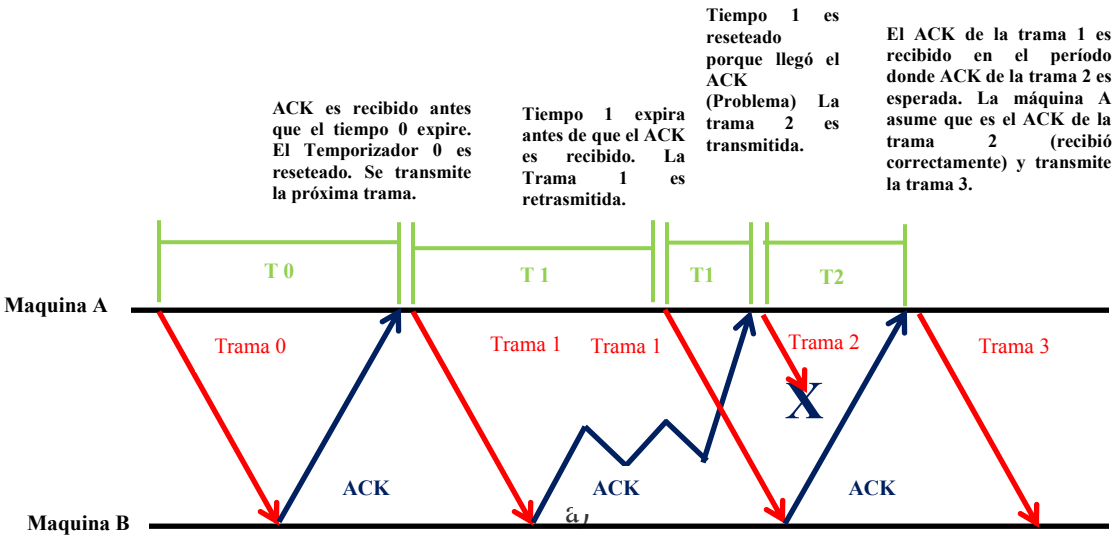


Figura 73: ACK fuera del tiempo establecido

d) Bit de Paridad

Si se analiza el caso de que el ACK llega fuera del tiempo establecido, existe la posibilidad de una ambigüedad debido al comportamiento de los ACKs. Esto ocurre porque la máquina A no tiene ninguna manera de identificar los ACKs recibidos a que trama pertenece. Para evitar esta ambigüedad se debe incorporar un número de secuencia en los ACKs para informar a la máquina transmisora (máquina A) que trama está siendo reconocida.

Debido a la estructura del algoritmo de parada y espera ARQ, un solo bit de secuencia de paridad para e impar colocado en los ACKs puede ser la solución. El uso de este bit se ilustra en la figura 74.

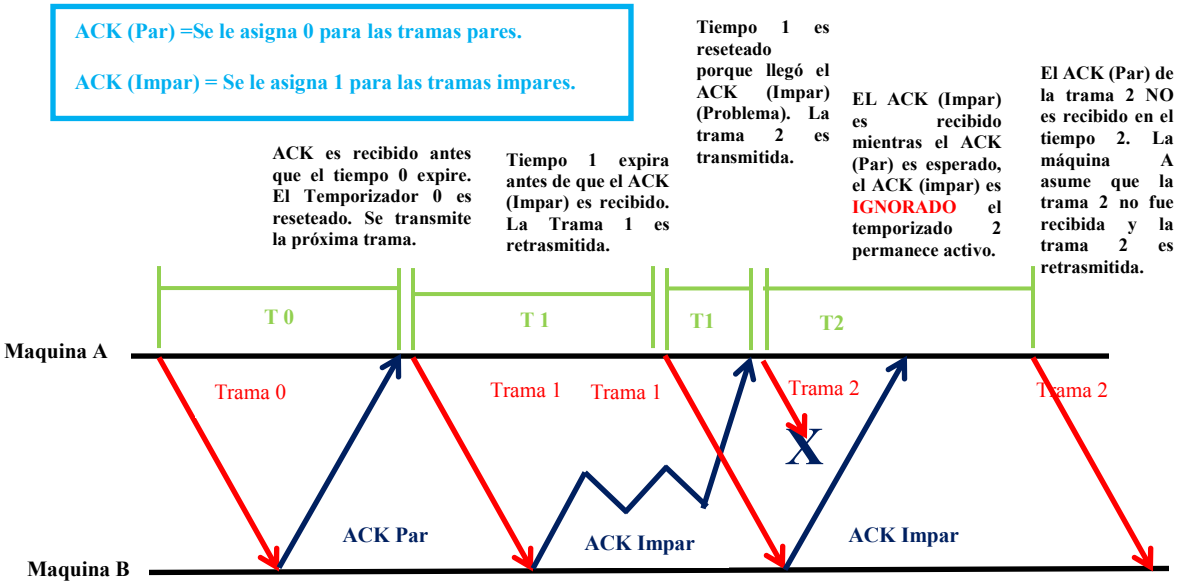


Figura 74: Bit de paridad

1. La máquina A transmite la trama 0 e inicia el temporizador 0 que expira después del tiempo previsto para la llegada del ACK (par).
2. La trama A llega a la máquina B dentro del tiempo t0.
3. La máquina B transmite el ACK (par).
4. El ACK (par) de la trama es recibido por la máquina A dentro del tiempo establecido, antes de que expire t0.
5. La máquina A transmite la trama 1 e inicia el temporizador 1 que expira después del tiempo previsto para la llegada del ACK (impar).
6. La trama 1 llega a la máquina B dentro del tiempo t1.
7. La máquina B transmite el ACK (impar) para reconocer la trama 1, pero el ACK impar se retrasa, por lo que llega a la máquina A después del tiempo t1 que ya ha expirado.
8. El temporizador 1 expira, y la máquina A decide volver a transmitir la trama 1.
9. Después de la retransmisión de la trama 1, el ACK (impar) es recibido por la máquina A fuera del tiempo establecido, la máquina A asume que es el ACK (impar) de la reciente trama 1 transmitida.
10. La máquina A transmitirá la trama 2 e iniciará el temporizador 2.
11. Se asume que la trama 2 se pierde o tiene algún error.

12. Se supone que el ACK (impar) de retransmisión de la trama 1 es recibido en el período que se espera el ACK (par) de la trama 2. La máquina A ignorará al ACK (par) debido a que está esperando un ACK de una trama impar. El temporizador 2 no se restablece y se mantiene activo en espera del ACK (impar).
13. El temporizador 2 expira después de un período de tiempo porque no se ha recibido el ACK (par).
14. La máquina A decide retransmitir la trama 2 y se repite el proceso en espera del ACK (par)

Conclusión: La transmisión recupera la trama perdida.

6.3 *GO-BACK-NARQ*.- Es uno de los tres esquemas de ARQ básicos, que son utilizados en los sistemas de comunicación por paquetes, ya que es un protocolo mucho más complejo [48], [49], [50]. Permite enviar tramas incluso si las tramas anteriores se recibieron sin una señal de reconocimiento. Este protocolo se basa en la secuencia de tramas, ver figura 75.

A continuación, se propone un escenario para *Go-Back-N*:

1. Se supone que el transmisor siempre dispone de tramas de datos para enviar. El transmisor inicia los números de secuencia por 0. Cada nueva trama generada se numera secuencialmente.

2. El transmisor envía tramas al receptor consecutivamente y las almacena en la ventana de transmisión a la espera de ser reconocidas.

3. El receptor acepta las tramas de manera ordenada. Es decir, descarta las tramas fuera de secuencia. Por tanto, al recibir una trama de datos se comprueba si tiene errores.

    a. Si los tiene, se descarta.

    b. Si no los tiene, se comprueba si su número de secuencia corresponde al de la trama esperada.

        1. Si corresponde, se entrega al nivel superior y se envía una trama de reconocimiento ACK indicando el número de la trama recibida correctamente.

2. Si no corresponde, se descarta la trama y se envía una trama NACK indicando el número de secuencia de la trama esperada. A partir de este momento, el receptor no envía ningún otro tipo de trama hasta recibir la trama esperada correctamente.

4. Al recibir una trama ACK, el transmisor elimina la trama reconocida de la ventana de transmisión.

5. Al recibir una trama NACK, el transmisor reenvía toda la ventana de transmisión

Es necesario el uso de un temporizador para el correcto funcionamiento del protocolo. Nótese que el protocolo quedaría atascado si:

• Se pierde toda la ventana de transmisión.

• Se pierden tramas de reconocimiento (tanto ACK como NACK).

Por tanto, el transmisor dispone de un temporizador T0. Este temporizador se inicia al enviar la primera trama y estará activo siempre que haya alguna trama en tránsito.

- El temporizador se reinicia al recibir un ACK y al recibir un NACK.
- Cuando vence el temporizador se reenvía toda la ventana. Esto implica que se vuelve a poner en marcha el temporizador al comenzar las retransmisiones.
- La duración del temporizador deber ser ligeramente superior al retardo de ida y vuelta (RTT, *Round Trip Time*) [51].

  Conclusión: La transmisión recupera la trama 1 perdida.

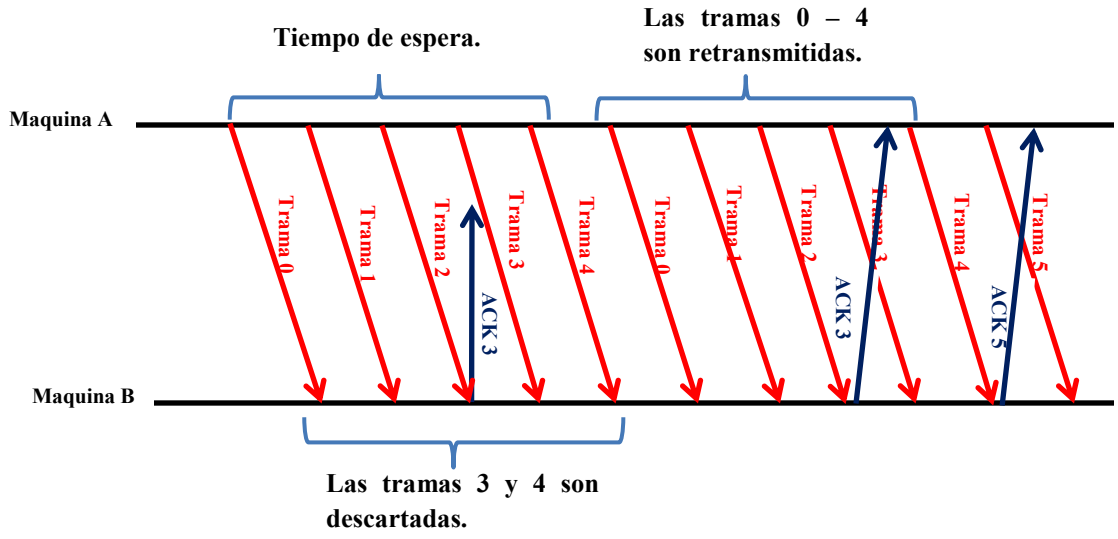


Figura 75*: Go-Back*-NARQ

6.4 ARQ DE REPETICIÓN SELECTIVA.- La repetición selectiva difiere del *Go-Back-N* ya que retrasmite únicamente los paquetes que se han perdido debido a errores, por este motivo es necesario que el receptor disponga de un buffer de almacenamiento de tramas [50], [52].

En este caso, el receptor acepta todas las tramas, independientemente de si llegan en secuencia o no, pero sólo las entrega de manera ordenada al nivel superior. Por su parte, el transmisor sólo retransmite las tramas que no han sido reconocidas correctamente. Por tanto, es necesario un buffer de almacenamiento tanto en transmisor como receptor, es decir, existe una ventana de recepción y ventana de transmisión. El receptor almacenará las tramas recibidas en su ventana de recepción y las irá entregando si llegan secuencialmente. En caso de producirse un error, el receptor sigue aceptando las siguientes tramas, pero sólo las entrega cuando dispone de una secuencia completa, es decir, cuando ha recibido de nuevo y correctamente la trama errónea (entrega ordenada), ver figura 76.

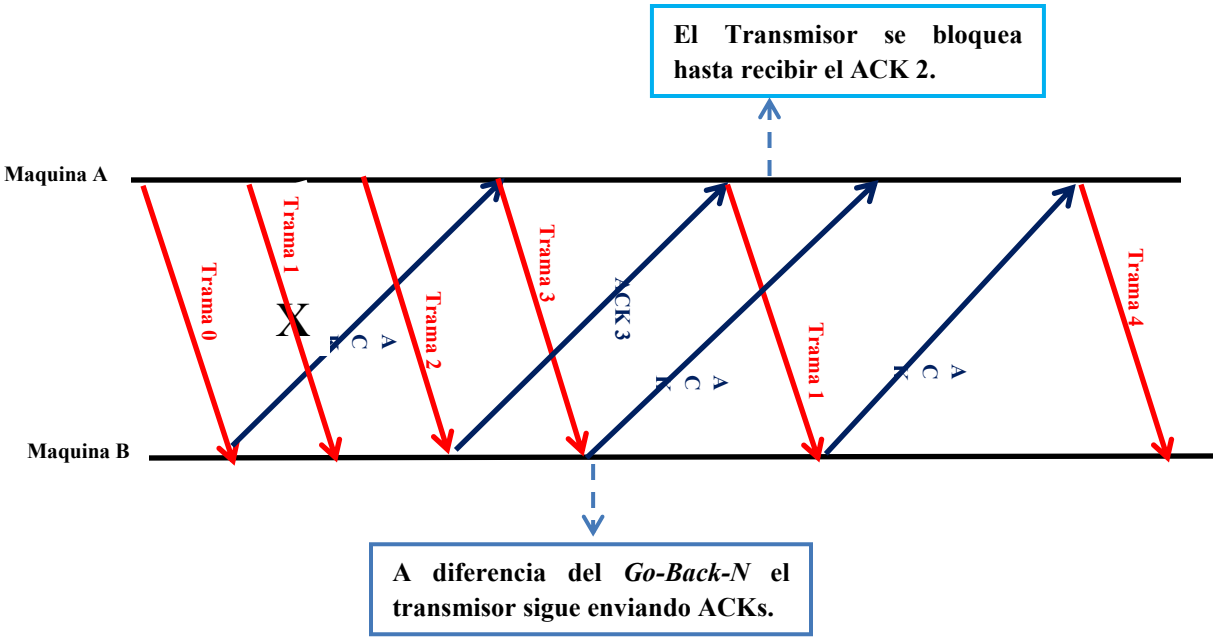


Figura 76: ARQ de repetición selectiva

6.5 SIMULACIÓN

Este modelo simula un sistema *Go-Back-N* solicitud automática de repetición (ARQ). Al modelar la transmisión y el flujo de tramas de datos, el modelo permite visualizar el comportamiento del sistema y analizar su rendimiento en diferentes condiciones. A continuación se muestra cómo utilizar los subsistemas del transmisor y el receptor de este modelo figura 77.

6.5.1 Estructura del Modelo

Los paquetes se crean y se transmiten sobre un canal, los mismos que son modificados por el ruido del canal. Para cada trama recibida, el receptor envía un mensaje de vuelta al transmisor ACK o NACK que es un mensaje de verificación basado en un código de redundancia cíclica (CRC). Al recibir un mensaje NACK, el transmisor retransmite todas las tramas que aún no han sido reconocidos con un ACK.

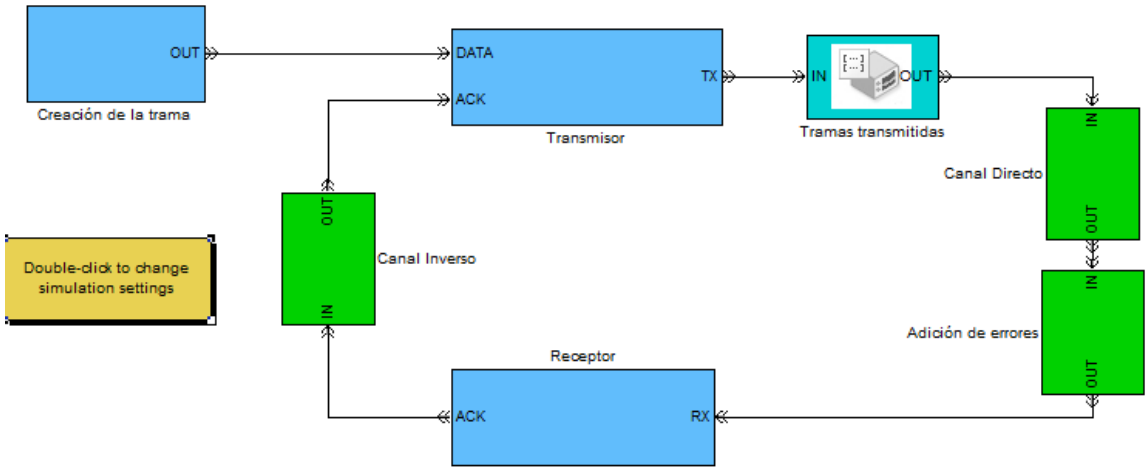


Figura 77: Esquema de simulación de un *G-Back-N (ARQ)*

a) Creación de Tramas

Este subsistema genera tramas de datos con un tiempo de intergeneración exponencial. Cada trama tiene un número de secuencia único que lo identifica figura 78.

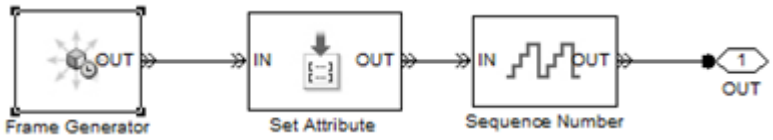


Figura 78: Subsistema de creación de tramas

b) Transmisor

El transmisor envía tramas y también retransmite tramas que han tenido errores basado en mensajes de ACK y NACK enviados por el receptor. Si el bit es igual a 1 corresponde a un mensaje ACK, mientras que 0 corresponde a un mensaje NACK figura 79.

El transmisor utiliza un diagrama de flujo de estados para controlar la transmisión y la retransmisión de tramas. El estado transmisor puede tener cualquiera de los siguientes valores:

- Transmisión: La colocación de una nueva trama de datos en el enlace.
- Retransmisión: Retransmitir las tramas de datos a través del enlace.
- Esperando la retransmisión: Esperando el enlace esté disponible para iniciar la retransmisión de tramas rechazadas.

- No transmitir porque no se disponga de nuevos datos o el número máximo de tramas pendientes se ha alcanzado.

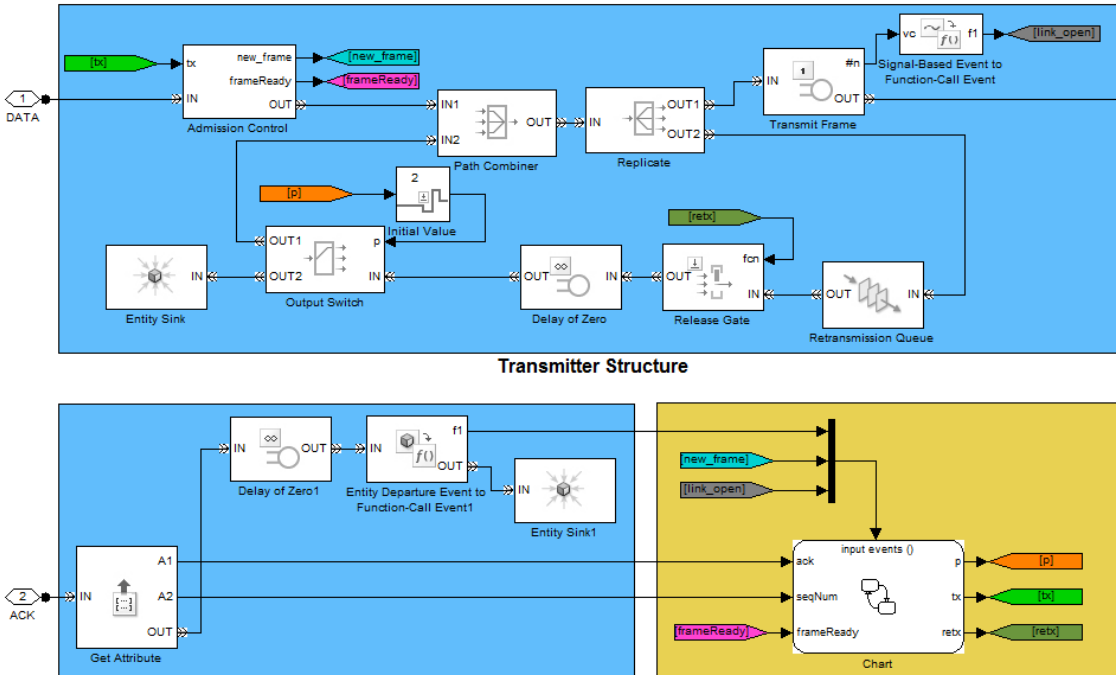


Figura 79: Subsistemas del bloque transmisor

c) Canal Directo e Inverso

Los canales directo e inverso representan el retardo de propagación, que se supone más largo que el retardo asociado con la transmisión. Es decir, el transmisor termina de enviar la trama completa antes de que el receptor comience a recibir la misma. Los paquetes son potencialmente dañados, pero no se han perdido durante la transmisión. El canal de retorno se supone que es libre de errores figura 80.

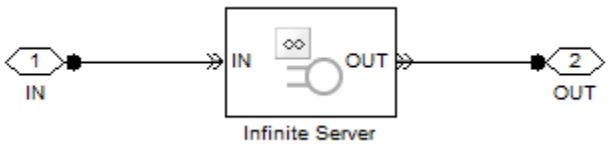


Figura 80: Canal directo e inverso

d) Adición de Errores

Este subsistema adiciona errores a ciertos paquetes con una determinada probabilidad. El modelo supone que el receptor puede detectar el error a través de una comprobación CRC, pero no puede corregir el error figura 81.

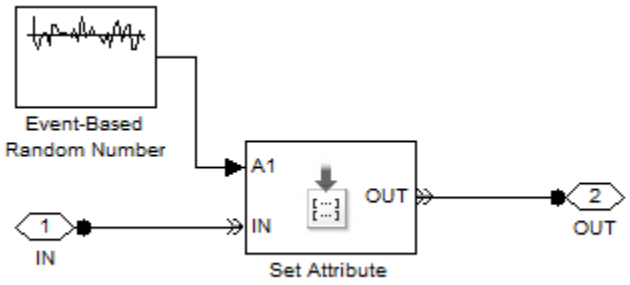


Figura 81: Subsistema del bloque de adición de errores

e) Receptor

El receptor envía un mensaje de ACK o NACK de vuelta al transmisor para cada trama y acepta o rechaza la trama de acuerdo con la comprobación del CRC. Después de rechazar una trama, el receptor continuo rechazando tramas hasta que se recibe correctamente la trama rechazada. Un estado de flujo controla el estado del receptor, o bien de aceptar o rechazar tramas de datos figura 82.

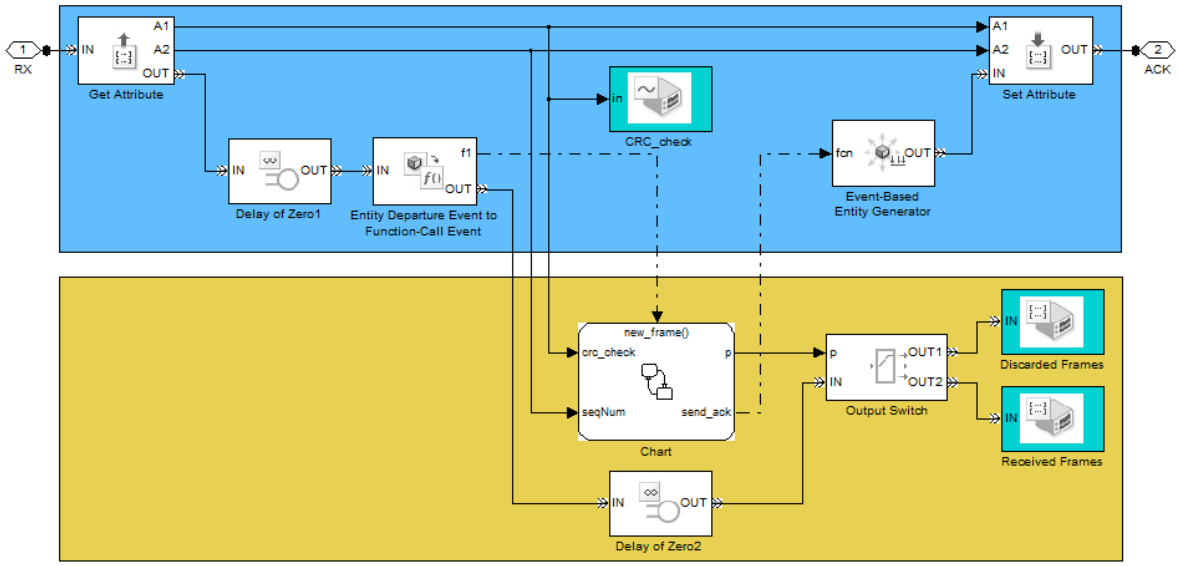


Figura 82: Subsistema del bloque receptor

f) Resultados y Pantallas

- Resultado de la comprobación CRC para cada trama recibida, donde un valor de 0 indica un error figura 83.

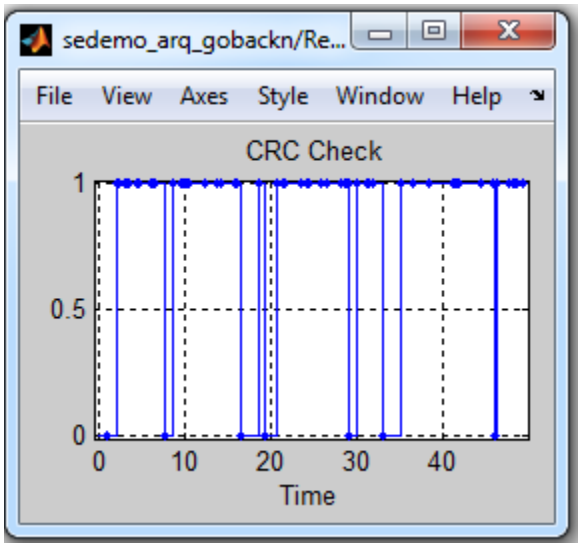


Figura 83: Verificación del CRC

- El número de tiempo y secuencia de las tramas transmitidas figura 84.

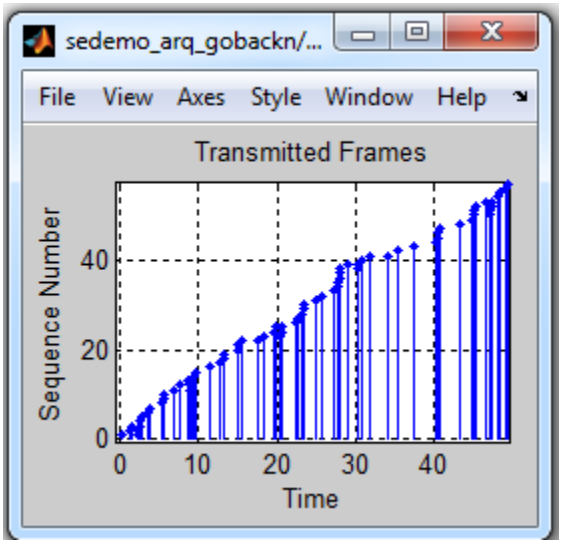


Figura 84: Trama de transmisión

- El número de tiempo y secuencia de tramas rechazadas por el receptor figura 85.

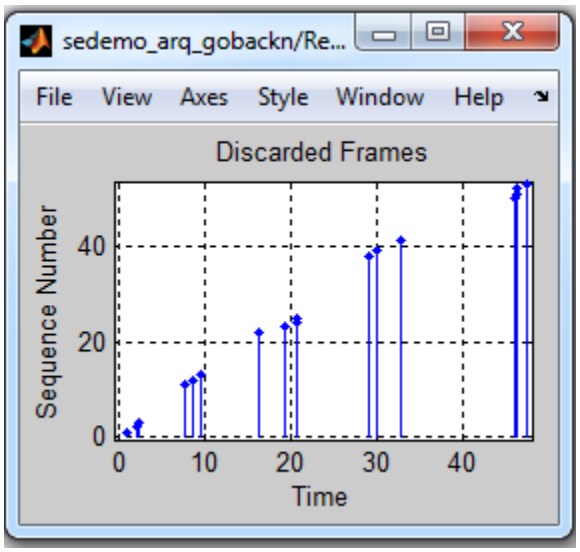


Figura 85: Tramas rechazadas por el receptor

- El número de tiempo y secuencia de tramas aceptadas por el receptor figura 86.

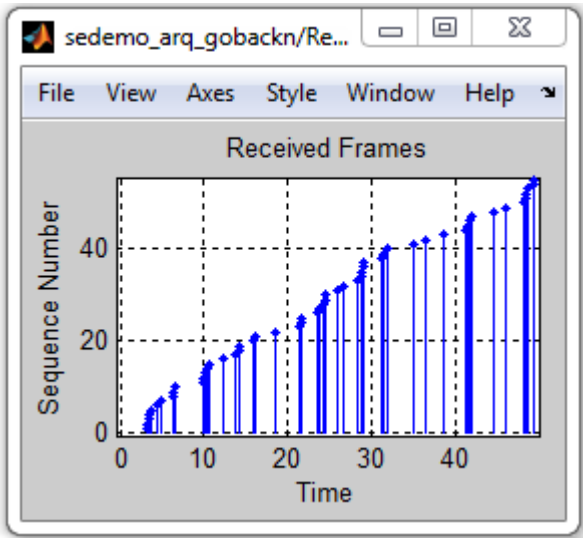


Figura 86: Ttramas en el receptor

g) Ajuste de Parámetros para la Simulación

El Ajuste de parámetros para el modelo *Go-Back-N* solicitud automática de repetición (ARQ) figura 87 son los siguientes:

Tamaño de la Ventana Deslizante: Número máximo de tramas no confirmadas permitidos en tránsito en cualquier momento.

Tiempo a Transmitir la Trama: la cantidad de tiempo que tarda el transmisor para enviar todos los bits en una trama dada.

Tiempo de Propagación: Cantidad de tiempo para que una trama se propague desde el transmisor al receptor o viceversa. Esto debería ser más largo que el tiempo para transmitir una trama.

Probabilidad de Trama Errónea: Probabilidad de que una trama se dañara en el camino.

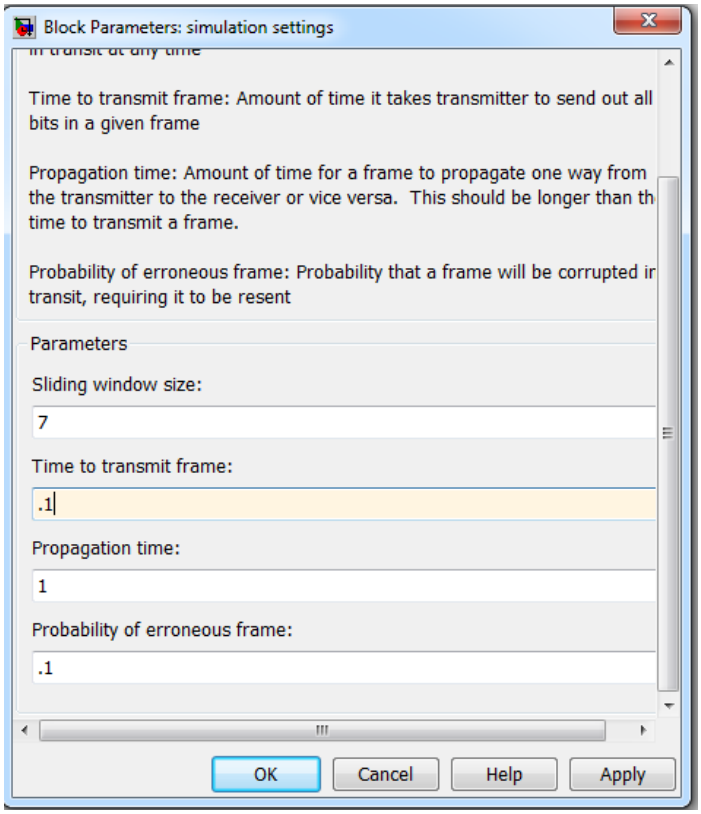


Figura 87: Parámetros de simulación

CAPÍTULO 7

TÉCNICAS DE CORRECCIÓN DE ERRORES FORWARD ERROR CORRECTION

***Forward Error Correction*.-** Que significa corrección de errores hacia adelante, esta técnica se basa en la idea de reconstruir la información deteriorada por los errores, obviamente la reconstrucción tiene lugar en el equipo receptor, para ello deben emplearse en los códigos un gran número de bits lo que disminuye la efectividad del código [53].

(FEC) Se utiliza en el control de paridad a través de paquetes de red. FEC se lleva a cabo mediante la adición de redundancia a la información transmitida utilizando algoritmos predeterminados (CRC, Reed Solomon, BCH, etc.) para darse cuenta de los errores y corregirlos [54].

7.1 Distancia *Hamming*.- A finales de la década de 1940 Claude Shannon fue el que desarrollo de la teoría de la información. Al mismo tiempo, *Richard Hamming*, un colega de Shannon en los Laboratorios Bell, encontró la necesidad de realizar corrección de errores en su trabajo en las computadoras. La comprobación de paridad ya estaba siendo utilizada para detectar errores. *Hamming* encuentra un modelo más sofisticado de comprobación de paridad al - permitir la corrección de errores individuales junto con la detección de errores dobles.

La idea entonces fue establecer un set de caracteres en el que todos tienen entre sí la misma distancia de *Hamming,* luego en el extremo receptor se verifica cada carácter recibido para determinar si es uno de los caracteres válidos, si no lo es se busca el carácter que tenga la menor distancia de *Hamming* con este y se le asigna ese valor como "correcto" [44]. Un código de *Hamming* de distancia 3 puede detectar errores dobles y corregir los simples (de un solo bit). En el caso de errores dobles la "corrección" es errónea ya que como no se sabe si el error es simple o doble se presupone simple y como tal se corrige. El código de *Hamming*-significa una codificación binaria de la información donde la apariencia de una simple error es detectado y corregido [27], [55].

Los códigos de *Hamming* a propuesta de este se mejoraron agregando a los bits de información una serie de bits de comprobación, a partir de estos últimos se puede detectar la posición de bits erróneos y corregirlos.

Códigos *Hamming* fueron la primera clase importante de códigos binario lineal diseñados para la corrección de errores [56].

7.1.1 El Código de *Hamming* para una Trama.

Se considera un mensaje que tiene cuatro bits de datos (D) que va a ser transmitido como una palabra de código de 7 bits mediante la adición de tres bits de control de error. Esto se llama un código (7,4). Los tres bits a ser añadidos son incluso tres bits de paridad (P), donde se calcula la paridad de cada uno de los diferentes subconjuntos de los bits de mensaje, como se muestra en la tabla 22 [57], [58].

Tabla 22: Código de *Hamming* (7,4)

7	6	5	4	3	2	1	
D	D	D	P	D	P	P	Palabra de código de 7 bits
D	-	D	-	D	-	P	Paridad Par
D	D	-	-	D	P	-	Paridad Par
D	D	D	P	-	-	-	Paridad Par

Los tres bits de paridad son (1,2,4) los mismos que están relacionados con los bits de datos (3,5,6,7) como se muestra en la figura 88.

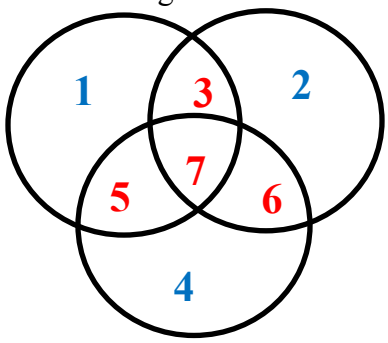


Figura 88: Bits de paridad y bits de datos

Se puede observar que el cambio del bit 7 afecta a los tres bits de paridad, en cualquier otro caso solamente afecta a dos bits de paridad [58].

De acuerdo a la tabla 23, el mensaje de 1101 sería enviado como 1100110

Tabla 23: Análisis de los bits de paridad

7	6	5	4	3	2	1	
1	1	0	0	1	1	0	Palabra de código de 7 bits
1	-	0	-	1	-	0	Paridad Par
1	1	-	-	1	1	-	Paridad Par
1	1	0	0	-	-	-	Paridad Par

Ahora se puede observar que si se produce un error en cualquiera de los bits de datos, el error afectará a diferentes combinaciones de los tres bits de paridad, dependiendo de la posición de bit [57], [59].

Por ejemplo, se supone que el mensaje anterior 1100110 se envía y en el transcurso del camino se produce un error y llega al receptor como 1110110.

El error anterior (en el bit 5) se puede corregir mediante el análisis de cuál de los tres bits de paridad se vio afectada por el bit que tiene error, ver tabla 24.

Tabla 24: Análisis del bit con error

7	6	5	4	3	2	1		ANÁLISIS	
1	1	1	0	1	1	0	Palabra de código de 7 bits		
1	-	1	-	1	-	0	Paridad Par	1	INCORRECTO
1	1	-	-	1	1	-	Paridad Par	0	CORRECTO
1	1	1	0	-	-	-	Paridad Par	1	INCORRECTO

Cuando se realiza el análisis de la trama de 7 bits que llegó al receptor todos los bits de paridad deben ser 0, pero como existió un error afecto a 2 bits de paridad. Si se analiza los bits de paridad actuales tenemos 101 que corresponde al número 5, lo que nos dice que es el bit 5 el que tiene un error [57].

Se puede resumir que el código *Hamming* tiene las siguientes características:

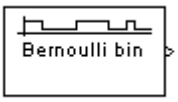
- Detección de 2 bits erróneos (suponiendo que no se intenta la corrección).
- Corrección del error de un solo bit.
- Existe una redundancia de 3 bits para un mensaje de 4 bits.
- El peso de *Hamming* de una cadena binaria es el número de unos en la cadena [55].

El código de *Hamming* usados para la detección de errores en el canal simétrico binario satisface la envolvente 2-p, donde p es los bits de verificación de paridad igual a n – k [60].

7.1.2 Simulación

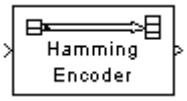
Para la simulación se utiliza los siguientes bloques:

a) **Generador Binario de *Bernoulli*:** Es un generador de números binarios de forma aleatorios, el mismo que utiliza la distribución de *Bernoulli.* La distribución de *Bernoulli* produce ceros con una probabilidad p y unos con una probabilidad 1-p. La distribución de *Bernoulli* tiene un valor promedio de 1-p y una varianza de p(1-p).



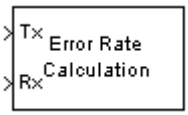
b) **Codificador *Hamming*:** El bloque codificador *Hamming* crea un código de *Hamming* con K = longitud del mensaje y N = longitud de la trama. El número N debe tener la forma 2^M-1, donde M es un número entero mayor que o igual a 3. Entonces K es igual a N-M.

Este bloque acepta un vector columna de señal de entrada de longitud K. La señal de salida es un vector columna de longitud N, (N,K), por ejemplo si se tiene (7,4) donde la longitud de la trama N = 7 y la longitud del mensaje es K = 4.



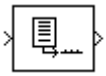
c) **Decodificador *Hamming***: Este bloque recupera el vector de la trama y el vector del mensaje binario de *Hamming*. Para la correcta decodificación, los valores de los parámetros de este bloque deben coincidir con los del bloque codificador de *Hamming*.

d) **Cálculo de la Tasa de Error**: Este bloque realiza el cálculo del BER dividiendo el número total de bits erróneos recibidos para el número total de bits transmitido o producidos por el Generador Binario de *Bernoulli*.



e) **Canal Binario Simétrico**: El objetivo del canal es introducir errores binarios en la señal transmitida. Este bloque acepta una señal de entrada escalar o vectorial. El bloque procesa cada elemento del vector de forma independiente, y se introduce un error en un lugar determinado con una probabilidad de error.

f) **Sin Buffer**: El bloque sin buffer tiene una entrada Mi-por-N y una salida de 1-por-N. Es decir, las entradas son sin búfer y en modo de fila, de tal manera que cada fila de la matriz se convierte en una muestra independiente de tiempo en la salida. La velocidad a la que el bloque recibe las entradas es generalmente menor que la velocidad a la que el bloque produce las salidas.



g) **Operador Relacional**: El bloque del operador relacional compara dos entradas usando el parámetro del operador relacional que se especifique. La primera entrada corresponde al puerto de entrada superior y la segunda entrada al puerto de entrada inferior.

7.1.2.1 Simulación Utilizando el Codificador y Decodificador de *Hamming*

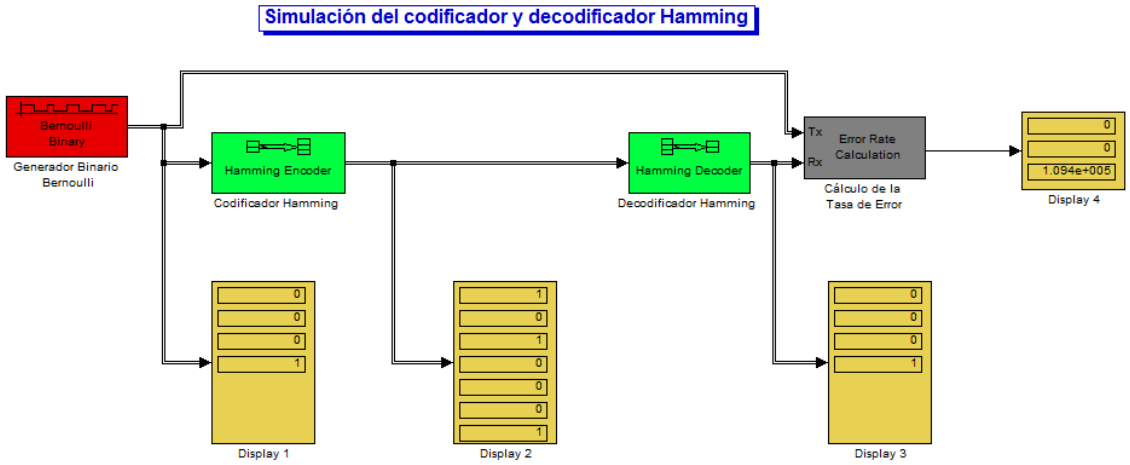


Figura 89: Simulación utilizando el codificador y decodificador de *Hamming*

En la figura 89 se muestra la simulación en la que se puede observar que no existe ningún error debido a que no pasa por ningún canal por tal motivo los *diplays* están visualizando los siguientes códigos.

1. *Display* 1: Indica el código de datos a ser transmitido (1000).
2. *Display* 2: Indica el código a la salida del codificador en donde se le añade 3 bits. redundantes para formar una trama de 7 bits (1000101).
3. *Display* 3: Decodifica la trama y elimina los códigos redundantes entregando. únicamente los bits del mensaje (1000).
4. *Display* 4: Indica los siguientes parámetros.
   - BER = 0
   - Bits de errores = 0
   - Bits Transmitidos = $1.094*10^5$

7.1.2.2 Simulación del Código de *Hamming* Introduciendo Errores en el Canal

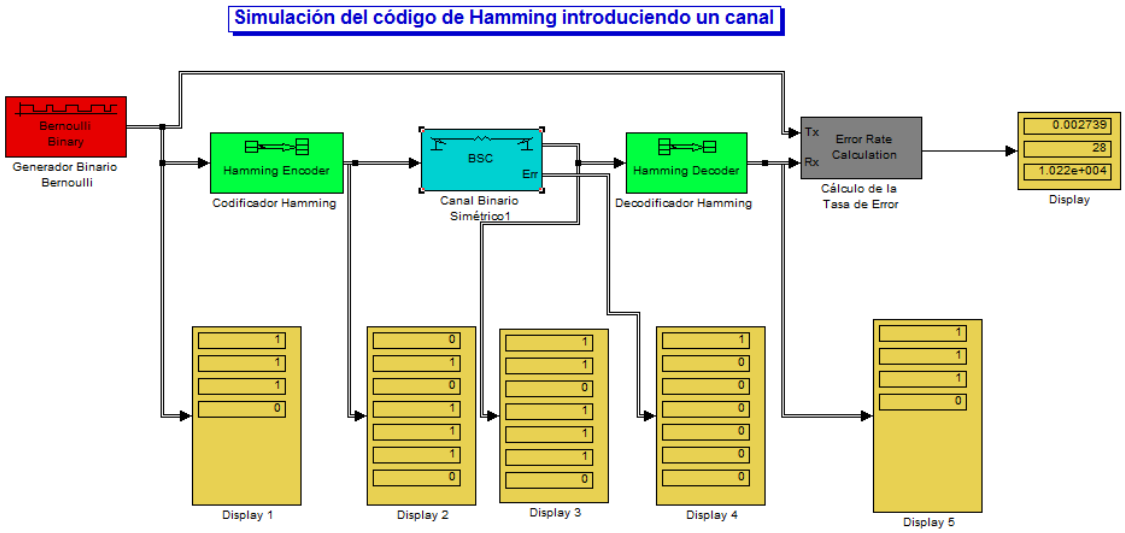


Figura 90 Simulación del código de *Hamming* introduciendo un canal

En la figura 90 se puede observar la simulación en la que se ha introducido un canal a la que se ha colocado una probabilidad de error de 0.02%, el mismo que ha producido diferentes cambios en el proceso, los mismos que se pueden observar en los *displays*.

Display 1: Indica el código de datos a ser transmitido (1110).

Display 2: Indica el código a la salida del codificador en donde se le añade 3 bits redundantes para formar una trama de 7 bits (0101110).

Display 3: Introduce algunos errores, por tal motivo cambia la trama a ser transmitida (1101110).

Display 4: Compara entre los bits de ingreso al canal y los bits de salida del canal, si existe un error coloca un 1 y en caso contrario un 0, en este caso existe un error en el bit 1.

Display 5: Decodifica la trama y elimina los códigos redundantes entregando únicamente los bits del mensaje (1110), como se puede observar a pesar que existe un error el mensaje es correcto, por tal motivo existió la corrección del error.

Display 6: Indica los siguientes parámetros:

- BER = 0.002739 = 28/10220
- Bits de errores = 28
- Bits Transmitidos = 10220

7.1.2.3 Simulación de Corrección de Errores por Medio del Código de *Hamming*

El bloque de operador relacional compara la señal transmitida, que viene desde el bloque de generador aleatorio de Bernoulli, con la señal recibida, procedente del bloque decodificador *Hamming* como se indica en la figura 91 El bloque de salida entrega un 0 cuando las dos señales son iguales y un 1 cuando son diferentes.

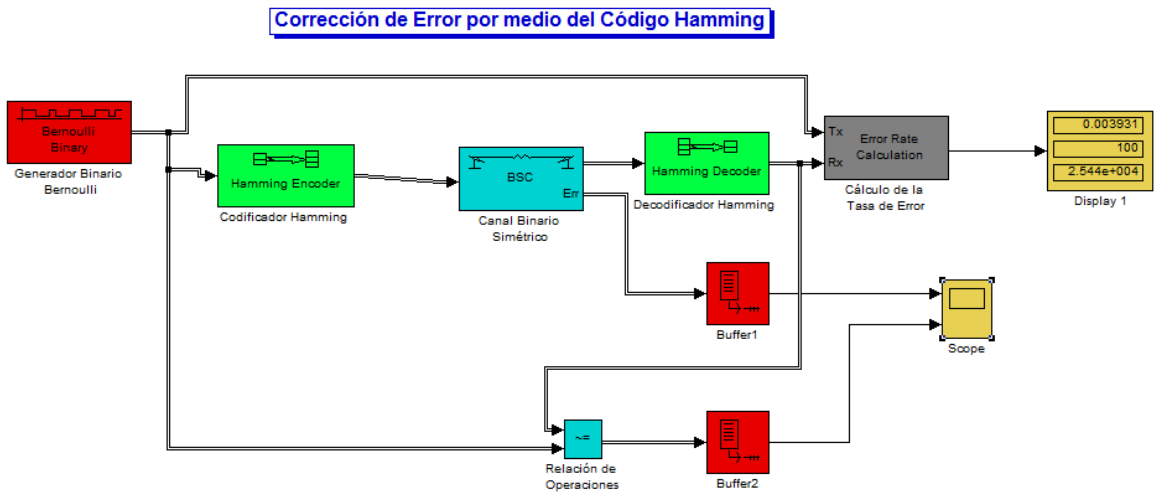


Figura 91: Simulación de corrección de errores por medio del código de *Hamming*

Cuando se ejecuta la simulación se muestran los datos de error. Al final de cada uno de los 5000 pasos de tiempo, el ámbito de aplicación aparece como se muestra en la figura 92, luego borra los datos mostrados y muestra los siguientes 5.000 puntos de datos.

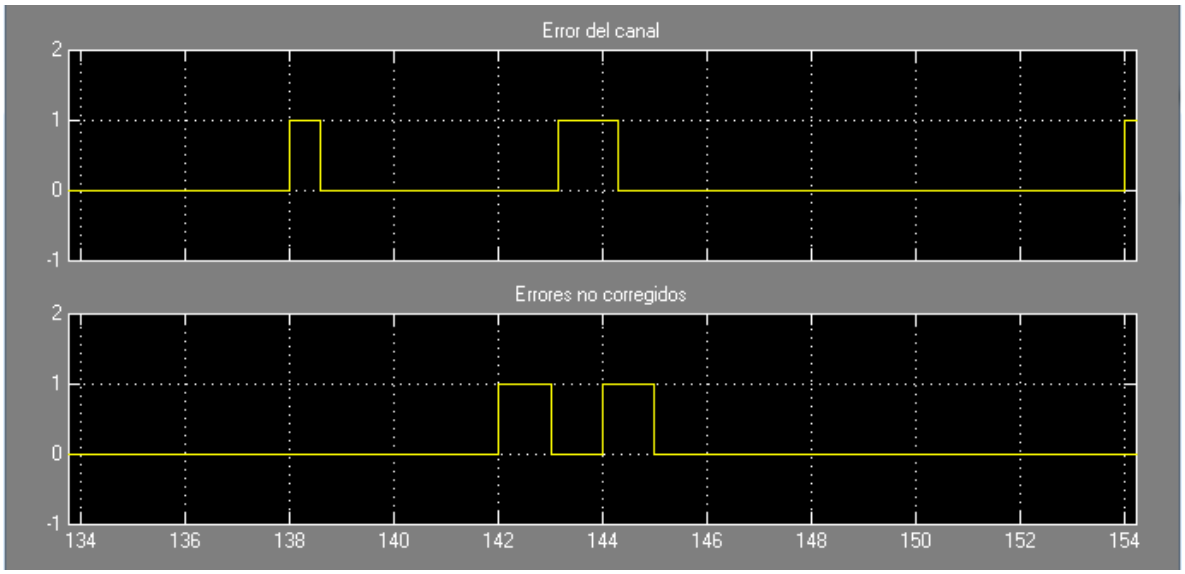


Figura 92: Corrección de errores

La figura superior muestra los errores de canal generados por el bloque de canal binario simétrico. El ámbito inferior muestra los errores que no se corrigieron debido al decodificador.

Si se analiza la figura 92 se puede observar que el pulso rectangular más angosto ubicado a la izquierda de la figura superior representa un error que se corrige. El pulso ubicado en el centro de la figura superior representa también un código que se corrige, pero debido al decodificador se produce 2 errores.

7.1.2.4 Simulación de Corrección de Errores por Medio del Código de *Hamming* Exportando Datos a *MatLab.*

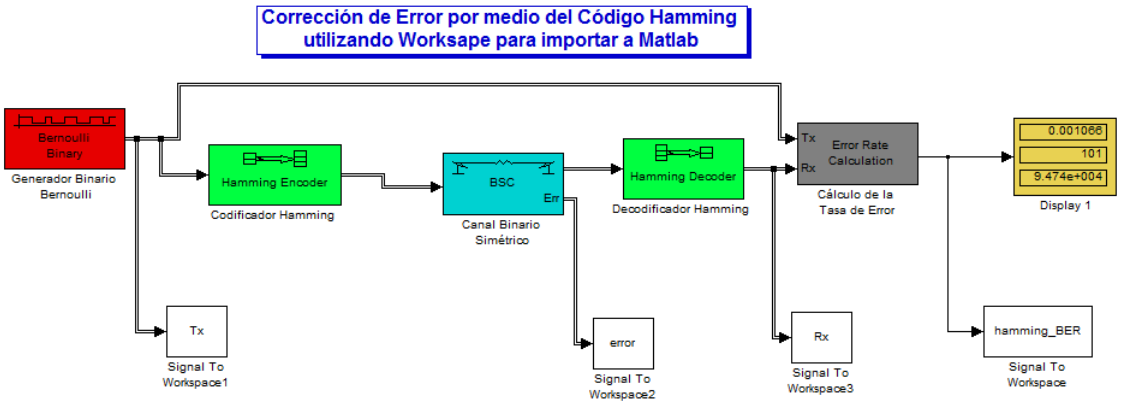


Figura 93: Esquema para transferir datos del *Simulink* al Espacio de trabajo de *MatLab*

7.2 CÓDIGOS *BOSE CHAUDHURI Y HOCQUENGHEM.*- Los códigos BCH forman una de las clases más grandes de códigos cíclicos de corrección de errores.

Esta clase de códigos es una generalización notable de los códigos de *Hamming* [61] para la corrección de múltiples error [62]. Los códigos binarios BCH fueron descubiertos por *Hocquenghem* en 1959 y de forma independiente por *Bose y Chaudhuri* en 1960.

Los códigos BCH constituyen una de las clases más importantes y poderosas de los códigos de bloques lineales. Para cualquier enteros positivos m ($m \geq 3$) y t ($t < 2^m-1$), existe un código BCH binario con los siguientes parámetros:

- Longitud del bloque: $n = 2^m - 1$.
- Número de bits de dato: $k = 2^m - m - 1$.
- Número de bits de chequeo: $n - k \leq mt$.
- Distancia mínima: $d_{min} >= 2t + 1$; t = bits de corrección.

7.2.1 Generación de Código de Campo de Galois.- Pará desarrollar la corrección de errores de un único bit (t=1) se partirá de una matriz de polinomios basado en campo de *Galois* GF = (2^3). A partir de estas mismas formulaciones, es posible construir códigos BCH con distancia predecibles y la capacidad de corregir errores de varios bits [63].

El campo de *Galois* GF (2^3) o GF (8) tiene ocho elementos numerados [0 .. 7], como se muestra en la tabla 25. La relación específica entre estos elementos está definido por un polinomio primitivo P (x) con grado 3. En este ejemplo, sea P (x) = $x^3 + x + 1$. Si algún elemento a es una raíz de P (x), entonces P (a) = 0. En la siguiente tabla P (a) = $a^3 + a + 1$; $a^3 = a + 1$.

Tabla 25: Campo de *Galois* = 2^3

a^0	$=a^0$	=1	=1	=1	**= 001**	**=1**	$=a^7$	$=a^{14}$	$=a^{21}$…
a^1	$=a^0$ x a	=(1) x a	=a	=a	**= 010**	**=2**	$=a^8$	$=a^{15}$	$=a^{22}$…
a^2	$=a^1$ x a	=(a) x a	$=a^2$	$=a^2$	**= 100**	**=4**	$=a^9$	$=a^{16}$	$=a^{23}$…
a^3	$=a^2$ x a	$=(a^2)$ x a	$=a^3$	=a+1	**= 011**	**=3**	$=a^{10}$	$=a^{17}$	$=a^{24}$…
a^4	$=a^3$ x a	=(a+1) x a	$=a^2+a$	$=a^2+a$	**= 110**	**=6**	$=a^{11}$	$=a^{18}$	$=a^{25}$…
a^5	$=a^4$ x a	=(a^2+a) x a	$=a^3+a^2$	$=a^2+a+1$	**= 111**	**=7**	$=a^{12}$	$=a^{19}$	$=a^{26}$…
a^6	$=a^5$ x a	=(a^2+a+1) x a	$=a^3+a^2+a$	$=a^2+1$	**= 101**	**=5**	$=a^{13}$	$=a^{20}$	$=a^{27}$…

Observe que los valores se reducen a siete términos de 3 bits cada uno y que la secuencia se repite periódicamente para N > 6, de tal manera que $a^7 = a^0$ y así sucesivamente. Estos valores servirán para desarrollar una matriz de verificación capaz de corregir los errores de múltiplos bits. En general, las operaciones dentro de esta GF (8) utiliza el módulo de aritmética 2 para los valores de 3-bit. Por ejemplo 011 + 101 = 110, ó 3 + 5 = 6.

La potencia de i…n de cada uno de los elementos de a^n puede ser descrito como a^{ni}. Para cada uno de los valore de i….n se puede escribir la siguiente tabla como valores octales ver tabla 26.

Tabla 26: Valores del campo de *Galois* para i=1….n

i	**a^{6i}**	**a^{5i}**	**a^{4i}**	**a^{3i}**	**a^{2i}**	**a^{1i}**	**a^{0i}**
1	**5**	**7**	**6**	**3**	**4**	**2**	**1**
2	**7**	**3**	**2**	**5**	**6**	**4**	**1**
3	**6**	**2**	**7**	**4**	**5**	**3**	**1**
4	**3**	**5**	**4**	**7**	**2**	**6**	**1**
5	**4**	**6**	**5**	**2**	**3**	**7**	**1**
6	**2**	**4**	**3**	**6**	**7**	**5**	**1**
7	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	A	**B**	**C**	**D**	**E**	**F**	**G**

.

Las filas de la tabla se repiten periódicamente para i > 7. La fila para i = 7 se incluye para completar.

7.2.1.1 Control y Corrección de Errores

a) Para un solo bit de corrección de errores: si t=1 entonces D = 2t+1 = 3

La matriz H de (3*7) para la corrección de errores de un solo bit se define en la tabla 26 con i = 1 como se muestra a continuación:

	A	B	C	D	E	F	G
H=	5	7	6	3	4	2	1
	1	1	1	0	1	0	0
H=	0	1	1	1	0	1	0
	1	1	0	1	0	0	1

La corrección del error de un único bit es posible porque cada columna de 3-bit es única. Por ejemplo, el bit único de error $E(x) = x^4$ devuelve el síndrome 6 de la columna y ningún otro error de un solo bit.

La corrección de dos bits de error no es posible, los dos diferentes bits de error no tienen un único resultado. Por ejemplo si tenemos los dos bits de error dados por $E(x) = x^3 + x^6$ devuelve un síndrome de D + A = 3 + 5 = 011 + 101 = 110 = 6, pero existe otra combinación que también me dan un síndrome de 6 como es: B + G = 7 + 1 = 111 + 001 = 110 = 6 lo que implica que existiera un error en x^5+x^0 que es diferente al error que se desea corregir.

En general, esta matriz proporciona una distancia de *Hamming* mínima D = 3.

b) Corrección de doble bit de error: Se tiene que t=2 entonces D = 2t+1 = 5.

Para lograr la corrección de dos bit de error es necesario que la distancia de *Hamming* sea igual a 5 (D = 5). Con este fin, la matriz de verificación debe ser ampliada para dar síndromes diferentes para cada uno de los esperados patrones de error. El reto, por supuesto, es determinar cómo ampliar esta matriz para asegurar que todos los errores de uno y dos bits dará un síndrome único. Se debe tener en cuenta que en cada fila (i) el

ordenamiento de cada columna entrega un síndrome diferente para cualquier error específico, por lo que la combinación de dos filas de la tabla es la clave para determinar el error.

Usando el mismo GF =8 como la base para esta nueva matriz, se incorpora otra fila como se muestra a continuación (i = 3). Esta fila adicional sumará tres filas a la matriz binaria original.

Con i = (1 y 3), la matriz H = (6 x 7) corrige los dos bits de errores como se muestra a continuación.

	A	B	C	D	E	F	G
H=	5	7	6	3	4	2	1
	6	2	7	4	5	3	1

	1	1	1	0	1	0	0
	0	1	1	1	0	1	0
H=	1	1	0	1	0	0	1
	1	0	1	1	1	0	0
	1	1	1	0	0	1	0
	0	0	1	0	1	1	1

La corrección de un solo bit es todavía posible en la matriz de (6 x 7), porque cada columna tiene 6-bit que es único. El resultado está asegurado desde la matriz de (3 x 7) y no depende de nuevas filas. El único bit de error $E(x) = x^4$ tiene ahora un síndrome igual a 67 que corresponde a los 6 bits codificado en octal (columna C) y no hay otro error de un solo bit.

La corrección de los dos bits de erros ahora es posible y tienen un resultado único. El error de dos bits $E(x) = x^3 + x^6$ = A + D = 011100+101110 + 56 = 110010 = 62 y si se analiza B + G = 72 + 11 = 111010 + 001001 = 110011 = 63 es ahora otro resultado diferente. La tabla 27 muestra los únicos síndromes de 6 bits (octal) para corregir un solo error (D = 3) o dos errores (D = 5).

Tabla 27: Valores de los síndromes para 6 bits codificados en octal

		0000000	0000001	0000010	0000100	0001000	0010000	0100000	1000000
			G	**F**	**E**	**D**	**C**	**B**	**A**
0000000		00	11	23	45	34	67	72	56
0000001	**G**	11	--	32	54	25	76	63	47
0000010	**F**	23	32	--	66	17	44	51	75
0000100	**E**	45	54	66	--	71	22	37	13
0001000	**D**	34	25	17	71	--	53	46	62
0010000	**C**	67	76	44	22	53	--	15	31
0100000	**B**	72	63	51	37	46	15	--	24
1000000	**A**	56	47	75	13	62	31	24	--

7.2.1.2 Matriz de Verificación que no Funciona Correctamente

La siguiente matriz de verificación H = (6 x 7) está definida para corregir dos bits de errores (con i = 1 e i = 2) como se muestra a continuación. Sin embargo esta elección tiene fallas ya que no corrige correctamente.

	A	**B**	**C**	**D**	**E**	**F**	**G**
H=	**5**	**7**	**6**	**3**	**4**	**2**	**1**
	7	**3**	**2**	**5**	**6**	**4**	**1**

	1	**1**	**1**	**0**	**1**	**0**	**0**
	0	**1**	**1**	**1**	**0**	**1**	**0**
H=	**1**	**1**	**0**	**1**	**0**	**0**	**1**
	1	**0**	**0**	**1**	**1**	**1**	**0**
	1	**1**	**1**	**0**	**1**	**0**	**0**
	1	**1**	**0**	**1**	**0**	**0**	**1**

A primera vista, esta matriz parece tan buena como el ejemplo anterior, pero por desgracia, nada se ha ganado mediante la inclusión de esta particular elección de la nuevas filas con i = 2.

El error del único bit $E(x) = x^4$ devuelve ahora el síndrome = 62 de 6-bit (octal) de la columna C y no hay otro error de un solo bit. El error de dos bits $E(x) = x^3 + x^6$ devuelve un síndrome que es la suma de los bits A + D = 57 + 35 = 101111+011101 =110010 = 62, el mismo que el error de un solo bit, por tal motivo la nueva fila con i = 2 no contribuye a la distinción de dos errores de bit.

Sucede que en este sistema numérico, elevar al cuadrado un término es una operación lineal (en otras palabras, $x^2 + y^2 = (x + y)^2$) y cada valor en la fila (i = 2) es el cuadrado del valor correspondiente encontrado en (i = 1). Además, ninguna de las filas pares contribuirá a entregar una información adicional necesaria para dar un síndrome único. En particular, los valores de las filas i = (1,2) son cuadrados el uno con respecto al otro, como lo son i = (2,4). Los valores i = (3,6) describen otro par de filas relacionadas con ese sistema cuadrático.

Como ejemplo se toma de la fila (i = 1), el valor de A = 5 y D = 3, entonces A + D = 101 + 011 = 110 = 6.

$$X^2 + Y^2 = A^2 + D^2 = 5^2 + 3^2 = 7 + 5 = 111 + 101 = 010 = 2$$

$$(X + Y)^2 = (A+D)^2 = (5 + 3)^2 = (101 + 011)^2 = (110)^2 = (6)^2 = 2$$

$$\text{Por lo tanto } X^2 + Y^2 = (X + Y)^2$$

En la tabla 28 se muestra los parámetros (n,k,t) para la construcción de los códigos BCH y en la tabla 29 se indican los polinomios generadores.

Tabla 28: Parámetros de los códigos BCH [64]

n	*k*	*t*	*n*	*k*	*t*	*n*	*k*	*t*	*n*	*k*	*t*	*n*	*k*	*t*
7	4	1	63	30	6	127	64	10	255	207	6	255	99	23
15	11	1		24	7		57	11		199	7		91	25
	7	2		18	10		50	13		191	8		87	26
	5	3		16	11		43	14		187	9		79	27
31	26	1		10	13		36	15		179	10		71	29
	21	2		7	15		29	21		171	11		63	30
	16	3	127	120	1		22	23		163	12		55	31
	11	5		113	2		15	27		155	13		47	42
	6	7		106	3		8	31		147	14		45	43
63	57	1		99	4	255	247	1		139	15		37	45
	51	2		92	5		239	2		131	18		29	47
	45	3		85	6		231	3		123	19		21	55
	39	4		78	7		223	4		115	21		13	59
	36	5		71	9		215	5		107	22		9	63

Tabla 29: Polinomios generadores para los códigos BCH [64]

n	*k*	*t*	*P*(*X*)
7	4	1	$X^3 + X + 1$
15	11	1	$X^4 + X + 1$
15	7	2	$X^8 + X^7 + X^6 + X^4 + 1$
15	5	3	$X^{10} + X^8 + X^5 + X^4 + X^2 + X + 1$
31	26	1	$X^5 + X^2 + 1$
31	21	2	$X^{10} + X^9 + X^8 + X^6 + X^5 + X^3 + 1$

7.3 TÉCNICA DE CORRECCIÓN DE ERRORES UTILIZANDO CÓDIGOS *REED-SOLOM*

Los códigos *Reed-Salomon* (códigos RS) son una subclase de los códigos BCH no binarios, ya que el codificador de un código RS opera sobre un bloque de bits en vez de bits individuales como en el caso de los códigos binarios. Así, la data es procesada en trozos de m bits, llamados símbolos. Y un código RS (n,k) tiene los siguientes parámetros [65], [63]:

- Longitud del símbolo: m bits por símbolo.
- Longitud del bloque: $n = 2^m - 1$ símbolos = $m(2^m - 1)$ bits.
- Longitud de la data: k símbolos.

- Tamaño del código de chequeo: n – k = 2t símbolos = m(2t) bits.
- Distancia mínima: $d_{min} = 2t + 1$ símbolos.

7.3.1 Campo de *Galois* GF (2^3) = GF (8)

Para el análisis se tomará como base el polinomio P (x) = $x^3 + x + 1$, donde los términos (a^n) sirven para describir el campo de *Galois* GF (8) que son equivalentes al valor de 3 bits en octal.

0	**a^0**	**a^1**	**a^2**	**a^3**	**a^4**	**a^5**	**a^6**
0	**1**	**2**	**4**	**3**	**6**	**7**	**1**

Como el polinomio P (a) = $a^3 + a + 1 = 3 + 2 + 1 = 011 + 010 + 001 = 000 = 0$, debido a que es una raíz de P(x).

El polinomio M (x) se puede definir mediante los valores mencionados anteriormente.

Por ejemplo, el polinomio M (x) tiene tres coeficientes que se puede representar de la siguiente manera:

M(x) $= a^1 x^3 + a^2 x^2 + a^1 x + a^1$

$= 2x^3 + 4x^2 + 2x + 2$, también se puede escribir como (2 4 2 2)

$= (010)\, x^3 + (100)\, x^2 + (010)\, x + (010)$

= El valor de los 12 bits binarios son: 010100010010

7.3.2 Formulación del Generador Polinomial G (x).- La forma general del del generador polinomial G(x) del código RS es:

$$G(x) = (x - a^1)\,(x - a^2)\,(x - a^3)\,(x - a^4) \ldots (x - a^{2t})$$

Donde t es el número de múltiples bits que pueden ser corregidos en el símbolo y 2t es el número de símbolos de paridad. Los códigos RS tienen la capacidad de corregir errores en un símbolo en lugar de corregir errores de bits. Para GF (8), el código resultante

es (n, k) = (7, 7-2t), donde n, k y t se refieren a un número de símbolo de 3bits. El tamaño de la global es m(2m-1) = 3 × 7 = 21 bits.

7.3.3 Polinomio Generador G (x) para el Código RS basado en GF (8)

La longitud de la palabra de código máxima con 3-bits de símbolos es 2^3-1 = 7 símbolos, y si t = 1, entonces G (x) que se define como (n, k) = (7,5) código RS que es capaz de corregir una falla 3 bits por símbolo.

$$G(x) = (x + a^1)\,(x + a^2)$$

$$G(x) = x * x + (a^1 * x + a^2 * x) + a^1 * a^2$$

$$G(x) = x^2 + x(a^1 + a^2) + a^3$$

$$G(x) = 1x^2 + 6x + 3$$

Este es un polinomio de grado 3 con 2 símbolos de redundancia.

Si se tiene t = 2, entonces G (x) que se define como (n, k) = (7,3), el mismo que es capaz de corregir 2 fallas de 3 bits símbolos.

$$G(x) = (x + a^1)\,(x + a^2)\,(x + a^3)\,(x + a^4)$$

$$G(x) = [x^2 + x(a^1 + a^2) + a^1 * a^2][\,x^2 + x(a^3 + a^4) + a^3 * a^4]$$

$$G(x) = (1x^2 + 6x + 3)(1x^2 + 5x + 1)$$

$$G(x) = 1x^4 + 3x^3 + 1x^2 + 2x + 3$$

Este es un polinomio de grado 4 con 4 símbolos de redundancia.

Considere un ejemplo con una palabra de código (7,5) con t = 1 utilizando G (x) = $1x^2 + 6x + 3$ y F(x) como mensaje de ingreso.

$$F(x) = 2x^4 + 3x^3 + 2x^2 + 4x + 4$$

Se multiplica F (x) * x^2 se añade dos ceros, y luego se divide para G (x).

$$F(x) * x^2 = 2x^6 + 3x^5 + 2x^4 + 4x^3 + 4x^2 + 0x + 0$$

$$(F(x) * x2) / G(x) = (2\ 4\ 1\ 5\ 4), \text{residuo } (1\ 7)$$

Para obtener la palabra de código de *Reed Solomon* se coloca el residuo (1 7) por los dos ceros.

$$C(x) = \mathbf{2}x^6 + \mathbf{3}x^5 + \mathbf{2}x^4 + \mathbf{4}x^3 + \mathbf{4}x^2 + \mathbf{1}x + \mathbf{7}$$

Los siete símbolos de 3 bits en la palabra de código entero comprenden 21 bits.

Cualquier error en cualquier carácter se puede corregir, como se muestra a continuación.

Se considera el siguiente error E(x) de un solo símbolo:

$$E(x) = 3x^6$$

Se adiciona el error a la palabra de código M(x) = C(x) + E(x):

$$C(x) = \mathbf{2}x^6 + 3x^5 + 2x^4 + 4x^3 + 4x^2 + 1x + 7$$

$$M(x) = 2x^6 + 3x^5 + 2x^4 + 4x^3 + 4x^2 + 1x + 7 + 3x^6$$

$$M(x) = \mathbf{1}x^6 + 3x^5 + 2x^4 + 4x^3 + 4x^2 + 1x + 7$$

Ahora se divide el valor de M(x) /G(x) si el resultado es cero no existe error caso contrario existe error en el mensaje recibido. Para realizar los diferentes cálculos se utiliza una calculadora de código de *Galois* [63].

$$M(x) / G(x) = (1\ 5\ 2\ 7\ 6), \text{residuo } (1\ 6)$$

Se divide el valor de E(x)/G(x)

$$E(x) / G(x) = (3\ 1\ 3\ 2\ 2), \text{residuo } (1\ 6)$$

Se puede observar que E(x) y M(x) tienen el mismo residuo (1 6) y es diferente de cero.

El residuo (1 6) corresponde al error $E(x) = 3x^6$ y se compara con el error de la palabra M (x). Este único error en M (x) se puede corregir fácilmente ya que originalmente M (x) = C (x) + E (x). Cualquier único símbolo con error podría ser corregido de esta

manera, porque cada símbolo es de 3-bits, los códigos RS proporcionan ráfagas de corrección de errores.

A continuación se proporciona una tabla con el lugar en donde se encuentra el error para diferentes valores de E(x) y para un polinomio generador $G(x) = 1x^2 + 6x + 3$, un mensaje de ingreso $F(x) = 2x^4 + 3x^3 + 2x^2 + 4x + 4$ y un polinomio primitivo del campo de *Galois* $P(x) = x^3 + x + 1$ tabla 30.

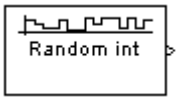
Tabla 30: Lugar de errores para deferentes valores de E(x)

	x^6	x^5	x^4	x^3	x^2	x^1	x^0
1	(6 2)	(7 2)	(7 3)	(1 1)	(6 3)	(1 0)	(0 1)
2	(7 4)	(5 4)	(5 6)	(2 2)	(7 6)	(2 0)	(0 2)
3	**(1 6)**	(2 6)	(2 5)	(3 3)	(1 5)	(3 0)	(0 3)
4	(5 3)	(1 3)	(1 7)	(4 4)	(5 7)	(4 0)	(0 4)
5	(3 1)	(6 1)	(6 4)	(5 5)	(3 4)	(5 0)	(0 5)
6	(2 7)	(4 7)	(4 1)	(6 6)	(2 1)	(6 0)	(0 6)
7	(4 5)	(3 5)	(3 2)	(7 7)	(4 2)	(7 0)	(0 7)

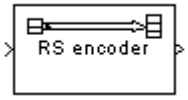
7.3.4 Simulación de Código de Corrección de Errores *Reed-Solomon*

A continuación se describirá los bloques que no han sido utilizados en simulaciones anteriores.

a) **Generador de Enteros Aleatorio**: El bloque generador de enteros aleatorio genera uniformemente distribuciones enteras aleatorios en el intervalo [0, M-1], donde M es el número M-ario que se define en el cuadro de diálogo.

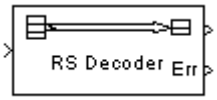


b) **Codificador RS con Ingreso de Números Enteros:** A este bloque codificador RS debe ingresar números enteros, el mismo que crea un código Reed-Solomon con la longitud del mensaje, K, y longitud de palabra de código.



c) **Bloque de Ganancia**: El bloque de ganancia multiplica la entrada por un valor constante (ganancia). La entrada y la ganancia pueden ser un escalar, vector o matriz.

d) **Decodificador RS**: El bloque decodificador RS recupera un mensaje de una palabra de código de *Reed-Salomon*. Para el correcto funcionamiento los valores deben ser enteros correspondientes al bloque aleatorio de enteros que ingresaron al codificador RS.



7.3.4.1 Simulación de la Técnica de Corrección de Errores *Reed-Salomon*

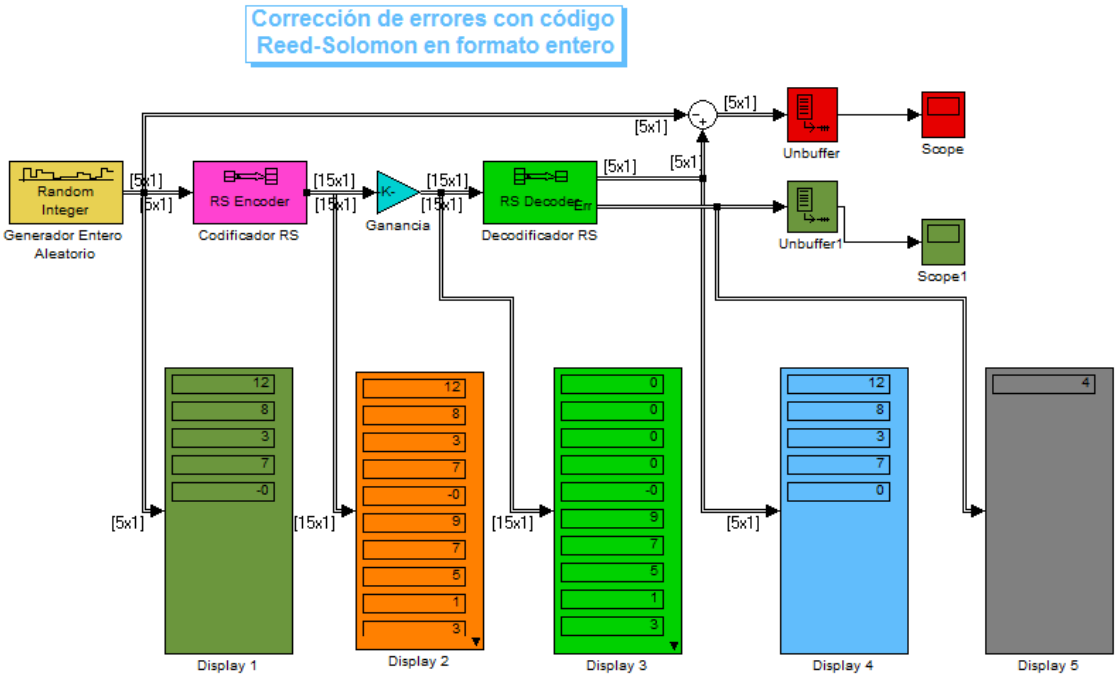


Figura 94: Simulación de la corrección de errores utilizando la técnica de Reed-Solomon

En la figura 94 se visualiza la simulación de un código Reed-Solomon en formato de números enteros, en la cual se ilustra las longitudes de los vectores apropiadas del código el ingreso y salida del codificador y decodificador, también con la utilidad del *Scope*

se puede visualizar los errores en cada palabra del código como su corrección. Los *displays* indican los siguientes parámetros:

1. *Display* 1: Valores de los códigos al ingreso del codificador RS.
2. *Display* 2: Valores a la salida del codificador RS e ingreso al bloque de ganancia.
3. *Display* 3: Valores a los que se añaden un vector [0 0 0 0 0] para el decodificador RS.
4. *Display* 4: Valores a la salida del decodificador, como se puede observar son iguales al ingreso de codificador.
5. *Display* 5: La cantidad de errores producidos iguales o menores a 5 debido al vector de ceros que se ingresó al decodificador RS.

Con la ayuda del *Scope* se puede visualizar la diferencia entre el mensaje original y el mensaje recuperado; el decodificador fue capaz de corregir todos los errores que se produjeron figura 95.

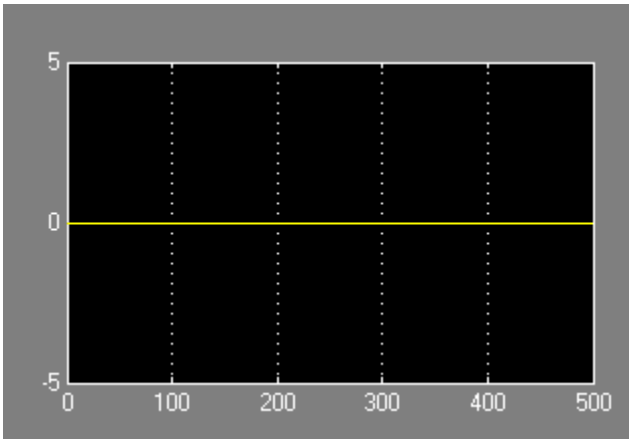


Figura 95: Resultado entre Tx y Rx

La figura 96 nos indica el número de errores que el decodificador detectado al intentar recuperar el mensaje. A menudo, el número es cinco porque el bloque de ganancia reemplaza los primeros cinco símbolos en cada palabra de código con ceros. Sin embargo, el número de errores es menor que cinco cada vez que una palabra de código correcta contiene uno o más ceros en los primeros cinco lugares.

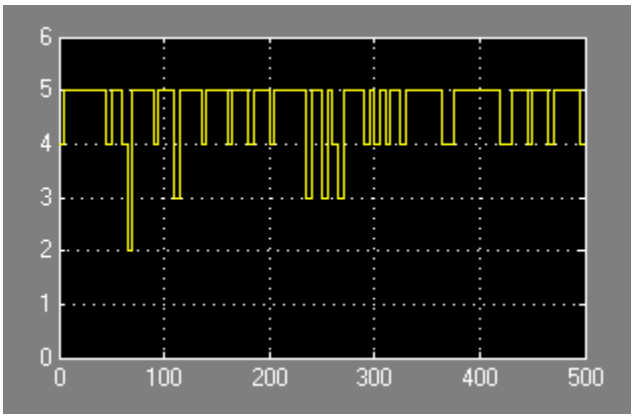


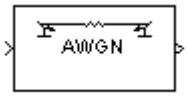
Figura 96: número de errores que el decodificador detectado

7.3.4.2 Simulación de Corrección de Errores *Reed-Salomon* con un Canal AWGN

A continuación se describirá los bloques que no han sido utilizados en simulaciones anteriores.

a) **Modulador QAM**: El bloque QAM representa un modulador rectangular de banda base que modula mediante M-aria cuadratura tanto en amplitud como en fase.

b) **Canal AWGN**: El bloque de canal AWGN añade ruido blanco gaussiano a una señal de entrada real o complejo. Cuando la señal de entrada es real, este bloque añade verdadero ruido Gaussiano y produce una señal de salida real. Cuando la señal de entrada es complejo, este bloque añade ruido gaussiano complejo y produce una señal de salida compleja. Este bloque hereda su tiempo de muestreo de la señal de entrada.



c) **Demodulador QAM**: El bloque demodulador rectangular QAM en banda base demodula una señal de que fue modulada con modulación de amplitud en cuadratura con una constelación en un entramado rectangular.

d) **Bloque de Suma**: El bloque de Suma acumulativa calcula la suma acumulada a lo largo de la dimensión especificada en la entrada o en el tiempo.

e) **Bloque de Terminación:** Se utiliza este bloque para terminar o tapar bloques que ya no tengan conexión con otros bloques.

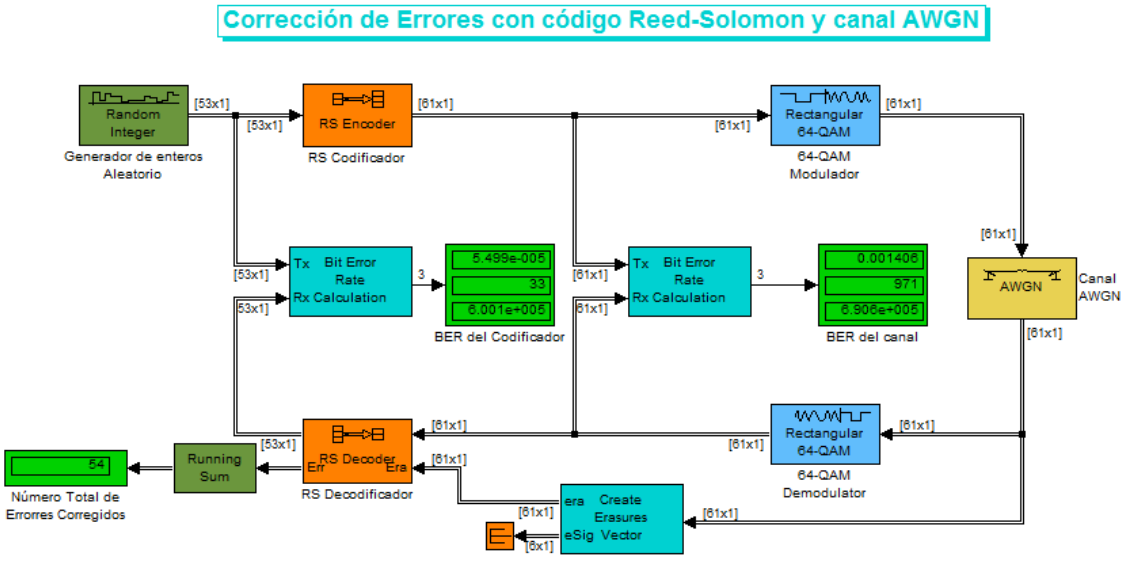


Figura 97: Simulación de la corrección de errores con código Reed-Solomon y canal AWGN

La figura 97 grafica una simulación con la utilización de la técnica de Reed-Solomon para la corrección de errores, al cual se le ha introducido un canal AWGN, por tal motivo es necesario utilizar un modulador y un demodulador QAM debido que se ha

introducido por medio del canal un ruido blanco *gausiano*, los *displays* nos indican los BER en diferentes puntos.

1. ***Display* del BER del Canal**: Indica el error producido por el canal, por tal motivo se toma las señales de entrada al modulador QAM y salida del demodulador QAM los valores son los siguientes:

   - BER = 0.001406
   - Símbolos de errores = 971
   - Símbolos Transmitidos = $6.908*10^5$

2. ***Display* del BER del Codificador**: Indica el error producido entre el transmisor Tx y el receptor Rx, por tal motivo se toma las señales de entrada al codificador RS y la señal a la salida del decodificador RS los valores que indica el *display* son los siguientes:

- BER = $5.499*10^{-5}$
- Símbolos de errores = 33
- Símbolos Transmitidos = $6.001*10^5$

3. ***Display* del número total de errores corregidos**: Indica la cantidad de errores que ha corregido el decodificador RS, que en esta simulación es de 54.

7.4 CÓDIGOS CONVOLUCIONALES.- Los códigos convolucionales son ampliamente usados en los sistemas de comunicación inalámbricos y generan bits redundantes para el chequeo y control de errores de una forma continua y tienen la ventaja de ser códigos con memoria [66].

Un código convolucional queda especificado por tres parámetros (n, k, m) [67]:

- n es el número de bits de la palabra codificada.
- k es el número de bits de la palabra de datos.
- m es la memoria del código o longitud del registro.

7.4.1 Codificación.- El codificador convolucional al ser un estado que tiene memoria se puede representar por una máquina de estados de transición, para comprender el

funcionamiento del codificador se desarrollará un ejemplo, ver figura 98, que tiene 1 bit para representar la palabra de datos, 2 bits de la palabra de codificación para cada bit de la palabra de datos y 3 bits de la memoria del código o longitud del registro (2,1,3), cuya tasa de código será k/n = 1/2.

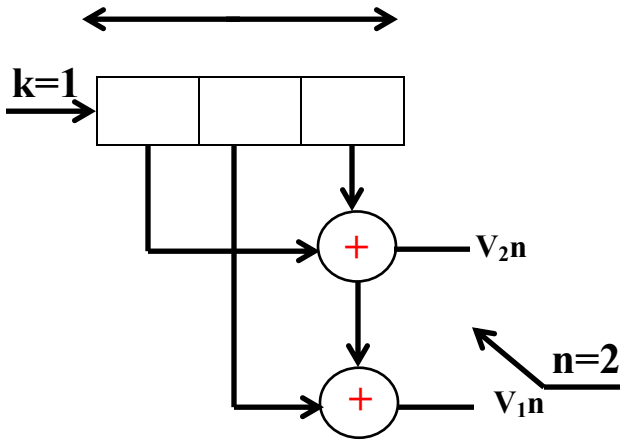


Figura 98: Memoria de un codificador convolucional (2,1,3)

- En el instante inicial el codificador está cargado de ceros, los mismos que se trasladan de un puesto cada que ingres un bit, el mismo que puede ser 0 ó 1.
- Si ingresa un 0 tenemos la máquina de estados como el de la figura 99 a y si ingresa un 1 tendremos la máquina de estados como la figura 99 b.

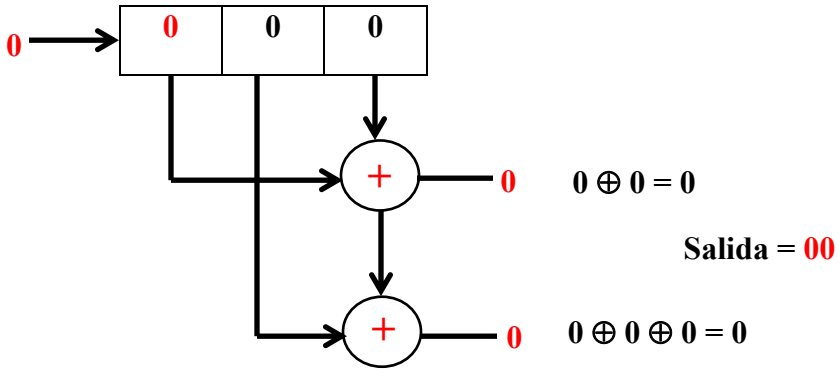


Figura 99 a: Cuando ingresa un 0

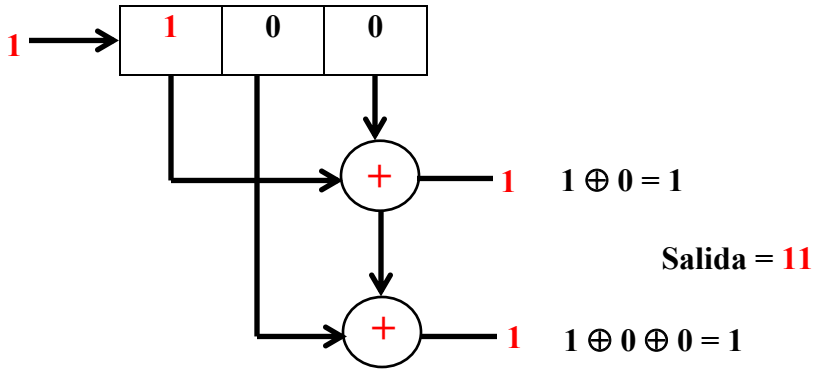


Figura 99 b: Cuando ingresa un 1

Figura 99: Comportamiento de la máquina de estados para el ingreso del primer bit

- Se parte del estado en el que se ingresó un 1 ya que con 0 se formará un bucle, es decir la máquina de estado tiene los valores (1,0,0).
- Si ingresa un 1 tenemos la máquina de estados como el de la figura 100 a y si ingresa un 1 tendremos la máquina de estados como la figura 100 b.

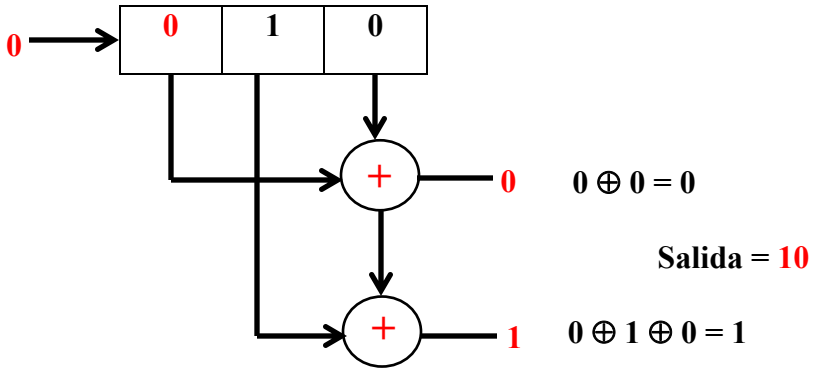


Figura 100 a: Cuando ingresa un 0

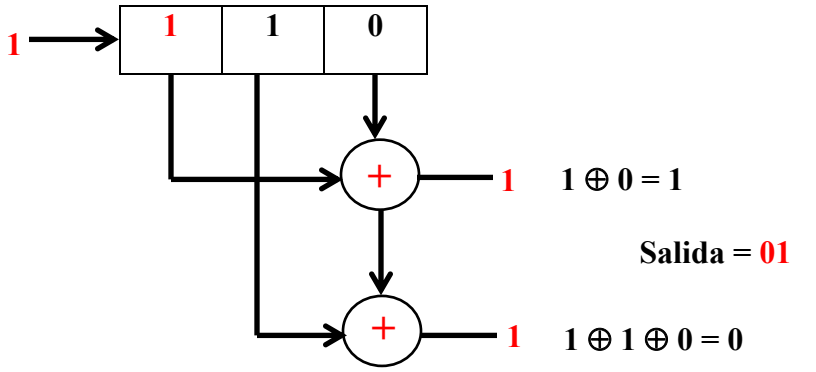


Figura 100 b: Cuando ingresa un 1

Figura 100: Comportamiento de la máquina de estados para el ingreso del segundo bit

Este proceso se realiza sucesivamente hasta completar un ciclo.

En la figura 101 se representa todo el proceso, para lo cual cada estado del codificador convolucional se representa mediante una caja y las transiciones entre los estados vienen dadas por líneas que conectan dichas cajas. Para saber de una manera rápida en qué estado se encuentra el codificador, basta con observar los dos bits del registro de memoria más alejados de la entrada.

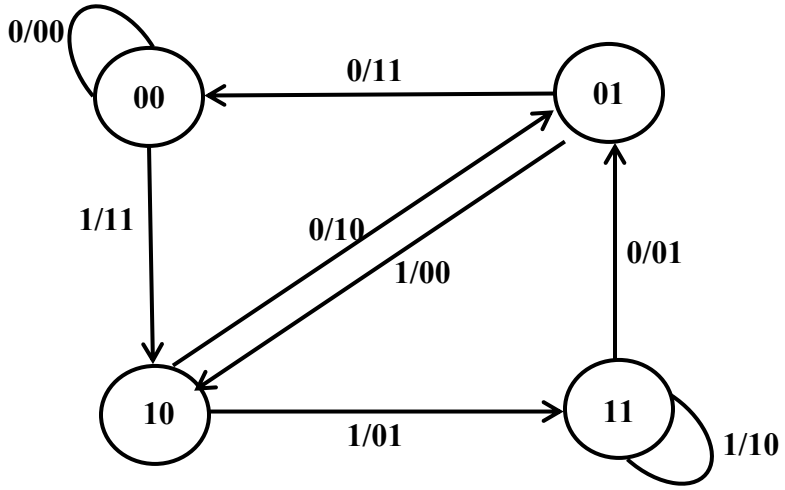


Figura 101: Estados del codificación convolucional (1,2,3)

7.4.2 Descodificación.- Para realizar la descodificación se utiliza el algoritmo de decodificación de *Viterbi*. En la etapa de descodificación los símbolos transmitidos por el canal, son reconvertidos en el mensaje original una vez que han alcanzado su destino.

Para conseguir descifrar el mensaje original a partir de los símbolos que se recibe del codificador pudiendo este haber sufrido alteraciones debido a la presencia de ruido, se necesita hacer uso de las tablas de estado siguiente y de salida mostradas en las tablas 31.

Tabla 31: Estado para el símbolo siguiente y de salida

	Estado siguiente	
Estado Actual	**Bit de entrada = 0**	**Bite de entrada = 1**
00	00	10
01	00	10
10	01	11
11	01	11

	Símbolo de Salida	
Estado Actual	**Bit de entrada = 0**	**Bite de entrada = 1**
00	00	11
01	11	00
10	10	01
11	01	10

La forma de entender la descodificación de manera adecuada se ve detalladamente gracias a los llamados diagramas de *Trellis* [66], [67].

La figura 102 muestra el diagrama de *Trellis* para un ejemplo de 15 bits de entrada:

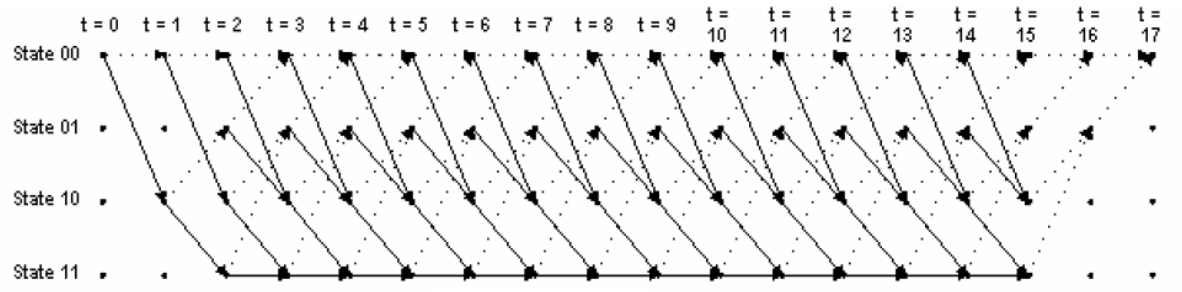


Figura 102 Diagrama de *Trellis* [66]

Para un mensaje de 15 bits como el mostrado en el ejemplo, con dos "*flushing bits*" obtenidos en la codificación, habrá 17 instantes de tiempo además del instante t=0 que representa la condición inicial del codificador.

Además, en el ejemplo, las líneas punteadas representan la transición de estado cuando el BIT que llega es un cero, y las líneas continuas, cuando el BIT es un 1.

Se puede observar también que las transiciones de estado del diagrama de *Trellis* corresponden exactamente con las tablas de estado siguiente y símbolo de salida mostradas anteriormente, y que dado que el estado inicial del codificado es 00 y que los dos "*flushing bits*" son 0, el diagrama comienza en el estado 00 y acaba en este mismo estado.

La figura 103 muestra los estados que se alcanzan en cada instante de tiempo para un mensaje de 15 bits como el siguiente:

Mensaje original: 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 0 0 marcados en rojo = *flushing bits*

Salida del codificador: 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11 10 11

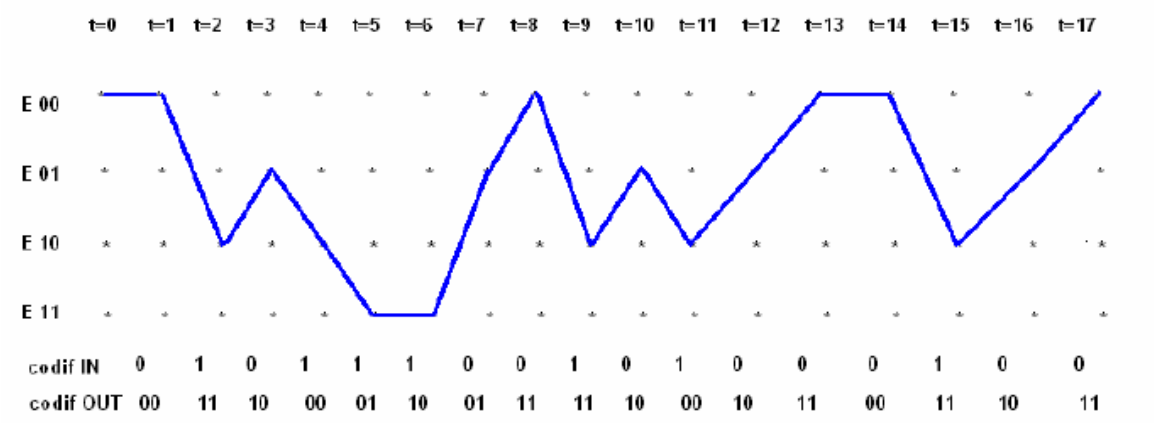


Figura 103: Estados de salida para cada uno de los 15 bits de ingreso [66]

Al recibir el siguiente mensaje codificado con unos cuantos errores producidos por la presencia de ruido:

Salida del codificador: 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11 10 11

Símbolos recibidos: 00 11 11 00 01 10 01 11 11 10 00 00 11 00 11 10 11

Cada vez que se recibe un par de símbolos, se mide la "distancia" entre lo que se ha recibido, y todos los posibles pares de símbolos que se podía haber recibido.

Por ejemplo: entre los instantes t=0 y t=1, sabiendo que el estado inicial es 00 y gracias a las tabla de salida, se determina que los únicos dos símbolos que se pude haber recibido son el 00 si el BIT de entrada es 0 o el 11 si es 1. De esta forma, se mide la medir la distancia entre estos símbolos y el símbolo realmente recibido usando para ello es la distancia de *Hamming*.

La distancia de *Hamming* se obtiene contando el número de bits diferentes que hay entre el símbolo recibido y los símbolos que podían haberse recibido en consecuencia de la transición de estado tabla 32.

Tabla 32: Cálculo de la distancia *Hamming* en t=1

Estado i-1	**Estado i**	**Símbolo Recibido**	**Símbolo Posible**	**Distancia *Hamming***
00	00	00	00	0
00	10	00	11	2

Así pues, la distancia de *Hamming* para t=1 y teniendo en cuanta que el símbolo recibido es el 00 será, ver figura 104:

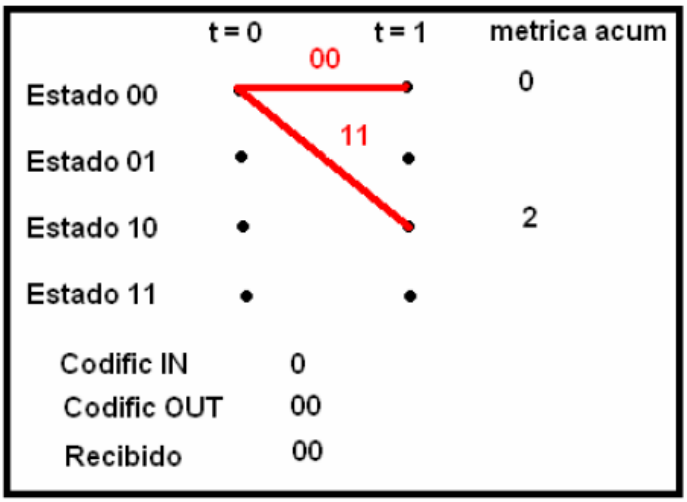


Figura 104: Diagrama de estado para t =1 [66]

Haciendo lo mismo para t=2, teniendo en cuanta que el símbolo recibido es 11, y sabiendo que podemos estar tanto en el estado 00 como en el 10, los posibles símbolos serán, ver tabla 33:

Tabla 33: Cálculo de la distancia *Hamming* en t=2

Estado i-1	**Estado i**	**Símbolo Recibido**	**Símbolo Posible**	**Distancia *Hamming***
00	00	00	00	0
00	10	00	11	2
10	01	11	10	1
10	11	11	01	1

Hay que destacar que el error métrico acumulado se corresponde con la suma del error acumulado en los estados anteriores más el error acumulado para el instante actual.

Ejemplo:

El estado 11, tiene como predecesor al estado 10, que tenía un error acumulado de 2. Se suma a este error el error actual que es 1, el error medio acumulado será 3, como muestra la figura 105.

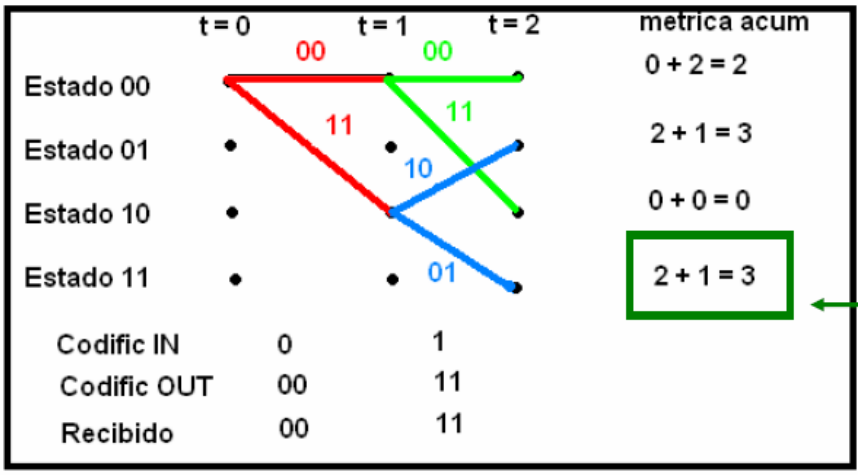


Figura 105: Diagrama de estado para t =2 [66]

El diagrama de *Trellis* para t = 3 se hace ya más complicado, ya que ahora se tiene dos caminos diferentes que se lleva desde cada uno de los cuatro estados validos de t = 2 a los cuatro estados validos de t = 3.

La manera de llevar a cabo esto es, comparar las métricas acumuladas por cada una de las dos ramas, y quedarse con aquella que sea menor. Si se encuentre que los dos valores son iguales, bastara con salvar dicho valor.

Para conseguir el mejor camino se debe quedarse con la menor de las cuatro métricas de los cuatro estados, si existen dos estados que compartan la misma métrica, ver figura 106. En este caso, se puede realizar diferentes acciones, como por ejemplo quedarse siempre con la rama de arriba o al revés. En realidad esta decisión no debería afectar al desarrollo de la decodificación.

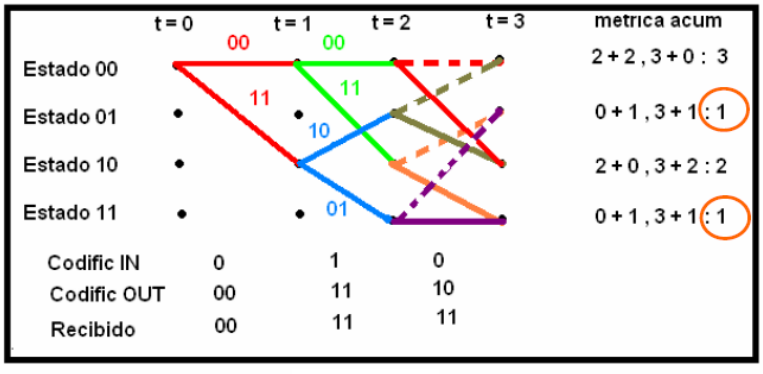


Figura 106: Diagrama de estado para t =3 [66]

Nótese como en este ejemplo, el símbolo codificado no es el mismo que el recibido. Esto es debido a la presencia de ruido. Si continuásemos desarrollando este diagrama de la misma manera hasta llegar a t = 17 se obtendría el siguiente resultado final, ver figura 107.

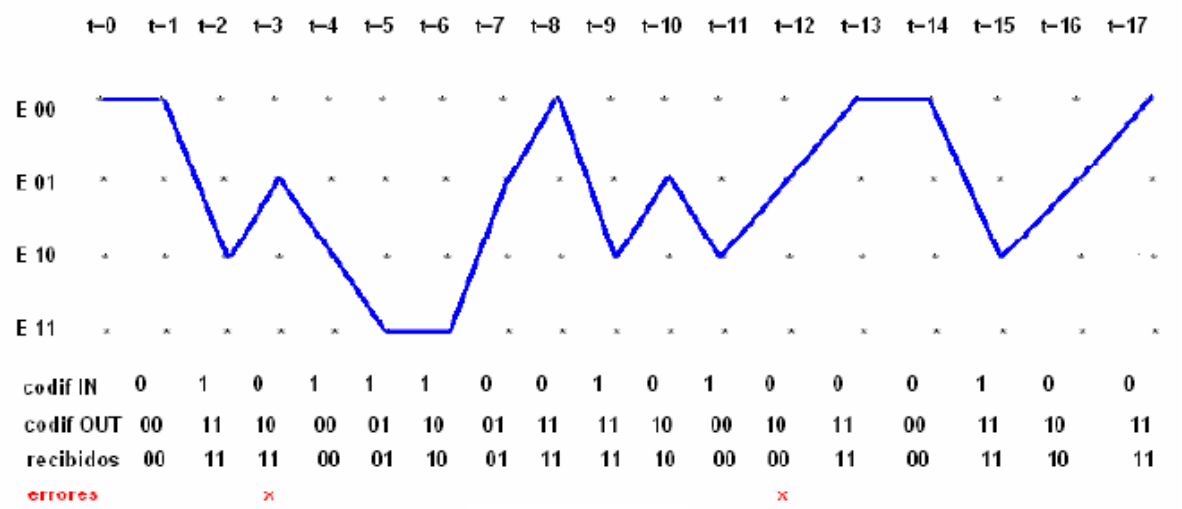


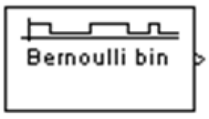
Figura 107: Obtención de los errores [66]

7.4.3 Conclusión.- Existe la corrección de error en t=3, a pesar de que existe un error a la salida del codificador (10) y los bits recibidos (11), pero debido al diagrama de *Trellis* y al código de *Hamming* dicho valor corresponde al bit 0, el mismo análisis se puede realizar para t = 11, por esta razón los códigos en el emisor son iguales a los del receptor, ver figuras 103 y 107.

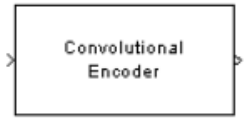
7.4.4 Simulación

Para realizar la simulación se ha partido de los siguientes bloques:

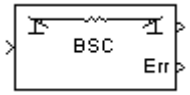
a) **Generador Binario de *Bernoulli*:** Es un generador de números binarios de forma aleatorios, el mismo que utiliza la distribución de *Bernoulli*. La distribución de *Bernoulli* produce ceros con una probabilidad p y unos con una probabilidad 1-p. La distribución de *Bernoulli* tiene un valor promedio de 1-p y una varianza de p(1-p).



b) **Codificador Convolucional:** El bloque codificador convolucional codifica una secuencia de vectores de entrada binarios para producir una secuencia de vectores de salida binarios. Este bloque puede procesar múltiples símbolos a la vez. Este bloque puede aceptar entradas que varían en longitud durante la simulación.

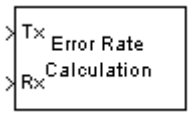


c) **Canal Binario Simétrico**: El objetivo del canal es introducir errores binarios en la señal transmitida. Este bloque acepta una señal de entrada escalar o vectorial. El bloque procesa cada elemento del vector de forma independiente, y se introduce un error en un lugar determinado con una probabilidad de error.



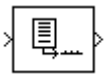
d) **Decodificador *Viterbi***: El decodificador de *Viterbi* descodifica los bits de entrada para producir símbolos binarios de salida. Este bloque puede procesar varios símbolos a la vez para un rendimiento más rápido. Este bloque puede dar a sus secuencias de bits que varían en longitud durante la simulación.

e) **Cálculo de la Tasa de error**: Este bloque realiza el cálculo del BER dividiendo el número total de bits erróneos recibidos para el número total de bits transmitido o producidos por el Generador Binario de *Bernoulli*. Si las entradas son bits, entonces el bloque calcula la tasa de error de bit. Si las entradas son símbolos, entonces se calcula la tasa de error de símbolo.



f) **Sin buffer**: El bloque sin buffer tiene una entrada Mi-por-N y una salida de 1-por-N. Es decir, las entradas son sin búfer y en modo de fila, de tal manera que cada fila de la matriz se convierte en una muestra independiente de tiempo en la salida. La

velocidad a la que el bloque recibe las entradas es generalmente menor que la velocidad a la que el bloque produce las salidas.



g) **Operador Relacional:** El bloque del operador relacional compara dos entradas usando el parámetro del operador relacional que se especifique. La primera entrada corresponde al puerto de entrada superior y la segunda entrada al puerto de entrada inferior.

7.4.4.1 Simulación de Corrección de Errores Utilizando Códigos Convolucionales

El bloque de operador relacional compara la señal transmitida, que viene desde el bloque de generador aleatorio de Bernoulli, con la señal recibida, procedente del bloque decodificador *Viterbi* como se indica en la figura 108. El bloque de salida entrega un 0 cuando las dos señales son iguales y un 1 cuando son diferentes.

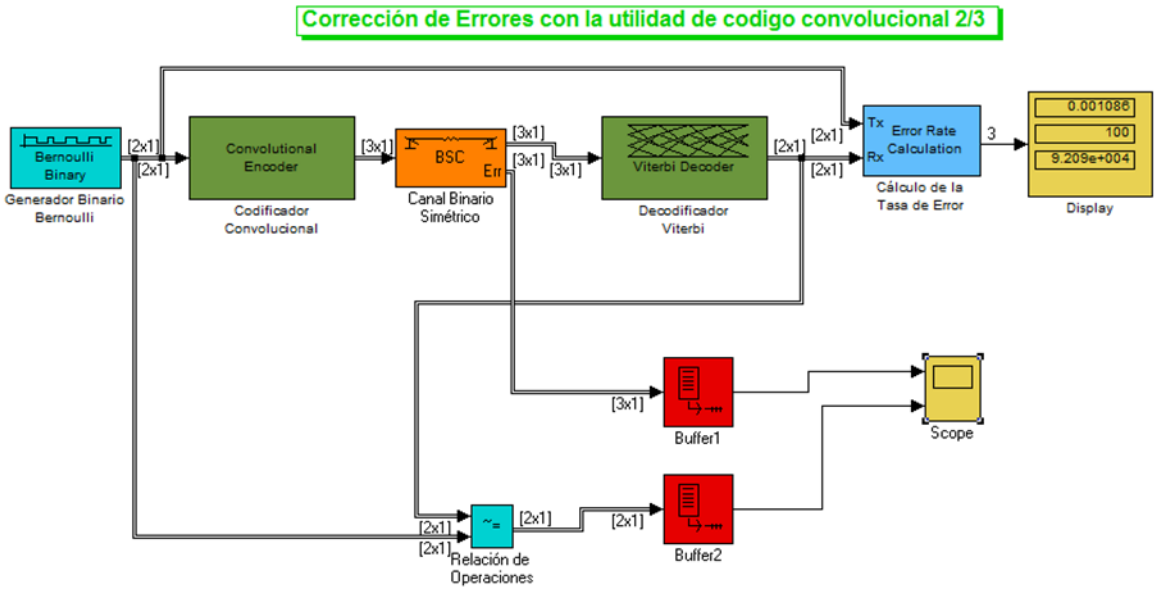


Figura 108: Simulación de corrección de errores por medio de códigos convolucionales

Cuando se ejecuta la simulación se muestran los datos de error. Al final de cada uno de los 5000 pasos de tiempo, el ámbito de aplicación aparece como se muestra en la figura 109, luego borra los datos mostrados y muestra los siguientes 5.000 puntos de datos.

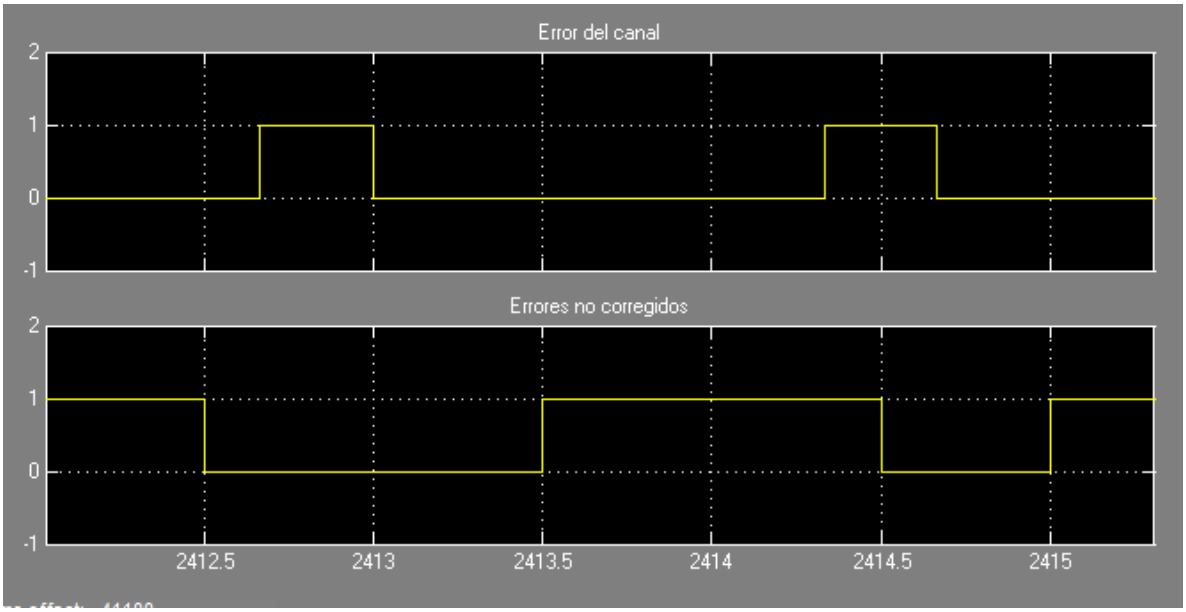


Figura 109: Corrección de errores

La figura superior muestra los errores de canal generados por el bloque de canal binario simétrico. El ámbito inferior muestra los errores que no se corrigieron debido al decodificador.

Si se analiza la figura 109 se puede observar que el pulso rectangular ubicado a la izquierda de la figura superior representa dos errores que se corrigen. El pulso ubicado a la derecha de la figura superior representa 2 códigos, de los cuales solamente uno de ellos se corrige.

7.5 RESULTADOS

A continuación se presenta una serie de resultados obtenidos al realizar las diferentes simulaciones con la ayuda del *Simulink* de ***MatLab.***

Los resultados se obtuvieron de los siguientes análisis:

1. Estudio del comportamiento del código BCH para diferentes valores de trama.

2. Comportamiento General del código BCH, *Hamming* y Convolucional para una valor de n = 7 y k=4 bits (7,4).

7.5.1 Comportamiento del Código BCH para Diferentes Valores de Trama

En la figura 110 se ha simulado un código BCH (15,11,1) en donde se ha obtenido que los primeros errores (BER = 0.00974) se dan cuando se ha trasmitido 308 bits o 28 tramas de 11 bits de datos como se puede observar en la figura 111.

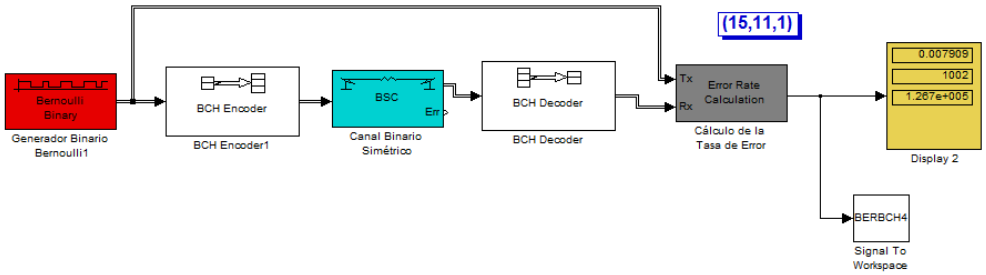


Figura 110: Simulación de un código BCH (15,11,1)

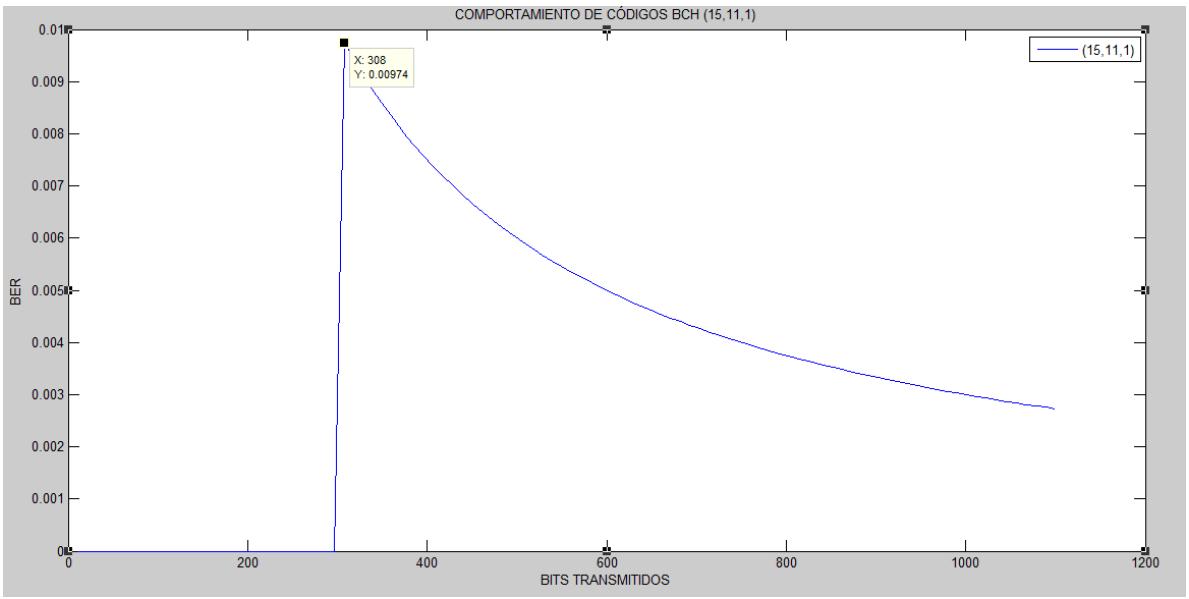


Figura 111: Comportamiento de un código BCH (15,11,1)

En la figura 112 se ha simulado un código BCH (15,7,2) en donde se ha obtenido que los primeros errores (BER = 0.002268) se dan cuando se ha trasmitido 882 bits o 126 tramas de 7 bits da datos como se puede observar en la figura 113.

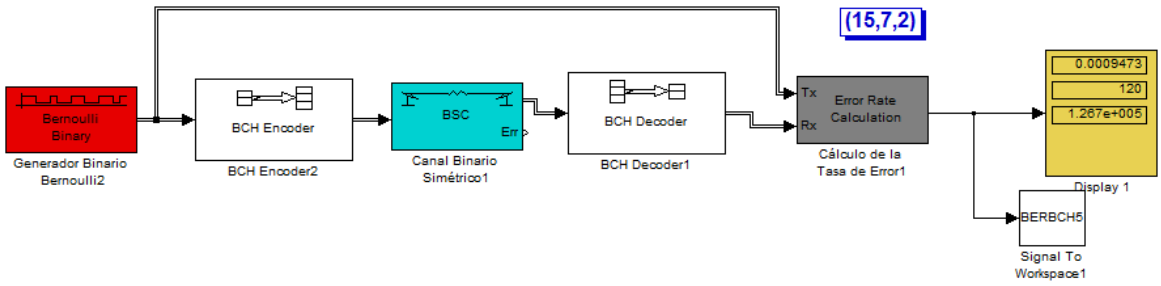


Figura 112: Simulación de un código BCH (15,7,2)

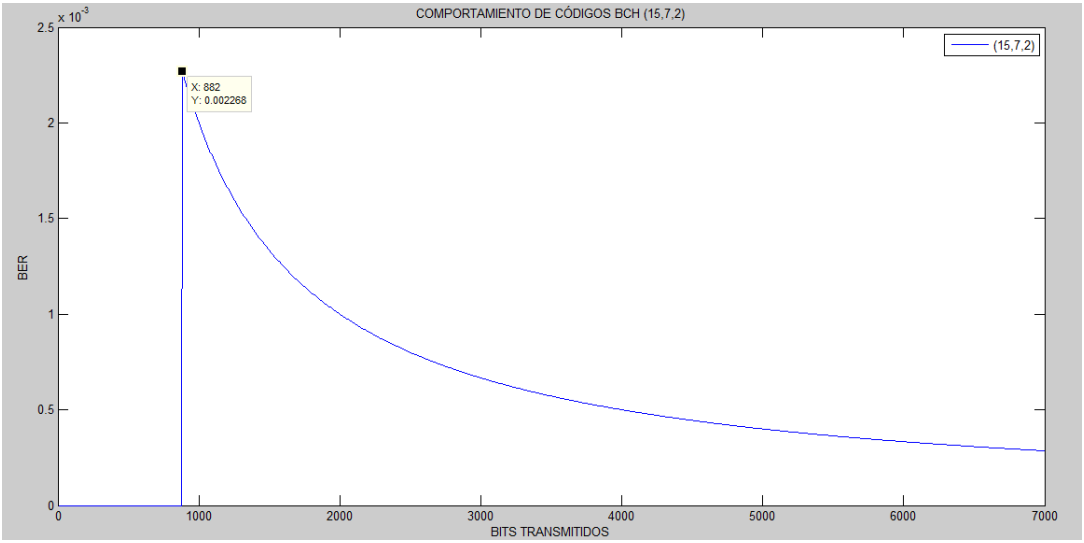


Figura 113: Comportamiento de un código BCH (15,7,2)

En la figura 114 se ha simulado un código BCH (15,5,3) en donde se ha obtenido que los primeros errores (BER = 0.000111) se dan cuando se ha trasmitido 18030 bits o 3606 tramas de 5 bits de datos como se puede observar en la figura 115.

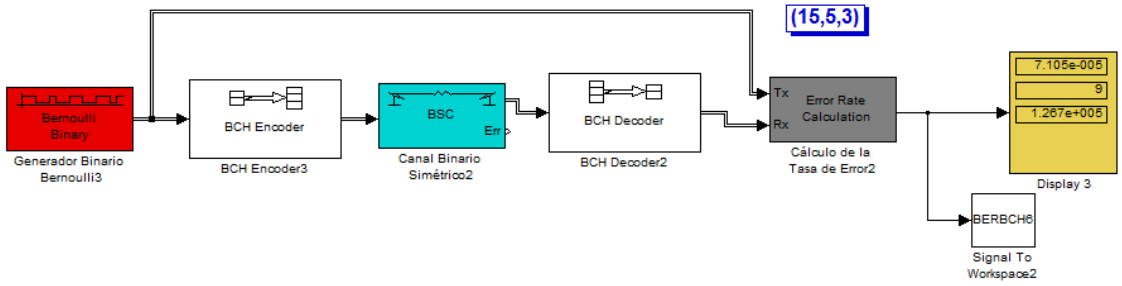


Figura 114: Simulación de un código BCH (15,5,3)

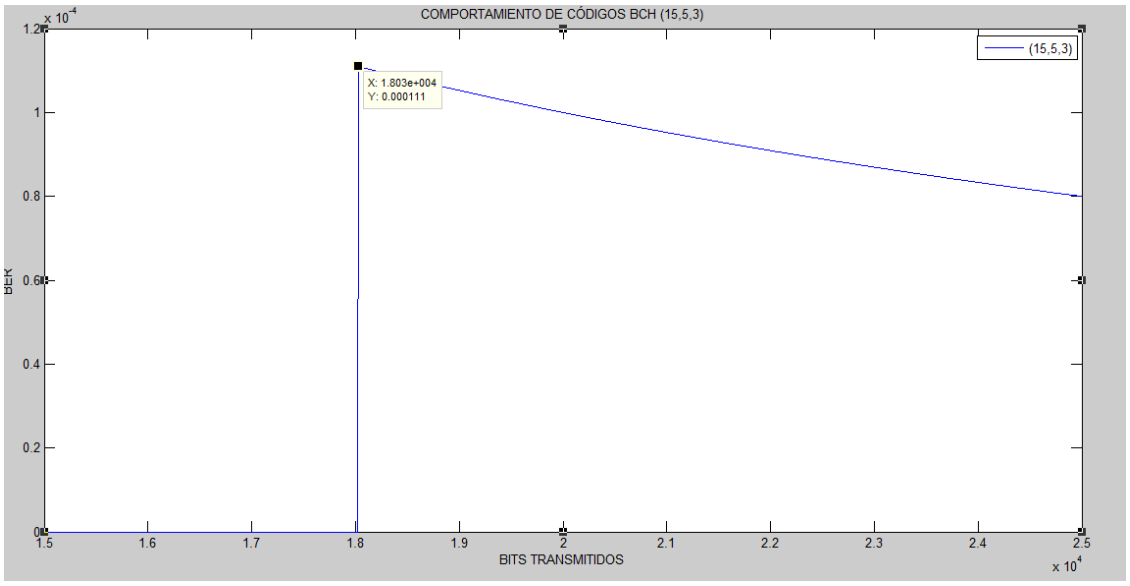


Figura 115: Comportamiento de un código BCH (15,5,3)

En la figura 116 se puede observar los resultados en forma general del comportamiento del código BCH para diferentes tramas de datos: (15,11,1) (15,7,2) y (15,5,3) siendo el valor de n =15 que es la longitud total de la trama, los valores de 11, 7 y 5 la longitud de los bits de datos y los valores de 1,2,3 los bits de corrección en cada trama, siendo la que presenta un menor BER el código BCH (15,5,3), debido a que existe tres bits de corrección por tal motivo la tasa de error BER va hacer menor para cualquier cantidad de bits transmitidos.

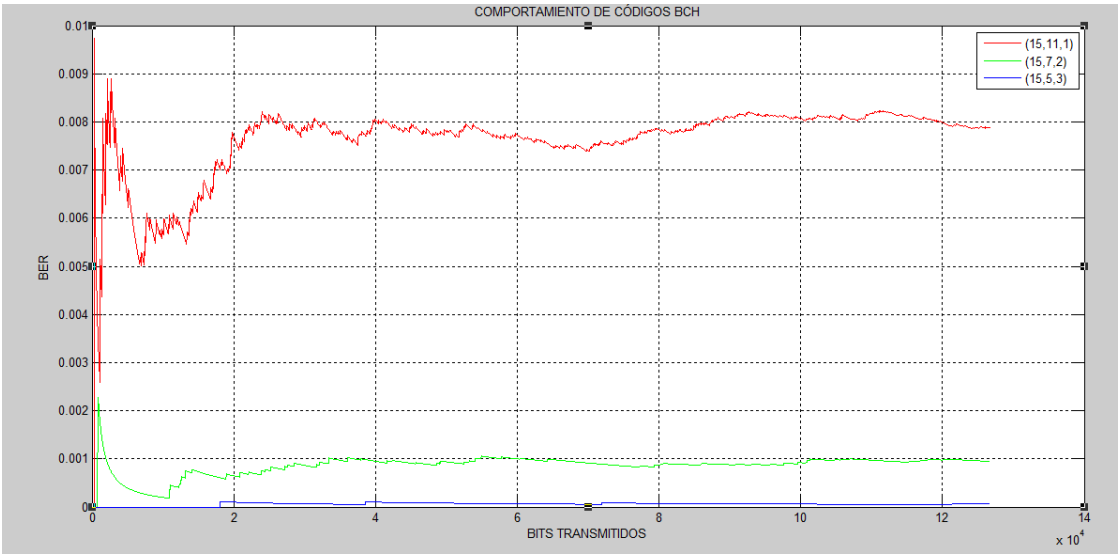


Figura 116: Comportamiento de un código BCH (15,11,1), (15,7,2) y (15,5,3)

7.5.2 Comportamiento General del Código BCH, *Hamming* y Convolucional para una Valor de n = 7 y k=4 bits (7,4)

Si se analiza la figura 118 se puede observar el comportamiento del código *Hamming* que tiene un BER de 0.02273 cuando se ha transmitido 44 bits o 11 tramas de 4 bits de datos.

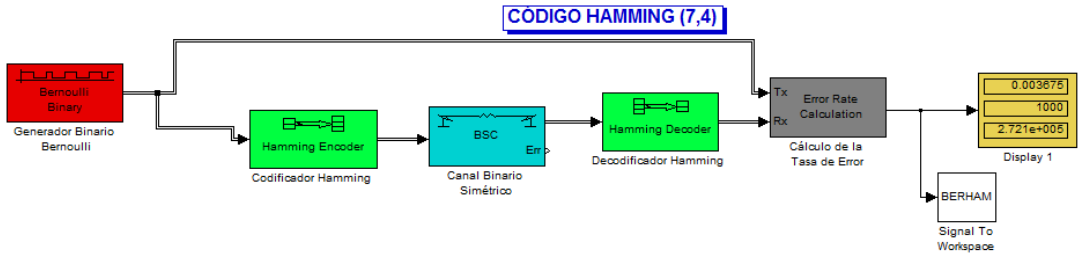


Figura 117: Simulación del código *Hamming* (7,4)

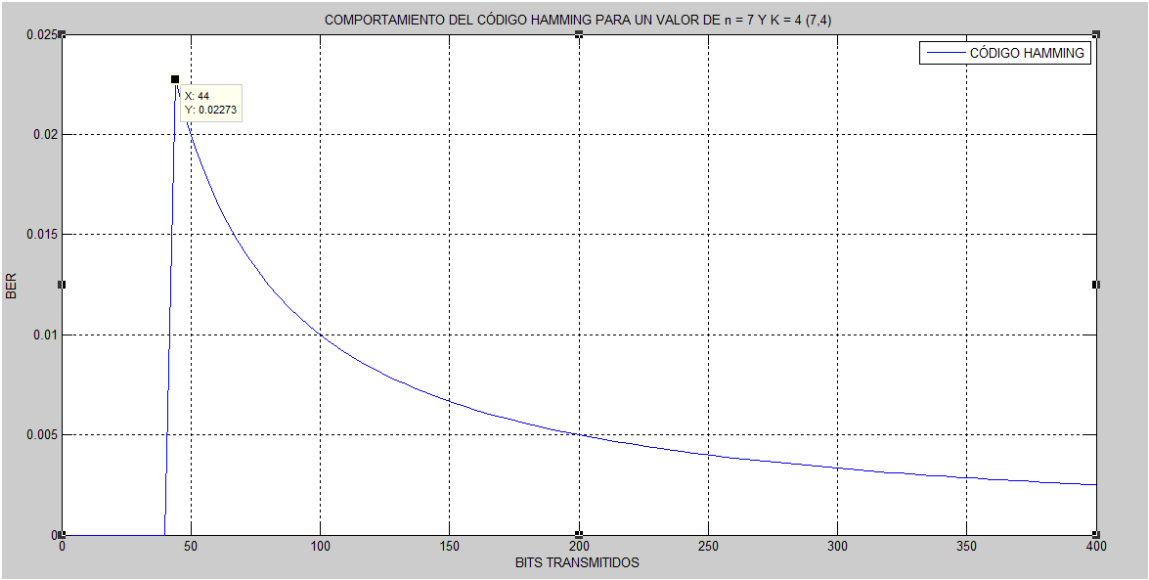


Figura 118: Comportamiento del código *Hamming* (7,4)

En la figura 120 se puede observar el comportamiento del código BCH que tiene un BER de 0.02273 cuando se ha transmitido 44 bits o 11 tramas de 4 bits de datos, este resultado es igual a la del código *Hamming*.

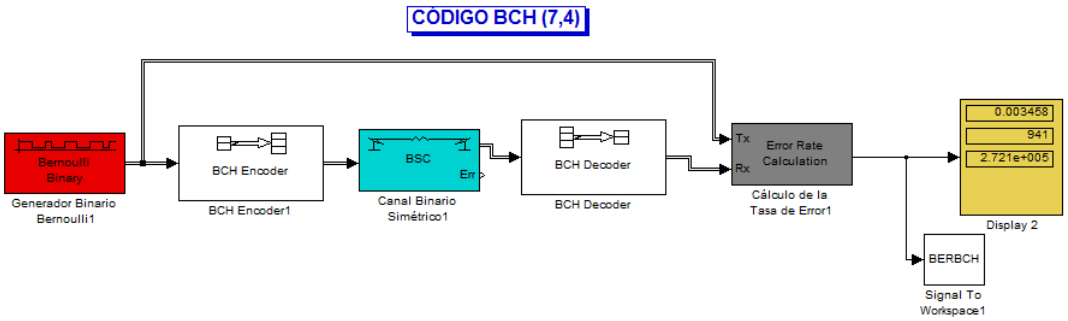


Figura 119: Simulación del código BCH (7,4)

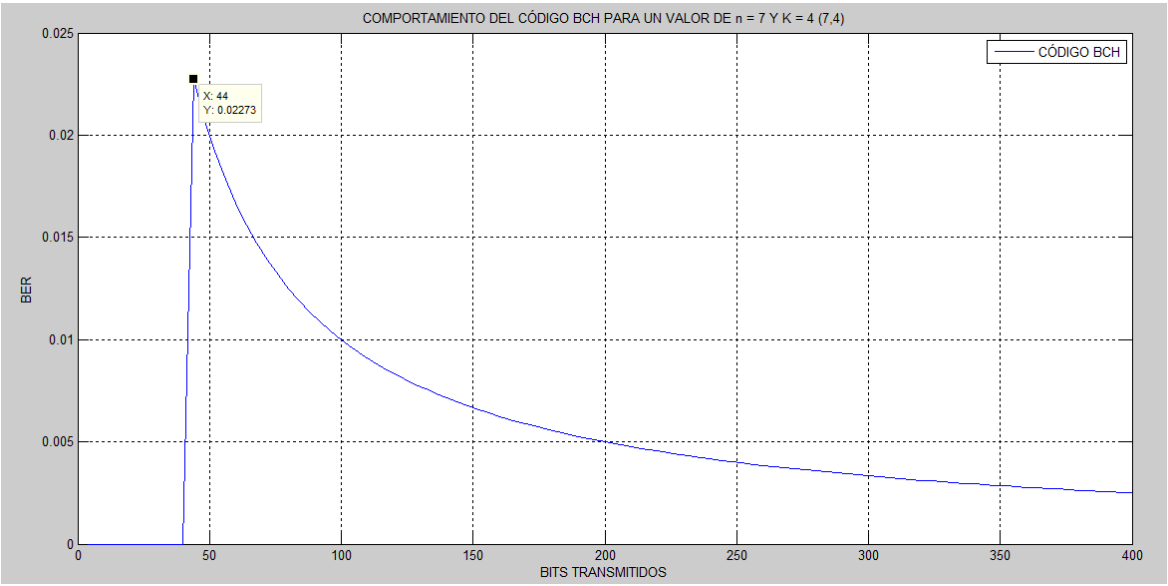


Figura 120: Comportamiento del código BCH (7,4)

En la figura 122 se puede observar el comportamiento del código Convolucional que tiene un BER de 0.0005371 cuando se ha transmitido 7448 bits o 1879 tramas de 4 bits de datos.

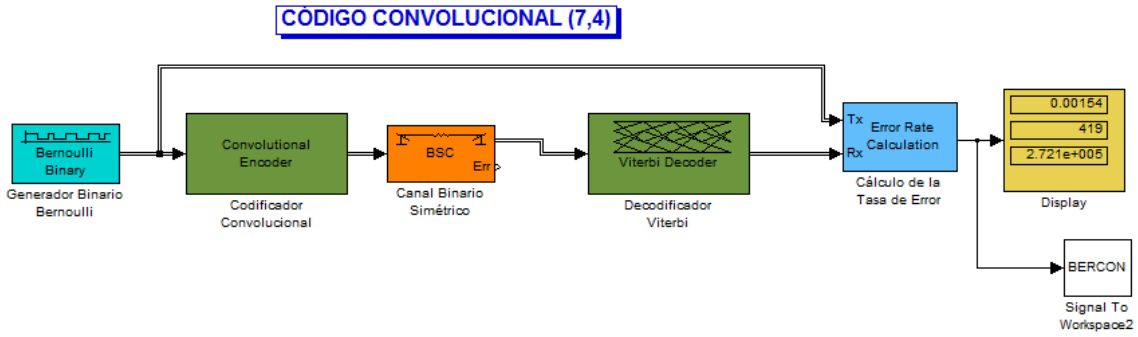


Figura 121: Simulación del código convolucional (7,4)

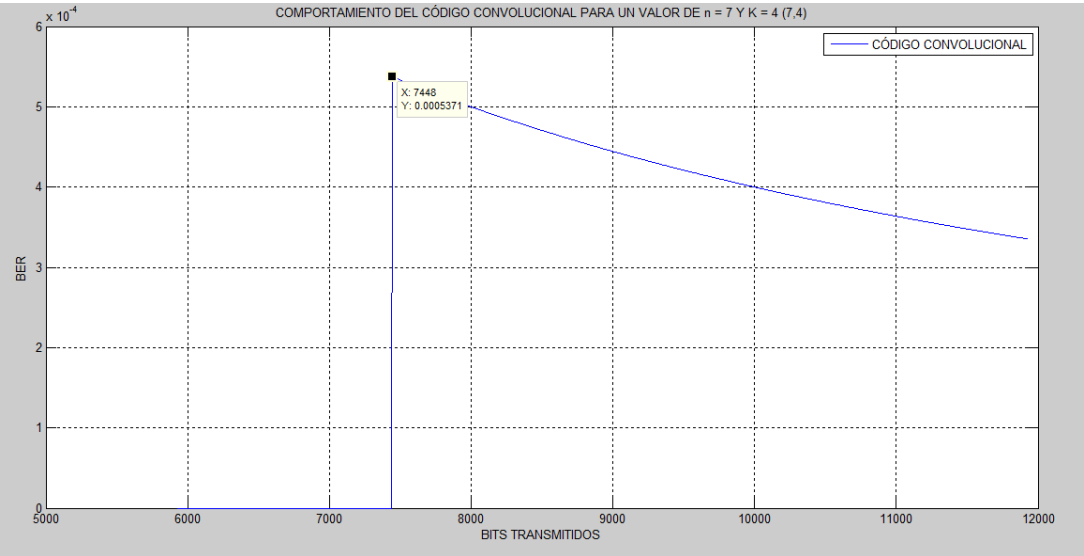


Figura 122: Comportamiento del código convolucional (7,4)

Si se analiza la figura 123 en donde el BER está en función del número de bits transmitidos, se puede observar que tiene un mejor comportamiento el código convolucional para la corrección de errores con relación al código *Hamming* y BCH, el resultado del BER que nos entrega la figura 123 para una transmisión de 25000 bits es de 0.003617 para los códigos *Hamming* y BCH en cambio para esa misma cantidad de bits transmitido el BER del código convolucional es de 0.001698, siendo este código un 27.7% más efectivos que los anteriores.

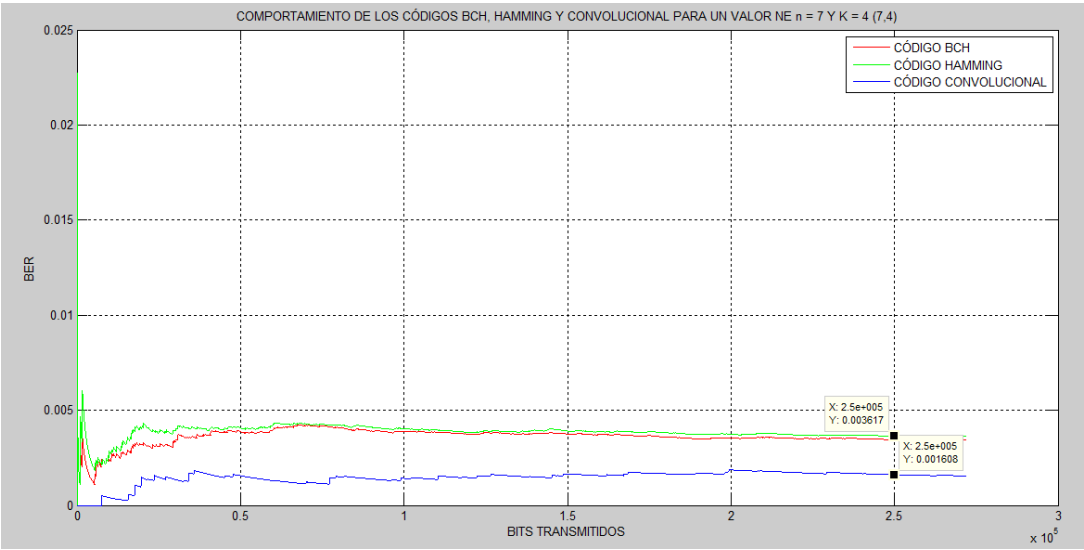


Figura 123: Comportamiento del código BCH, *Hamming* y convolucional en función del número de bits transmitidos para n = 7 y k = 4 (7,4)

La figura 124 nos indica la relación del BER en función del número de tramas de 4 bits de datos, en donde para 60.000 tramas el BER del código *Hamming* y BCH es de 0.003679, mientras que para esa misma cantidad de tramas el código convolucional tiene un BER de 0.001663, siendo este código un 27.18% más efectico para la corrección de errores que los otros dos códigos.

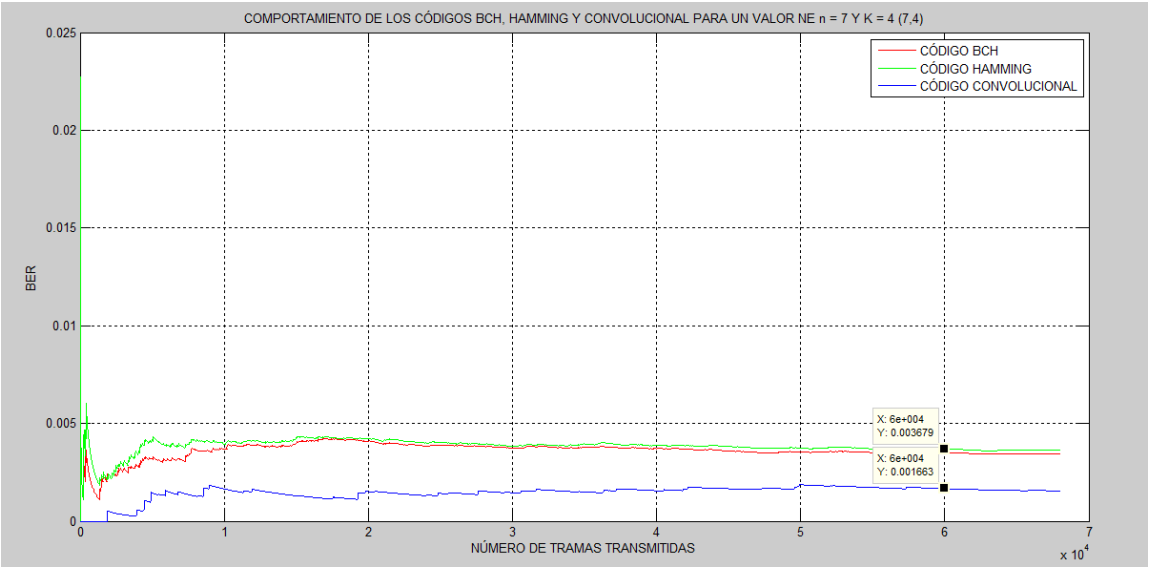


Figura 124: Comportamiento del código BCH, *Hamming* y convolucional en función del número de tramas de 4 bits de datos transmitidas para n = 7 y k = 4 (7,4)

CONCLUSIONES

Una vez realizado esta investigación sobre Análisis, Modelado y Simulación de Algoritmos sobre Técnicas de Comunicación Tolerante a Fallas Aplicadas en Sistemas Industriales se llegó a las conclusiones siguientes:

El protocolo que actualmente se está utilizando a nivel industrial es ModBus por sus ventajas y sobre todo por ser un protocolo abierto, por este motivo se realizó un estudio de todas sus características como de las principales funciones que utiliza esta comunicación entre un maestro y una unidad de terminal remota (RTU).

Es importante realizar un estudio detallado del Código de Redundancia Cíclica (CRC) que tienen al final de cada trama ModBus con la finalidad de implementar otros códigos y analizar su comportamiento frente a futuras fallas que se pueden dar en las comunicaciones industriales.

En las comunicaciones digitales al transmitir los datos a través de un canal se producen errores por presencia del ruido u otros factores como el eco la interferencia intersímbolo lo que provoca una pérdida de bits de información que prácticamente puede afectar a la transmisión de datos. Por este motivo se realizó un estudio de las diferentes técnicas para detectar y corregir los errores.

En este documento se desarrolló y se simuló diferentes técnicas para detectar los errores y a su vez se aplicó algunos códigos especiales para agregar redundancia. El objetivo de trabajar con estos bits de redundancia es corregir los errores ocurridos durante la transmisión de bloques de datos.

Al implementar las simulaciones con la ayuda de Simulink de MatLab se pudo determinar que se detectaron los errores y por medio de diferentes técnicas se llegaron a corregir los errores como se puede obtener en diferentes gráficas y en el cálculo del BER para cada una de las simulaciones.

Es importante resaltar que en este trabajo se llegó determinar algunos resultados sobre el comportamientos del código BCH para diferentes valores de trama como también

se realizó una comparación entre tres técnicas de corrección de errores: BCH, Hamming y Convolucional.

Como propuesta se sugiere realizar un estudio sobre las diferentes fallas que se pueden implementar a nivel industrial y realizarlos prácticamente con el objetivo de implementar sistemas tolerantes a fallas.

Es importante realizar un estudio detallado del Código de Redundancia Cíclica (CRC) que tienen al final de cada trama ModBus con la finalidad de implementar otros códigos y analizar su comportamiento frente a futuras fallas que se pueden dar en las comunicaciones industriales.

REFERENCIAS

[1] Domingo Juan, Gámiz Juan, Grau Antonio, Martínea Herminio (2003) *"Comunicaciones en el Entorno Industrial*" Editorial UOC, Primera Edición

[2] http://www.sisman.utm.edu.ec/libros/

[3] Revista (2013) Automática e Instrumentación: "*Comunicaciones Industriales*" Información Económica Industrial, http://www.automaticaeinstrumentacion.com/?p=344161

[4] Kaschel Hector, Pinto Ernesto. "*Analisis del Estado del Arte de los Buses de Campo Aplicados al Control de Procesos Industriales*", Facultad de Ingeniería, departamento de Ingeniería Eléctrica, Universidad de Santiago de Chile.

[5] Zhao Jue ; Yang Shun (2012) "*Design of ModBus-Profibus Fieldbus Bridge Based on the STM32 and VPC3 + C*" Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference, Page(s): 411 – 414

[6] Comunicaciones Industriales "*Profibus (PA/DP/FMS)"* Universidad Politécnica de Cartagena ETS de Ingenieros Industriales Dpto. de Tecnología Electrónica

[7] Li Hui; Zhang Hao; Peng Daogang, (2009) "*Research and Application on Interbus Operator Terminal*", Computer Science and Information Technology, 2009. IEEE International Conference on Digital Object Identifier: 2009, Page(s): 309 - 312.

[8] http://www.anybus.com/technologies/interbus.shtml

[9] Schiffer, V.; Vandesteeg, K.W.; Vasko, D.A.; Lenner, J.A. (2000) "*Introduction to DeviceNet Safety*" Factory Communication Systems, IEEE International Workshop on Digital Object, Page(s): 293 - 300.

[10] http://www.instrumentacionycontrol.net/cursos-libres/automatizacion/curso-redes-industriales/item/278-la-red-devicenet-interconectando-dispositivos-de-control-para-el-intercambio-de-datos.html

[11] Villajulca José Carlos. (2010) "*La Red DeviceNet: Interconectando Dispositivos de Control para el Intercambio de Datos, Instrumentación y Control.net*" Agosto 2010.

http://www.instrumentacionycontrol.net/cursos-libres/automatizacion/curso-redes-industriales/item/278-la-red-devicenet-interconectando-dispositivos-de-control-para-el-intercambio-de-datos.html

[12] Pantoni, R.P.; Brandao, D.; Torrisi, N.M.; Mossin, E.A. (2008), "*Integration of an open and non-proprietary device description technology in a foundation fieldbus simulator"* Factory Communication Systems. IEEE Page(s): 435- 444

[13] Foundation fieldbus Solutions from ABB (2013) "*The integration of FOUNDATION fieldbus into IndustrialIT takes full advantage of the advanced features offered by the protocol*". http://www.abb.com/cawp/gad02181/c1256d71001e0037c1256b5a003158ce.aspx

[14] http://www.eaeeie.org/ineit-mucon/html/sensors/sensorEN/desc-eng.html

[15] Huang Xiao-ping ; Li Jian-liang (2012), "*The Application of LonWorks in the Building Air Conditioner Intelligent Control System*" Intelligent Human-Machine Systems and Cybernetics (IHMSC), IEEE 2012 , Page(s): 205- 207

[16] http://dz.prosyst.com/pdoc/mbserver_5.0/Tutorial/lon/bundles/overview/overview.html

[17] Catálogo lamonde for Automation (2011) "*SDS (Honeywell) DL205 I/O base controller (F2-SDS-1)*"

http://www.lamonde.com/acatalog/SDS__Honeywell_Smart_Distributed_System__Field_IO.html

[18] Belden Sending All The Right Signals (2013) "*Smart Distributed Systems – SDS*" http://www.belden.com/products/industrialcable/smart-distributed-systems-sds.cfm

[19] Farsi, M.; Ratcliff, K.; Barbosa, M. (1999) "*An introduction to CANopen*" Computing & Control Engineering Journal Volume: IEEE, Page(s): 161- 168

[20] Peñaloza Calderón, J. A. (2009). *"Red de PLC´ S y variadores de velocidad con protocolos Ethernet y Modbus*" Universidad Pontificie Bolivariana.

[21] Forouzan Behrouz (2007), "*Transmisión de Datos y Redes de Comunicación"*, Cuarta Edición McGraw Hill.

[22] Briseño Márquez José E (2005) "*Transmisión de Datos"*, Universidad de los Andes, Facultad de Ingeniería Mérida Venezuela.

[23] Alonso Miguel José (1996), "*Protocolos de Comunicación para Sistemas Abiertos",* Buenos Aires, Adison-Wesley Iberoamericana.

[24] Gil Pablo y otros. (2010) "*Redes y Transmisión de datos"*, Universidad de Alicate.

[25] Desset C. Vandendorpe L., Macq B.(2004) "*Computing the Word-Symbol and Bit Error Rates for Block Error-Correcting Codes*". IEEE Trans Commun. Jun 2004.

[26] Aguilera. (2010) "*Seguridad Informática"*, Editorial Editex Madrid 2010.

[27] Tabirca, T.; Ciurea, E.; Tabirca, S. (1998) "*Rules for Hamming-Code Generating*" Optimization of Electrical and Electronic Equipments, IEEE 1998.

[28] Bastidas Ibañez Javier (2012), "*Sistemas Tolerantes a Fallos"*, Departamento de Informática, universidad de Valladolid.

[29] http://www.angelfire.com/ak2/karenteamo/tdatos.html

[30] Herrera Pérez Enrique (2004), "*Comunicaciones Digital y Ruido"*, Editorial Limusa S.A, México D.F.

[31] Bhuia C.T. (2005) *"Informmation Technology Network and Internet",* New Age International Publishers New Delhi, First Edition 2005.

[32] MODBUS IDA. (2006) "*Modbus Application Protocol Specification V1.1b*". United States of America: Modbus-Ida, 2006.

[33] Dao-gang Peng; Hao Zhang; Li Yang; Hui Li (2008) "*Design and Realization of Modbus Protocol Based on Embedded Linux System*" Embedded Software and Systems Symposia, 2008. Publication IEEE: 2008, Page(s): 275 – 280.

[34] Vélez Correa José, Cervera Manjares Norwin (2005) "*Diseño e Implementación de una Red Industrial Bajo Protocolo Modbus que Permita la Comunicación Digital con*

Instrumentos de Campo", Corporación Universitaria Rafael Núñez, Facultad de Ingeniería de Sistemas Cartagena De Indias D.T Y C. Marzo 2005.

[35] Modbus Plus de Modicon (2004), "*Guía para la Planificación y la Instalación de la Red*" Versión 6.0 890 USE 100 03 Noviembre 2004.

[36] Hernández Joffre Víctor Hugo, Jiménez Anaya Eduardo, Medina Suzunaga Gabriel, Montero Guerrero Paola (2009) "*Control de Dispositivos Electrónicos Mediante Protocolo Modbus RS-485 y SMS*" Instituto Politécnico Nacional, Escuela Superior de Ingeniería Mecánica, México Noviembre 2009.

[37] Modicon Modbus Protocol (1996) "*Reference Guide*" PI–MBUS–300 MODICON, Inc., Industrial Automation Systems June 1996.

[38] www.arcesio.net/modems/rs232.ppt

[39] http://perso.wanadoo.es/pictob/comserie.htm

[40] Francisco Andrés Candelas Herías (2011) "*Comunicación con RS-485 y MODBUS*", Universidad de Alicante

[41] MODBUS (2002) "*Over Serial Line Specification & Implementation guide V1.0*" http://www.modbus.org/docs/Modbus_over_serial_line_V1.pdf

[42] Manual de instrucciones (2008) Central de medida 710 "*Manual de referencia Schneider*" http://tecno-plc.com/joomla/images/stories/pdfs/PM710.pdf

[43] Adam Christian Gabriel y otros (2006), "*Software Educativo Emulador de Detección y*

Corrección de Errores en Transmisión afectadas por Inconvenientes en la Señal", Universidad Tecnológica Nacional, Facultad Regional de Córdova.

[44] Tenembau Andrew S (2003), "*Redes de Computadora"*, Vrije Universiteit, Amsterdam, The Netherlands, Traducción Elisa Núñez Ramos, Pearson Educación, México.

[45] Hsu John Y (1996), "*Computer Networks: Architecture, protocols and software"*, Artech House 1996.

[46] Gibson Jerry (1989) "*Principles of Digital and Analog Communications*", Second Edition, New York, Macmillan.

[47] Qin, Yang; Lie-Liang (2010), "*Throughput Analysis of Stop-and-Wait Automatic Repeat Request Scheme for Network Coding Nodes*" Vehícular Technology Conference IEEE 2010

[48] Kamtorn Ausavapattanakun and Aria Nosratinia, (2007) "*Analysis of Go-Back-N ARQ in Block Fading Channels*", IEEE Transactions on Wirwlwss Communications, August 2007.

[49] Shu Lin; Costello, D.; Miller, M (1984), "*Automatic Repeat Request Error Control Schemes*" Communications Magazine, IEEE 1984.

[50] Heredia Gonzáles José, (20 07) Applet – Simulador Didáctico de Protocolos de ventana Protocolos (ARQ), Universidad Politécnica de Cartagena, Junio 2007.

[51] Egea López Esteban, Práctica de redes de ordenadores, Simulación de Algoritmos Automatic Repeat Request (ARQ), Escuela Técnica Superior de Ingeniería de Telecomunicaciones, Universidad Politécnica de Cartagena, 2007.

[52] Zorzi, M.; Rao, R.R (1995) "*Throughput analysis of Go-Back-N ARQ in Markov channels with unreliable feedback*" International Conference on Digital 1995.

[53] Bhargava, V. (1983) "*Forward error correction schemes for digital communications"* Communications Magazine, IEEE Publication 1983.

[54] Demir U; Aktas O (2006) "*Raptor versu Reed Solomon forward error correction codes*",Computer Networks, IEEE International Symposium on Digital 2006.

[55] Savari, S.A.; Kliewer, J. (2010), "*When Huffman Meets Hamming: A Class of Optimal Variable-Length Error Correcting Codes*" Data Compression Conference IEEE, Digital Object Identifier: 2010.

[56] Esteban L. (2005) "*Hamming Codes Are Rate-Efficient Array Codes*", Department of Electrical Engineering, IEEE University of California, Los Angeles, 2005.

[57] Kumar, U.K.; Umashankar, B.S. (2007)"*Improved Hamming Code for Error Detection and Correction*" Wireless Pervasive Computing, 2nd International Symposium on Digital Object Identifier: IEEE 2007.

[58] Echaiz Javier "Código de corrección de errores" D.C.I.C. – U.N.S. Organización de Computadoras Universidad Nacional del Sur, Argentina.

[59] Helleseth, T.; Klove, T.; Levenshtein, V.I. (2005), "*Error-Correction Capability of Binary Linear Codes*", IEEE Digital Object Identifier: (2005).

[60] Gupta, M.; Bhullar, J.S.; Bansal, B.N (2010). "*Undetected Error Probability of Hamming Code for Any Number of Symbols*" Information Theory and Information Security, 2010 IEEE International Conference on Digital 2010.

[61] Geer G. Vlugt M. (1994) "On generalized Hamming weights of BCH codes" Information Theory, IEEE Publication 1994.

[62] Yingquan Wu, (2011) "*Erasure-Only List Decoding of Reed - Solomon and BCH Codes with Applications to Their*" IEEE International Symposium on Digital Object, Santa Clara, USA, Publication 2011.

[63] Richard Tervo, Ph.D., P.Eng (2012) "*Course Information Digital Communications*" Department of Electrical and Computer Engineering - University of New Brunswick, Fredericton, NB, Canada - January 2012.

[64] Stallings W. (2002) "*Wireless Communications and Networks".* Prentice Hall. 2002.

[65] Morelos Zaragoza (2002). "*The Art of error Correcting Coding"*. R John Wiley and Sons 2002.

[66] Díaz Escalona, Concepción Fernández Sánchez (2006) "Emulador hardware para arquitecturas reconfigurables de grano grueso", Proyecto de Sistemas Informáticos, Universidad Complutense de Madrid (2006).

[67] Haykyn S. (2001) *Sistemas de Comunicación*. Limusa Wiley. 2001.

Software

Modscan32 de WINTECH