# INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA
UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"

INGENIERÍA EN COMUNICACIONES Y ELECTRÓNICA

**"MULTICODIFICADOR REED-SOLOMON EN SOFTWARE"**

# T E S I S

**QUE PARA OBTENER EL TÍTULO DE**

**INGENIERO EN COMUNICACIONES Y ELECTRÓNICA**

**PRESENTAN:**

**GUSTAVO DURAN HERRERA**
**JULIO RICARDO FRANCO GILES**

**ASESORES:**

**M. EN C. DAVID VÁZQUEZ ÁLVAREZ**
**M. EN C. GABRIELA SÁNCHEZ MELÉNDEZ**

**MÉXICO, D.F, 2014**

# INSTITUTO POLITÉCNICO NACIONAL

## ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELECTRICA

## UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"

# TEMA DE TESIS

**QUE PARA OBTENER EL TITULO DE** — **INGENIERO EN COMUNICACIONES Y ELECTRÓNICA**
**POR LA OPCIÓN DE TITULACIÓN** — **TESIS COLECTIVA Y EXAMEN ORAL INDIVIDUAL**
**DEBERA(N) DESARROLLAR** — **C. GUSTAVO DURAN HERRERA**
**C. JULIO RICARDO FRANCO GILES**

**"MULTICODIFICADOR REED-SOLOMON EN SOFTWARE"**

IMPLEMENTAR EN LENGUAJE DE PROGRAMACIÓN C++ UN MODULO DE SOFTWARE QUE REPRESENTE UN MÚLTIPLE CODIFICADOR REEE-SOLOMON PARA CÓDIGOS BCH, UTILIZANDO EL CONCEPTO DE RADIO DEFINIDO POR SOFTWARE

- JUSTIFICACIÓN
- RADIO DEFINIDO POR SOFTWARE Y SUS APLICACIONES
- CÓDIGOS BCH (BOSE-CHAUDHURI-HOCQUENGHEM) Y CODIGO REED-SOLOMON
- HERRAMIENTAS DE SOFTWARE Y HARDWARE PARA APLICACIONES DE COMUNICACIONES MÓVILES
- CODIFICADORES REED-SOLOMON, REALIZACIÓN DE PRUEBAS
- CONCLUSIONES
- RECOMENDACIONES PARA TRABAJOS FUTUROS
- ANEXOS
- REFERENCIAS
- GLOSARIO

**MÉXICO D.F. A 4 DE JUNIO DE 2014**

**A S E S O R E S**

**M. EN C. DAVID VÁZQUEZ ÁLVAREZ**

**M. EN C. GABRIELA SÁNCHEZ MELÉNDEZ**

**ING. PATRICIA LORENA RAMÍREZ RANGEL**
**JEFE DEL DEPARTAMENTO DE**
**INGENIERÍA EN COMUNICACIONES Y ELECTRÓNICA**

# ÍNDICE.

# LISTA DE FIGURAS

# LISTA DE TABLAS

# OBJETIVO

Implementar en lenguaje de programación C++ un módulo de software que represente un múltiple codificador Reed-Solomon para códigos BCH, utilizando el concepto de Radio Definido por Software.

# OBJETIVOS ESPECÍFICOS

- Revisión de la arquitectura del radio definido por software y sus diferentes aplicaciones en las comunicaciones móviles actuales.
- Revisión de los códigos BCH.
- Revisión del codificador Reed-Solomon para encontrar generalidades que permitan definir un múltiple codificador.
- Representación de módulos de codificación en lenguaje C++ utilizando herramientas para programación (IDE eclipse).
- Realización de pruebas y comprobación de resultados.

# JUSTIFICACIÓN

Durante los últimos 5 años se ha ido viendo un avance muy grande en la tecnología de las comunicaciones móviles, que ha hecho inevitable cambiar de dispositivos una y otra vez incluso cada medio año con la finalidad de cumplir las exigencias que demandan los usuarios de estas tecnologías.

De tal manera que las empresas que se dedican a este giro comercial se ven obligadas a reconfigurar sus equipos (hardware y software) para poder hacer frente a las características que demanda la nueva tecnología, aun cuando esto signifique modificar por completo el sistema, es decir, comprar nuevo hardware y su respectivo software.

Lo anterior se ve muy claramente en los dispositivos móviles de telefonía celular, el cambio de tecnología 4G a 4G LTE, no solo volvió obsoleto el software que se programó en los dispositivos, sino que también hizo inservible el hardware que en ellos se instaló, es claro que el avance tecnológico impulsa el desecho de dispositivos electrónicos.

La presente investigación se realiza primordialmente para estudiar la metodología del funcionamiento de un codificador Reed-Solomon; siendo este una parte esencial de la estructura general de un “Transmisor de Video Digital”, por sus siglas en ingles DVB (Digital Video Broadcast), es decir, efectuar un código modificable y reprogramable, mediante los equipos de radio desarrollados por programas o “Radio Software”, por sus siglas en ingles SDR (Software Defined Radio) produciendo facilidad en su uso y adaptación a las necesidades del usuario.

Por lo tanto, es necesario dar origen a un proceso más eficiente y compacto el cual genere beneficios expresados en la optimización de los procesos, a diferencia de las generaciones pasadas, a fin de mejorar la calidad, el control de la gestión, la satisfacción y la respuesta a los clientes.

El desarrollo de este proyecto permitirá mejorar la forma en que operan las comunicaciones móviles dentro de los siguientes dispositivos: transmisores DVB, redes inalámbricas, radios de campo y dispositivos de almacenamiento. Ofreciendo soluciones más fiables para las comunicaciones cotidianas y para los usuarios de cada país y grupos sociales. El DVB está presente alrededor de todo el mundo por medio de la televisión, y siendo ésta considerada como un medio de acceso a la información; se ha suscitado la necesidad de afinar la transmisión de señales. De tal forma que el desarrollo del codificador Reed-Solomon como componente en la estructura del DVB es sumamente importante.

Es así, que lo anterior se expresará mediante una serie de argumentos basados en procedimientos teóricos y técnicos; diseñando software programado en lenguaje C++, que ofrezca la facilidad de elegir entre un tipo u otro de codificador Reed-Solomon.

# CAPÍTULO 1. RADIO DEFINIDO POR SOFTWARE Y SUS APLICACIONES.

## 1.1 Radio Definido por Software

Radio Definido por Software (SDR por sus siglas en Ingles Software Defined Radio) es un aspecto del dominio de procesamiento digital de señales (DSP por sus siglas en Ingles Digital Signal Processing) donde señales de radio son digitalizadas y procesadas para obtener modulación o demodulación de los distintos modos.

SDR es una técnica de radio muy utilizada actualmente esto no quiere decir que es nueva ya que lleva varios años en el mercado, es por ello que hablaremos un poco en este capítulo de sus antecedentes para ver cómo ha sido su evolución. Los equipos receptores y transceptores de radiocomunicaciones son equipos constituidos por una gran cantidad de componentes electrónicos, los cuales forman circuitos sintonizadores, etapas de frecuencia intermedia, detectores, amplificadores de baja frecuencia, etc., con esto se quiere dar a entender que están constituidos por **"Hardware"**, el SDR está conformado por componentes que funcionan por medio de programas en un ordenador, es decir, están constituidos por **"Software"**

### *1.1.1 Antecedentes*

El término "receptor digital" fue acuñado en 1970 por un investigador en un laboratorio del Departamento de Defensa. Un laboratorio llamado la Sala de Oro en TRW en California creó una herramienta de análisis de banda base de software llamado Midas, que por supuesto fue definido por software [1].

El término "software de radio" fue acuñado en 1984 por un equipo de la División de E-Systems Inc. Garland Texas, para referirse a un receptor digital de banda y fue publicado en el boletín de la empresa E-Team. A 'Radio Proof," se desarrolló allí y se popularizó el Radio Software en varias agencias gubernamentales. Este Radio Software 1984 fue un receptor de banda base digital que proporciona cancelación de interferencia programable y demodulación de señales de banda ancha.

En 1991, Joe Mitola reinvento el termino del software de radio para un proyecto de construcción de una estación base GSM que combinaría receptor digital de Ferdensi con bloqueadores de comunicaciones de control digital. E-Systems Melpar vendió la idea de radio de software para la Fuerza Aérea de los EE.UU.. Melpar construyo un prototipo de terminal táctico. Ese prototipo no duró mucho tiempo porque cuando la División ECI E-Systems fabrica las primeras unidades de producción limitada, decidieron "tirar esos C30", sustituyéndolos por RF convencional filtrado de transmisión y recepción, volviendo a radio digital de banda base. La Fuerza Aérea no dejaría a Mitola publicar los detalles técnicos de ese prototipo. Así que en vez y con el permiso de la USAF, en 1991 Mitola describe los principios de la arquitectura

sin detalles de la implementación en un documento, "Radio Software: Estudio, análisis crítico y direcciones futuras", que se convirtió en la era de la primera publicación IEEE para emplear el término en 1992. Cuando Mitola presentó la ponencia en la conferencia, el expositor de GEC Marconi inició su presentación tras Mitola con "Joe tiene toda la razón acerca de la teoría de un radio de software y estamos construyendo una".

Pocos meses después de la Conferencia Nacional Telesystems 1992, en un E-Systems en su programa corporativo, un vicepresidente de la División de Garland E-Systems se opuso a la utilización de Melpar del término "software de radio". La publicación de Mitola de radio de software en el IEEE abrió el concepto a la amplia comunidad de ingenieros de radio. Su presentación en mayo de 1995 edición especial de la revista IEEE Communications con la cubierta de "Radio Software" fue ampliamente considerado como un suceso con miles de citas académicas. Mitola fue presentado por Joao da Silva en 1997 en la Primera Conferencia Internacional de Radio Software como el "padrino" de radio software en gran parte por su voluntad de compartir una tecnología tan valiosa "de interés público".

El término **"radio definido por software"** fue acuñado en 1995 por Stephen Blust, que publicó una solicitud de información de BellSouth Wireless en la primera reunión de Información en el foro Modular Systems en 1996, organizado por la USAF y la DARPA en torno a la comercialización de su programa clandestino II. Mitola se opuso al término de Blust, pero finalmente lo aceptó como una vía pragmática hacia la radio de software ideal. Aunque la idea se puso en práctica por primera vez con un IF ADC a principios de 1990, las radios definidas por software tienen sus orígenes en el sector de la defensa desde finales de 1970, tanto en los EE.UU. y Europa. Alrededor de un año después de la Primera Conferencia Internacional de Bruselas. Una de las primeras iniciativas de radio de software público fue el proyecto militar de la Fuerza Aérea de los EE.UU. DARPA-llamado SpeakEasy. El objetivo principal del proyecto SpeakEasy era utilizar procesamiento programable para emular más de 10 radios militares existentes, que operan en bandas de frecuencia entre 2 y 2000 MHz. Otro objetivo del diseño era clandestino para poder incorporar fácilmente nueva codificación y estándares de modulación en el futuro, por lo que las comunicaciones militares pueden seguir el ritmo de los avances en técnicas de codificación y modulación [2].

Por lo que su uso inicio en el área militar y fue ahí donde se desarrolló principalmente.

### *1.1.2 Concepto de Radio Definido por Software*

El termino radio se utiliza mediante la combinación de transmisor (Tx) y receptor (Rx), lo que generalmente se conoce como transceptor. Una de las principales características del radio, se encuentra en el intercambio de información, y no solo puede transportar audio, sino eventualmente video y transmisiones multimedia [3].

Un SDR, tiene casi todos sus componentes definidos y funcionando en forma de programas en un ordenador, a excepción de un mínimo de componentes físicos. Y mientras no sea activado ese software o conjunto de programas, el equipo de radio no será utilizado como tal, sino que será un simple conjunto de placas electrónicas externas, incapaces de funcionar.

Al poder moldear el SDR a nuestra conveniencia únicamente modificando o remplazando sus programas de software, es posible usarlo para un sinfín de propósitos específicos dependiendo del usuario. El SDR es capaz de ser reprogramado o reconfigurado para funcionar con diferentes protocolos y formas de onda a través de la carga dinámica de nuevas formas de onda y protocolos.

Dentro del SDR existe otro término importante llamado Radio Cognitivo (Cognitive Radio, CR). Un dispositivo de CR es un sistema de radiofrecuencia capaz de variar sus parámetros basándose en su interacción con el entorno en el que opera [4].

Dada esta definición, las dos características principales de un dispositivo de este tipo son:

- Capacidad cognitiva: Tecnología necesaria para capturar la información de su entorno de radiofrecuencia e identifica las partes del espectro que no estén siendo utilizadas.
- Auto-reconfiguración: Tecnología necesaria para que el dispositivo pueda variar, de manera dinámica, distintos parámetros relacionados con la transmisión o recepción (frecuencia, potencia, modulación, etc.), de acuerdo con su entorno.

Dependiendo el uso y el área del SDR puede ser:

- Un sistema Multibanda el cual soporta más de una banda de frecuencias dedicadas, usado para estándares wireless (como GSM 900, GSM 1800, GSM 1900).
- Un sistema Multiestandar el cual soporta más de una interface aérea. Los sistemas Multiestandar pueden trabajar dentro de una familia estándar (como ULTRA-FDD, ULTRA-TDD para UMTS) a través de diferentes redes (como DECT, GSM, UMTS, WLAN).
- Un sistema Multiservicio el cual provee diferentes servicios (Datos, telefonía, video).
- Un sistema Multicanal el cual soporta dos o más transmisiones independientes y reciben canales simultáneamente.

El SDR tiene la cualidad de poder ser transformado a través del uso de software o de lógica re definible, continuamente esto se hace con Procesadores Digitales de Señales o con FPGAs (Field Programmable Gate Arrays) de propósito general o con tarjetas de desarrollo XMOS.

## 1.2 Aplicación en el Digital Video Broadcast (DVB)

Digital Video Broadcasting (DVB) es una organización que promueve estándares aceptados internacionalmente de televisión digital, en especial para HDTV y televisión vía satélite, así como para comunicaciones de datos vía satélite (unidireccionales, denominado DVB-IP, y bidireccionales, llamados DVB-RCS) [5].

El acceso unidireccional, no es de banda ancha, ya que se realiza combinando el acceso a Internet tradicional, vía RTB/RDSI, más el módem univía de acceso satelital DVB.

### *1.2.1 Principios de DVB*

El DVB (Digital Video Broadcasting) es un organismo encargado de crear y proponer los procedimientos de estandarización para la televisión digital compatible. Está constituido por más de 270 instituciones y empresas de todo el mundo. Los estándares propuestos han sido ampliamente aceptados en Europa y casi todos los continentes, con la excepción de Estados Unidos, Canadá y Japón donde coexisten con otros sistemas propietarios. Todos los procedimientos de codificación de las fuentes de vídeo y audio están basados en los estándares definidos por MPEG. No obstante, hemos visto que los estándares MPEG sólo cubren los aspectos y metodologías utilizados en la compresión de las señales de audio y vídeo y los procedimientos de multiplexación y sincronización de estas señales en tramas de programa o de transporte. Una vez definida la trama de transporte es necesario definir los sistemas de modulación de señal que se utilizarán para los distintos tipos de radiodifusión (satélite, cable y terrestre), los tipos de códigos de protección frente a errores y los mecanismos de acceso condicional a los servicios y programas.

El DVB ha elaborado distintos estándares en función de las características del sistema de radiodifusión. Los estándares más ampliamente utilizados en la actualidad son el DVB-S y el DVB-C que contemplan las transmisiones de señales de televisión digital mediante redes de distribución por satélite y cable respectivamente. La transmisión de televisión digital a través de redes de distribución terrestres utilizando los canales UHF convencionales se contempla en el estándar DVB-T, que actualmente se está implantando en la mayor parte de los países europeos. Además de estos estándares también están especificados sistemas para la distribución de señales de televisión digital en redes multipunto, sistemas SMATV (Satellite Master Antenna Televisión). También existen estándares que definen las características de la señalización en el canal de retorno en sistemas de televisión interactiva, la estructura de transmisión de datos para el cifrado y descifrado de programas de acceso condicional, la transmisión de subtítulos, y la radiodifusión de datos (nuevos canales de teletexto) mediante sistemas digitales [6].

### *1.2.2 Transmisión*

Los sistemas DVB distribuyen los datos por:

- satélite (DVB-S y DVB-S2)
- cable (DVB-C y DVB-C2)
- televisión terrestre (DVB-T y DVB-T2)
- televisión terrestre para dispositivos portátiles (DVB-H)
- televisión satelital para dispositivos portátiles (DVB-SH)

Estos estándares definen la capa física y la capa de enlace de datos de un sistema de distribución. Los dispositivos interactúan con la capa física a través de una interfaz paralela síncrona (SPI), una interfaz serie síncrona (SSI) o una interfaz serie asíncrona (ASI). Todos los datos se transmiten en flujos de transporte MPEG-2 con algunas restricciones adicionales (DVB-MPEG). Se está experimentando en varios países un estándar para distribución comprimida en el tiempo (DVB-H) para distribución a dispositivos móviles [7].

Estos estándares se diferencian principalmente en los tipos de modulación utilizados, debido a las diferentes restricciones técnicas:

- DVB-S (SHF) utiliza QPSK, 8PSK O 16-QAM.
- DVB-S2 (SHF) utiliza QPSK, 8PSK, 16APSK o 32APSK en los retransmisores.
- DVB-C (VHF/UHF) utiliza QAM, 16-QAM, 32-QAM, 64-QAM, 128-QAM o 256-QAM (64-QAM en general).
- DVB-T (VHF/UHF) 16-QAM o 64-QAM (o QPSK) en combinación con COFDM y soporta modulación jerárquica.

### *1.2.3 Contenido*

Además de la transmisión de audio y vídeo, DVB también define conexiones de datos (DVB-DATA - EN 301 192) con canales de retorno (DVB-RC) para diferentes medios (DECT, GSM, RTB/RDSI, satélite, etc.) y protocolos (DVB-IPTV: protocolo de Internet; DVB-NPI: protocolo de red independiente) [8].

Para facilitar la conversión, estos estándares también soportan las tecnologías existentes tales como el teletexto (DVB-TXT) y el sincronismo vertical (DVB-VBI). Sin embargo, para muchas aplicaciones hay disponibles alternativas más avanzadas como, por ejemplo, DVB-SUB para los subtítulos.

El codificador Reed-Solomon que es empleado en los estándares DVB es el RS (204,188). Para desarrollar la metodología a emplear en el diseño de codificadores RS, opte por trabajar con el codificador RS (7,3) para comenzar y asi implementarlo en un entorno de desarrollo integrado, IDE (de sus siglas en ingles de Integrated Development Environment) para Eclipse.

# CAPÍTULO 2. CÓDIGOS BCH (BOSE-CHAUDHURI-HOCQUENGHEM) Y CÓDIGO REED-SOLOMON

El objetivo del actual capítulo es presentar un breve pero concreto marco teórico que permita conocer algunos conceptos y fundamentos en los que se basa el funcionamiento de los códigos Reed-Solomon, en los cuales se encuentran: los códigos BCH (Bose-Chaudhuri-Hocquenghem), los códigos cíclicos, los polinomios de palabra-código y detección y corrección de errores.

## 2.1 Definición de códigos BCH

Los códigos BCH constituyen una de las clases más importantes y poderosas de los códigos de bloques lineales, fueron inventados en 1959 por Hocquenghem, e independientemente cerca de 1960 por Bose y Rayo-Chaudhuri [9].

Los códigos BCH más comunes son los códigos BCH binarios, aunque para el estudio de los codificadores Reed-Solomon, es preciso usar los del tipo no-binario, los cuales están caracterizados por cualquier entero positivo m, igual o mayor a 3 y t menor que (2m – 1)/2 mediante los siguientes parámetros:

Longitud del bloque: n = 2m – 1; donde m>=3
Dimensión del código: n = k – t, donde t=grado del polinomio generador (gx); el cual depende del número de errores que pueda corregir el código.
Distancia mínima: dmin >= 2t + 1

En palabras menos complejas, la función de un código BCH es recibir un paquete de información de longitud *k*, éste es procesado, convirtiéndolo en un bloque de información de longitud *n*, donde n>k, observe la siguiente figura.

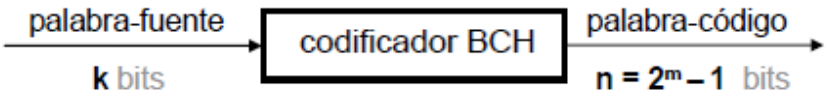


*Figura 2.1 Representación de inserción de bits de paridad en la información.*

## 2.2 Códigos cíclicos

Los códigos cíclicos son códigos de corrección de errores por bloques que siguen los principios generales enunciados en el concepto anterior. Los mismos pueden ser codificados y decodificados usando registros. Para un código cíclico, un código válido (c0, c1,…, cn-1), desplazado hacia la izquierda un bit (cn-1, c0,…, cn-2), es también un código válido. La entrada de longitud fija (k) toma y produce un código (n-k) [10].

Se pueden encontrar una gran variedad de códigos cíclicos, tales como el Código de Redundancia Cíclica (CRC) comúnmente utilizado en comunicaciones digitales,

el código Golay, el código Hamming, así como el código Bose-Chaudhuri-Hocquenqhem (BCH) y el Reed-Solomon, que es la principal área de interés de este trabajo.

### *2.2.1 Polinomios de palabra-código*

Para poder representar matemáticamente una operación de los códigos cíclicos es necesario hacer uso de polinomios. Es así que empezamos por ver a los elementos de una palabra-código de tamaño *n* como los coeficientes de un polinomio de grado *n-1*. Por ejemplo, la palabra-código con elementos $x_0$, $x_1$,..., $x_{n-1}$ puede ser representada en forma de polinomio como:

$$X(D) = x_0 + x_1D, \ldots, + x_{n-1}D_{n-1}$$

Dónde: D es una variable Real arbitraria.

### *2.2.2 Polinomio generador*

Un conjunto de polinomios de palabra-código de grado *n-1* o menos es siempre el que especifica a un código cíclico (n, k). Dentro de este conjunto se encuentra un polinomio especial de grado mínimo *n-k* como un factor, que se denota por *g(D)*. Aquel factor especial es seleccionado como el Polinomio Generador del Código [11].

### *2.2.3 Codificación de un código cíclico*

Multiplicar el polinomio del mensaje *m(D)* por $D^{n-k}$

$$D^{n-k}\, m(D) = m_0D^{n-k} + m_1D^{n-k+1} + \ldots + m_{k-1}D^{n-1}$$

Dividir $D^{n-k}\, m(D)$ por el polinomio generador *g(D)*, obteniendo el residuo *b(D)*.

$$\frac{D^{n-k}\, m(D)}{g(D)} = a(D) + \frac{b(D)}{g(D)}$$

Agregar *b(D)* a $D^{n-k}\, m(D)$ para obtener el polinomio de la palabra-código *x(D)*.

$$x(D) = b(D) + D^{n-k}\, m(D)$$

### *2.2.4 Síndrome en la detección de error*

Considerando que la palabra-código recibida con error sea:

$$Y(D) = y_0 + y_1D+, \ldots,+ y_{n-1}D_{n-1}$$

El polinomio del síndrome se obtiene con el residuo de la división del polinomio de la palabra código entre el polinomio generador *g(D)*.

$$\frac{Y(D)}{g(D)} = q(D) + \frac{s(D)}{g(D)}$$

Dónde: q es el cociente y s es el síndrome.

De manera similar al código Hamming, con el síndrome y el patrón de errores se hace la detección del error o los errores y su posición dentro de la palabra-código.

## 2.3 Detección y corrección de error

En comunicaciones prácticamente todas las señales digitales producidas en la actualidad llevan asociados el proceso de detección o corrección de errores. Este proceso se ocupa de la detección mediante los métodos CRC y BIP y corrección de errores mediante FEC a bloques y convolucional.

Para detectar que hubo un error, al enviarse un marco se guarda en una tabla cuándo se envió y se le asocia un tiempo para recibir su confirmación. Si no se recibe la confirmación por parte del receptor, se re-envía el marco. El problema que puede surgir es que si se perdió la confirmación, el receptor puede tener marcos duplicados, lo cual se soluciona al asignar un número de secuencia a cada marco, para descartar los duplicados y re-enviar su confirmación.

Otra forma de detectar un error (que ya no fue la pérdida del marco, sino la corrupción de su contenido), es insertar un código de chequeo, y para esta labor se utilizan códigos basados en el concepto de "distancia de Hamming".

La distancia de Hamming para un código cualquiera se define como el número de bits diferentes al hacer un XOR entre todos sus símbolos.

Si los símbolos de un código difieren al menos en 2X+1 bits, al variar X bits (dañar X bits) obtengo un nuevo símbolo que se parecerá más en un bit a un código válido que a otro código válido y por lo tanto puede decir que el símbolo dañado en realidad es el más parecido realizando así su corrección.

Para el diseño estándar de protocolos, se han especificado algunas cadenas de chequeo, bien conocidas como CRC-12, CRC-16 y CRC-CCITT con CRC=12,16 bits y CCITT=16 bits respectivamente. Estas cadenas se interpretan con polinomios de la siguiente manera.

CRC-12 = 1100000001111 = $X^{12} + X^{11} + X^3 + X^2 + X + 1$.
CRC-16 = 11000000000000101 = $X^{16} + X^{15} + X^2 + 1$
CRC-CCITT = 10001000000100001 = $X^{16} + X^{12} + X^5 + 1$

Se observa que la posición del bit con un uno representa la potencia del polinomio. Cada uno de estos polinomios se conocen como "generador polinomio" y las siglas CRC significan "Cyclic Redundancy Code" (Código de Redundancia Cíclica) [14].

## 2.4 Definición de codificador

Podemos definir a un codificador como un circuito combinacional con dos veces más entradas que salidas cuya finalidad es presentar en la salida el código binario correspondiente a la entrada activada.

Existen dos tipos fundamentales de codificadores [16]:

- Los primeros solo admiten una entrada activada, codificando en la salida el valor binario de la misma y cero cuando no existe ninguna activa.

- En los segundos puede haber más de una entrada activada, existiendo prioridad en aquella cuyo valor decimal es más alto.

La codificación para control de errores corresponde a una rama de las matemáticas aplicadas llamada teoría de la información. Una aplicación específica corresponde a los códigos Reed-Solomon; los algoritmos que maneja esta aplicación pueden ser implementados tanto en software como en hardware.

Dado que actualmente la tecnología vuelve indispensable el uso de dispositivos que manejen lógica reconfigurable a alta escala de integración, de tal suerte que no sea necesario cambiar hardware sino que baste solo con reconfigurar el software cargado en ellos para satisfacer las especificaciones demandadas por los usuarios, y que los beneficios que ofrece esta tecnología a los diseñadores de sistemas digitales, al hacer uso de un lenguaje de programación tan común como lo es C++, permite plantear el diseño de estos módulos de codificación en software programado en este lenguaje.

## 2.5 Bases del código Reed-Solomon

El código Reed-Solomon es un código corrector de errores basado en bloques en donde el codificador procesa un bloque de símbolos de datos, a los que agrega redundancia para producir un bloque de símbolos codificados. El estudio de este codificador ha permitido calificarlo como el más ventajoso, debido a su eficiencia en cuanto a ganancia del código y su fácil aplicación tecnológica al poseer uno de los algoritmos de decodificación más eficientes desarrollado por Berlekamp [17].

El uso de los códigos Reed-Solomon como corrector de errores, actualmente, es muy visto en sistemas de comunicaciones inalámbricas o móviles (telefonía celular, enlaces de microondas, etc.), comunicaciones satelitales, televisión digital/DVB, módem de alta velocidad (ASDL, xSDL), dispositivos de almacenamiento (cintas, discos compactos, DVD, Blu-Ray, códigos de barras, etc.)

## 2.6 Características de los códigos Reed-Solomon

El código Reed-Solomon forma parte de un subconjunto de los códigos BCH, códigos cíclicos que presentan entre sus parámetros *(n, k, t)* una relación entre los símbolos de mensaje *(k)* , de la palabra-código *(n)* y del máximo número de errores que son posibles de corregir *(t)*, y son de bloques lineales. Un código Reed-Solomon se especifica como RS(n, k) con símbolos de *s* bits, esto significa que el codificador toma *k* símbolos formados por *s* bits cada uno y añade símbolos de paridad para hacer una palabra-código de *n* símbolos.

En una palabra-código existirán *n-k* símbolos de paridad de *s* bits cada uno. Un decodificador para RS(n, k, t) tiene la capacidad de corregir hasta *t* símbolos que contienen errores en una palabra-código, donde *2t = (n-k)* [18].

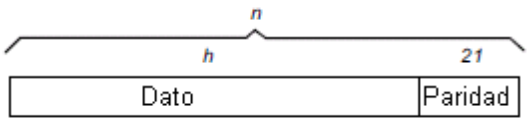


*Figura 2.2 Palabra código Reed-Solomon.*

El circuito digital que procesa las tramas

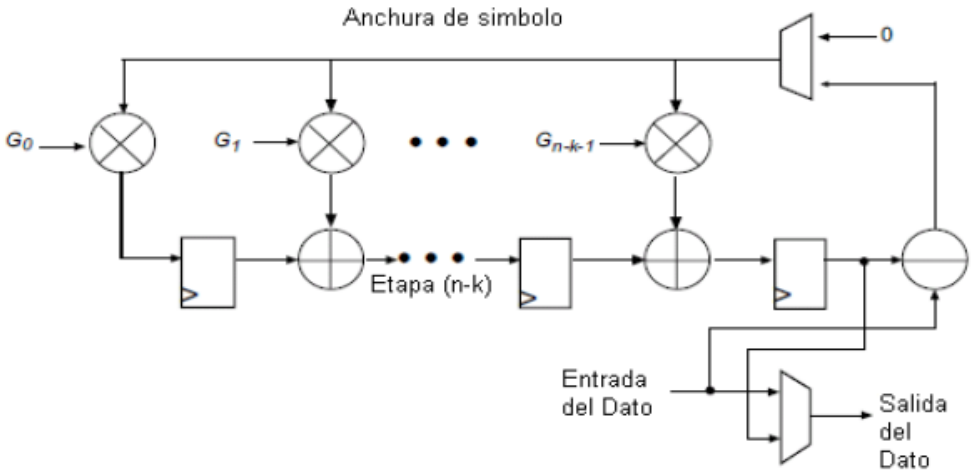


*Figura 2.3 Arquitectura genérica de un codificador Reed-Solomon.*

## 2.7 Campos de Galois aplicados a la codificación Reed-Solomon

Los códigos Reed-Solomon se basan en un área especializada de la matemática llamada campos de Galois o campos finitos. Un campo finito tiene la propiedad de que las operaciones aritméticas sobre elementos del campo siempre tienen un resultado en el campo. Un codificador o decodificador Reed-Solomon debe ser capaz de realizar estas operaciones aritméticas.

## 2.8 Generador Polinomial Campos de Galois

Como se mencionó en la sección 2.2.2, una palabra-código Reed-Solomon es generada por un polinomio especial o polinomio generador. Aquellas palabras-código que sean exactamente divisibles por tal polinomio generador, se consideran palabras-código válidas y este polinomio es representado por la siguiente ecuación:

$$g(x) = (x - a^{t})(x - a^{t+1})(x - a^{t+2t})$$

Al aplicar la operación c(x) = g(x) * i(x), donde g(x) es el polinomio generador, i(x) es el bloque de información y c(x) es un palabra-código válida, se puede obtener la codificación Reed-Solomon [20].

El primer paso corresponde a la definición del campo de Galois para la codificación, el cual estará definido en función de la longitud del símbolo -entiéndase m, bits/símbolo-, permitiendo así conocer el polinomio irreducible del campo GF (2m).

Las bases teóricas que sustentan este codificador están dadas por el polinomio en su forma general.

$$g(x) = \prod_{t=0}^{n-k-1} (x - a^{hx(Comienza\ generador+t)})$$

Al expandir el polinomio se obtiene la ecuación siguiente.

$$G(x) = G_{n-k-1}x^{n-k-1} + G_{n-k-2}x^{n-k-2} + \cdots + G_1x + G_0$$

Dónde:

n: longitud de la palabra codificada (en símbolos).
k: longitud del mensaje codificado (en símbolos).
m: longitud del símbolo (bits)

El segundo paso corresponde a definir el polinomio generador donde se obtiene:

$$G(x) = \alpha^2x^3 + \alpha^5x^2 + \alpha^5x + \alpha 6.$$

# CAPÍTULO 3. HERRAMIENTAS DE SOFTWARE Y HARDWARE PARA APLICACIONES DE COMUNICACIONES MÓVILES

En este capítulo analizaré el hardware y software de programación que son comúnmente utilizados en aplicaciones de comunicaciones móviles como es el caso de este multicodificador Reed-Solomon y mencionaré aquellos que cumplan más acertadamente con mis intenciones constituyendo así la elección en donde será implementado el código que desarrollaré posteriormente.

A continuación daré una breve descripción de los diferentes lenguajes de programación que son utilizados para programar el hardware que cumple con las necesidades de las comunicaciones móviles actuales pero antes es necesario explicar que un lenguaje de programación es una herramienta que nos permite comunicarnos e instruir a un microprocesador para que realice una tarea específica. Cada lenguaje de programación posee una sintaxis y un léxico particular, es decir, forma de escribirse que es diferente en cada uno por la forma que fue creado y por la forma que trabaja su compilador para revisar, acomodar y reservar el mismo programa en memoria.

## 3.1 Lenguaje C++

C++ es un lenguaje de programación orientado a objetos que toma la base del lenguaje C y le agrega la capacidad de abstraer tipos.

La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitieran la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma [22].

### *3.1.1 Antecedentes del Lenguaje C++*

C++ es un lenguaje de programación creado por Bjarne Stroustrup en los laboratorios de AT&T en 1983. Stroustrup tomó como base el lenguaje de programación más popular en aquella época el cual era C.

El C++ es un derivado del mítico lenguaje C, el cual fue creado en la década de los 70 por la mano del finado Dennis Ritchie, el cual surgió como un lenguaje orientado a la programación de sistemas y de herramientas recomendado sobre todo para

programadores expertos, y que no llevaba implementadas muchas funciones que hacen a un lenguaje más comprensible [21].

Sin embargo, aunque esto en un inicio se puede convertir en un problema, en la práctica es su mayor virtud, ya que permite al programador un mayor control sobre lo que está haciendo. Años más tarde, un programador llamado Bjarne Stroustrup, creó lo que se conoce como C++.

Necesitaba ciertas facilidades de programación, incluidas en otros lenguajes pero que C no soportaba, al menos directamente, como son las llamadas clases y objetos, principios usados en la programación actual. Para ello rediseño C, ampliando sus posibilidades pero manteniendo su mayor cualidad, la de permitir al programador en todo momento tener controlado lo que está haciendo, consiguiendo así una mayor rapidez que no se conseguiría en otros lenguajes.

C++ pretende llevar a C a un nuevo paradigma de clases y objetos con los que se realiza una comprensión más humana basándose en la construcción de objetos, con características propias solo de ellos, agrupados en clases.

### *3.1.2 Características del lenguaje C++*

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos.

Una particularidad del C++ es la posibilidad de redefinir los operadores, y de poder crear nuevos tipos que se comporten como tipos fundamentales.

C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que tiene menos automatismos, con lo que obliga a usar librerías de terceros.

El nombre C++ fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C [23].

## 3.2 Lenguaje para Descripción de Hardware (VHDL)

El lenguaje descriptivo del hardware (HDL) es una clase de lenguaje de programación para la descripción formal de circuitos electrónicos. El HDL puede describir la operación del circuito, su diseño y estructura, asimismo verificar las pruebas de su operación por medio de la simulación.

### *3.2.1 Antecedentes*

Históricamente, los primeros diseños digitales se realizaron siguiendo las técnicas clásicas de síntesis (descripción esquemática y realización de prototipos). A medida

que el grado de complejidad de los circuitos aumentaba, dando lugar a circuitos de mayor grado de integración se tuvo que recurrir a métodos que permitieran simular su comportamiento.

En breve, se enlista los HDL's más viejos y menos capaces que fueron desapareciendo con el tiempo.

- AHDL (Altera HDL, una lengua propietaria de Altera).
- Átomo (síntesis del comportamiento y HDL de alto nivel basados en Haskell).
- Bluespec (HDL de alto nivel basado originalmente en Haskell, ahora con una síntaxis basado Verilog).
- JHDL (basado en Java).
- MyHDL (basado en Python).
- RHDL (basado en Ruby)

Los Lenguajes de Descripcion de Hardware (HDL- del inglés Hardware Description Language) remplazan la descripción esquemática de un sistema digital por una descripción textual. Tanto a nivel académico como industrial se desarrollaron varios HDL's con sus respectivas herramientas de síntesis y simulación, pero recién en la década de 1980 surgió la necesidad de crear un lenguaje normalizado. El Departamento de Defensa de los EE.UU., desarrollo un HDL denominado VHSIC (Very High Speed Integarted Circuit) Hardware Description Language, hoy conocido como VHDL que fue normalizado por el IEEE en 1987 (Norma 1076) y posteriormente se lo extendió (1993-2002).

### *3.2.2 Características*

- Permite especificar el comportamiento o la estructura de un sistema empleando jerarquías.
- Permite hacer descripciones desde un alto nivel de abstracción (algorítmico) hasta un nivel de componentes (estructural), pudiendo coexistir ambos en una misma descripción.
- Los modulos creados en VHDL pueden usarse en otros diseños y con otras tecnologías, siempre que no utilicen modulos prediseñados por un dado fabricante que se aplique solo a dispositivos de su producción.
- Permite crear bancos de prueba (test benches) para simular las descripciones.
- El carácter textual de la descripción, permite realizar en forma natural la documentación del diseño, particularmente importante si se trata de sistemas complejos.

Estos dos lenguajes de programación son los más recurridos al momento de querer programar aplicaciones para comunicaciones móviles, ahora mencionare los dispositivos en los cuales es posible implementar el código a desarrollar.

## 3.3 FPGA (Del Ingles Field Programmable Gate Array)

Es un circuito integrado destinado a ser configurado por el cliente o el diseñador después de la fabricación, de ahí el nombre programable en campo. La configuración de la FPGA se especifica generalmente usando un lenguaje de descripción de hardware, similar a la utilizada para un circuito integrado específico de aplicación. Las FPGAs modernas tienen grandes recursos de puertas lógicas y bloques de memoria RAM para ejecutar cálculos digitales complejos.

Las FPGAs contienen componentes lógicos programables llamados "bloques lógicos" y una jerarquía de interconexiones reconfigurables. Los bloques lógicos se pueden configurar para realizar funciones combinacionales complejas, o puertas lógicas meramente sencillas como AND y XOR. En la mayoría de las FPGAs los bloques lógicos también incluyen elementos de memoria, que pueden ser simples flip-flops o más bloques completos de memoria.

Algunas FPGAs tienen características analógicas, además de las funciones digitales. La función analógica más frecuente es la rapidez de respuesta programable y fuerza de accionamiento en cada pin de salida, lo que le sirve al ingeniero para establecer las respuestas lentas en los pines de carga ligera. Otra característica analógica relativamente frecuente son los comparadores diferenciales en los pines de entrada diseñados para ser conectados a los canales de señalización diferencial.

### *3.3.1 Historia*

La industria de la FPGA programable brotó de memoria de sólo lectura y los dispositivos lógicos programables. PROM y PLDs ambos tenían la opción de ser programado en lotes en una fábrica o en el campo.

A finales de 1980 el Departamento de Guerra de Superficie Naval financió un experimento propuesto por Steve Casselman para desarrollar un equipo que implementaría 600.000 puertas reprogramables. Casselman fue un éxito y una patente relacionada con el sistema se publicó en 1992.

Algunos de los conceptos fundamentales industrys y tecnologías para arrays programables lógicos, puertas y bloques lógicos se fundan en las patentes concedidas a David de Westminster y LUVERNE R. Peterson en 1985.

Xilinx co-fundadores Ross Freeman y Bernard Vonderschmitt inventaron la primera matriz de puertas programables comercialmente viable en 1985. Xilinx continuó sin respuesta y creciendo rápidamente desde 1985 hasta mediados de la década de 1990. En 1993, Actel cubría el 18 por ciento del mercado.

La década de 1990 fue un periodo explosivo para las FPGAs, tanto en sofisticación y el volumen de producción. En la década de 1990, las FPGAs se utilizaron sobre

todo en telecomunicaciones y redes. A finales de la década, las FPGAs encontraron su camino en las aplicaciones de consumo, automotriz e industrial.

### *3.3.2 Aplicaciones*

Las aplicaciones de las FPGAs incluyen el procesamiento digital de señales, radio definido por software, imagen médica, la visión artificial, reconocimiento de voz, la criptografía, la bioinformática, emulación de hardware, la radioastronomía, detección de metales y una gama cada vez mayor de otras áreas.

Las FPGAs comenzaron inicialmente como competidores de CPLD y compitió en un espacio similar al de la lógica de procesamiento para el PCBs. En cuanto a su tamaño, capacidad y velocidad aumentó, comenzaron a asumir funciones cada vez mayores con el estado en el que algunos están comercializados como sistemas completos de fichas. En particular, con la introducción de los multiplicadores dedicados en arquitecturas FPGA a finales de 1990, las solicitudes que habían sido tradicionalmente la única reserva de DSPs comenzaron a incorporar las FPGAs en su lugar.

Tradicionalmente, las FPGAs se han reservado para aplicaciones específicas en que el volumen de la producción es pequeño. Por estas aplicaciones de bajo volumen, la prima que pagan las empresas en los costos de hardware por unidad de un chip programable es más viable.

### *3.3.3 Arquitectura*

Básicamente, en una FPGA la lógica se divide en un gran número de bloques lógicos programables que son individualmente más pequeños que un PLD. Se encuentran distribuidos a través de todo el chip en un mar de interconexiones programables y todo el arreglo se encuentro rodeado de bloques de E/S programables (IOBs). Un bloque lógico programable (CLB o *slice*) de FPGA es menos eficiente que un PLD típico, pero un chip FPGA contiene muchos más bloques lógicos que los PLD que contiene un CPLD del mismo tamaño.

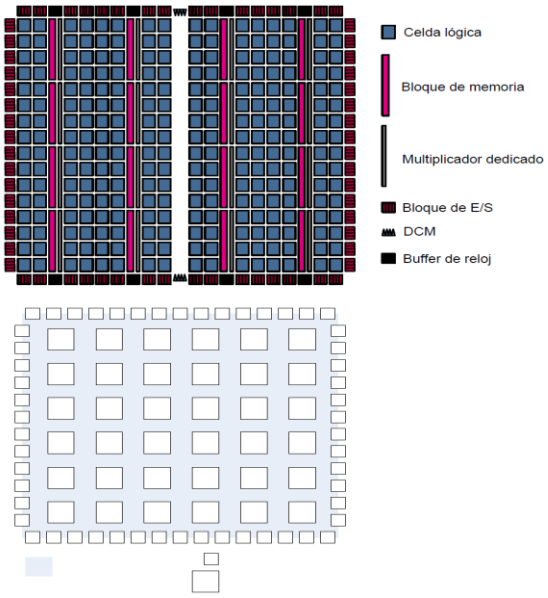


*Figura 3.1 Arquitectura Genérica de una FPGA.*

***Slices*:** En los *slices* se realiza la mayor parte de la funcionalidad de la FPGA y suelen estar agrupados de 2 en 2 o de 4 en 4 formando bloques lógicos configurables (CLBs). Dentro de este componente encontramos los módulos LUT, registros y multiplexores programables en un número que depende de familia de FPGA, pero la arquitectura básica común es la que se muestra en la figura.

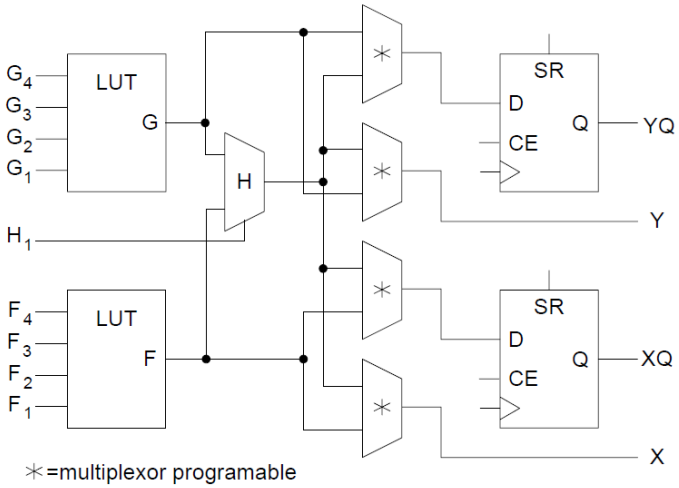


*Figura 3.2 Slice simplificado de una FPGA.*

Los elementos programables más importantes son los generadores reprogramables de función lógica, realizadas por las denominadas LUT (Look-up Table) o tablas de búsqueda, que son celdas de memoria SRAM y multiplexores para seleccionar la salida.

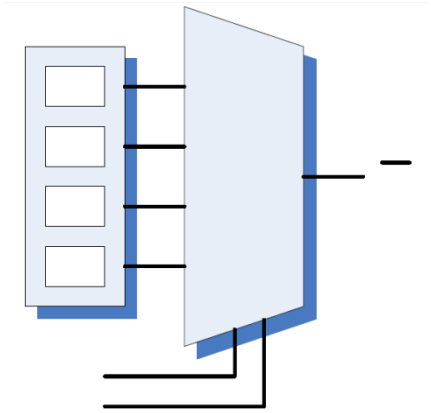

*Figura 3.3 LUT de 2 entradas.*

Los generadores de función pueden diseñarse para cualquier número de variables que se desee sin más que aumentar el tamaño de la memoria SRAM y la ubicación de selectores que escojan un solo valor almacenado para cada combinación de valores de las variables.

En general, los *slices* contienen alguna lógica adicional aparte de las LUTs para aumentar las prestaciones y la eficiencia de estos bloques, como biestables para obtener salidas registradas o lógica para implementar eficientemente comparadores, contadores o sumadores serie. Además, como las funciones lógicas se generan en realidad a través de memorias SRAM, los propios *slices* se pueden configurar para usarlos como bloques de memoria en lugar de lógica, es lo que se denomina memoria distribuida para diferenciarla de bloques de memoria específicos que pudiera haber en la FPGA. Los slices más próximos suelen agruparse siguiendo esta filosofía en grupos denominados CLBs o bloques lógicos configurables.

• **IOBs:** Los bloques de Entrada/Salida de las FPGAs cumplen la misma función que las macroceldas de salida en otros dispositivos lógicos programables, pero con más controles lógicos, entre los que se incluyen, configuraciones de entrada y salida combinacionales o registradas, alta impedancia, elementos de retardo, controles analógicos y otros.

• **Interconexión programable:** Cada CLB en la FPGA se encuentra incrustado en la estructura de interconexión, que se componen en realidad de cables con conexiones programables para ellos. Inicialmente, se disponía de unas interconexiones heterogéneas de propósito general, aunque en la década pasada se evolucionó hacia una estructura de interconexión jerárquica, tal como se muestra en la siguiente figura. Las líneas del grupo del reloj están optimizadas para su uso como entradas de reloj a los CLB, proporcionando un retardo corto. El conjunto de líneas simples se optimizan para conectividad flexible entre bloques adyacentes, pero en mayor cantidad y sin la limitación unidireccional de las líneas directas.

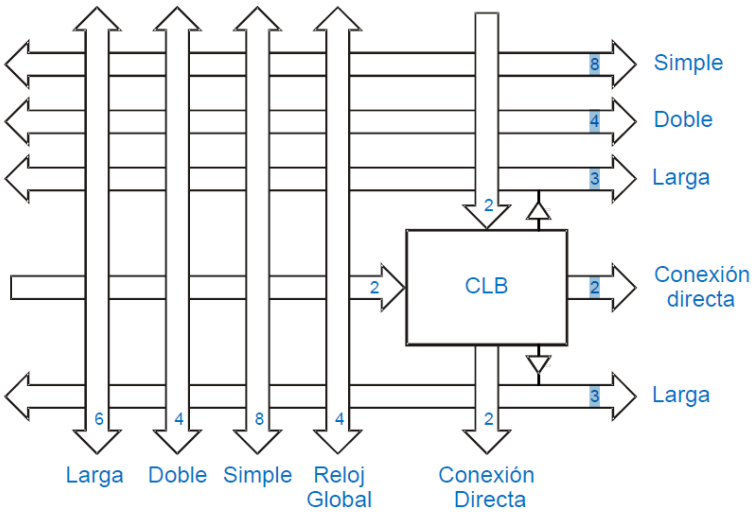


*Figura 3.4 Estructura de interconexión de una FPGA XC4000 de Xilinx.*

Sería posible conectar dos CLBs no adyacentes usando líneas simples, pero deberían pasar por un conmutador programable para cada salto, lo que agregaría retardos adicionales. Las líneas de los grupos doble viajan pasados dos CLBs antes de llegar a un conmutador, de modo que proporcionan retardos más cortos para conexiones más largas. Para conexiones muy largas, se emplean los grupos largos, que no pasan por ningún conmutador programable y recorren todo la FPGA en vertical u horizontal.

• **Otros componentes dentro de una FPGA:** Las FPGAs muchas veces se evalúan en función de la flexibilidad de sus arquitecturas y la consistencia de los resultados obtenidos de un ajuste después de que se han efectuado pequeños cambios de diseño. De esta manera, los fabricantes proporcionan recursos extra en sus arquitecturas para ayudar a asegurar resultados consistentes e implementar algunos sistemas de manera muy eficiente.

Así, si las FPGAs más antiguas eran muy homogéneas, como las XC4000, las que imperan hoy en día, familias como la Spartan-3 o Virtex-4, representantes de la gama baja y la gama alta de las FPGA de Xilinx respectivamente, disponen de diversos dispositivos embebidos dentro del dispositivo, como memorias, multiplicadores, DCMs (administradores de reloj), e incluso microprocesadores.

### *3.3.4 Diseño de FPGA y programación*

Al definir el comportamiento de la FPGA, el usuario proporciona un lenguaje de descripción de hardware o un diseño esquemático. La forma de HDL se adapta más a trabajar con grandes estructuras porque es posible sólo especificarlos numéricamente en lugar de tener que dibujar cada pieza a mano. Sin embargo, la introducción del esquema puede permitir una mejor visualización de un diseño.

A continuación, mediante una herramienta electrónica de automatización de diseño, se genera una lista de conexiones. La lista de conexiones puede ser instalada en la arquitectura de una FPGA real mediante un proceso denominado lugar y la ruta, por lo general realizado por el lugar propio de la compañía de FPGA y el software de ruta. El usuario deberá validar los mapas, lugar y la ruta a través de los resultados de análisis de tiempos, la simulación, y otros métodos de verificación. Una vez que el proceso de diseño y validación se ha completado, el archivo binario generado se utiliza para configurar el FPGA. Este archivo se transfiere a la FPGA/CPLD a través de una interfaz en serie o a un dispositivo de memoria externa, como una EEPROM.

Las más comunes son las HDL y VHDL Verilog aunque en un intento de reducir la complejidad en el diseño de las HDL que se han comparado con el equivalente de los lenguajes de montaje, hay movimientos para aumentar el nivel de abstracción a través de la introducción de lenguajes alternativos. Lenguaje de programación gráfica de LabVIEW de National Instruments tiene un FPGA módulo complementario a disposición de destino y hardware FPGA programa.

A fin de simplificar el diseño de los sistemas complejos en FPGAs existen bibliotecas de funciones complejas predefinidas y circuitos que se han probado y optimizado para acelerar el proceso de diseño. Estos circuitos predefinidos son comúnmente llamados núcleos IP.

En un flujo de diseño típico, un desarrollador de aplicaciones FPGA simulará el diseño en varias etapas a lo largo del proceso de diseño. Inicialmente la descripción RTL en VHDL o Verilog se simula mediante la creación de bancos de pruebas para simular el sistema y observar los resultados. Entonces, después de que el motor de síntesis ha trazado el diseño de una lista de conexiones, la lista de conexiones se traduce a una descripción de nivel de la puerta donde se repite la simulación para confirmar la síntesis procedió sin errores. Por último, el diseño se presenta en la FPGA en la que se pueden añadir retardos de propagación de punto y la simulación correr otra vez con estos valores back-anotados en la lista de conexiones.

## 3.4 XMOS

Un dispositivo XMOS es un sistema embebido que tiene la característica particular de contener un procesador con múltiples núcleos en un solo circuito integrado. Estos núcleos pueden compartir la carga de procesamiento de tareas (task) mediante el uso de un determinado número de hilos (threads) que operan en verdadero tiempo real.


*Figura 3.5 Logo de la empresa diseñadora del microcontrolador XMOS.*

Estos dispositivos XMOS reúnen todas las capacidades de los procesadores incluidos en los actuales dispositivos de lógica reconfigurable, tales como los DSP's, ASIC's y las FPGA's, pero a diferencia de estos, un XMOS se puede programar a través de un flujo de diseño unificado en lenguaje de programación C, C++ o XC, lo que le otorga un valor agregado debido a la popularidad de estos lenguajes.

### *3.4.1 Historia*

XMOS es una joven empresa privada de semiconductores sin fábrica que se dedica al diseño y documentación de Xcore, una tecnología de microcontrolador multinúcleo fácilmente escalable, que calcula multitarea en paralelo, XMOS fue fundada en julio de 2005 por Ali Dixon (entonces estudiante de último año en la Universidad de Bristol ), James Foster (ex director general de Oxford Semiconductor), Noel Hurley (anteriormente en ARM Holdings ), David May (ex arquitecto jefe de Inmos ) y Hitesh Mehta (Acacia Capital Partners). Recibió financiación inicial de la Universidad de Bristol.

En el otoño de 2006, XMOS asegura la financiación de Amadeus Capital Partners, DFJ Esprit, y Foundation Capital. El nombre XMOS es una referencia suelta a Inmos.

En junio de 2009, se anunció que las firmas de capital de riesgo estaban considerando la financiación de nuevas empresas relacionadas con XMOS. En diciembre de 2009 XMOS lanzó un sitio web de la comunidad, el intercambio XCore como un sitio para permitir y fomentar el debate y la colaboración innovadora y emprendedora.

### *3.4.2 Aplicaciones*

Los dispositivos XMOS son usados en sistemas en tiempo real embebidos que proveen interfaces de hardware y capacidad de procesamiento de datos. El número de núcleos usados dependen de la complejidad de la aplicación y son fácilmente escalables.

La tecnología del procesador XMOS es de uso general, por lo que se ha explotado en una variedad de mercados, incluyendo audio, azulejos LED, la comunicación, la

robótica y la innovación amateur. Esto permite a terceros establecer los productos y empresas basándose principalmente en la tecnología actual.

El procesador de Xcore es adecuado para una amplia gama de exigentes aplicaciones integradas y está siendo utilizado hoy en día con rápida expansión en audio, aplicaciones industriales y de automoción.

Es así que se considera una excelente opción para la implementación del código Reed-Solomon que se desarrolló en el presente trabajo.

### *3.4.3 Arquitectura*

Como ya se mencionó anteriormente, los dispositivos XMOS hacen uso de tecnología multinúcleo que tiene la siguiente estructura.

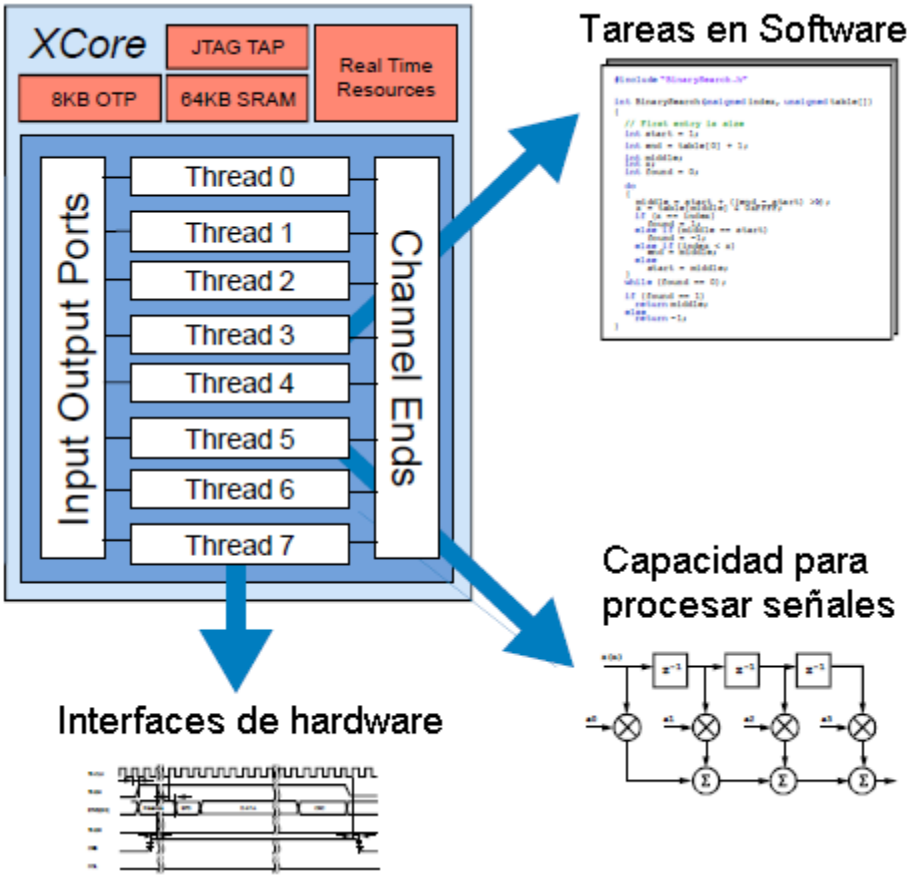


*Figura 3.6 Arquitectura general de un microcontrolador XMOS.*

El mecanismo de interconexión provee la habilidad de trasferir datos entre núcleos directamente. Su programación se hace a través de lenguajes de alto nivel usando herramientas de desarrollo de software. Este dispositivo posee una tecnología que le da la capacidad de responder a eventos sin necesidad de interrupciones debido a que cada uno de sus hilos (thread) se ejecuta al menos cada 20ns.

### *3.4.4 Programación*

Los procesadores XMOS se pueden programar con C, C++ o XC e incluso en ensamblador nativo.

Existen compiladores de C y C++ basados en LLVM, así como un compilador de XC desarrollado por XMOS. Algunas de las características de este hardware son mejor explotadas mediante el uso del lenguaje XC, que es un subconjunto del lenguaje de programación C extendido con construcciones de programación de alto nivel para la concurrencia explícita y temporizada de señales de E/S.

Así puede ser configurado para soportar una amplia gama de interfaces y periféricos, y responde mucho más rápido que los microcontroladores convencionales para ofrecer un rendimiento preciso en tiempo real.

### *3.4.5 Herramientas de software para programación*

XMOS también ha desarrollado herramientas de programación gratuitas que soportan C y C + + (a través de LLVM) y XC (el lenguaje desarrollado para explotar mejor la arquitectura) a través de un compilador originado por XMOS. Las herramientas de programación son multiplataforma y se pueden utilizar desde la línea de comandos o desde una interfaz gráfica de usuario basada en Eclipse que no tiene ningún costo adicional.

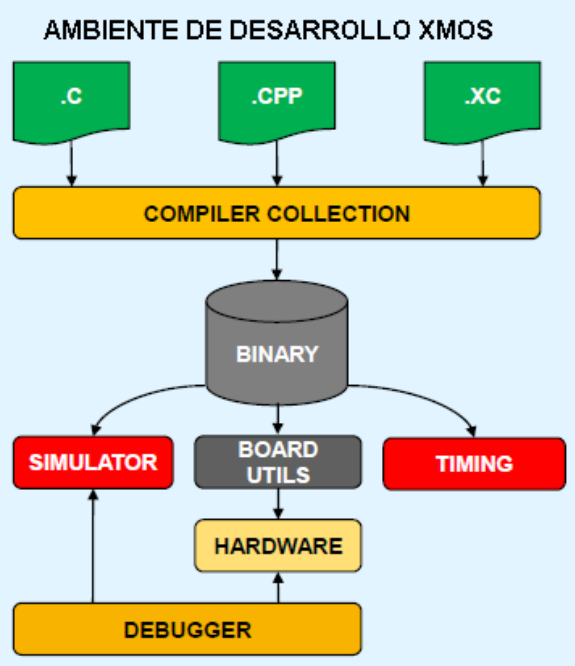


*Figura 3.7 Diagrama general del proceso de programación de un dispositivo XMOS.*

### *3.4.5.1 xTIMEcomposer Studio*

Esta suite completa de herramientas proporciona todo lo necesario para desarrollar y desplegar productos a base de Xcore (microcontroladores multinúcleo). El entorno de diseño xTIMEcomposer estudio se basa en las herramientas estándar de la industria, como el depurador GDB y compiladores LLVM, y se extendió con herramientas únicas tales como el XMOS Timing Analyzer (XTA) y el alcance del software xScope que aprovechan el determinismo de la arquitectura Xcore. Todas las herramientas están integradas en el IDE de Eclipse y son multiplataforma. xTIMEcomposer es gratis y está disponible en la web de XMOS.

*Figura 3.8 Logo del IDE xTIMEcomposer Studio desarrollado por XMOS.*

### *3.4.5.2 XMOS Timing Analyzer (XTA)*

Para verificar que el tiempo se ha cumplido, se introdujo una herramienta revolucionaria; XMOS Timing Analyser (XTA). XTA funciona analizando el binario de la aplicación y los informes del peor y mejor camino de los casos temporales a través de su código. Es una herramienta formal, lo que significa que ofrece 100% de cobertura de código sin la necesidad de un banco de pruebas. Se elimina la necesidad de más de la especificación de potencia de procesamiento.

*Figura 3.9 Logo de la herramienta XMOS Timing Analyzer desarrollada por XMOS.*

### *3.4.5.3 xSCOPE*

La depuración en circuito puede ser un reto. Una vez que el sistema está sujeto a los estímulos en tiempo real puede ser difícil de localizar las causas de un comportamiento inesperado.

Esto significa que la depuración en circuito requiere una forma de monitorear el sistema y exportación de datos sin afectar al código que está observando. La respuesta es la recopilación de datos intrusiva baja, que debe permitir que el sistema funcione sin cambiar su comportamiento o la afectación por introducción de sincronización. También es conveniente observar los datos pertinentes y de alto nivel, como variables de estado, valores de entrada / salida en los lazos de control o incluso observar directamente los flujos de datos. Esto es exactamente lo que ofrece xScope; en tiempo real, la instrumentación en el circuito de usuario

especificó, sondas de datos, sin afectar su diseño o el funcionamiento del dispositivo.


*Figura 3.10 Logo de la herramienta xScope desarrollada por XMOS.*

Una vez que conozco tanto el hardware como el software para desarrollar aplicaciones de comunicaciones móviles actuales, he llegado a la conclusión de usar el lenguaje de programación C++ porque además de ser un lenguaje comúnmente utilizado y el más estudiado y practicado dentro de mí carrera, en mi opinión no requiere de una capacitación adicional para ser comprendido por desarrolladores de aplicaciones móviles que deseen dar continuidad a este proyecto, y es más simple convertir a VHDL desde C++ que viceversa.

En lo que respecta a una elección particular de hardware es necesario hacer un propio análisis de la aplicación específica de este multicodificador, porque no siempre es conveniente usar una FPGA, pero también no siempre es conveniente usar un procesador multinúcleo XMOS, ya que en algunos casos esta tecnología podría llegar a verse sobrada y el FPGA seria en tal caso una mejor opción.

## 3.5 Eclipse

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE).

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios [24].

### *3.5.1 Antecedentes*

Eclipse comenzó como un proyecto de IBM Canadá. Fue desarrollado por OTI (Object Technology International) como reemplazo de VisualAge también desarrollado por OTI. En noviembre del 2001, se formó un consorcio para el desarrollo futuro de Eclipse como código abierto. En 2003, fue creada la fundación independiente de IBM [25].

Resumen de las versiones de Eclipse:

| Versión | Fecha de lanzamiento | Versión de plataforma |
|---|---|---|
| Kepler | 26 de junio de 2013 | 4.3 |
| Juno | 27 de junio de 2012 | 4.2 |
| Indigo | 22 de junio de 2011 | 3.7 |
| Helios | 23 junio de 2010 | 3.6 |
| Galileo | 24 de junio de 2009 | 3.5 |
| Ganymede | 25 junio de 2008 | 3.4 |
| Europa | 29 de junio de 2007 | 3.3 |
| Callisto | 30 de junio de 2006 | 3.2 |
| Eclipse 3.1 | 28 de junio 2005 | 3.1 |
| Eclipse 3.0 | 28 de junio de 2004 | 3.0 |

*Tabla 3.1 Versiones de Eclipse.*

### *3.5.2 Entorno de desarrollo integrado (del inglés IDE)*

Un entorno de desarrollo integrado, llamado también IDE (sigla en inglés de *integrated development environment*), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes [26].

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, PHP, Python, Java, C#, Delphi, Visual Basic, etc.

# CAPÍTULO 4. CODIFICADORES REED-SOLOMON, REALIZACIÓN DE PRUEBAS

Durante capítulos anteriores de este trabajo, se ha presentado una investigación que constituye una base teórica en la implementación de códigos Reed-Solomon para aplicaciones de radio definido por software, así como sus orígenes, importancia y la intención de aplicar ésta tecnología en dispositivos de lógica reprogramable.

Si bien los codificadores Reed-Solomon ya existen, no se les puede dar un uso libre, es decir, el módulo de software no se puede extraer y utilizar, sino más bien se utiliza sólo como una herramienta de comprobación. Siendo que el concepto de radio definido por software requiere que dicho módulo de codificación Reed-Solomon sea libre, es ahí donde surge la necesidad de implementar esta programación.

Ahora toca el turno de hablar de la parte práctica de los codificadores Reed-Solomon que constituyen la base funcional de este multicodificador, me refiero a los códigos RS(7,3) y RS(15,11) para ser más precisos, así se ha recurrido a herramientas de comprobación que manejan los codificadores Reed-Solomon, específicamente MATLAB. Estos módulos de comprobación de códigos Reed-Solomon han sido de gran ayuda a la hora de calcular los Campos Finitos de Galois (GF ($2^m$)), ya que permiten obtener los coeficientes del polinomio generador específicos de cada tipo de código Reed-Solomon, partiendo del hecho de que para cada tipo de código existe un solo polinomio generador. Al ingresar los coeficientes de los multiplicadores de Galois con la ayuda de MATLAB se permite así ahorrar carga computacional al dispositivo reprogramable.

## 4.1 Ejemplos Prácticos de Funcionamiento de códigos Reed-Solomon

Con el objeto de comprender mejor todo el proceso de codificación de los códigos Reed-Solomon, se desarrollarán dos ejemplos prácticos sin omitir ningún paso. Todo el proceso efectuado en estos ejemplos constituye, en último término, el algoritmo computacional empleado en la programación del software que permita realizar todo el proceso automáticamente.

### *4.1.1 Codificador Reed-Solomon (7, 3)*

Como primer paso definiremos las herramientas a emplear que son:

- Campos de Galois.
- Tabla de verdad de la compuerta XOR.
- Arquitectura del codificador Reed-Solomon.

Campos de Galois.

Estos están definidos en la siguiente tabla, estos valores ya son predeterminados, pueden ser calculados como se explicó anteriormente para fines prácticos tomaremos en cuenta la siguiente tabla:

| CAMPOS DE GALOIS PARA UN RS(7, 3) | | | | | |
|---|---|---|---|---|---|
| **4** | | **7** | | **5** | |
| X0= | 0 | X0= | 0 | X0= | 0 |
| X1= | 4 | X1= | 7 | X1= | 5 |
| X2= | 3 | X2= | 5 | X2= | 1 |
| X3= | 7 | X3= | 2 | X3= | 4 |
| X4= | 6 | X4= | 1 | X4= | 2 |
| X5= | 2 | X5= | 6 | X5= | 7 |
| X6= | 5 | X6= | 4 | X6= | 3 |
| X7= | 1 | X7= | 3 | X7= | 6 |

*Tabla 4.1 Multiplicadores de Galois para RS(7, 3).*

Tabla de verdad de una compuerta XOR:

| XOR | | |
|---|---|---|
| **A** | **B** | **f** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Tabla 4.2 Tabla de verdad de la compuerta XOR.*

Arquitectura genérica del codificador Reed-Solomon:

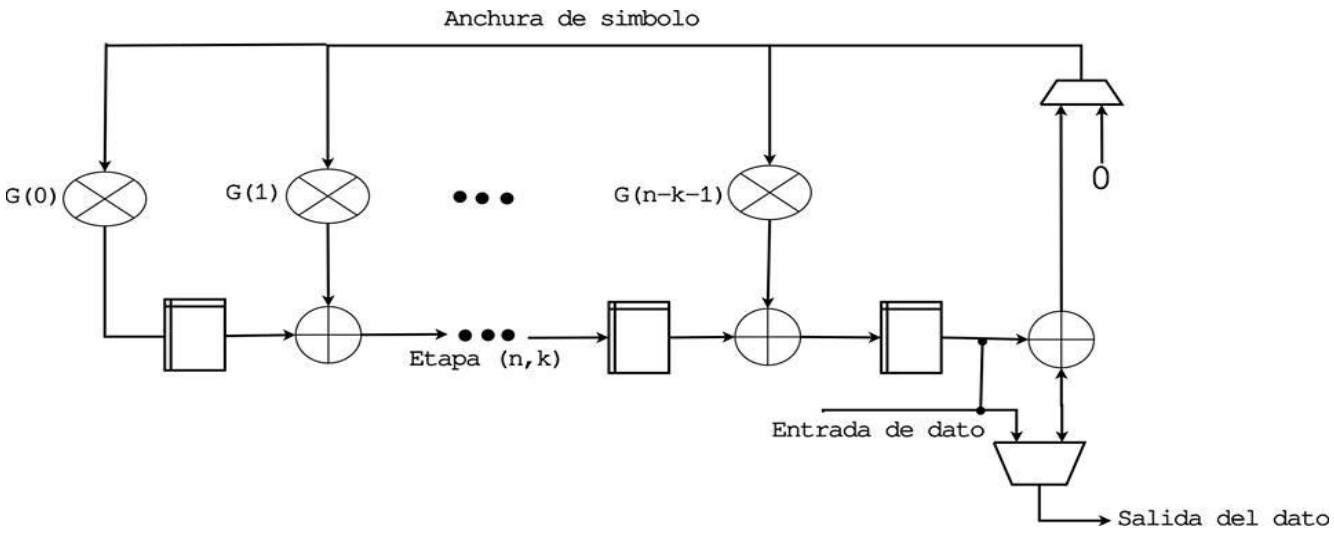


*Figura 4.1 Arquitectura genérica del codificador Reed-Solomon.*

Arquitectura a emplear para ejemplificar el codificador Reed-Solomon (7, 3):

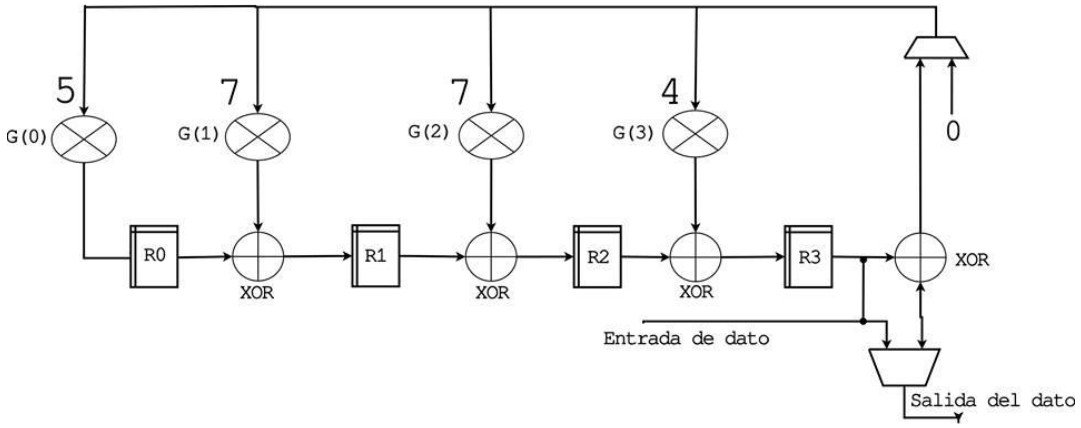


*Figura 4.2 Arquitectura del codificador Reed-Solomon (7, 3).*

Ahora los símbolos que introduciremos al codificador Reed-Solomon serán (1, 3, 7) y empezaremos a revisar como fluyen estos dentro de la arquitectura del codificador.

El primer símbolo a ingresar es el (1) así que este lo convertimos a su forma binaria que es la siguiente [001], en este momento cabe resaltar que todos los registros se encuentran vacíos así que entra casi directo, como se observa en el siguiente esquema:

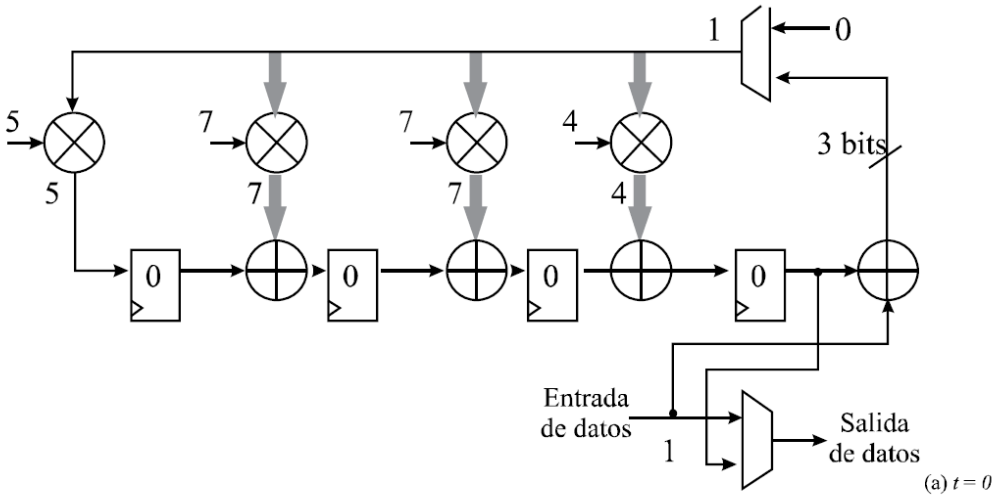


*Figura 4.3 Inserción del símbolo "1" en el codificador Reed-Solomon (7, 3).*

Los valores obtenidos de los campos de Galois los obtenemos de la tabla de estos, únicamente en esta ocasión que los registros se encuentran vacíos pasan directos y comenzaremos a llenar una tabla donde iremos vaciando nuestros registros para que quede claro.

| **ENTRADA (1,3,7)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **Primera vuelta** | 5 | 101 | 7 | 111 | 7 | 111 | 4 | 100 |
| **Segunda vuelta** | | | | | | | | |
| **Tercera vuelta** | | | | | | | | |

*Tabla 4.3 Registros obtenidos después de la inserción del primer símbolo al RS(7, 3).*

Ahora para la segunda vuelta hay que poner más atención ya que los registros no están vacíos y hay que realizar las operaciones correspondientes de las compuertas XOR para obtener los nuevos registros. Iniciamos identificando el símbolo que introduciremos que para nuestro ejemplo es el (3) el cual lo convertimos en su forma binaria que es la siguiente [011].

Como podemos observar en nuestro diagrama a la primera compuerta donde llega nuestro símbolo de entrada tiene a su otra entrada el registro tres R3 así que realizamos la operación entre el símbolo (3) de entrada y el (4) que se encuentra dentro del R3, esto apoyándonos de la tabla de verdad de la compuerta XOR.

| | Símbolo | Binario |
|---|---|---|
| R3 | 4 | 100 |
| Entrada | 3 | 011 |
| **Resultado** | **7** | **111** |

*Tabla 4.4 Operación de la compuerta XOR con el segundo símbolo de entrada y el R3 almacenado.*

Por lo tanto el símbolo que llegara a nuestros campos de Galois será el 7 así que apoyándonos de nuestra tabla buscaremos el X7 para cada campo, como lo mostramos en la siguiente tabla:

| **CAMPOS DE GALOIS PARA UN RS(7, 3)** | | | |
|---|---|---|---|
| **5** | **7** | **7** | **4** |
| X7= 6 | X7= 3 | X7= 3 | X7= 1 |

*Tabla 4.5 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "7".*

Ahora en el siguiente esquema veremos las operaciones que tendremos que realizar de las compuertas XOR'S tomando en cuenta los valores de los campos de Galois que entran a estas compuertas y el registro obtenido de la primera vuelta que a su vez es la otra entrada de cada compuerta, cabe resaltar que para actualizar el registro [0] no hace falta hacer ninguna operación ya que no tiene ninguna compuerta de por medio:

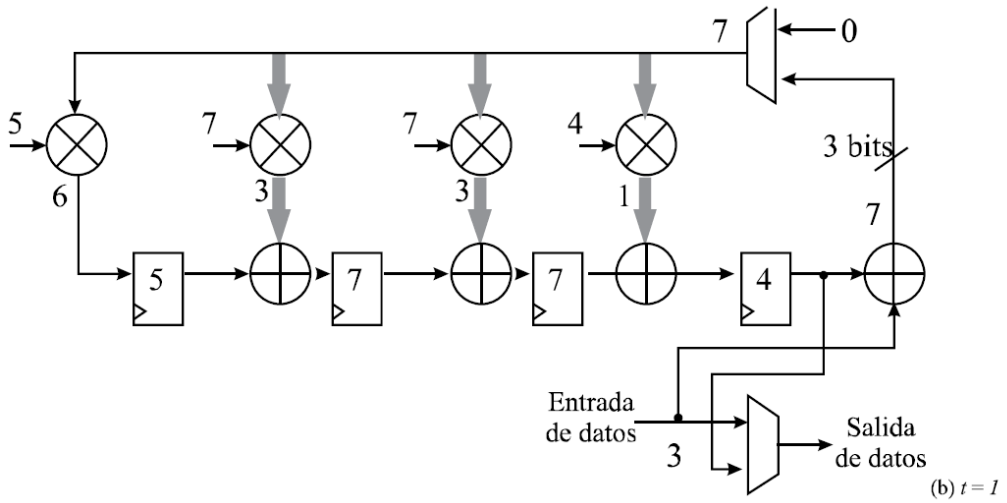


*Figura 4.4 Inserción del símbolo "3" en el codificador Reed-Solomon (7, 3).*

A continuación se muestran las operaciones que se realizan para actualizar los registros 1, 2 y 3:

| | Símbolo | Binario | | Símbolo | Binario | | Símbolo | Binario |
|---|---|---|---|---|---|---|---|---|
| R0 | 5 | 101 | R1 | 7 | 111 | R2 | 7 | 111 |
| G1 | 3 | 011 | G2 | 3 | 011 | G3 | 1 | 001 |
| **Resultado** | **6** | **110** | **Resultado** | **4** | **100** | **Resultado** | **6** | **110** |

*Tabla 4.6 Operaciones de las compuertas XOR's.*

Nuevamente se vacían los nuevos símbolos almacenados en nuestros registros a nuestra tabla de registros llenando así la segunda vuelta:

| **ENTRADA (1,3,7)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **Primera vuelta** | 5 | 101 | 7 | 111 | 7 | 111 | 4 | 100 |
| **Segunda vuelta** | 6 | 110 | 6 | 110 | 4 | 100 | 6 | 110 |
| **Tercera vuelta** | | | | | | | | |

*Tabla 4.7 Registros obtenidos después de la inserción del segundo símbolo al RS(7, 3).*

Ahora se procede a introducir el tercer símbolo que es el [7] en su forma binaria [111], se realiza la operación de la compuerta XOR donde se involucra el símbolo de entrada actual y el símbolo almacenado en nuestro registro tres que en este caso es el [6] que en su forma binaria es [110], la operación es la siguiente:

| | Símbolo | Binario |
|---|---|---|
| R3 | 6 | 110 |
| Entrada | 7 | 111 |
| **Resultado** | **1** | **001** |

*Tabla 4.8 Operación de la compuerta XOR con el tercer símbolo de entrada y el R3 almacenado.*

Por lo tanto el símbolo que llegará a nuestros campos de Galois será el 1 así que de la tabla buscaremos el X1 para cada campo, como lo mostramos en la siguiente tabla:

| **CAMPOS DE GALOIS PARA UN RS(7,3)** | | | |
|---|---|---|---|
| **5** | **7** | **7** | **4** |
| X1= 5 | X1= 7 | X1= 7 | X1= 4 |

*Tabla 4.9 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "1".*

En el siguiente esquema se muestran las operaciones que se tienen que realizar de las compuertas XOR'S tomando en cuenta los valores de los campos de Galois que entran a estas compuertas y el registro obtenido de la primera vuelta que a su vez es la otra entrada de cada compuerta, cabe resaltar que para actualizar el registro [0] no hace falta hacer ninguna operación ya que no tiene ninguna compuerta de por medio:

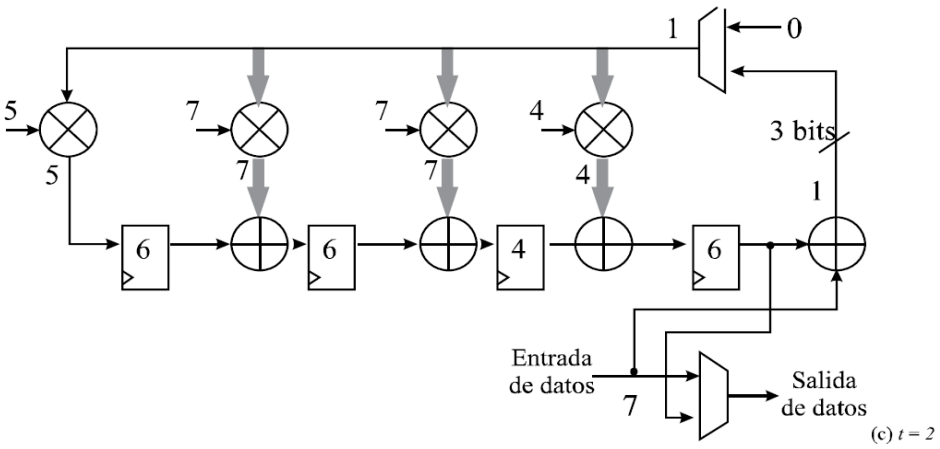


*Figura 4.5 Inserción del símbolo "7" en el codificador Reed-Solomon (7, 3).*

Ahora se observan las operaciones que se realizan para actualizar los registros 1, 2 y 3:

| | **Símbolo** | **Binario** | | **Símbolo** | **Binario** | | **Símbolo** | **Binario** |
|---|---|---|---|---|---|---|---|---|
| R0 | 6 | 110 | R1 | 6 | 110 | R2 | 4 | 100 |
| G1 | 7 | 111 | G2 | 7 | 111 | G3 | 4 | 100 |
| **Resultado** | **1** | **001** | **Resultado** | **1** | **001** | **Resultado** | **0** | **000** |

*Tabla 4.10 Operaciones de las compuertas XOR's.*

Nuevamente se vacían los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la tercera vuelta:

| **ENTRADA (1,3,7)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **Primera vuelta** | 5 | 101 | 7 | 111 | 7 | 111 | 4 | 100 |
| **Segunda vuelta** | 6 | 110 | 6 | 110 | 4 | 100 | 6 | 110 |
| **Tercera vuelta** | 5 | 101 | 1 | 001 | 1 | 001 | 0 | 000 |

*Tabla 4.11 Registros obtenidos después de la inserción del tercer símbolo al RS(7, 3).*

En el siguiente esquema observaremos los últimos símbolos almacenados en los registros de codificador Reed-Solomon:

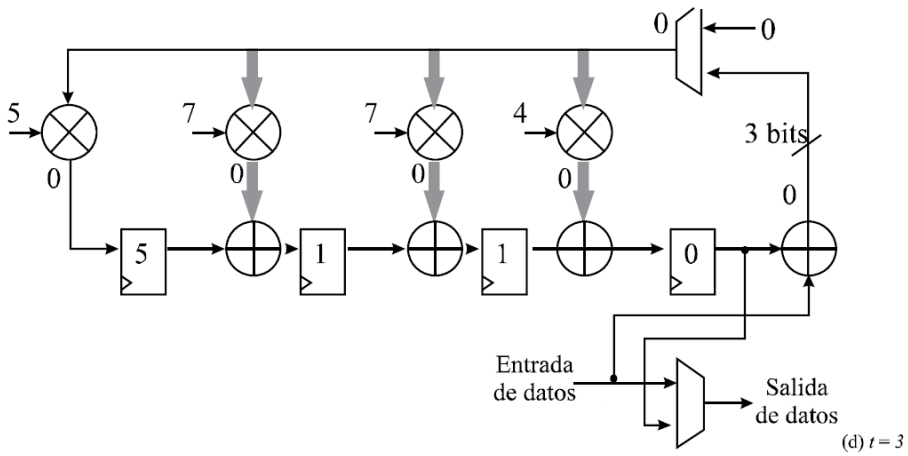


*Figura 4.6 Símbolos almacenados en los registros del codificador Reed-Solomon (7, 3).*

Ahora los símbolos almacenados en los registros los vaciamos de derecha a izquierda y se agregan a la entrada inicial:

Salida: [1, 3,7, 0, 1, 1,5]

*Los símbolos en rojo son el código de redundancia.

### *4.1.2 Codificador Reed-Solomon (15,11)*

Como primer paso definiremos las herramientas a emplear que son:

- Campos de Galois.
- Tabla de verdad de la compuerta XOR.
- Arquitectura del codificador Reed-Solomon.

Campos de Galois.

Estos están definidos en la siguiente tabla, estos valores ya son predeterminados, pueden ser calculados como se explicó anteriormente.

| CAMPOS DE GALOIS PARA UN RS(15,11) | | | | | | | |
|---|---|---|---|---|---|---|---|
| **12** | | **1** | | **3** | | **15** | |
| X0= | 0 | X0= | 0 | X0= | 0 | X0= | 0 |
| X1= | 12 | X1= | 1 | X1= | 3 | X1= | 15 |
| X2= | 11 | X2= | 2 | X2= | 6 | X2= | 13 |
| X3= | 7 | X3= | 3 | X3= | 5 | X3= | 2 |
| X4= | 5 | X4= | 4 | X4= | 12 | X4= | 9 |
| X5= | 9 | X5= | 5 | X5= | 15 | X5= | 6 |
| X6= | 14 | X6= | 6 | X6= | 10 | X6= | 4 |
| X7= | 2 | X7= | 7 | X7= | 9 | X7= | 11 |
| X8= | 10 | X8= | 8 | X8= | 11 | X8= | 1 |
| X9= | 6 | X9= | 9 | X9= | 8 | X9= | 14 |
| X10= | 1 | X10= | 10 | X10= | 13 | X10= | 12 |
| X11= | 13 | X11= | 11 | X11= | 14 | X11= | 3 |
| X12= | 15 | X12= | 12 | X12= | 7 | X12= | 8 |
| X13= | 3 | X13= | 13 | X13= | 4 | X13= | 7 |
| X14= | 4 | X14= | 14 | X14= | 1 | X14= | 5 |
| X15= | 8 | X15= | 15 | X15= | 2 | X15= | 10 |

*Tabla 4.12 Multiplicadores de Galois para RS(15, 11).*

Tabla de verdad de una compuerta XOR:

| XOR | | |
|---|---|---|
| **A** | **B** | **f** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Tabla 4.13 Tabla de verdad de la compuerta XOR.*

Arquitectura genérica del codificador Reed-Solomon:

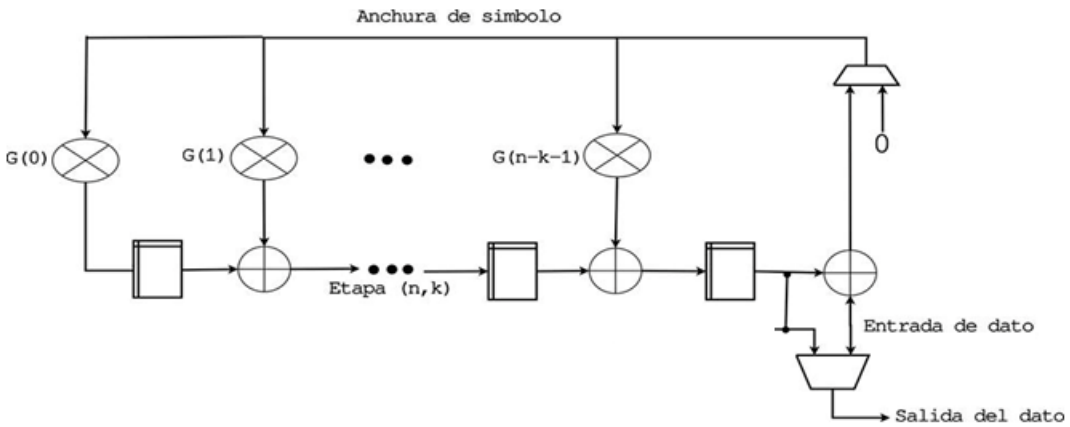


*Figura 4.7 Arquitectura genérica del codificador Reed-Solomon.*

Arquitectura a emplear para ejemplificar el codificador Reed-Solomon (15,11):

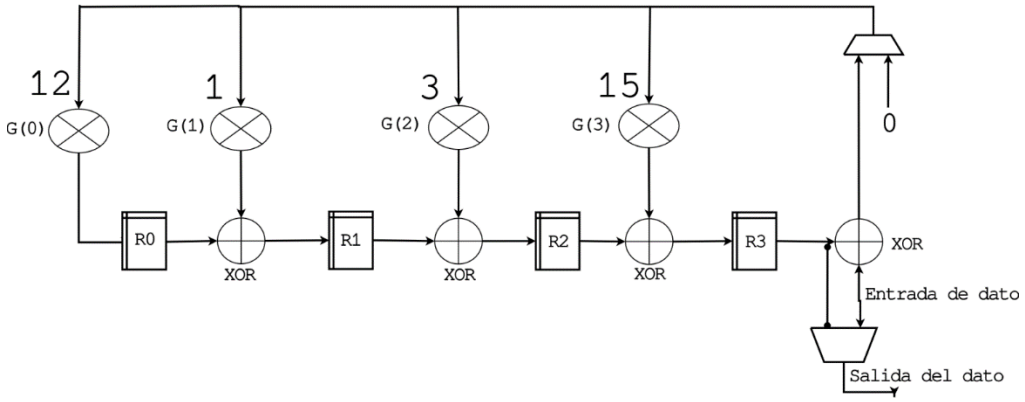


*Figura 4.8 Arquitectura del codificador Reed-Solomon (15, 11).*

Ahora los símbolos que introduciremos al codificador Reed-Solomon serán (3,6,9,12,2,4,6,8,13,15,1) y empezaremos a revisar como fluyen estos dentro de la arquitectura del codificador.

Como tenemos 11 símbolos por lo tanto necesitaremos de 11 vueltas a nuestro codificador y al finalizar cada una de ellas actualizaremos nuestros registros, los símbolos ingresaran de izquierda a derecha iniciando por el símbolo 3 en la vuelta número uno y finalizando con el símbolo 1 en la vuelta número 11.

**Vuelta número 1.**
**Símbolo de entrada ‘3’ [0011].**

Operación XOR del símbolo 3 que en binario es [0011] con el R3 que tiene almacenado el símbolo 0 que en binario es [0000]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 0 | 0000 |
| Entrada | 3 | 0011 |
| **Resultado** | **3** | **0011** |

*Tabla 4.14 Operación de la compuerta XOR con el primer símbolo de entrada y el R3 almacenado.*

El símbolo 3 es el que ingresará a los multiplicadores de Galois y obtendremos a la salida de estos los siguientes valores:

| **CAMPOS DE GALOIS PARA UN RS(15,11)** | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X3= 7 | X3= 3 | X3= 5 | X3= 2 |

*Tabla 4.15 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "3".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 7 que en su representación binaria es [0111].

Para actualizar los registros restantes se necesita realizar las operaciones de las compuertas XOR's teniendo en cuenta que los registros en este momento se encuentran en 0 y su forma binaria es [0000], en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenarán:

| | **Símbolo** | **Binario** | | **Símbolo** | **Binario** | | **Símbolo** | **Binario** |
|---|---|---|---|---|---|---|---|---|
| R0 | 0 | 0000 | R1 | 0 | 0000 | R2 | 0 | 0000 |
| G1 | 3 | 0011 | G2 | 5 | 0101 | G3 | 2 | 0010 |
| **Resultado** | **3** | **0011** | **Resultado** | **5** | **0101** | **Resultado** | **2** | **0010** |

*Tabla 4.16 Operaciones de las compuertas XOR's.*

En este momento se vacían los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 1:

| **ENTRADA (3,6,9,12,2,4,6,8,13,15,1)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |

*Tabla 4.17 Registros obtenidos después de la inserción del primer símbolo al RS(15, 11).*

### Vuelta número 2.
### Símbolo de entrada '6' [0110].

Operación XOR del símbolo 6 que en binario es [0110] con el R3 que tiene almacenado el símbolo 2 que en binario es [0010]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 2 | 0010 |
| Entrada | 6 | 0110 |
| **Resultado** | **4** | **0100** |

*Tabla 4.18 Operación de la compuerta XOR con el segundo símbolo de entrada y el R3 almacenado.*

El símbolo 4 es el que ingresará a los multiplicadores de Galois y obtendremos a la salida de estos los siguientes valores:

| **CAMPOS DE GALOIS PARA UN RS(15,11)** | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X4= 5 | X4= 4 | X4= 12 | X4= 9 |

*Tabla 4.19 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "4".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 5 que en su representación binaria es [0101].

Para actualizar los registros restantes necesitamos realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenarán:

| | **Símbolo** | **Binario** | | **Símbolo** | **Binario** | | **Símbolo** | **Binario** |
|---|---|---|---|---|---|---|---|---|
| R0 | 7 | 0111 | R1 | 3 | 0011 | R2 | 5 | 0101 |
| G1 | 4 | 0100 | G2 | 12 | 1100 | G3 | 9 | 1001 |
| **Resultado** | **3** | **0011** | **Resultado** | **15** | **1111** | **Resultado** | **12** | **1100** |

*Tabla 4.20 Operaciones de las compuertas XOR's.*

En este momento vaciamos los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 2:

| ENTRADA (3,6,9,12,2,4,6,8,13,15,1) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |

*Tabla 4.21 Registros obtenidos después de la inserción del segundo símbolo al RS(15, 11).*

### Vuelta número 3.
### Símbolo de entrada '9' [1001].

Operación XOR del símbolo 9 que en binario es [1001] con el R3 que tiene almacenado el símbolo 12 que en binario es [1100]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 12 | 1100 |
| Entrada | 9 | 1001 |
| **Resultado** | **5** | **0101** |

*Tabla 4.22 Operación de la compuerta XOR con el tercer símbolo de entrada y el R3 almacenado.*

El símbolo 5 es el que ingresará a los multiplicadores de Galois y se obtendrá a la salida de estos los siguientes valores:

| CAMPOS DE GALOIS PARA UN RS(15,11) | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X5= 9 | X5= 5 | X5= 15 | X5= 6 |

*Tabla 4.23 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "5".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 9 que en su representación binaria es [1001].

Para actualizar los registros restantes necesitamos realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenarán:

| | Símbolo | Binario | | Símbolo | Binario | | Símbolo | Binario |
|---|---|---|---|---|---|---|---|---|
| R0 | 5 | 0101 | R1 | 3 | 0011 | R2 | 15 | 1111 |
| G1 | 5 | 0101 | G2 | 15 | 1111 | G3 | 6 | 0110 |
| **Resultado** | **0** | **0000** | **Resultado** | **12** | **1100** | **Resultado** | **9** | **1001** |

*Tabla 4.24 Operaciones de las compuertas XOR's.*

En este momento se vacían los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 3:

| **ENTRADA (3,6,9,12,2,4,6,8,13,15,1)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |
| **3** | 9 | 1001 | 0 | 0000 | 12 | 1100 | 9 | 1001 |

*Tabla 4.25 Registros obtenidos después de la inserción del tercer símbolo al RS(15, 11).*

### Vuelta número 4.
### Símbolo de entrada '12' [1100].

Operación XOR del símbolo 12 que en binario es [1100] con el R3 que tiene almacenado el símbolo 9 que en binario es [1001]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 9 | 1001 |
| Entrada | 12 | 1100 |
| **Resultado** | **5** | **0101** |

*Tabla 4.26 Operación de la compuerta XOR con el cuarto símbolo de entrada y el R3 almacenado.*

El símbolo 5 es el que ingresará a los multiplicadores de Galois y obtendremos a la salida de estos los siguientes valores:

| **CAMPOS DE GALOIS PARA UN RS(15,11)** | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X5= 9 | X5= 5 | X5= 15 | X5= 6 |

*Tabla 4.27 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "5".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro

con el valor obtenido del multiplicador que en este caso es 9 que en su representación binaria es [1001].

Para actualizar los registros restantes necesitamos realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenaran:

| | **Símbolo** | **Binario** | | **Símbolo** | **Binario** | | **Símbolo** | **Binario** |
|---|---|---|---|---|---|---|---|---|
| R0 | 9 | 1001 | R1 | 0 | 0000 | R2 | 12 | 1100 |
| G1 | 5 | 0101 | G2 | 15 | 1111 | G3 | 6 | 0110 |
| **Resultado** | **12** | **1100** | **Resultado** | **15** | **1111** | **Resultado** | **10** | **1010** |

*Tabla 4.28 Operaciones de las compuertas XOR's.*

En este momento se vacían los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 4:

| **ENTRADA (3,6,9,12,2,4,6,8,13,15,1)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |
| **3** | 9 | 1001 | 0 | 0000 | 12 | 1100 | 9 | 1001 |
| **4** | 9 | 1001 | 12 | 1100 | 15 | 1111 | 10 | 1010 |

*Tabla 4.29 Registros obtenidos después de la inserción del cuarto símbolo al RS(15, 11).*

### Vuelta número 5.
### Símbolo de entrada '2' [0010].

Operación XOR del símbolo 2 que en binario es [0010] con el R3 que tiene almacenado el símbolo 10 que en binario es [1010]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 10 | 1010 |
| Entrada | 2 | 0010 |
| **Resultado** | **8** | **1000** |

*Tabla 4.30 Operación de la compuerta XOR con el quinto símbolo de entrada y el R3 almacenado.*

El símbolo 8 es el que ingresará a los multiplicadores de Galois y obtendremos a la salida de estos los siguientes valores:

| **CAMPOS DE GALOIS PARA UN CODIFICADOR RS(15,11)** | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X8= 10 | X8= 8 | X8= 11 | X8= 1 |

*Tabla 4.31 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "8".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 10 que en su representación binaria es [1010].

Para actualizar los registros restantes necesitamos realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenarán:

| | **Símbolo** | **Binario** | | **Símbolo** | **Binario** | | **Símbolo** | **Binario** |
|---|---|---|---|---|---|---|---|---|
| R0 | 9 | 1001 | R1 | 12 | 1100 | R2 | 15 | 1111 |
| G1 | 8 | 1000 | G2 | 11 | 1011 | G3 | 1 | 0001 |
| **Resultado** | **1** | **0001** | **Resultado** | **7** | **0111** | **Resultado** | **14** | **1110** |

*Tabla 4.32 Operaciones de las compuertas XOR's.*

En este momento vaciamos los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 5:

| **ENTRADA (3,6,9,12,2,4,6,8,13,15,1)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |
| **3** | 9 | 1001 | 0 | 0000 | 12 | 1100 | 9 | 1001 |
| **4** | 9 | 1001 | 12 | 1100 | 15 | 1111 | 10 | 1010 |
| **5** | 10 | 1010 | 1 | 0001 | 7 | 0111 | 14 | 1110 |

*Tabla 4.33 Registros obtenidos después de la inserción del quinto símbolo al RS(15, 11).*

### Vuelta número 6.
### Símbolo de entrada '4' [0100].

Operación XOR del símbolo 4 que en binario es [0100] con el R3 que tiene almacenado el símbolo 14 que en binario es [1110]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 14 | 1110 |
| Entrada | 4 | 0100 |
| **Resultado** | **10** | **1010** |

*Tabla 4.34 Operación de la compuerta XOR con el sexto símbolo de entrada y el R3 almacenado.*

El símbolo 10 es el que ingresará a los multiplicadores de Galois y se obtendrá a la salida de estos los siguientes valores:

| **CAMPOS DE GALOIS PARA UN RS(15,11)** | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X10= 1 | X10= 10 | X10= 13 | X10= 12 |

*Tabla 4.35 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "10".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 1 que en su representación binaria es [0001].

Para actualizar los registros restantes necesitamos realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenarán:

| | **Símbolo** | **Binario** | | **Símbolo** | **Binario** | | **Símbolo** | **Binario** |
|---|---|---|---|---|---|---|---|---|
| R0 | 10 | 1010 | R1 | 1 | 0001 | R2 | 7 | 0111 |
| G1 | 10 | 1010 | G2 | 13 | 1101 | G3 | 12 | 1100 |
| **Resultado** | **0** | **0000** | **Resultado** | **12** | **1100** | **Resultado** | **11** | **1011** |

*Tabla 4.36 Operaciones de las compuertas XOR's.*

En este momento vaciamos los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 6:

| ENTRADA (3,6,9,12,2,4,6,8,13,15,1) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |
| **3** | 9 | 1001 | 0 | 0000 | 12 | 1100 | 9 | 1001 |
| **4** | 9 | 1001 | 12 | 1100 | 15 | 1111 | 10 | 1010 |
| **5** | 10 | 1010 | 1 | 0001 | 7 | 0111 | 14 | 1110 |
| **6** | 1 | 0001 | 0 | 0000 | 12 | 1100 | 11 | 1011 |

*Tabla 4.37 Registros obtenidos después de la inserción del sexto símbolo al RS(15, 11).*

### Vuelta número 7.
### Símbolo de entrada '6' [0110].

Operación XOR del símbolo 6 que en binario es [0110] con el R3 que tiene almacenado el símbolo 11 que en binario es [1011]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 11 | 1011 |
| Entrada | 6 | 0110 |
| **Resultado** | **13** | **1101** |

*Tabla 4.38 Operación de la compuerta XOR con el séptimo símbolo de entrada y el R3 almacenado.*

El símbolo 13 es el que ingresará a los multiplicadores de Galois y obtendremos a la salida de estos los siguientes valores:

| CAMPOS DE GALOIS PARA UN RS(15,11) | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X13= 3 | X13= 13 | X13= 4 | X13= 7 |

*Tabla 4.39 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "13".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 3 que en su representación binaria es [0011].

Para actualizar los registros restantes necesitamos realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenarán:

| | Símbolo | Binario | | Símbolo | Binario | | Símbolo | Binario |
|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 0001 | R1 | 0 | 0000 | R2 | 12 | 1100 |
| G1 | 13 | 1101 | G2 | 4 | 0100 | G3 | 7 | 0111 |
| **Resultado** | **12** | **1100** | **Resultado** | **4** | **0100** | **Resultado** | **11** | **1011** |

*Tabla 4.40 Operaciones de las compuertas XOR's.*

En este momento se vacían los nuevos símbolos almacenados en nuestros registros a la tabla de registros llenando así la vuelta número 7:

| **ENTRADA (3,6,9,12,2,4,6,8,13,15,1)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |
| **3** | 9 | 1001 | 0 | 0000 | 12 | 1100 | 9 | 1001 |
| **4** | 9 | 1001 | 12 | 1100 | 15 | 1111 | 10 | 1010 |
| **5** | 10 | 1010 | 1 | 0001 | 7 | 0111 | 14 | 1110 |
| **6** | 1 | 0001 | 0 | 0000 | 12 | 1100 | 11 | 1011 |
| **7** | 3 | 0011 | 12 | 1100 | 4 | 0100 | 11 | 1011 |

*Tabla 4.41 Registros obtenidos después de la inserción del séptimo símbolo al RS(15, 11).*

**Vuelta número 8.**
**Símbolo de entrada '8' [1000].**

Operación XOR del símbolo 8 que en binario es [1000] con el R3 que tiene almacenado el símbolo 11 que en binario es [1011]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 11 | 1011 |
| Entrada | 8 | 1000 |
| **Resultado** | **3** | **0011** |

*Tabla 4.42 Operación de la compuerta XOR con el octavo símbolo de entrada y el R3 almacenado.*

El símbolo 3 es el que ingresará a los multiplicadores de Galois y se obtendrá a la salida de estos los siguientes valores:

| **CAMPOS DE GALOIS PARA UN RS(15,11)** | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X3= 7 | X3= 3 | X3= 5 | X3= 2 |

*Tabla 4.43 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "3".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 7 que en su representación binaria es [0111].

Para actualizar los registros restantes se necesitan realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenarán:

| | **Símbolo** | **Binario** | | **Símbolo** | **Binario** | | **Símbolo** | **Binario** |
|---|---|---|---|---|---|---|---|---|
| R0 | 3 | 0011 | R1 | 12 | 1100 | R2 | 4 | 0100 |
| G1 | 3 | 0011 | G2 | 5 | 0101 | G3 | 2 | 0010 |
| **Resultado** | **0** | **0000** | **Resultado** | **9** | **1001** | **Resultado** | **6** | **0110** |

*Tabla 4.44 Operaciones de las compuertas XOR's.*

En este momento se vacían los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 8:

| **ENTRADA (3,6,9,12,2,4,6,8,13,15,1)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |
| **3** | 9 | 1001 | 0 | 0000 | 12 | 1100 | 9 | 1001 |
| **4** | 9 | 1001 | 12 | 1100 | 15 | 1111 | 10 | 1010 |
| **5** | 10 | 1010 | 1 | 0001 | 7 | 0111 | 14 | 1110 |
| **6** | 1 | 0001 | 0 | 0000 | 12 | 1100 | 11 | 1011 |
| **7** | 3 | 0011 | 12 | 1100 | 4 | 0100 | 11 | 1011 |
| **8** | 7 | 0111 | 0 | 0000 | 9 | 1001 | 6 | 0110 |

*Tabla 4.45 Registros obtenidos después de la inserción del octavo símbolo al RS(15, 11).*

### Vuelta número 9.
### Símbolo de entrada '13' [1101].

Operación XOR del símbolo 13 que en binario es [1101] con el R3 que tiene almacenado el símbolo 6 que en binario es [0110]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 6 | 0110 |
| Entrada | 13 | 1101 |
| **Resultado** | **11** | **1011** |

*Tabla 4.46 Operación de la compuerta XOR con el noveno símbolo de entrada y el R3 almacenado.*

El símbolo 11 es el que ingresará a los multiplicadores de Galois y se obtendrá a la salida de estos los siguientes valores:

| **CAMPOS DE GALOIS PARA UN RS(15,11)** | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X11= 13 | X11= 11 | X11= 14 | X11= 3 |

*Tabla 4.47 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "11".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 13 que en su representación binaria es [1101].

Para actualizar los registros restantes necesitamos realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenarán:

| | **Símbolo** | **Binario** | | **Símbolo** | **Binario** | | **Símbolo** | **Binario** |
|---|---|---|---|---|---|---|---|---|
| R0 | 7 | 0111 | R1 | 0 | 0000 | R2 | 9 | 1001 |
| G1 | 11 | 1011 | G2 | 14 | 1110 | G3 | 3 | 0011 |
| **Resultado** | **12** | **1100** | **Resultado** | **14** | **1110** | **Resultado** | **10** | **1010** |

*Tabla 4.48 Operaciones de las compuertas XOR's.*

En este momento se vacían los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 9:

| ENTRADA (3,6,9,12,2,4,6,8,13,15,1) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |
| **3** | 9 | 1001 | 0 | 0000 | 12 | 1100 | 9 | 1001 |
| **4** | 9 | 1001 | 12 | 1100 | 15 | 1111 | 10 | 1010 |
| **5** | 10 | 1010 | 1 | 0001 | 7 | 0111 | 14 | 1110 |
| **6** | 1 | 0001 | 0 | 0000 | 12 | 1100 | 11 | 1011 |
| **7** | 3 | 0011 | 12 | 1100 | 4 | 0100 | 11 | 1011 |
| **8** | 7 | 0111 | 0 | 0000 | 9 | 1001 | 6 | 0110 |
| **9** | 13 | 1101 | 12 | 1100 | 14 | 1110 | 10 | 1010 |

*Tabla 4.49 Registros obtenidos después de la inserción del noveno símbolo al RS(15, 11).*

### Vuelta número 10.
### Símbolo de entrada '15' [1111].

Operación XOR del símbolo 15 que en binario es [1111] con el R3 que tiene almacenado el símbolo 10 que en binario es [1010]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 10 | 1010 |
| Entrada | 15 | 1111 |
| **Resultado** | **5** | **0101** |

*Tabla 4.50 Operación de la compuerta XOR con el décimo símbolo de entrada y el R3 almacenado.*

El símbolo 5 es el que ingresará a los multiplicadores de Galois y se obtendrá a la salida de estos los siguientes valores:

| CAMPOS DE GALOIS PARA UN RS(15,11) | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X5= 9 | X5= 5 | X5= 15 | X5= 6 |

*Tabla 4.51 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "5".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 9 que en su representación binaria es [1001].

Para actualizar los registros restantes necesitamos realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenarán:

| | Símbolo | Binario | | Símbolo | Binario | | Símbolo | Binario |
|---|---|---|---|---|---|---|---|---|
| R0 | 13 | 1101 | R1 | 12 | 1100 | R2 | 14 | 1110 |
| G1 | 5 | 0101 | G2 | 15 | 1111 | G3 | 6 | 0110 |
| **Resultado** | **8** | **1000** | **Resultado** | **3** | **0011** | **Resultado** | **8** | **1000** |

*Tabla 4.52 Operaciones de las compuertas XOR's.*

En este momento se vacían los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 10:

| **ENTRADA (3,6,9,12,2,4,6,8,13,15,1)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |
| **3** | 9 | 1001 | 0 | 0000 | 12 | 1100 | 9 | 1001 |
| **4** | 9 | 1001 | 12 | 1100 | 15 | 1111 | 10 | 1010 |
| **5** | 10 | 1010 | 1 | 0001 | 7 | 0111 | 14 | 1110 |
| **6** | 1 | 0001 | 0 | 0000 | 12 | 1100 | 11 | 1011 |
| **7** | 3 | 0011 | 12 | 1100 | 4 | 0100 | 11 | 1011 |
| **8** | 7 | 0111 | 0 | 0000 | 9 | 1001 | 6 | 0110 |
| **9** | 13 | 1101 | 12 | 1100 | 14 | 1110 | 10 | 1010 |
| **10** | 9 | 1001 | 8 | 1000 | 3 | 0011 | 8 | 1000 |

*Tabla 4.53 Registros obtenidos después de la inserción del décimo símbolo al RS(15, 11).*

**Vuelta número 11.**
**Símbolo de entrada '1' [0001].**

Operación XOR del símbolo 1 que en binario es [0001] con el R3 que tiene almacenado el símbolo 8 que en binario es [1000]:

| | Símbolo | Binario |
|---|---|---|
| R3 | 8 | 1000 |
| Entrada | 1 | 0001 |
| **Resultado** | **9** | **1001** |

*Tabla 4.54 Operación de la compuerta XOR con el último símbolo de entrada y el R3 almacenado.*

El símbolo 9 es el que ingresará a los multiplicadores de Galois y se obtendrá a la salida de estos los siguientes valores:

| CAMPOS DE GALOIS PARA UN RS(15,11) | | | |
|---|---|---|---|
| **12** | **1** | **3** | **15** |
| X9= 6 | X9= 9 | X9= 8 | X9= 14 |

*Tabla 4.55 Resultado de los multiplicadores de Galois al tener como entrada el símbolo "9".*

Ahora para actualizar el R0 no es necesario realizar ninguna operación ya que no interfiere ninguna compuerta XOR así que se actualiza directamente este registro con el valor obtenido del multiplicador que en este caso es 6 que en su representación binaria es [0110].

Para actualizar los registros restantes necesitamos realizar las operaciones de las compuertas XOR's teniendo en cuenta los registros almacenados, en la siguiente tabla se muestran las operaciones correspondientes así como también los nuevos registros que se almacenaran:

| | **Símbolo** | **Binario** | | **Símbolo** | **Binario** | | **Símbolo** | **Binario** |
|---|---|---|---|---|---|---|---|---|
| R0 | 9 | 1001 | R1 | 8 | 1000 | R2 | 3 | 0011 |
| G1 | 9 | 1001 | G2 | 8 | 1000 | G3 | 14 | 1110 |
| **Resultado** | **0** | **0000** | **Resultado** | **0** | **0000** | **Resultado** | **13** | **1101** |

*Tabla 4.56 Operaciones de las compuertas XOR's.*

En este momento vaciamos los nuevos símbolos almacenados en los registros a la tabla de registros llenando así la vuelta número 11:

| ENTRADA (3,6,9,12,2,4,6,8,13,15,1) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Número de vuelta** | **R0** | | **R1** | | **R2** | | **R3** | |
| | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario | Símbolo | Binario |
| **1** | 7 | 0111 | 3 | 0011 | 5 | 0101 | 2 | 0010 |
| **2** | 5 | 0101 | 3 | 0011 | 15 | 1111 | 12 | 1100 |
| **3** | 9 | 1001 | 0 | 0000 | 12 | 1100 | 9 | 1001 |
| **4** | 9 | 1001 | 12 | 1100 | 15 | 1111 | 10 | 1010 |
| **5** | 10 | 1010 | 1 | 0001 | 7 | 0111 | 14 | 1110 |
| **6** | 1 | 0001 | 0 | 0000 | 12 | 1100 | 11 | 1011 |
| **7** | 3 | 0011 | 12 | 1100 | 4 | 0100 | 11 | 1011 |
| **8** | 7 | 0111 | 0 | 0000 | 9 | 1001 | 6 | 0110 |
| **9** | 13 | 1101 | 12 | 1100 | 14 | 1110 | 10 | 1010 |
| **10** | 9 | 1001 | 8 | 1000 | 3 | 0011 | 8 | 1000 |
| **11** | 6 | 0110 | 0 | 0000 | 0 | 0000 | 13 | 1101 |

*Tabla 4.57 Registros obtenidos después de la inserción del último símbolo al RS(15, 11).*

Ahora los símbolos almacenados en los registros se vacían de derecha a izquierda y se agregan a la entrada inicial:

Salida: [3,6,9,12,2,4,6,8,13,15,1,**13,0,0,6**]

*Los símbolos en rojo son el código de redundancia.

## 4.2 Creación de algoritmo para codificador RS(7, 3)

El algoritmo que permite la realización del proceso de codificación fue implementado en la plataforma de software: IDE eclipse, esto debido a que es un estándar en el desarrollo de software.

El proceso del diseño en software de un codificador RS (7, 3) se representa en el diagrama de flujo siguiente, el cual puede hacerse general para el diseño en software de cualquier tipo de codificador RS, como lo fue el codificador RS (15,11); debido a que se realizan subrutinas. Se muestra la subrutina de los campos de Galois correspondientes al codificador RS (7, 3) generados por MATLAB.

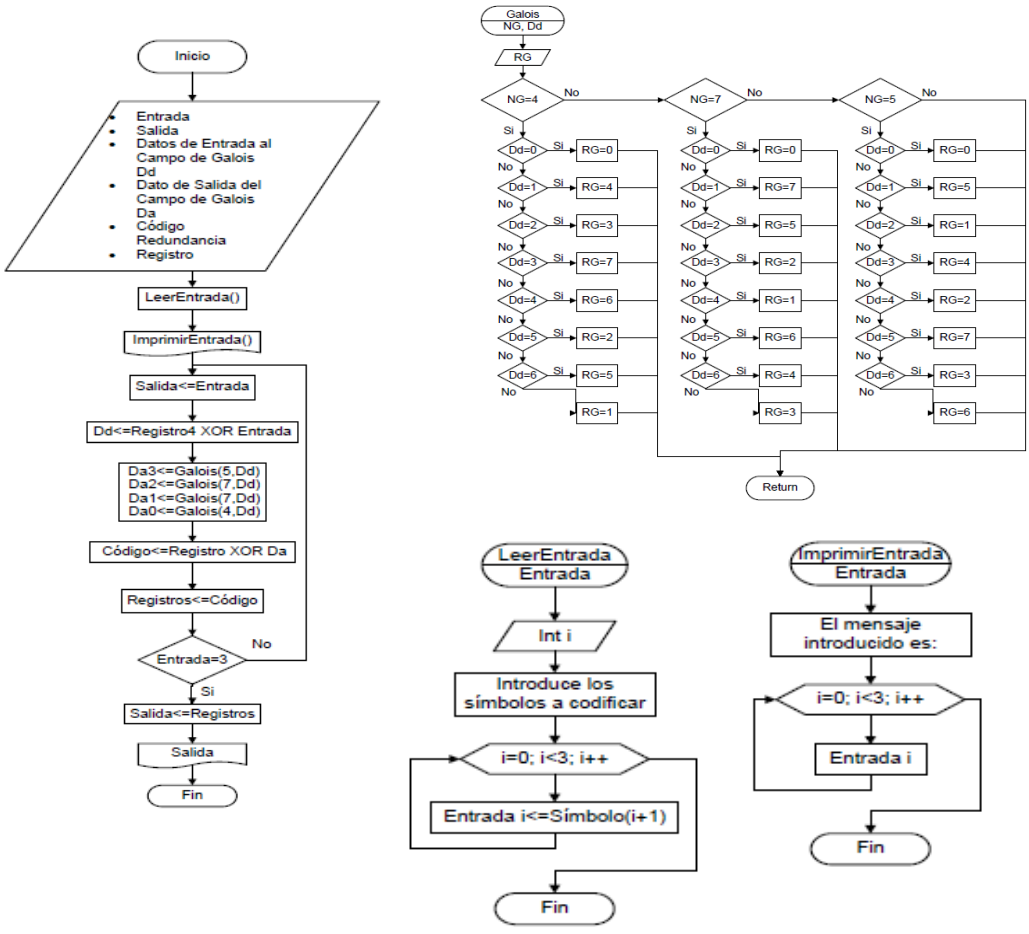


*Figura 4.9 Diagrama de flujo del software.*

## 4.3 Programación en IDE: eclipse

Una vez que se posee el algoritmo de codificación Reed-Solomon, en base a los ejemplos desarrollados, se procede a traducir el algoritmo en lenguaje de programación C++ para generar las pruebas correspondientes a la implementación del código en C++.

En siguientes figuras se muestra la implementación del codificador Reed-Solomon en IDE eclipse.

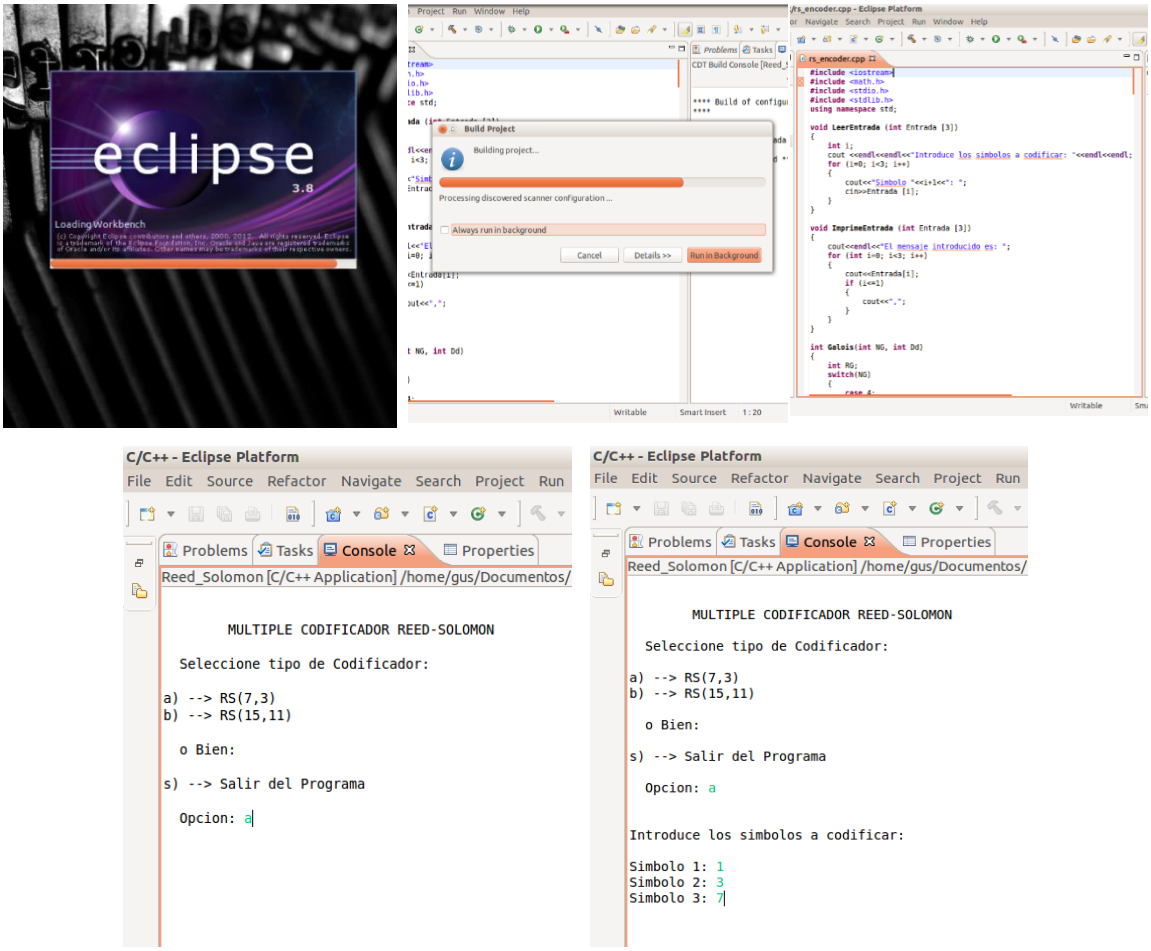


*Figura 4.10 Implementación del codificador Reed-Solomon en IDE eclipse.*

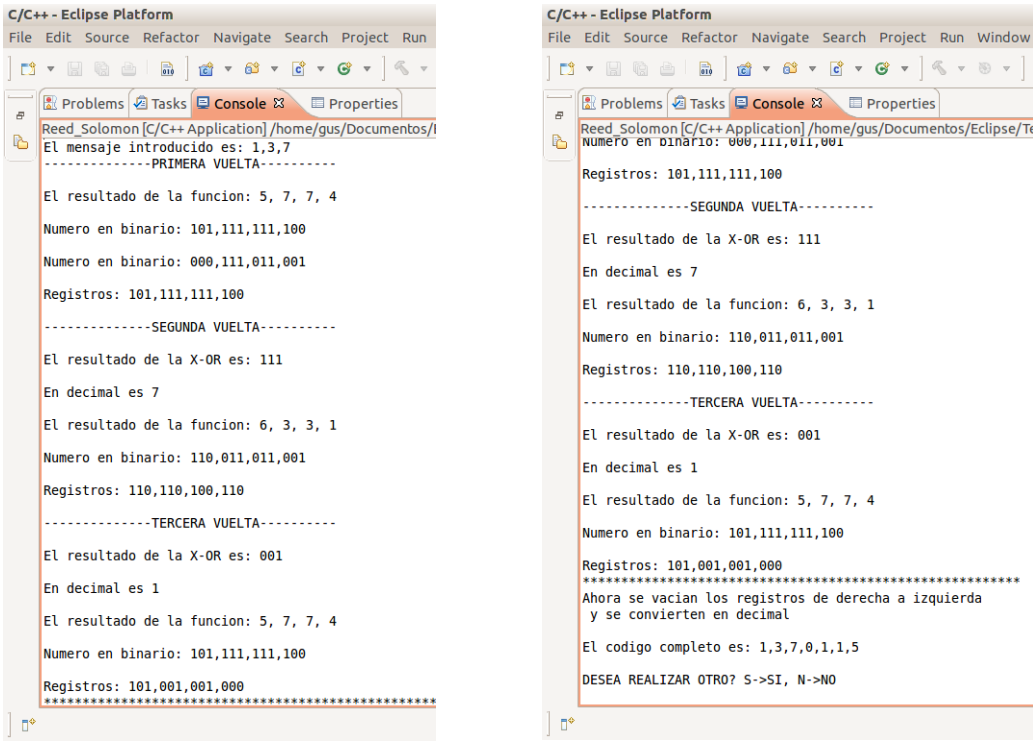


*Figura 4.11 Resultados del mensaje introducido al RS(7, 3).*

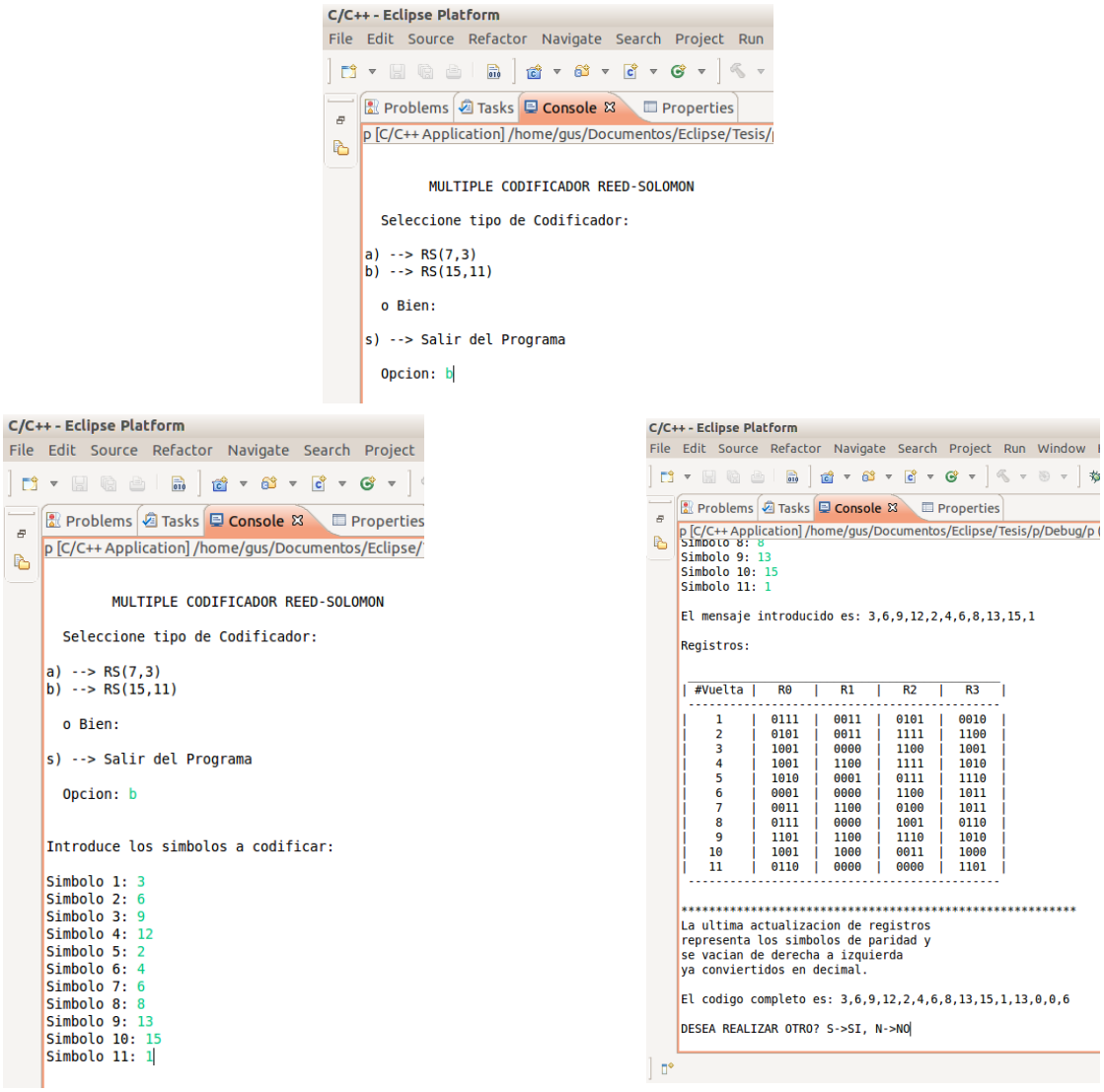


*Figura 4.12 Resultados del mensaje introducido al RS(15, 11).*

## 4.4 Comprobación del código Reed-Solomon en MATLAB

La intención de la presente sección es comprobar mediante el uso de la herramienta matemática: MATLAB, el funcionamiento del módulo de software diseñado en C++.

No se pretende programar en MATLAB el algoritmo diseñado, dado que MATLAB posee su propio algoritmo o módulo de software para el cálculo de códigos Reed-Solomon a través de funciones, solo se utiliza para comprobar que el programa desarrollado en IDE eclipse es correcto. Se procede a ingresar en MATLAB los mismos símbolos que ingresan en IDE eclipse, primero para un codificador RS (7,3), con un mensaje a codificar: msg = [1, 3, 7].

```
>> m=3; %número de bits por símbolo
>> n=7; %número de símbolos de palabra código
```

```
>> k=3; %número de símbolos de mensaje
>> genpoly = rsgenpoly(n, k, 11, 0); %coeficientes polinomio generador
>> msg = gf([1, 3, 7], m); %mensaje en un arreglo de campo de Galois
>> code = rsenc(msg, n, k, genpoly); %palabra código resultante
```

Y se obtiene así la palabra código:

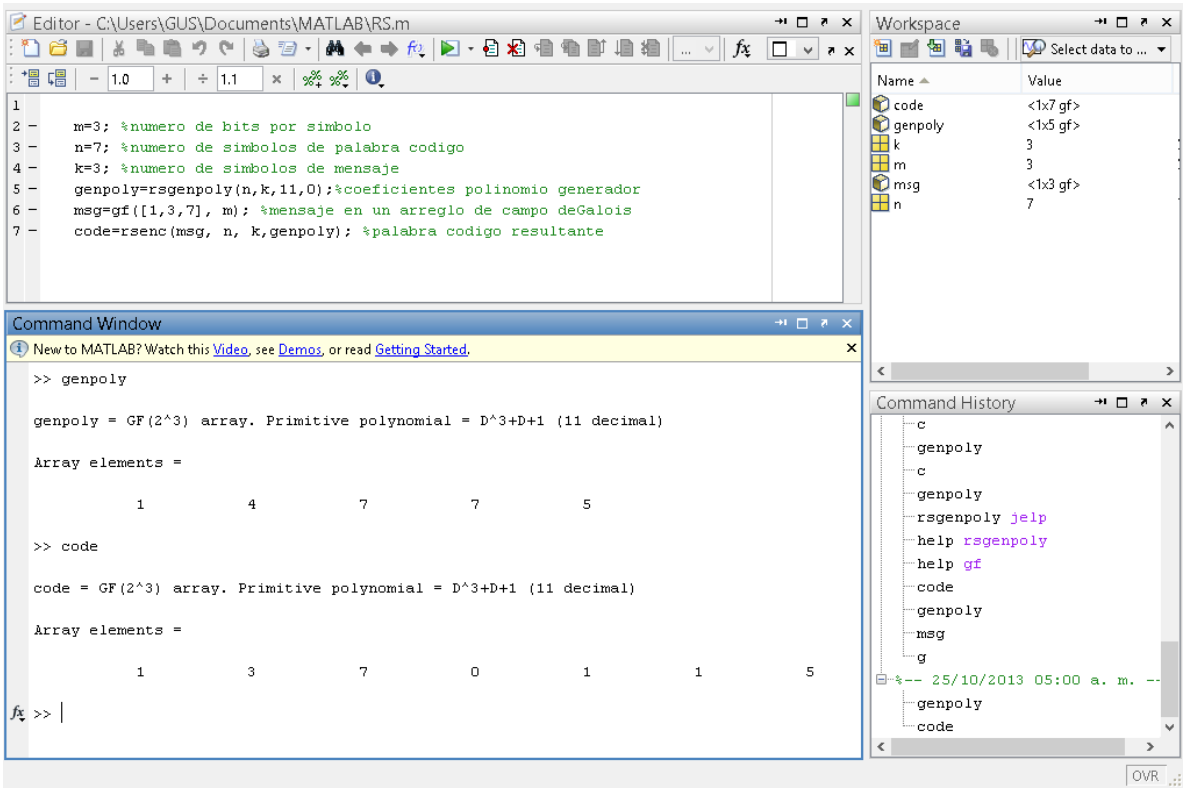


*Figura 4.13 Comprobación en MATLAB de los resultados del mensaje introducido al RS(7, 3).*

Ahora toca realizarlo para el codificador RS (15,11), con un mensaje a codificar: msg = [3,6,9,12,2,4,6,8,13,15,1]. Lo anterior queda de la siguiente manera:

```
>> m=4; %número de bits por símbolo
>> n=15; %número de símbolos de palabra código
>> k=11; %número de símbolos de mensaje
>> genpoly = rsgenpoly(n, k, 19, 0); %coeficientes polinomio generador
>> msg = gf([3,6,9,12,2,4,6,8,13,15,1], m); %mensaje en un arreglo de campo de
Galois
>> code = rsenc(msg, n, k, genpoly); %palabra código resultante
```

Y se obtiene así la palabra código:

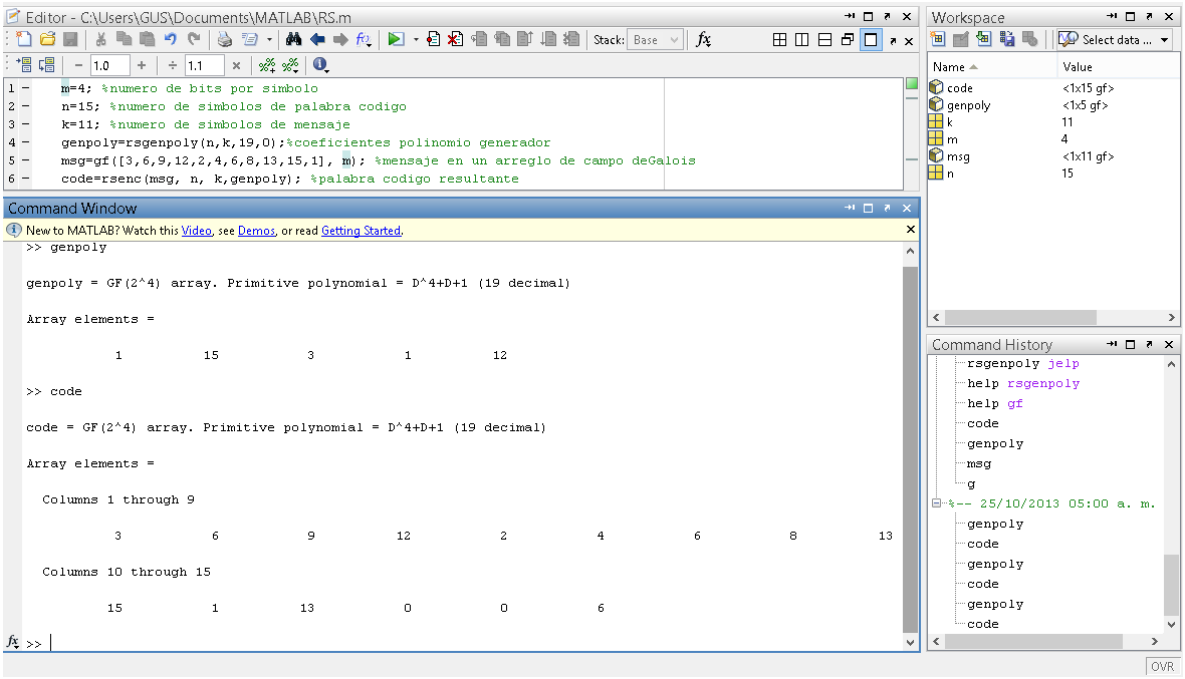


*Figura 4.14 Comprobación en MATLAB de los resultados del mensaje introducido al RS(15, 11).*

Comprobando así que el algoritmo y bloques funcionales del programa en C++ programado en IDE eclipse, funcionan correctamente y pueden ser considerados una base fundamental para el concepto de multicodificador Reed-Solomon aplicado al radio definido por software.

## 4.5 Resultados

Los resultados que se obtuvieron al realizar todas las pruebas correspondientes a cada codificador Reed-Solomon diseñado, fueron que, el codificador Reed-Solomon propio de MATLAB nos permitió obtener los coeficientes del polinomio generador del código Reed-Solomon, así como los campos de Galois pertenecientes a determinado código Reed-Solomon. Al comparar los resultados que arroja MATLAB con los que produce el módulo de software en C++, no se encuentran discrepancias, esto solo se puede interpretar como un correcto funcionamiento del algoritmo diseñado para C++ y del software programado en la plataforma de IDE eclipse.

# CONCLUSIONES

El desarrollo del presente trabajo de investigación tuvo a bien llevar a cabo un estudio y análisis del funcionamiento del código Reed-Solomon para encontrar las generalidades que permitieron crear un algoritmo para definir un codificador Reed-Solomon genérico, adaptable a cualquier especificación que se utilice en los estándares actuales de codificación.

Es importante mencionar que se logró crear este algoritmo debido a las características que presenta el código Reed-Solomon como poseer una estructura matemática bien definida y el establecimiento de un polinomio generador para obtener diferentes códigos para diferentes longitudes de información con el simple hecho de cambiar algunos parámetros del polinomio lo que da como consecuencia que se tenga una estructura general.

El algoritmo que se creó es capaz de llevar a cabo el proceso de codificación Reed-Solomon automáticamente sin la necesidad de modificar en absoluto ningún parámetro que se encuentre definido en él, sin embargo es necesario anexar en una librería externa los coeficientes de los multiplicadores de Galois correspondientes al codificador que se desee utilizar y se pueden obtener fácilmente mediante la herramienta matemática MATLAB usando el código descrito en el capítulo 4.

Con lo anterior se puede considerar que se ha creado un módulo de software que se pude adaptar fácilmente a cualquier dispositivo que trabaje con lógica reprogramable, permitiendo contribuir al concepto de Radio Definido por Software y su crecimiento como tecnología que impulsa la convergencia con nuevas formas para la solución de los problemas actuales en las comunicaciones móviles e inalámbricas.

El presente trabajo muestra con lo anterior un avance significativo en el establecimiento de nuevas generaciones de comunicaciones móviles basadas en tecnologías de Radio Definido por Software, abarcando una etapa muy importante que beneficia a que se tenga un control más preciso de la información, a una mayor eficiencia en la transmisión de datos manteniendo y mejorando la calidad que se desea seguir brindando en las comunicaciones móviles.

# RECOMENDACIONES PARA TRABAJOS FUTUROS

En esta área se recomienda los siguientes puntos:

- Implementar el módulo de software que se desarrolló en IDE “eclipse”, en una tarjeta de desarrollo XMOS, para así poder comparar el rendimiento y eficiencia de ésta tarjeta con una FPGA.

- Realizar el diseño del decodificador Reed-Solomon para ser programado en una tarjeta de desarrollo XMOS por medio del concepto de Radio Definido por Software y consumar la programación del código con la funcionalidad de ambos bloques, tanto codificador como decodificador.

- Considerar la implementación en una tarjeta de desarrollo XMOS los elementos restantes que componen el transmisor para culminar el esquema, estos son: la codificación de la fuente, el entrelazado, el codificador convolucional, el filtrado y las diferentes modulaciones.

# ANEXOS

# ANEXO A: PROGRAMACIÓN EN C++

```
#include <iostream>  //Librerias necesarias.
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
using namespace std;

int n=0; //Número de simbolos de palabra código.
int k=0; //Número de simbolos de mensaje a codificar.
int m=0; //Número de bits por cada símbolo.

void LeeEntrada (int *ent) //Función que lee los k símbolos de entrada.
{
	cout<<endl<<"Introduce los simbolos a codificar: "<<endl<<endl;
	for (int i=0; i<k; i++)
	{
		do{
		cout<<"Simbolo "<<i+1<<": ";
		cin>>ent[i];
		}while(ent[i]<0 || ent[i]>(pow(2,m)-1));
	}
}

void ImprimeEntrada (int *ent) //Función que imprime los k símbolos de entrada.
{
	for (int i=0; i<k; i++)
	{
		cout<<ent[i];
		if (i<k-1)
		{
			cout<<",";
		}
	}
}

void DecBin (int *ent, bool **Bin) //Función que convierte un vector de
{                                                    //decimales en una matriz
de estos numeros
	int i,j,bent;		//expresados en binario.
	for(i=0;i<k;i++)
	{
		bent=ent[i];
		for(j=0;j<m;j++)
		{
```

```
                        Bin[i][j]=bent%2;
                        bent=bent/2;
                }
        }
}

void RegIni(bool **reg)       //Función que inicializa la matriz de registros
{                             //expresados en binario.
        for(int i=0;i<(n-k);i++)
        {
                for(int j=0; j<m; j++)
                {
                        reg[i][j]=0;
                }
        }
}

int BinDec (bool *DatoBin)  //Función que convierte un número en binario
{                                                    //a un número decimal.
        int Dd=0,a,temp;
        for(int i=0;i<m;i++)
        {
                a=DatoBin[i];
                temp = a*pow(2,i);
                Dd += temp;
        }
        return Dd;
}

int MultGalois_7_3(int NG, int Dd)  //Función que realiza la multiplicación
{                                                          //de Galois para un
codificador RS(7,3)
        int RG;
                switch(NG)
                {
                        case 4:
                        switch (Dd)
                        {
                                case 0:
                                        RG=0;break;
                                case 1:
                                        RG=4;break;
                                case 2:
                                        RG=3;break;
                                case 3:
                                        RG=7;break;
                                case 4:
```

```
                RG=6;break;
        case 5:
                RG=2;break;
        case 6:
                RG=5;break;
        case 7:
                RG=1;break;
}
break;
case 5:
switch (Dd)
{
        case 0:
                RG=0;break;
        case 1:
                RG=5;break;
        case 2:
                RG=1;break;
        case 3:
                RG=4;break;
        case 4:
                RG=2;break;
        case 5:
                RG=7;break;
        case 6:
                RG=3;break;
        case 7:
                RG=6;break;
}
break;
case 7:
switch (Dd)
{
        case 0:
                RG=0;break;
        case 1:
                RG=7;break;
        case 2:
                RG=5;break;
        case 3:
                RG=2;break;
        case 4:
                RG=1;break;
        case 5:
                RG=6;break;
        case 6:
                RG=4;break;
```

```
                        case 7:
                                RG=3;break;
                }
                break;
        }
        return RG;
}

int MultGalois_15_11(int NG, int Dd) //Función que realiza la multiplicación
{                                                       //de Galois para un
codificador de RS(15,11)
        int RG;
        switch(NG)
        {
                case 12:
                switch (Dd)
                {
                        case 0:
                                RG=0;break;
                        case 1:
                                RG=12;break;
                        case 2:
                                RG=11;break;
                        case 3:
                                RG=7;break;
                        case 4:
                                RG=5;break;
                        case 5:
                                RG=9;break;
                        case 6:
                                RG=14;break;
                        case 7:
                                RG=2;break;
                        case 8:
                                RG=10;break;
                        case 9:
                                RG=6;break;
                        case 10:
                                RG=1;break;
                        case 11:
                                RG=13;break;
                        case 12:
                                RG=15;break;
                        case 13:
                                RG=3;break;
                        case 14:
                                RG=4;break;
```

```
        case 15:
                RG=8;break;
}
break;
case 1:
        RG=Dd;break;
case 3:
switch (Dd)
{
        case 0:
                RG=0;break;
        case 1:
                RG=3;break;
        case 2:
                RG=6;break;
        case 3:
                RG=5;break;
        case 4:
                RG=12;break;
        case 5:
                RG=15;break;
        case 6:
                RG=10;break;
        case 7:
                RG=9;break;
        case 8:
                RG=11;break;
        case 9:
                RG=8;break;
        case 10:
                RG=13;break;
        case 11:
                RG=14;break;
        case 12:
                RG=7;break;
        case 13:
                RG=4;break;
        case 14:
                RG=1;break;
        case 15:
                RG=2;break;
}
break;
case 15:
switch (Dd)
{
        case 0:
```

```
                                RG=0;break;
                        case 1:
                                RG=15;break;
                        case 2:
                                RG=13;break;
                        case 3:
                                RG=2;break;
                        case 4:
                                RG=9;break;
                        case 5:
                                RG=6;break;
                        case 6:
                                RG=4;break;
                        case 7:
                                RG=11;break;
                        case 8:
                                RG=1;break;
                        case 9:
                                RG=14;break;
                        case 10:
                                RG=12;break;
                        case 11:
                                RG=3;break;
                        case 12:
                                RG=8;break;
                        case 13:
                                RG=7;break;
                        case 14:
                                RG=5;break;
                        case 15:
                                RG=10;break;
                }
                break;
        }
        return RG;
}

void Galois(int Dd, bool **gal) //Función que seleccionar Campo de Galois.
{
        int RG;
        if(n==7&&k==3)
        {
                int galdec[4]={5,7,7,4};
                for(int i=0;i<(n-k);i++)
                {
                        RG=MultGalois_7_3(galdec[i],Dd);
                        for(int j=0;j<m;j++)
```

```
                {
                        gal[i][j]=RG%2;
                        RG=RG/2;
                }
        }
    }
    else if(n==15&&k==11)
    {
        int galdec[4]={12,1,3,15};
        for(int i=0;i<(n-k);i++)
        {
            RG=MultGalois_15_11(galdec[i],Dd);
            for(int j=0;j<m;j++)
            {
                gal[i][j]=RG%2;
                RG=RG/2;
            }
        }
    }
    /*else if(n==15&&k==9)
        {
            int galdec[6]={1,2,3,4,5,6};
            for(int i=0;i<(n-k);i++)
            {
                RG=MultGalois_15_9(galdec[i],Dd);
                for(int j=0;j<m;j++)
                {
                    gal[i][j]=RG%2;
                    RG=RG/2;
                }
            }
        }*/
}

void RespReg(bool **reg, bool **resp) //Función que realiza un respaldo de los
{                                     //regisros actules.
    for(int i=0;i<n-k;i++)
    {
        for(int j=0; j<m; j++)
        {
            resp[i][j]=reg[i][j];
        }
    }
}

void Paridad(bool **Bin, int *CodD) //Función que agrega los símbolos de paridad
{                                   //al mensaje fuente.
```

```
	for(int j=0;j<n-k;j++)
	{
		CodD[j]=BinDec(Bin[j]);
	}

}

void ImprimeParidad(int *ent) //Función que imprime simbolos de paridad.
{
	cout<<",";
	for (int i=(n-k-1);i>=0; i--)
	{
		cout<<ent[i];
		if (i>0)
		{
			cout<<",";
		}
	}
}

void encoder() //Función que realiza la codificacion Reed-Solomon.
{
	char resp;
	int *Entrada, *Cod, Dd;
	bool *EntXor, **Entbin, **Reg, **RegB, **MGBin;
	Entrada=new int[k];
	Cod=new int[n-k];
	EntXor=new bool[m];
	Entbin=new bool*[k];
	Reg=new bool*[n-k];
	RegB=new bool*[n-k];
	MGBin=new bool*[n-k];
	for(int i=0;i<k;i++)
	{
		Entbin[i]=new bool[m];
	}
	for(int i=0;i<(n-k);i++)
	{
		Reg[i]=new bool[m];
		RegB[i]=new bool[m];
		MGBin[i]=new bool[m];
	}
	do
	{
		LeeEntrada(Entrada);
		cout<<endl<<"El mensaje introducido es: ";
		ImprimeEntrada(Entrada);
```

```
            DecBin(Entrada,Entbin);
            RegIni(Reg);
            for(int i=0;i<k;i++)
            {
                    for(int j=0;j<m;j++)
                    {
                            EntXor[j]=Entbin[i][j] xor Reg[n-k-1][j];
                    }
                    Dd=BinDec(EntXor);
                    Galois(Dd,MGBin);
                    RespReg(Reg,RegB);
                    for(int j=0;j<m;j++)
                    {
                            Reg[0][j]=MGBin[0][j];
                    }
                    for(int j=1;j<n-k;j++)
                    {
                            for(int y=0;y<m;y++)
                            {
                                    Reg[j][y]=RegB[j-1][y] xor MGBin[j][y];
                            }
                    }
            }
            Paridad(Reg,Cod);
            cout<<endl<<endl<<"El codigo completo es: ";
            ImprimeEntrada(Entrada);
            ImprimeParidad(Cod);
            cout<<"\n\nDESEA REALIZAR OTRO? (S/N): ";
            cin>>resp;
    }while(resp=='s'||resp=='S');
    delete []Entrada;
    delete []Cod;
    delete []EntXor;
    for(int i=0;i<k;i++)
    {
            delete []Entbin[i];
    }
    delete []Entbin;
    for(int i=0;i<(n-k);i++)
    {
            delete []Reg[i];
            delete []RegB[i];
            delete []MGBin[i];
    }
    delete []Reg;
    delete []RegB;
    delete []MGBin;
```

```
}

char Menu(void) //Función que muestra un menú de seleccion de codificador.
{
	char op, pause;
	//system("clear");
	cout <<endl<<endl<< "\tMULTIPLE CODIFICADOR REED-SOLOMON" << endl << endl;
	cout << "  Seleccione tipo de Codificador:" << endl << endl;
	cout << "a) --> RS(7,3)" << endl;
	cout << "b) --> RS(15,11)" << endl<< endl;
	cout << "  o Bien:" << endl << endl;
	cout << "s) --> Salir del Programa" << endl <<endl;
	cout << "  Opcion: ";
	cin>>op;
	switch(op)
	{
		case 'a':
				n=7,k=3,m=3;
				encoder();
				break;
		case 'b':
				n=15,k=11,m=4;
				encoder();
				break;
		case 's':
				break;
		default:
				cout<<"\n\n\tOpcion Invalida, ingrese opcion valida"<< endl;
				cout<<"\n\nIngrese cualquier caracter para continuar . . .
";
				cin >> pause;
				break;
	}
	return op;
}

int main(void)
{
	char op;
	do
	{
		op=Menu();
	}while(op!='s');
	cout<<"\nFuncion Principal: Programa Terminado."<< endl;
return 0;}
```

# ANEXO B: ARTÍCULOS PRESENTADOS EN CONGRESOS

CP 36
PON 70

## PROPUESTA DE MULTICODIFICADOR REED-SOLOMON PARA UN PROCESADOR EN PARALELO XCMOS

**David Vázquez Álvarez, Gabriela Sánchez Meléndez, Gustavo Durán Herrera, Julio R. Franco Giles**
Departamento de Ingeniería en Comunicaciones y Electrónica
ESIME Zacatenco I.P.N,, D.F., México. Teléfono: (55) 5729-6000 Ext:54797
E-mail: dvazqueza@ipn.mx, gsanchezgsm@yahoo.com, jlaraariza@gmail.com, julio7r7@hotmail.com.

*Resumen:* En este artículo presentamos las bases teóricas necesarias para lograr comprender el funcionamiento del codificador Reed-Solomon y una propuesta de implementación orientada a la tecnología de procesamiento paralelo (XCMOS) que permita su configuración a través de lenguaje de programación C++.

*Abstract:* This article contains the necessary theoretical basis to understand the encoders Reed-Solomon function and a implementation proposal oriented to parallel processing (XCMOS) that allows its configuration through the programming language C++.

### I. INTRODUCCIÓN

El codificador Reed-Solomon encuentra su aplicación en la tecnología de las telecomunicaciones gracias a su flexibilidad y su eficiente algoritmo, tanto de codificación como de decodificación, lo que le ha valido ser parte esencial de diferentes estándares para transmisión de datos.

Es así que en una de estas áreas de aplicación encontramos al DVB (de sus siglas en inglés Digital Video Broadcast). El DVB es una organización que promueve estándares aceptados internacionalmente para la transmisión de televisión digital en especial HDTV (de sus siglas en inglés High Definition TV) y televisión vía satelital, así como para las comunicaciones de datos vía satélite.

ROC&C'2013 / CP 36 PONENCIA RECOMENDADA POR EL COMITE DE COMPUTACION DEL IEEE SECCION MEXICO Y PRESENTADA EN LA REUNION INTERNACIONAL DE OTOÑO, ROC&C'2013. ACAPULCO, GRO., DEL 10 AL 14 DE NOVIEMBRE DEL 2013.

Esta organización es mucho más que alta definición; ya que ofrece soluciones modificables y aceptables para cada país y para todas las clases sociales.

Toda la codificación de audio y vídeo está basada en los estándares ya establecidos por MPEG.

Existen múltiples estándares creados por la DVB, la razón de esta variedad se debe principalmente a los diferentes medios de transmisión de la señal, ya sea por antenas de televisión convencional, satélite, cable, o para dispositivos más pequeños (por ejemplo los móviles), algunos de sus estándares son:

- Televisión terrestre: **DVB-T** y para dispositivos portátiles: **DVB-H**.
- Cable: **DVB-C** y **DVB-C2**.
- Satélite: **DVB-S** y **DVB-S2** y para dispositivos portátiles: **DVB-SH**. [2]

Tanto el DVB estándar como los DVB de diferente transmisión, se compone de cinco elementos:

- Codificación fuente.
- Codificador Reed-Solomon.
- Entrelazado.
- Codificación convolucional.
- Filtrado.
- Modulación.

El codificador Reed-Solomon empleado en los estándares DVB es el RS (204,188). Para tener bases comprobables de que el múltiple codificador funcionará con, por lo menos, tres tipos de codificadores Reed-Solomon, se ha optado por empezar con el codificador RS (7,3) y así ir incrementando el tamaño del codificador paulatinamente, pero con la seguridad de que niveles inferiores funcionan correctamente.

## II. CÓDIGO REED-SOLOMON

El código debe su nombre a la unión de los apellidos de sus inventores, Irving S. Reed y Gustave Solomon en el año de 1960. Es este código es actualmente aplicado a la detección y corrección de errores en dispositivos de almacenamiento, CD, comunicaciones móviles, comunicaciones satelitales y por cable, modem de alta velocidad y en transmisiones de televisión digital.

El código Reed-Solomon forma parte de una clase de códigos de bloque estándar de detección y corrección de errores en los datos transmitidos sobre un canal de comunicaciones. Pertenecen a la categoría FEC (del inglés Forward Error Correction) que significa que corrige los errores de datos alterados en el receptor, haciendo uso de una serie de bits adicionales que le permite recuperar el mensaje o dato original.

Un código Reed-Solomon se especifica como RS(n, k) con símbolos de s bits. Lo anterior significa que el codificador toma k símbolos de s bit cada uno y añade símbolos de paridad para hacer una palabra de código de n símbolos.

Existen n-k símbolos de paridad de s bits cada uno. Un decodificador puede corregir hasta t símbolos que contienen errores en una palabra de código, donde (2t=n-k).

El siguiente diagrama muestra una típica palabra de código Reed-Solomon:

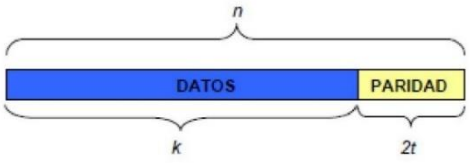


Figura 1. Diagrama de palabra código Reed-Solomon.

Un error de símbolo ocurre cuando un bit en un símbolo es erróneo o cuando todos los bits en un símbolo se encuentran erróneos.

Los códigos Reed-Solomon son particularmente útiles para corregir el llamado "burst error" (cuando una serie de bits en el código de palabra se reciben con error).

Para poder obtener tal estructura de codificación es necesario procesar la trama de datos a través de un circuito digital que opere bajo los fundamentos de campo finito o campo de Galois. Dicho circuito se representa por los bloques funcionales mostrados a continuación:

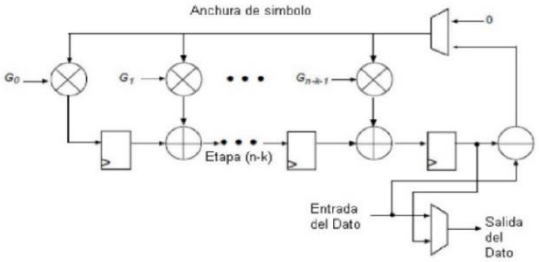


Figura 2. Circuito digital para código Reed-Solomon.

## III. RADIO DEFINIDO POR SOFTWARE

El termino radio se utiliza mediante la combinación de transmisor (Tx) y receptor (Rx), lo que generalmente se conoce como transceptor. Una de las principales características del radio, se encuentra en el intercambio de información, y no solo puede transportar audio, sino eventualmente video y transmisiones multimedia.

Un SDR, tiene casi todos sus "Componentes" definidos y funcionando en forma de programas en un ordenador, a excepción de un mínimo de componentes físicos.

Y mientras no sea activado ese software o conjunto de programas, el equipo de radio no será utilizado como tal, sino que será un simple conjunto de placas electrónicas externas, incapaces de cumplir con su funcionalidad.

El SDR es capaz de ser reprogramado o reconfigurado para funcionar con diferentes protocolos y formas de onda a través de la carga dinámica de nuevos programas de ordenador y protocolos.

La implementación de SDR se puede ver en tres etapas:
1.- Hardware que toma a señal desde su fuente y la acondiciona (rangos de frecuencia y amplitud) para que el digitalizador la pueda manejar.
2.- Digitalizador con capacidad en velocidad de muestreo que define el ancho de banda máximo de la señal a procesar.
3.- Software capaz de transformar la señal digitalizada en función de sus necesidades (estándares) particulares para obtener una nueva señal final deseada.

Las etapas se logran utilizando convertidores A/D y D/A ya que representa una ventaja el utilizar el procesamiento digital de señales, para operar tantas cadenas de señales como sea posible y reconfigurarlas cerca de la antena.

## IV. IMPLEMENTACIÓN

Como ya se mencionó, para tener bases comprobables de que el múltiple codificador funcionará, se ha optado por empezar con el codificador RS (7,3) e ir incrementando el tamaño del codificador paulatinamente.

Los principios que definen al codificador Reed-Solomon permiten generalizar de manera simple un algoritmo que nos represente la implementación de cualquier tipo de codificador Reed-Solomon, sin embargo no así con la parte matemática que los define.

Diseñar un algoritmo que genere los campos de Galois para todo tipo de codificadores Reed-Solomon es una tarea laboriosa que requiere de bastante colaboración, situación que hace complicado su desarrollo.

En vista de las limitantes mencionadas anteriormente, se decide, realizar tres tipos de codificadores Reed-Solomon, suficientes para describir la intención del proyecto, que es proponer un múltiple codificador Reed-Solomon en Software.

Estos tres tipos de codificadores son: RS(7,3), RS(15,9) y RS(31,23). Por el momento el único que tenemos implementado en software es el primero, el cual está desarrollado en lenguaje de programación C++ en un IDE de Eclipse, este es el molde para continuar con los siguientes dos y así tener ya nuestro múltiple-codificador Reed-Solomon, el diagrama de flujo de nuestro programa es el que a continuación les mostramos:

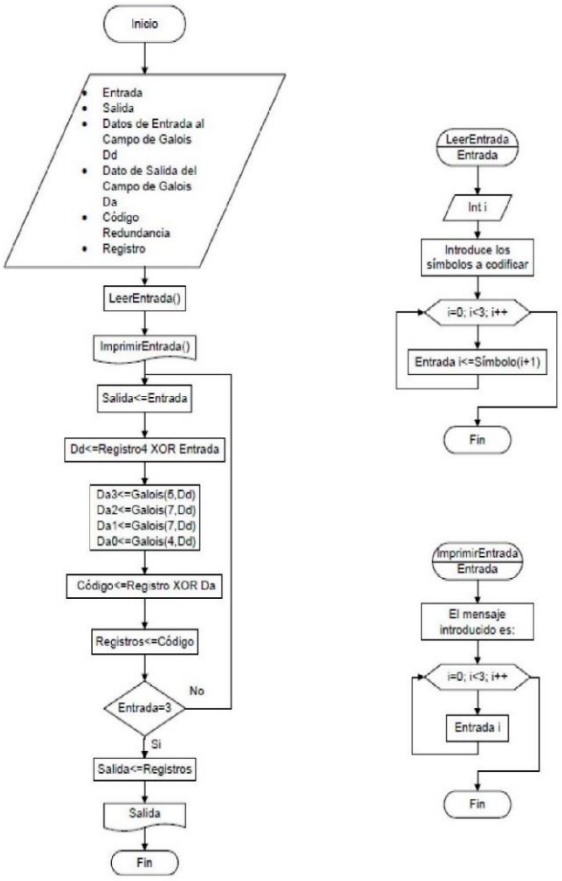


Figura 3. Diagrama de flujo de código Reed-Solomon.

Para comprobar el funcionamiento de los módulos y de nuestros resultados en nuestro programa hicimos varias pruebas de escritorio, así como también desarrollamos paso a paso cada uno de ellos, estando así seguros de que nuestro primer codificador Reed-Solomon en C++ está

correctamente programado y con esto podemos iniciar la programación de los siguientes dos.
En la próxima imagen mostramos las etapas para la ejecución del codificador Reed-Solomon (7,3), desde que iniciamos el IDE de Eclipse, corremos el programa y seleccionamos el codificador Reed-Solomon deseado.

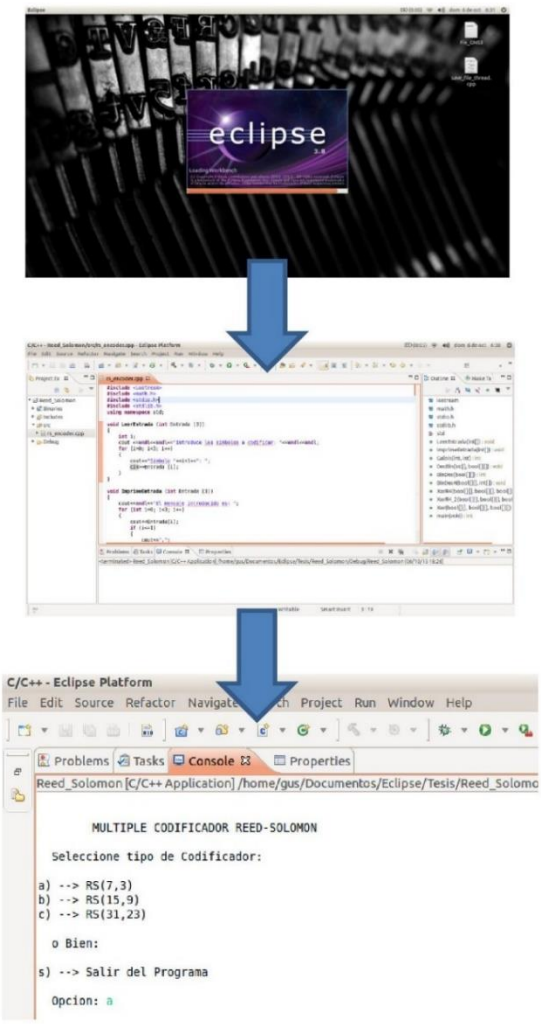


Figura 4.1 Implementación en IDE eclipse.

Ahora ya seleccionado el Codificado Reed-Solomon (7,3) ingresamos nuestros símbolos a codificar que serán en esta ocasión los siguientes [1, 3,7]:

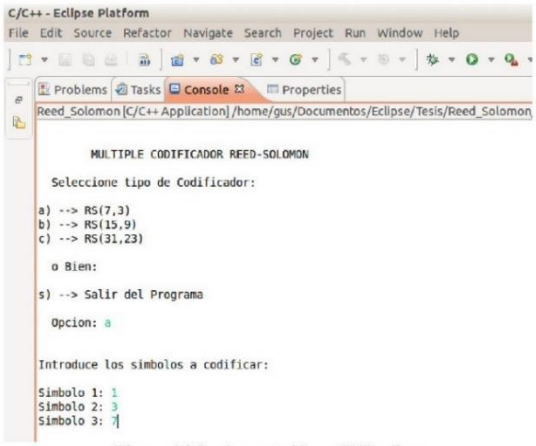


Figura 4.2 Implementación en IDE eclipse.

Iniciamos posteriormente nuestro programa obteniendo así los resultados equivalentes a cada vuelta realizada dentro del codificador lo cual mostramos en la siguiente imagen:

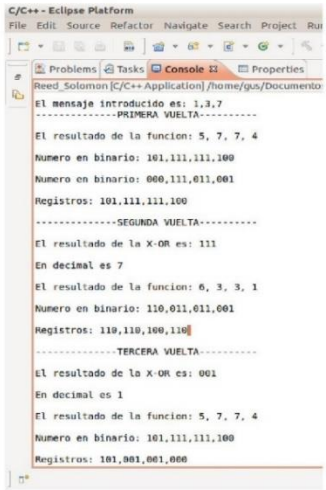


Figura 4.3 Implementación en IDE eclipse.

Finalmente obtenemos nuestro mensaje ya codificado, esto quiere decir nuestros símbolos de

entrada más sus símbolos de paridad, generados por nuestro codificador, esto se muestra en la siguiente figura:

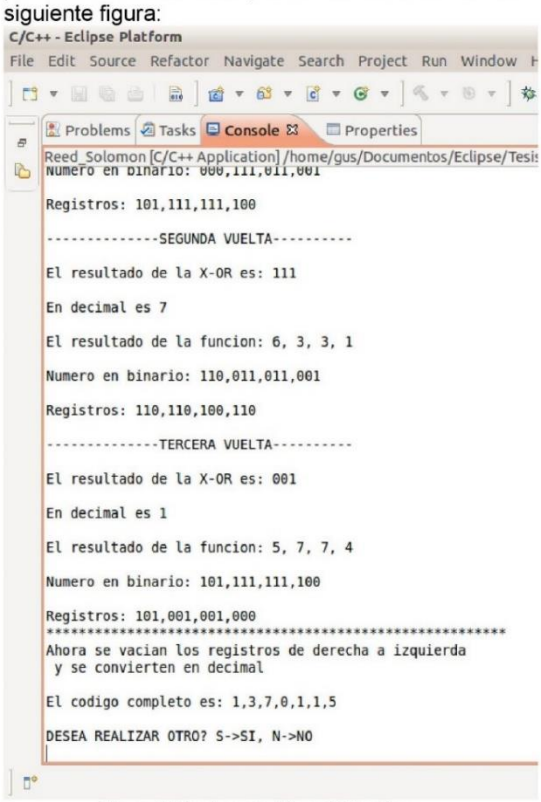


Figura 4.4 Implementación en IDE eclipse.

## V. CONCLUSIONES

Las tendencias en tecnología para las telecomunicaciones cambian actualmente tan de prisa que hace necesario actualizar los estándares para transmisión y recepción de datos multimedia constantemente, es entonces que el desarrollar módulos en software de estos estándares que permitan reconfigurar a través de Software permitirá desarrollar soluciones compactas y fiables para el mercado de las comunicaciones móviles.

El codificador Reed-Solomon es parte fundamental de este tipo de estándares que definen el bloque de codificación de canal para el concepto de radio definido por software, y el desarrollo de un múltiple codificador Reed-Solomon permite ampliar el panorama del objetivo del radio definido por software que es tender hacia la convergencia.

Con este trabajo se obtiene un módulo de software que representa a un múltiple codificador Reed-Solomon capaz de ser incorporado en un procesador paralelo que se puede reconfigurar través de software, ajustándose a cualquier estándar que haga uso de DVB.

En trabajos futuros queda pendiente la implementación de una subrutina que genere cualquier campo de Galois para cualquier estructura de codificador Reed-Solomon y así completar el concepto de convergencia.

## VI. REFERENCIAS

[1] David Vázquez Álvarez. Tesis de titulación "Técnicas de codificación para control de error en radio definido por software" I.P.N. SEPI Zacatenco, México 2009

[2] DVB. "La organización y los estándares Digital Video Broadcast" [En línea], fecha de consulta 04-10-13. Disponible: http://ficheros.molamiweb.com/webs/coyan/trabajo-DVB-texto.pdf

[3] Ángela Indira Rodríguez Ruiz, Juan Camilo Rodríguez Ibagué. "Modulación en televisión digital, transmisión de datos" [En línea], fecha de consulta 06-10-13. Disponible: http://www.slideshare.net/draconiano91/modulacion-en-television-digital

[4] Cecilia Urbina. "Reed-Solomon Codes" [En línea], fecha de consulta 06-10-13. Disponible: http://cecilia-urbina.blogspot.mx/2013/05/reed-solomon-codes.html

[5] Cecilia E. Sandoval Ruiz, Antonio Fedon. "Codificador y decodificador digital Reed-Solomon programados para hardware reconfigurable" [En línea], fecha de consulta 06-10-13. Disponible: http://www.redalyc.org/articulo.oa?id=47711102

[6] E & Q Engineering, solutions and innovation. "Radio Definida por Software" [En línea], fecha de consulta 06-10-13. Disponible: http://www.eqeng.com/site/en/node/38

[7] EcuRed. "Radio Definido por Software (SDR)". [En línea], fecha de consulta 06-10-13. Disponible: http://www.ecured.cu/index.php/Radios_definidos_por_Software

[8] XC-1A "XC-1A Hardware Manual" [En línea], fecha de consulta 06-10-13. Disponible: http://www1.futureelectronics.com/doc/XMOS%20LIMITED/X-CARDXC-1A.pdf

## VII. BIOGRAFÍAS

**M. en C. David Vázquez Álvarez**. Maestro en Ciencias en Ingeniería de Telecomunicaciones de la Sección de Estudios de Posgrado e Investigación (SEPI) del Instituto Politécnico Nacional (IPN). Actualmente es Subdirector Administrativo de la Escuela Superior de Ingeniería Mecánica y Eléctrica (ESIME) unidad Profesional Adolfo López Mateos (Zacatenco). Áreas de Interés Actual: Fibras Ópticas, Radio Definido por Software, Radio Cognitivo, Sistemas de Comunicaciones Móviles e Inalámbricas.

**M. en C. Gabriela Sánchez Meléndez**. Maestra en Ciencias en Ingeniería de Telecomunicaciones de la Sección de Estudios de Posgrado e Investigación (SEPI) del Instituto Politécnico Nacional (IPN). Actualmente es Profesora titular de la Escuela Superior de Ingeniería Mecánica y Eléctrica unidad Profesional Adolfo López Mateos (ESIME). Áreas de Interés: Fibras Ópticas, Telecomunicaciones, Aplicaciones de Software.

**Gustavo Durán Herrera**. Actualmente es alumno de 9° semestre en la especialidad de computación de la carrera de Ingeniería en Comunicaciones y Electrónica de la Escuela Superior de Ingeniería Mecánica y Eléctrica unidad Profesional Adolfo López Mateos impartida en el Instituto Politécnico Nacional (IPN). Áreas de Interés Actual: Radio Definido por Software, Radio Cognitivo, Sistemas de Comunicaciones Móviles e Inalámbricas.

**Julio Ricardo Franco Giles**. Actualmente es alumno de 9° semestre en la especialidad de computación de la carrera de Ingeniería en Comunicaciones y Electrónica de la Escuela Superior de Ingeniería Mecánica y Eléctrica unidad Profesional Adolfo López Mateos impartida en el Instituto Politécnico Nacional (IPN). Áreas de Interés Actual: Radio Definido por Software, Radio Cognitivo, Sistemas de Comunicaciones Móviles e Inalámbricas.

# REFERENCIAS

[1] http://centrodeartigos.com/articulos-noticias-consejos/article_145489.html. Fecha de revisión febrero 2013.

[2] http://ccc.inaoep.mx/Reportes/CCC-05-004.pdf. Fecha de revisión febrero 2013.

[3] http://www.tools4sdr.com/articles/Description+of+cognitive+radio+concept_2504610. Fecha de revisión: Marzo 2013.

[4] http://focus.ti.com/docs/solution/folders/print/357.html. Fecha de revisión: Marzo 2013.

[5] http://todoit.com.ve/xcalibur/8-tv-digital-diseno-hfc. Fecha de revisión: Marzo 2013.

[6] http://www.aiu.edu/publications/student/spanish/Comunnicacion%20de%20Systemas.html. Fecha de revisión: Marzo 2013.

[7] http://centrodeartigos.com/articulos-noticias-consejos/article_131048.html. Fecha de consulta: Marzo 2013.

[8] http://www.televisiondigital.es/tecnologias/AD/ForoTVAD/Conclusiones/3SG2_ASPECTOSTECNICOSRELACIONADOS.pdf Fecha de consulta: Abril 2013.

[9] http://www.google.com.mx/url?sa=t&rct=j&q=&esrc=s&frm=1&source=web&cd=13&ved=0CDwQFjACOAo&url=http%3A%2F%2Fingenierias.uanl.mx%2F25%2F25_codigos.pdf&ei=K6KLUsWZJsS52wXWyIGgAg&usg=AFQjCNFEvqC2Ii_Ul-mirukoil7tYFjAwA&sig2=cKEKsSRuS6dhnQIX8QdrBw. Fecha de consulta: Abril 2013.

[10] http://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r87939.PDF. Fecha de consulta: Abril 2013.

[11] http://chovis2.tripod.com/CRC.HTM Fecha de consulta: Mayo 2013.

[12] http://serdis.dis.ulpgc.es/~ii-ti/TI-WEB/curso%200506/TEMA7.html. Fecha de consulta: Mayo 2013.

[13] http://cs.uns.edu.ar/~ags/OC/downloads/APUNTES%20DE%20TEOR%C3%8DA/Notes/M%C3%B3ulo%2004%20%20Detecci%C3%B3n%20y%20Correcci%C3%B3n%20de%20Errores%20(Pt.%202). otes.pdf Fecha de consulta: Mayo 2013.

[14] http://arantxa.ii.uam.es/~ig/teoria/temas/IG_tema-4-2008-2009.pdf. Fecha de consulta: Mayo 2013.

[15] http://neo.lcc.uma.es/evirtual/cdd/tutorial/enlace/detec.html Fecha de consulta: Junio 2013.

[16] http://enciclopedia.us.es/index.php/Codificador. Fecha de consulta: Junio 2013.

[17] http://digital.csic.es/bitstream/10261/21259/1/Codbidimens.pdf Fecha de consulta: Junio 2013.

[18] http://www.uaq.mx/ingenieria/publicaciones/eureka/n22/sandov01.pdf Fecha de consulta Julio 2013.

[19] http://www.pclviewer.com/rs2/galois.html Fecha de consulta: Julio 2013.

[20] http://www.javeriana.edu.co/Facultades/ingenieria/revista/Vol11nr1Codificador.pdf Fecha de consulta: Julio 2013.

[21] http://www.articulandia.com/premium/article.php/12-03-2007El-Lenguaje-C.htm Fecha de consulta Agosto 2013.

[22] http://www.larevistainformatica.com/C++.htm Fecha de consulta Agosto 2013.

[23] http://www.zator.com/Cpp/E1_2.htm Fecha de consulta Agosto 2013.

[24] http://www.epicor.com/Products/Pages/Eclipse-Software.aspx Fecha de consulta Agosto 2013.

[25] http://www.edukanda.es/mediatecaweb/data/zip/678/XW07_M2101_02708/web/main/m1/v9_10_1.html Fecha de consulta Agosto 2013.

[26] http://mx.globedia.com/aspectos-teoricos-entorno-desarrollo-integrado-ide Fecha de consulta Agosto 2013.

# GLOSARIO

**ADC** (Analog to Digital Converter)
Convertidor Analógico Digital, es un dispositivo electrónico que su función es convertir voltajes en una representación digital de unos y ceros, que puede ser guardada, manipulada y convertida de vuelta a analógica.

**Codificar**
Representar cada uno de los símbolos provenientes de una fuente por medio de un conjunto de símbolos predefinidos.

**Códigos BCH**(Bose, Ray-Chaudhuri, Hocquenghem codes)
Códigos de corrección de errores de Bose, Ray-Chaudhuri y Hocquenghem.

**Corrección de errores**
Posibilidad que se tiene en las comunicaciones digitales de corregir ciertos errores que ocurran en una transmisión.

**Decodificador**
Dispositivo que recibe las órdenes enviadas desde la oficina central y convierte las señales digitales comprimidas en video analógico y en audio. El Set-Top Box o decodificador permite decodificar las señales de televisión transportadas por la red de cable. En un extremo el decodificador se conecta a la toma de usuario y por el otro al televisor o al aparato de vídeo.

**Detección**
Es el proceso de decidir cuál de las posibles señales que puede originar una fuente es la que con mayor probabilidad generó una señal recibida.

**Detección de errores**
Es la posibilidad que se tiene en las comunicaciones digitales de identificar la ocurrencia de ciertos errores en una transmisión.

**DSP**
Procesadores digitales de señal son microprocesadores específicamente diseñados para el procesado digital de señal. Estrictamente hablando, el término DSP se aplica a cualquier chip que trabaje con señales representadas de forma digital.

**DVB (Transmisión de Video Digital.)**
Es una solución propuesta para la televisión digital y la transmisión de datos por todo el rango de medios de entrega. Todos los sistemas DVB se basan en compresión de audio y video MPEG-2. DVB agrega al flujo múltiple de transporte MPEG los elementos necesarios para llevar los servicios de transmisión digital al hogar a través del cable, el satélite y los sistemas de transmisión terrestres.

**DVB-T** (Digital Video Broadcasting Terrestrial)
Difusión digital de vídeo terrestre.

**DVB-T2** (Digital Video Broadcasting Terrestrial version 2)
Difusión digital de vídeo terrestre versión 2.

**DVB-S:** (Digital Video Broadcasting by Satellite)
Difusión digital de vídeo por satélite.

**GSM**
El diminutivo de Sistema global de comunicaciones móviles, es la plataforma de telefonía móvil más utilizada en todo el mundo.

**HDTV (High Definition TV)**
Televisión de alta definición.

**IEEE**
Corresponde a las siglas de The Institute of Electrical and Electronics Engineers, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación, ingenieros en informática e ingenieros en telecomunicación.

**MPEG-2**
Norma técnica internacional de compresión de imagen y sonido. El MPEG-2 especifica los formatos en que deben de representarse los datos en el descodificador y un conjunto de normas para interpretar estos datos. Es un estándar definido específicamente para la compresión de vídeo, utilizado para la transmisión de imágenes en vídeo digital.

**Transceptor**
Es la unión de un Receptor con un Transmisor en un solo equipo. Es un dispositivo que realiza, dentro del mismo chasis, funciones de trasmisión y recepción, utilizando componentes de circuito comunes para ambas funciones. Dado que determinados elementos se utilizan tanto para la transmisión como para la recepción, la comunicación que provee un transceptor solo puede ser semidúplex, lo que significa que pueden enviarse señales entre dos terminales en ambos sentidos, pero no simultáneamente.

**Radio cognitivo**
Es un paradigma de la comunicación inalámbrica en la cual tanto las redes como los mismos nodos inalámbricos cambian los parámetros particulares de transmisión o de recepción para ejecutar su cometido de forma eficiente sin interferir con los usuarios autorizados.