Análisis y Simulación en Matlab del Método de Detección y Corrección de Errores Reed-Solomon (204,188) Utilizado en la Norma ISDB-TB e Implementación en un FPGA

L. Aguirre E. Gordón

CELEC EP TRANSELECTRIC
E-mail: lenin.aguirre@celec.gob.ec, esteban.gordon@celec.gob.ec

Resumen

El presente artículo busca entender el funcionamiento del método de detección y corrección de errores que se maneja dentro de la norma ISDB-TB. Se elabora un algoritmo para el codificador y decodificador utilizando Matlab y posterior implementación en la tarjeta FPGA Spartan 3E en lenguaje VHDL. Con el fin de facilitar la comprensión y el uso de los algoritmos se parte de un Reed-Solomon (15,9), el cual ayudará a facilitar los procesos matemáticos que posteriormente serán ejecutados en Matlab y con los cuales se desarrolla una interfaz gráfica amigable en donde se puede interactuar con la tarjeta FPGA desplegando la simulación y así verificar el análisis a continuación descrito.

***Palabras clave*— FPGA (Field Programmable Gate Array), Reed-Solomon, MATLAB (MATrix LABoratory), VHDL y ISDB-Tb.**

Abstract

This article seeks to understand the operation of the method of error detection and correction is handled within the ISDB-TB standard. An algorithm for the encoder and decoder using Matlab and subsequent implementation in Spartan 3E FPGA VHDL card is made. In order to facilitate the understanding and use of algorithms is part of a Reed-Solomon (15,9), which will help facilitate mathematical processes that will later be executed in Matlab and with which a user-friendly graphical interface is developed where you can interact with the FPGA card displaying the simulation and thus verify the analysis.

***Index terms*— FPGA (Field Programmable Gate Array), Reed-Solomon, MATLAB (MATrix LABoratory, VHDL and ISDB-Tb.**

Recibido: 09-09-2016, Aprobado tras revisión: 09-12-2016
Forma sugerida de citación: Aguirre, L; Gordón, E. (2017). "Análisis y Simulación en Matlab del Método de Detección y Corrección de Errores Reed-Solomon (204,188) Utilizado en la Norma ISDB-TB e Implementación en un FPGA" Revista Técnica ***"energía"***. No. 13, Pp. 117-126
ISSN 1390-5074.

1. INTRODUCCIÓN

La codificación Reed-Solomon (RS) es un esquema de codificación de bloque que puede detectar y corregir ráfagas de errores hasta un cierto límite, determinado por la cantidad de redundancia con la que se ha diseñado el código y se utiliza en el estándar ISDB-Tb. Este tipo de código surge de una familia llamada códigos de bloque, en que el codificador procesa un bloque de símbolos, a los que agrega redundancia para generar otro bloque de una longitud mayor de símbolos codificados. Se tiene una gran complejidad en el proceso computacional, pero este problema ha sido superado gracias a la implementación de circuitos integrados en gran escala. Los códigos Reed-Solomon son códigos FEC, no binarios, bloque lineal, cíclicos y son una subclase de los códigos BCH (su nombre hace referencia a las siglas de sus tres autores Bose, Ray-Chandhuri y Hocquengheim). El código fue inventado por Irving S. Reed y Gustave Solomon (de ahí su nombre) en el año de 1960. En el presente trabajo se creará un algoritmo a través del cual se puede generar el código RS de cualquier longitud y el cual podrá ser aplicado a varias aplicaciones.

2. RS(204,188)CARACTERÍSTICASGENERALES

Las señales de audio, video y datos ingresan al compresor MPEG, del cual se derivan las salidas TS (Transport Stream), estas múltiples salidas alimentan al remultiplexor el cual las transforma en un haz de transporte TSP (Transport Stream Packet), señal ráfaga de 188 bytes. Se debe entonces obligatoriamente aplicar el código Reed Solomon para que el TSP resultante sea convertido en otro de longitud de 204 bytes, consistiendo en 188 bytes de datos de programa y 16 bytes de paridad. El flujo de bits ya no es llamado TSP sino BTS (Broadcast Transport Stream). Debido al tamaño del TSP (204 bytes), un RS abreviado (204,188) se debe aplicar obligatoriamente en cada TSP como un código externo.

La codificación RS abreviada (204,188) se debe generar agregando 51 bytes en el comienzo de la entrada de los datos del código RS (255,239), y entonces después de aplicar la codificación externa esos 51 bytes se deben remover obligatoriamente. El código RS abreviado (204,188) puede corregir hasta 8 bytes aleatorios erróneos entre los 204 bytes [2].

A. Aritmética de Módulo 2

El presente estudio se basa en códigos binarios, en los cuales se utiliza un alfabeto simple con dos bits 0 y 1. La codificación y decodificación en este tipo de códigos envuelve operaciones de aritmética binaria de suma y multiplicación módulo 2 ($\oplus$,*), su correspondencia en la lógica binaria, es la operación OR EXCLUSIVA y AND respectivamente.

B. Teoría de los Campos de Galois

En un campo o extensión de un anillo se definen las dos operaciones conocidas como suma y multiplicación módulo dos en dicho campo, sus elementos cumplen los siguientes axiomas.

- Clausurativo
$$(\forall x,y \in G):(x \oplus y) \in G \quad (1)$$
- Asociativo
$$(\forall x,y,z \in G):(x \oplus y) \oplus z = x \oplus (y \oplus z) \in G \quad (2)$$
- Existencia de neutro
$$(\exists e \in G)(\forall x \in G):x \oplus e = e \oplus x = x \quad (3)$$
- Existencia del inverso
$$(\forall x \in G)(\exists i \in G):x \oplus y = y \oplus x = e \quad (4)$$
- Conmutativo
$$(\forall x \in G)(\exists y \in G):x \oplus y = y \oplus x \quad (5)$$
- Distributivo
$$(\forall x,x,z \in G):x * (y \oplus z) = (x * y) \oplus (x * z) \quad (6)$$

Al evaluar los axiomas en un campo se pueden dar dos opciones, la primera si el campo $(G,\oplus)$ cumple con los axiomas conmutativo y existencia del neutro, se dice que el campo es un anillo con elemento neutro y si el campo $(G,\oplus,*)$ es un anillo conmutativo con elemento neutro, y cumple con los axiomas de existencia de inverso y de neutro respecto a la operación suma módulo 2, se dice que el campo es un campo de Galois GF.[3] Dado un polinomio $p(x)$ en $F[x]$, donde $F[x]$ es el campo correspondiente a este polinomio, se asocia con $p(x)$ a un grupo de elementos denominado el grupo de Galois de $p(x)$. Las raíces del polinomio tienen una estrecha relación con su grupo de Galois. El grupo de Galois resulta ser el conjunto de permutaciones de las raíces del polinomio. El grupo de Galois de $p(x)$ queda definido como un cierto grupo de automorfismo del campo de descomposición de $p(x)$ sobre $F[x]$, es decir el conjunto de todas las raíces del polinomio. Los subgrupos del grupo de Galois y los subcampos del

campo de descomposición, mantienen una dualidad que expresa el teorema fundamental de la teoría de Galois [4].

C. Estructura del código RS

En la codificación (RS) se usa la representación RS(n, k) que tiene las siguientes características:

- m bits consecutivos agrupados que forman un símbolo.
- k símbolos de información en cada secuencia codificada.
- q símbolos de paridad en cada secuencia codificada.
- Longitud de la secuencia codificada, $n = k + q$ símbolos.
- $n = 2^m - 1$, donde 2^m es el número total de símbolos en el campo de Galois (GF) con los que se forma la secuencia codificada.
- Corrige hasta t símbolos erróneos en la secuencia codificada, donde $t = q/2$.
- Distancia mínima del código.

$$d_{min} = 2t + 1 = n - k + 1 \qquad (7)$$

El GF está formado por pp elementos, en donde pp siempre es un número primo. El campo $GF(p)$ se lo puede extender a un campo de $GF(p^m)$ elementos, en donde m es cualquier número entero positivo que tiene un valor mayor que 2. Para la generación de códigos RS, los elementos del campo de Galois serán $p = 2$. Los elementos del campo de Galois se representan con el *0, 1,* α y los subsiguientes se generan multiplicando el anterior por $\alpha\alpha$. Para cerrar la secuencia y dar un campo finito de elementos en $GF(2^m)$ se establece:

$$\propto^{(2^m-1)} + 1 = 0 \qquad (8)$$

Mediante la condición mencionada, cualquier elemento del campo que tenga una potencia igual o mayor que $2^m - 1$ se puede transformar a un elemento con una potencia menor que $2^m - 1$:

$$\propto^{(2^m+n)} = \propto^{(2^m-1)} \propto^{(n+1)} = \propto^{(n+1)} \qquad (9)$$

El campo finito $GF(2^m)$, está conformado por 2^m elementos consolidados de la siguiente manera:

$$GF(2^m) = \{0, \propto^0, \propto^1, \propto^2, \dots, \propto^{2^m-2}\} \qquad (10)$$

D. Polinomio Primitivo

Se define un polinomio primitivo o no reducible como aquel que no se puede factorizar en un producto de polinomios de menor grado [5].

Tabla 1: Polinomios primitivos de grado 2 a 8

$p_2(x) = x^2 + x + 1$
$p_3(x) = x^3 + x + 1$
$p_4(x) = x^4 + x + 1$
$p_5(x) = x^5 + x^2 + 1$
$p_6(x) = x^6 + x + 1$
$p_7(x) = x^7 + x + 1$
$p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$

Los polinomios primitivos se utilizan para la generación de los campos finitos de Galois, estos últimos permiten la generación de los códigos RS.

Se presenta los polinomios primitivos de grado 2 a 8 (ver Tabla 1).

E. Polinomio Generador

En general el polinomio $x^n + 1x^n + 1$ y sus factores, son una parte importante en la creación del RS. El polinomio generador ayuda a encontrar los elementos de un campo de Galois GF(2^n) [6].

Se constituye $g(x)g(x)$ como:

$$g(x) = 1 + \sum_{i=1}^{n-k-1} g_i X^i + X^{n-k} \qquad (11)$$

Donde el coeficiente g_i es igual a 0 o 1.

3. ALGORITMO DE CODIFICACIÓN REED-SOLOMON (15,9)

Para el entendimiento del proceso de codificación y decodificación, se describirá un código RS(15,9). Este proceso se puede generalizar para códigos más grandes, ya que no se altera la estructura y características, se puede decir que crecen linealmente con el tamaño de procesamiento de información [7].

Como ejemplo se desea codificar la secuencia 1111 0000 0000 1100 0000 0000 0000 0000 1000 empleando un código RS con $k = 9$ símbolos de información, $n = 15$ símbolos de tamaño de palabra código, $m = 4$ bits individuales en cada símbolo y capaz de corregir $t = 3$ errores. Para generar este código se emplean los campos de Galois de orden 2^4. Para el RS(15,9), el campo de Galios $GF(2^4)$ es el siguiente:

$$GF(2^4) = \{0, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{14}\} \quad (12)$$

Cualquier operación que se realice sobre un elemento genera otro elemento del campo, de esta manera el campo tiene un tamaño finito. Para el ejemplo RS(15,9) se emplea el polinomio primitivo de GF de orden 2^4, obtenido de la Tabla 1.

$$p(x) = 1 + x + x^4 \quad (13)$$

El grado del polinomio primitivo es $m = 4$. Con esto se obtiene $2^4 = 16$ elementos en el campo definido por $p(x)$. Se evalúa el polinomio $p(x)$ con diferentes valores de x para determinar cuando el polinomio da como resultado $p(x) = 0$. Los elementos binarios 1 y 0 no dan como resultado 0, ya que con la suma módulo 2, se tiene:

$$p(1) = 1 + 1 + 1^4 = 1 \quad (14)$$

$$p(0) = 1 + 0 + 0^4 = 1 \quad (15)$$

Se evalúa el polinomio $p(x)$ con α obteniendo lo siguiente:

$$p(\alpha) = 0 \quad (16)$$

$$1+\alpha+\alpha^4 = 0 \quad (17)$$

$$\alpha^4 = -1-\alpha \quad (18)$$

Dado que en el campo binario +1 =-1 y $\alpha = -\alpha$ debido a que $\alpha + \alpha = 0$, con lo cual α^4 se puede representar como:

$$\alpha^4 = -\alpha^4 \quad (19)$$

$$\alpha^4 = 1+\alpha \quad (20)$$

α^4 se expresa como una suma términos que tienen órdenes inferiores. De hecho todas las potencias de α pueden ser expresadas de la misma forma. Por ejemplo en α^5 se tiene:

$$\alpha^5 = \alpha^4 \times \alpha = (1+\alpha)\,\alpha = \alpha + \alpha^2 \quad (21)$$

De la misma manera se obtienen resultados similares para los demás exponentes de α. Además se puede comprobar que cualquier elemento que se encuentra fuera del campo de Galios $GF(2^4)$ es igual a uno de los elementos del campo $GF(2^4) = \{0, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{14}\}$ de la siguiente manera:

$$\alpha^{15} = \alpha^{14} \times \alpha = (1+\alpha^3)\,\alpha = \alpha + \alpha^4 = 1 + \alpha + \alpha = 1 = \alpha^0 \quad (22)$$

Esto quiere decir que la serie de elementos se repiten, entonces se tienen los 16 elementos del campo finito de $GF(2^4)$.

A. Representación Polinomial y Vectorial de los Elementos GF

Para un mejor entendimiento del campo de Galois se presenta su estructura en términos de 0s y 1s y de esta manera se relacionan los símbolos binarios (vectores) con los elementos del campo GF . Diferentes formas en las cual se puede representar el campo $GF(2^4)$ generado por el polinomio primitivo correspondiente (ver Tabla 2).

Tabla 2: Campos de Galois $GF(2^4)$

Forma Potencial	Forma Polinomial	Forma Vectorial
0	0	0 0 0 0
α^0	1	1 0 0 0
α^1	α	0 1 0 0
α^2	α^2	0 0 1 0
α^3	α^3	0 0 0 1
α^4	$1+\alpha$	1 1 0 0
α^5	$\alpha+\alpha^2$	0 1 1 0
α^6	$\alpha^2+\alpha^3$	0 0 1 1
α^7	$1+\alpha+\alpha^3$	1 1 0 1
α^8	$1+\alpha^2$	1 0 1 0
α^9	$\alpha+\alpha^3$	0 1 0 1
α^{10}	$1+\alpha+\alpha^2$	1 1 1 0
α^{11}	$\alpha+\alpha^2+\alpha^3$	0 1 1 1
α^{12}	$1+\alpha+\alpha^2+\alpha^3$	1 1 1 1
α^{13}	$1+\alpha^2+\alpha^3$	1 0 1 1
α^{14}	$1+\alpha^3$	1 0 0 1

B. Determinación del Polinomio Generador

Con el polinomio generador se encuentran los símbolos de paridad. Para la construcción del polinomio se deben tomar $2t$ raíces consecutivas, de tal forma que el polinomio generador representado por $g(x)$ tiene la siguiente forma:

$$g(x) = (x+\alpha)(x+\alpha^2)\ldots(x+\alpha^{2t}) \quad (23)$$

$$g(x) = (x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4)(x+\alpha^5)(x+\alpha^6) \quad (24)$$

$$g(x) = x^6 + \alpha^{10}x^5 + \alpha^{14}x^4 + \alpha^4 x^3 + \alpha^6 x^2 + \alpha^9 x + \alpha^6 \quad (25)$$

También se muestra el polinomio a utilizar en el codificador RS (15,9). Se observa que el grado del polinomio generador es igual al número de símbolos de paridad.

C. Cálculo de la Palabra Codificada

Para codificar la secuencia binaria i se seleccionan los bits de información de cuatro en cuatro que constituyen los símbolos de la secuencia de información y se construye el polinomio $i(x)$ transformando cuatro bits en un elemento de $GF(2^4)$ a partir de su representación polinómica, lo que da como resultado:

i=1111 0000 0000 1100 0000 0000 0000 0000 1000

$$i = \alpha^{12}\ 0\ 0\ \alpha^{4}\ 0\ 0\ 0\ 0\ 1$$

$$k = 9$$

Polinomio de los símbolos de información

$$i(x) = \alpha^{12} + 0x^1 + 0x^2 + \alpha^4 x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7 + 1x^8$$
$$i(x) = \alpha^{12} + \alpha^4 x^3 + x^8 \quad (26)$$

Al realizar la codificación RS es fundamental tomar en cuenta la estructura de la palabra de información o mensaje $i(x)$ y de la paridad $q(x)$.

Tabla 3: Ejemplo RS (15,9)

n									
Paridad $q(x)$						**Mensaje** $i(x)$			
						α^{12}	0	0	α^4	0	0	0	0	1
$q = n - k = 2t$ **símbolos**						k **símbolos**								

Para realizar la codificación se requiere desplazar los coeficientes del mensaje en $n-k$ posiciones:

$$i(x).x^{2t} = i(x).x^6 \quad (27)$$

$$i(x).x^{2t} = (\alpha^{12} + \alpha^4 x^3 + x^8)x^6 \quad (28)$$

$$i(x).x^{2t} = \alpha^{12}x^6 + \alpha^4 x^9 + x^{14} = I(x) \quad (29)$$

Posteriormente se divide $I(x)$ para $g(x)$:

$$\frac{I(x)}{g(x)} = \frac{u(x).x^{2t}}{g(x)} = \frac{u(x).x^{n-k}}{g(x)} = \frac{u(x).x^{q}}{g(x)} \quad (30)$$

Como residuo de la división se tiene:

$$q(x) = \alpha^{11} + \alpha^{12}x + \alpha^9 x^2 + \alpha^2 x^3 + \alpha^4 x^4 + \alpha^{11} x^5$$

La palabra codificada se encuentra expresada como:

$$V(x) = q(x)|I(x) \quad (31)$$

$$V(x) = \alpha^{11} + \alpha^{12}x + \alpha^9 x^2 + \alpha^2 x^3 + \alpha^4 x^4 \quad (32)$$

$$+\alpha^{11} x^5 + \alpha^{12} x^6 + \alpha^4 x^9 + x^{14} \quad (33)$$

$$V = (\alpha^{11}\alpha^{12}\alpha^9\alpha^2\alpha^4\alpha^{11}\ \alpha^{12}\ 0\ 0\ \alpha^4\ 0\ 0\ 0\ 0\ \alpha^0) \quad (34)$$

La Palabra codificada con la representación $GF(2^4)$ es:

$$V = \alpha^{11}\ \alpha^{12}\ \alpha^9\ \alpha^2\ \alpha^4\ \alpha^{11} \mid \alpha^{12}\ 0\ 0\ \alpha^4\ 0\ 0\ 0\ \alpha^0$$

Finalmente la palabra codificada con la representación binaria:

|0111|1111|0101|0010|1100|0111|1111|0000|0000|1100|0000|0000|0000|0000|1000

D. Ejemplo de Decodificación RS(15,9)

Para el ejemplo de la decodificación se insertan errores a la palabra codificada.

$$V = \alpha^9 + \alpha^{12} + \alpha^9 + \alpha^2 + \alpha^4 + \alpha^1 + \alpha^{12} + 0 + 0 + \alpha^4 + 0 + 0 + 0 + \alpha^5 + \alpha^0 \quad (35)$$
$$V = \alpha^9\ \alpha^{12}\ \alpha^9\ \alpha^2\ \alpha^4\ \alpha^1 \mid \alpha^{12}\ 0\ 0\ \alpha^4\ 0\ 0\ 0\ \alpha^5\ \alpha^0$$

|0101|1111|0101|0010|1100|0100|1111|0000|0000|1100|0000|0000|0000|0110|1000 (36)

E. Cálculo de los Síndromes

Al realizar la revisión de la paridad en la palabra codificada $v(\alpha)$ con o sin errores recibida por el decodificador se tiene como resultado el valor del síndrome. Es decir que se evaluará el polinomio con los $n-k$ símbolos de los cuales está conformado el síndrome, en el caso a estudiarse tenemos un RS(15,9) con 6 símbolos en el vector. Por lo tanto se tiene que el síndrome puede ser definido como:

$$S_i = v(x)|_{x=\alpha^i} \qquad i = 1, \ldots, n-k \quad (37)$$

Si al evaluar el vector síndrome de la palabra recibida, el resultado es igual a cero, dicha palabra será una palabra código válida, de otro modo al ser el resultado diferente de cero, se tendrá una palabra de código con errores. Primero se evaluará la palabra código con los seis símbolos pertenecientes al síndrome, es decir:

$$S_1 = v(\alpha), S_2 = v(\alpha^2), S_3 = v(\alpha^3), \ldots, S_6 = v(\alpha^6)$$

Se tendrá el vector síndrome de la siguiente manera:

$$S_i = (\alpha^1, \alpha^2, \alpha^0, \alpha^8, \alpha^0, \alpha^{13}) \quad (38)$$

El vector síndrome resultante es diferente de cero, por lo tanto la palabra código que llegó al receptor posee errores.

F. Cálculo del Polinomio Detector de Error

El algoritmo más eficiente para la decodificación de códigos bloque cíclicos binarios es el algoritmo de Berlekamp-Masey, cuya complejidad crece linealmente con la distancia mínima del código. Permitirá encontrar el polinomio localizador de errores, siempre y cuando el número de errores producidos no supere la capacidad correctora t del código. Caso contrario el algoritmo producirá un polinomio que no cumpla con las propiedades del polinomio localizador o un polinomio que cumpla con estas propiedades pero incorrecto, en ambos casos la decodificación será equivocada, siendo esta una limitante de Berlekamp-Masey. El polinomio localizador de errores permitirá el cálculo de la ubicación exacta dentro de la palabra que llega al receptor y tendrá la siguiente forma [8]:

$\Lambda(x) = \Lambda^{2t}(x) \Lambda(x) = \Lambda^{2t}(x)$, de grado menor o igual que t.

Berlekamp-Massey es un algoritmo iterativo que analizará los valores de los síndromes de forma como se muestra en los siguientes pasos:

a) Calcular los síndromes $S_1, S_2, \ldots S_{2t}$

b) Inicializar

$$\Lambda^0(x) = 1, L = 1, T(x) = x \; y \; k = 0 \qquad (39)$$

c) Calcular $\Delta_k = S_k + \sum_{i=1}^{L_{k-1}} \Lambda_i^{(k-1)} . S_{k-i}$

d) Modificar $\Lambda^{(k)}(x) = \Lambda^{(k-1)}(x) - \Delta_k T(x)$

e) Si $2L \geq k$, ir al paso f) caso contrario ir al paso g)

f) Igualar $L = k - L_k - 1$ y $T(x) = \Lambda^{(k-1)}(x) / \Delta_k$, ir al paso h)

g) Igualar $L = L_k - 1$

h) Igualar $T(x) = xT(x) T(x) = xT(x)$

i) Si $k < 2t$, entonces $k = k + 1$ e ir al Paso c)

Dónde:

Polinomio localizador de error $\Lambda(x)$

Polinomio: $\Lambda(x) = \Lambda^{2t}(x)$

$T^k(x)$: Polinomio auxiliar, permite calcular $\Lambda(x)$

S_k: Síndrome

Δ: Discrepancia entre síndromes

k: Iteraciones en curso

L: Complejidad lineal

Para el ejemplo se calculará el polinomio localizador de error, con los resultados de las iteraciones calculadas (ver Tabla 4).

Tabla 4: Algoritmo Berlekamp - Massey

S_k	$\Lambda^K(x)$	Δ_k	L_k	$T^K(x)$
-	1	-	0	x
α^1	$1+\alpha^1 x$	α^1	1	$\alpha^{14} x$
α^2	$1+\alpha^1 x$	0	1	$\alpha^{14} x^2$
α^0	$1+\alpha^1 x+\alpha^{13} x^2$	α^{14}	2	$\alpha^1 x+\alpha^2 x^2$
α^9	$1+\alpha^7 x+\alpha^5 x^2$	α^5	2	$\alpha^1 x^2+\alpha^2 x^3$
α^0	$1+\alpha^7 x+\alpha^9 x^2+\alpha^7 x^3$	α^5	3	$\alpha^{10} x+\alpha^2 x^2+\alpha^0 x^3$
α^{13}	$1+\alpha^0 x+\alpha^3 x^2+\alpha^1 x^3$	α^{14}	-	-

Donde se tiene como resultado al polinomio localizador de error a:

$$\Lambda(x) = 1 + \alpha^9 x + \alpha^4 x^2 + \alpha^3 x^3 \qquad (40)$$

Ya encontrado este polinomio es necesario factorizarlo para así determinar las ubicaciones del error, para esto procedemos a evaluarlo verificando uno a uno si cada elemento α^j sobre el cual se evalúa, es raíz del polinomio Λ(x).

$$\Lambda(\alpha^0), \Lambda(\alpha^1), \ldots, \Lambda(\alpha^{14}) \qquad (41)$$

Con estos resultados se puede concluir que las raíces de Λ(x) son:

$$\alpha^0, \alpha^2 \; y \; \alpha^{10}$$

La ubicación de los errores serán los correspondientes a los recíprocos de las raíces calculadas. Por lo tanto las posiciones de los errores son:

$$x^0, x^5 \; y \; x^{13}$$

Para determinar los valores de los errores se definirá el polinomio evaluador de errores $z(x)$, en la forma

$$z(x) = S(x) \times \Lambda(x) \qquad (42)$$

El polinomio z(x) relaciona la ubicación y magnitud del error

$$z(x) = 1 + (S_1 + \Lambda_1)x + (S_2 + \Lambda_1 S_1 + \Lambda_2)x^2 + \cdots + (S_t + \Lambda_1 S_{t-1} + \cdots + \Lambda_t)x^t$$

$$z(x) = 1 + (S_1 + \Lambda_1)x + (S_2 + \Lambda_1 S_1 + \Lambda_2)x^2 + (S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 + \Lambda_3)x^3$$

$$z(x) = 1 + \alpha^3 x + \alpha^{-inf} x^2 + \alpha^0 x^3 = 1 + \alpha^3 x + x^3$$

Se emplea el algoritmo de Forney, dada las posiciones de los errores, para el cálculo de la magnitud de estos (diferencia entre símbolo correcto y recibido) por medio de la siguiente expresión:

$$e_j = \frac{z(\propto_j^{-1})}{\prod_{i \neq j}(1 + \propto_i . \propto_j^{-1})} \quad (43)$$

Donde:

$$\propto_0 = \propto^0$$
$$\propto_5 = \propto^5$$
$$\propto_{13} = \propto^{13}$$

$$e_0 = \frac{1 + \propto^3 \left(\frac{1}{\propto^0}\right) + \propto^{-inf} \left(\frac{1}{\propto^0}\right)^2 + \propto^0 \left(\frac{1}{\propto^0}\right)^3}{\left(1 + \propto^5 . \frac{1}{\propto^0}\right)\left(1 + \propto^{13} . \frac{1}{\propto^0}\right)} \quad (44)$$

$e_0 = \propto^2 \rightarrow$ *magnitud error posición* 0
$e_5 = \propto^6 \rightarrow$ *magnitud error posición* 5
$e_{13} = \propto^5 \rightarrow$ *magnitud error posición* 13

Una vez calculada la magnitud del error se calcula el polinomio

$$e(x) = e_0 + e_5 x^5 + e_{13} x^{13} \quad (45)$$

$$e(x) = \propto^2 + \propto^6 x^5 + \propto^5 x^{13} \quad (46)$$

Como paso final de la decodificación y para obtener la palabra código transmitida se realizará la diferencia entre la palabra recibida $r(x)$ y el error $e(x)$.

$$V(x) = v(x) + e(x) \quad (47)$$

$$V(x) = \propto^{11} + \propto^{12} x + \propto^9 x^2 + \propto^2 x^3 + \propto^4 x^4 \quad (48)$$

$$+ \propto^{11} x^5 + \propto^{12} x^6 + \propto^4 x^9 + x^{14} \quad (49)$$

La palabra de información serán los últimos nueve símbolos de la palabra codificada retirando los seis símbolos del inicio pertenecientes a la paridad y desplazándolos las seis ubicaciones hacia atrás que se aumentaron al iniciar el proceso.

$$i(x) = \alpha^{12} + \alpha^4 x^3 + x^8 \quad (50)$$

4. REED-SOLOMON (204,188)

La codificación abreviada se debe generar agregando 51 bytes en el comienzo de la entrada de los datos obteniendo un código RS(255,239). Para este codificador se debe utilizar el campo de Galios:

$$GF(2^8)$$

Polinomio primitivo $p(x)$:

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \quad (51)$$

Polinomio generador $g(x)$:

$$g(x) = (x+\propto)(x+\propto^2)(x+\propto^3) \ldots\ldots (x+\propto^{16}) \quad (52)$$

Al final de la decodificación se deberá remover los 51 bytes agregados al inicio del codificador.

5. SIMULACIÓN REED-SOLOMON (204,188)

A continuación se presenta la simulación del método de detección y corrección de errores Reed-Solomon (204,188) desarrollado en Matlab y con la herramienta Simulink. También se incluye un bloque de transmisión de señal OFDM con esquema de modulación de las portadoras QPSK, con el fin de poder ingresar el flujo de bits provenientes del codificador RS al canal de prueba. El programa permite emular diferentes niveles de ruido en el canal, desvanecimientos rápidos y también los intervalos de guarda 1/4, 1/8, 1/16, 1/32 de la duración del símbolo activo para el modo 1 de la transmisión de una señal de TV digital ISDBT-b [2] [9].

La palabra de información es editable y obtenida aleatoriamente por medio del botón GENERAR. En la Fig. 1 se aprecia el ejemplo que se genera una palabra aleatoria. No se puede apreciar todo la palabra debido a su longitud de 188 bytes.

Figura 1: Simulación RS (204,188)

Después de que la palabra de información es ingresada en el programa se emula un canal por el cual se transmite la señal codificada. Dentro del canal se inserta ruido el cual producirá errores en la palabra. Para que la transmisión se acerque a realidad se implementa un modulador OFDM+QPSK [10].

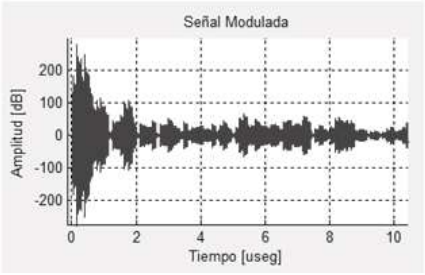

Figura 2: Señal modulada a transmitirse

En la Fig. 3 se aprecian los valores recuperados después del mapeo QPSK. Se presenta la palabra codificada con errores en forma discreta.

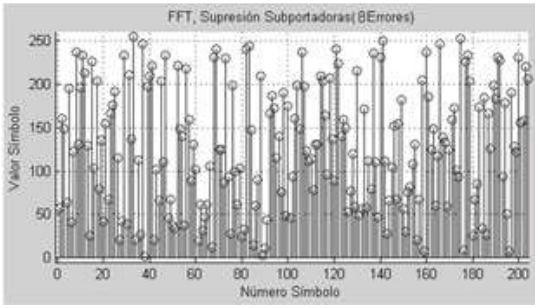

Figura 3: Señal recuperada con errores en el receptor

En la Fig. 4 se presenta la interfaz gráfica con el proceso completo de la decodificación RS (204,188).

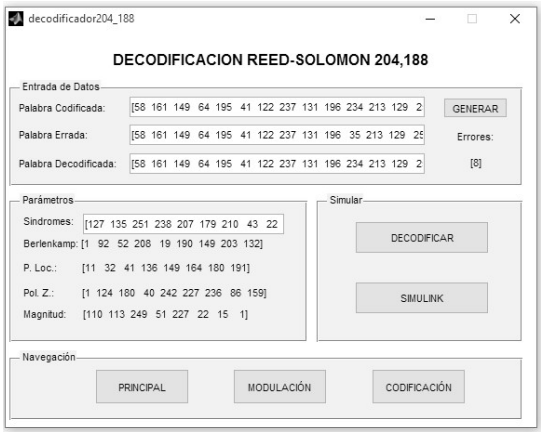

Figura 4: Interfaz gráfica con los valores completos de la decodificación.

6. IMPLEMENTACIÓN DE RS EN TARJETA FPGA SPARTAN 3E

Inicialmente se presentará algunas definiciones necesarias para el entendimiento de esta sección:

FPGA (Field Programmable Gate Array) es un dispositivo programable que contiene bloques de lógica que puede ser configurada mediante un lenguaje de descripción especializado.

VHDL, lenguaje definido por el IEEE usado para describir circuitos digitales o modelar fenómenos científicos respectivamente. VHDL es el acrónimo de VHSIC y HDL, donde VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language).

Para la elaboración del código VHDL se ha partido del entorno de Matlab, que consiste en una herramienta técnica de altas prestaciones para el cálculo numérico. Se ha transformado los algoritmos que estaban en lenguaje de Matlab a lenguaje VHDL siendo usada la herramienta Math Works, el generador HDL Code. La función del generador HDL coder es generar automáticamente un código HDL desde MATLAB, es decir que permite diseños sobre FPGAs. Ya que el objetivo de la creación del lenguaje fue principalmente lograr el diseño, modelado y documentación del Método de Detección y Corrección de errores Reed-Solomon se ha procedido a realizar la transformación de los algoritmos desde Matlab, facilitando la realización de scripts de los circuitos complejos desarrollados.

Para la implementación física se utiliza la tarjeta FPGA SPARTAN 3E, donde se cargará la configuración del algoritmo en VHDL. Para realizar las pruebas de funcionamiento se han configurado la entrada y salida de los datos a través del puerto serial. De esta manera el computador generará los datos, los cuales a través del puerto serial se envían hacia la tarjeta, en donde se aplicará el algoritmo de codificación cuando el usuario procede a ejecutarlo a través del botón EJECUTAR presente en la interfaz gráfica como se explicará posteriormente. La palabra codificada saldrá de la tarjeta por el mismo puerto serial hacia el computador donde se presentará por medio de un display en SIMULINK. De la misma manera se procederá la decodificación [11].

Figura 5: Sistema de comunicación entre PC y tarjeta FPGA

En la Fig. 6 se observa la prueba del codificador y decodificador RS en la tarjeta FPGA

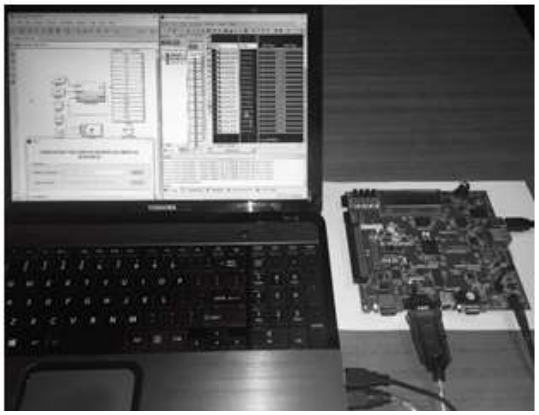

Figura 6: Conexión física entre la tarjeta y el computador

En la Fig. 7 se observa la interfaz FPGA con la palabra decodificada.

Figura 7: Interfaz gráfica con los valores completos de la decodificación.

7. RECURSOS UTILIZADOS EN LA TARJETA FPGA SPARTAN 3E

La Fig. 8 presenta el resumen detallado en porcentaje del nivel de uso de la tarjeta FPGA. En el campo "Number of Slices" se indica que se ha utilizado el 101% de la tarjeta, esto se debe a que la Spartan 3E tiene un 5 % de reserva del cual se ha utilizado el 5%.

Device Utilization Summary (estimated valu			
Logic Utilization	Used	Available	Utilization
Number of Slices	4742	4656	101%
Number of Slice Flip Flops	353	9312	3%
Number of 4 input LUTs	8281	9312	88%
Number of bonded IOBs	12	232	5%
Number of GCLKs	1	24	4%

Figura 8: Recursos utilizados en la tarjeta FPGA Spartan 3E para el codificador y decodificador RS (15,9)

La tarjeta FPGA Spartan 3E posee una memoria limitada lo que permite un número limitado de comandos. Por esta razón se podrá realizar la implementación del código Reed-Solomon (15,9) con un nivel de procesamiento del 100% y su funcionamiento es el correcto. Para el código Reed-Solomon (204,188) se requiere una tarjeta con una memoria más extensa, debido a que la tarjeta utilizada en el actual estudio no se logra procesar todo el algoritmo.

8. CONCLUSIONES

La complejidad de un algoritmo se define por medio del número de operaciones elementales necesarias en la resolución de un problema y a su vez las operaciones dependen del número de datos de entrada y de salida. De esta manera la complejidad aumenta linealmente según sea el tamaño de la información de entrada. Por esta razón se partió de un algoritmo pequeño con 15 bytes de entrada, Reed-Solomon (15,9), y posteriormente se replicó el algoritmo para el desarrollo del Reed-Solomon (204,188) con 255 bytes de entrada. El estudio, comprensión y visualización del código Reed-Solomon es más fácil con un código de dimensiones pequeñas.

La decodificación Reed-Solomon utiliza varios algoritmos para su resolución, entre los cuales se encuentra Berlekamp-Massey, estos algoritmos limitan al decodificador la corrección de errores cuando la palabra de información que ingresa al receptor sobrepasa el número de errores permitidos por el algoritmo. Al intentar decodificar una palabra que sobrepase el número máximo de errores no se logra decodificar, llegando la información con errores.

El algoritmo Reed-Solomon (204,188) puede corregir 8 bytes de los 255 bytes que ingresan, cada byte posee 8 bits, por lo tanto tiene la capacidad de corregir 64 bits siempre y cuando sean dentro de los mismos 8 bytes.

La herramienta de programación Matlab permite realizar funciones y algoritmos que dependen de entradas variables o constantes según sea la necesidad del usuario, por tal razón el algoritmo realizado es un genérico para cualquier Reed-Solomon a implementarse. El lenguaje de Matlab es flexible para realizar cálculos avanzados como los utilizados en el codificador y decodificador Reed-Solomon.

La tarjeta utilizada es la Spartan 3E, ya que es una tarjeta con múltiples aplicaciones, fácil de conseguir en el mercado y su valor es accesible para estudiantes. Tarjetas de mayor capacidad tienen costo elevado.

La tarjeta FPGA Spartan 3E es compatible con Matlab, por lo cual es factible realizar la transformación de funciones generadas en Matlab a programas en lenguaje VHDL.

Para el código Reed-Solomon (204,188) se satura la memoria de la tarjeta Spartan 3E, por lo cual no es posible realizar la implementación en dicha tarjeta, es necesario utilizar una tarjeta de mayor capacidad para poder implementarlo físicamente. La comprobación del correcto funcionamiento se logra a través de la simulación presentada. El costo de la tarjeta óptima para la implementación es alto y se debe solicitar su importación directa del fabricante. Se recomienda la tarjeta FPGA Virtex 6.

Dentro de las redes de transporte se poseen tecnología como SDH y OTN, las cuales trabajan con un sistema de detección y corrección de errores RS (255,239) el cual su algoritmo se ha elaborado en el presente proyecto. Por lo tanto el estudio se lo puede implementar sobre las redes de transporte de datos.

REFERENCIAS BIBLIOGRÁFICAS

[1] ARCOTEL, Adopción del estándar para Televisión Digital Terrestre, http://www.arcotel.gob.ec/wp-content/uploads/2015/07/Proyecto-resoluci%C3%B3n-norma-tecnica-tdt.pdf

[2] Asociación Brasilera de Normas Técnicas, "ABNT NBR 15601:2007 Televisión digital terrestre – Sistema de transmisión ISDB-Tb", 2007.

[3] Abramson Norman, "Teoría de la Información y Codificación", Editorial Paraninfo, quinta edición, Madrid, 1981.

[4] Haykin Simon, "Sistemas de Comunicación", Editorial Limusa Wiley, cuarta edición, New York, 2001.

[5] Artés Antonio, Pérez Fernando, "Comunicaciones Digitales", Editorial Limusa, primera edición, México, 2001.

[6] Herstein I. N., "Álgebra moderna", Editorial F. Trillas, S. A., primera edición, México, 1970.

[7] Espitia Jesus, Codificador Reed-Solomon en Software. http://tesis.ipn.mx/bitstream/handle/123456789/11486/44.pdf?sequence=1

[8] Casey Erin, Berlekamp-Massey Algorithm. http://www.math.umn.edu/~garrett/students/reu/MB_algorithm.pdf

[9] Rao Farhat Masood, Adaptive Modulation (QPSK, QAM). https://arxiv.org/ftp/arxiv/papers/1302/1302.7145.pdf

[10] D. Matie, "OFDM as possible modulation technique for multimedia applications in the range of mm waves", Introduction to OFDM, II edition, TUD-TVS. October 1998.

[11] Simpsons P, "FPGA Design: Best Practices for Teambased Design", Editorial Springer, New York, 2010.

Lenín A. Aguirre.- nació en Quito-Ecuador, el 11 de septiembre de 1990. Graduado en el Colegio Experimental "Sebastián de Benalcázar", donde obtuvo el título de Bachiller en Ciencias especialización Físico Matemáticas. En el año 2008 ingresó a la Escuela Politécnica Nacional donde obtuvo el título de Ingeniero Electrónica y Telecomunicaciones. Actualmente es Asistente Técnico en CELEC EP Unidad de Negocio Transelectric en la Subgerencia de Servicios del S.N.I.

Esteban P. Gordon.- nació en Quito-Ecuador, el 10 de noviembre de 1990. Graduado en el Colegio Experimental "Sebastián de Benalcázar", donde obtuvo el título de Bachiller en Ciencias especialización Físico Matemáticas. En el año 2008 ingresó a la Escuela Politécnica Nacional donde obtuvo el título de Ingeniero Electrónica y Telecomunicaciones. Actualmente es Asistente Técnico en CELEC EP Unidad de Negocio Transelectric en la Subgerencia de Servicios del S.N.I.