Implementación de un Codificador y Decodificador Reed Solomon (204,188) en una Tarjeta FPGA con el Algoritmo de Euclides Modificado

Oñate Cristian[1]; Lupera Morillo Pablo[1]

[1]Escuela Politécnica Nacional, Facultad de Ingeniería Eléctrica y Electrónica, Quito, Ecuador

***Abstract*—En este artículo se realiza un análisis de la teoría del código Reed-Solomon, para luego describir específicamente la codificación y decodificación del código Reed-Solomon (204, 188) en VHDL. Posterior a eso, se implementó el código descrito en la tarjeta FPGA Virtex 5 XUPV5-LX110T. El diseño del codificador se lo realizó usando su arquitectura genérica, mientras que para el decodificador se usaron como base los algoritmos de Euclides Modificado, Chien y Forney. Finalmente, se verificó su correcto funcionamiento y se determinó la cantidad de recursos utilizados de la tarjeta al implementar el algoritmo estudiado.**

***Index Terms*— algoritmo de Euclides modificado, corrección de errores, Reed Solomon.**

I. Introduction

En el intercambio de información a través de un canal de comunicaciones poco confiable o ruidoso, es inevitable encontrarse con errores en la recepción, esto es inadmisible si se pretende brindar una comunicación confiable y de calidad entre un emisor y un receptor, por lo que es necesario el uso de técnicas de codificación para el control de los errores (Castiñeira y Farrell, 2012).

En 1960 Irvin S. Reed y Gustave Solomon publicaron un artículo titulado "Polinomial Codes over Certain Finite Fields" presentando la creación del código que lleva sus nombres. Reed Solomon es un código usado para la detección y corrección de errores. La aplicación de este código fue complementada con la invención del algoritmo decodificador creado por Elwyn Berlekamp. Su primera aplicación significativa fue la codificación de imágenes digitales en la sonda espacial Voyager (Serrano, 2004).

En la actualidad tiene un amplio rango de aplicaciones en comunicaciones digitales y es usado para corregir errores en muchos sistemas de comunicación como: Wifi, Wimax (Lujan, Almagro, Cabrera, Suardíaz y Cerdan, n.d.), CDs y sondas espaciales (Bateman, 2003). También se destaca la aplicación del código Reed-Solomon en algunas tecnologías de comunicación digital, como: Digital Video Broadcasting (DVB), Integrated Services Digital Broadcasting Terrestrial (ISDB-T) y en Digital Audio Broadcasting (DAB) (Serrano, 2004).

II. Marco Teórico/Metodología

Los códigos Reed Solomon se representan como RS(n,k), en donde los parámetros definidos se expresan de la siguiente manera (Serrano, 2004):

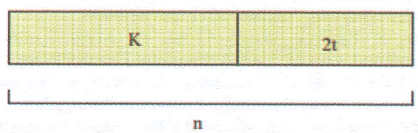

Figura 1 Parámetros del código RS(n,k) (Serrano, 2004).

n: número de símbolos a la salida del codificador (longitud de la palabra código).
k: número de símbolos de información que van a ser codificados.
$2t = n - k$: número de símbolos de paridad añadidos dentro de los n símbolos.
$t = \frac{n-k}{2}$: número máximo de errores que puede corregir el código.

Normalmente se los representa de manera sistemática, eso quiere decir que después de la codificación la palabra original se la puede distinguir claramente en la palabra codificada (Serrano, 2004).

A. *Codificación Reed Solomon*

C. Oñate, Escuela Politécnica Nacional, Quito, Ecuador. Estudiante del Departamento de Electrónica, Telecomunicaciones y Redes de Información DETRI (e-mail: tcristian697@gmail.com).

P. Lupera Morillo, Escuela Politécnica Nacional, Quito, Ecuador. Profesor del Departamento de Electrónica, Telecomunicaciones y Redes de Información DETRI (e-mail: pablo.lupera@epn.edu.ec).

La codificación tiene como función principal generar la palabra código, para esto la codificación sistemática añade n-k símbolos de paridad a la palabra original que ingresa al codificador, así ingresan k símbolos de información y salen n símbolos y esta es la palabra código. Los símbolos de paridad se los calcula multiplicando la palabra original por el polinomio X^{n-k} y este resultado se lo divide para el polinomio generador g(x), el residuo de esta división es el conjunto de los símbolos de paridad que se añade a la palabra original (Castiñeira y Farrell, 2012).

El polinomio generador es un polinomio de grado X^{n-k} sobre el campo de Galois, cuyo valor dependerá del campo sobre el que se esté trabajando. El polinomio generador g(x) de manera general se lo calcula de la siguiente manera (Castiñeira y Farrell, 2012).

$$g(x) = (x + \alpha)(x + \alpha^2)(x + \alpha^3) \dots (x + \alpha^{2t}) \quad (1)$$

Se presenta la descripción matemática de la codificación sistemática.

$$c(x) = m(x).X^{n-k} + [m(x).X^{n-k}] mod\ g(x) \quad (2)$$

m(x): polinomio de la palabra original recibida a la entrada del codificador,
c(x): la palabra codificada (palabra código),
g(x): es el polinomio generador,
X^{n-k} : es un término de grado n-k

B. DECODIFICACIÓN REED SOLOMON

El proceso de decodificación consiste en tomar la palabra enviada por el codificador y recuperar la palabra original. Si la palabra que se recibe tiene errores, el código Reed Solomon se encargará de detectar y corregir los errores que se produjeron en la transmisión, como este proceso es el inverso de la codificación ingresarán n símbolos al decodificador y se obtendrán los k símbolos correspondientes al mensaje original que se envió (Castiñeira y Farrell, 2012). El proceso de decodificación se lo describe a continuación:

- Cálculo del síndrome

La manera más sencilla de calcular el síndrome es evaluar la palabra codificada c(x) en cada una de las raíces del polinomio generador g(x), se necesitan 2t síndromes para poder detectar y corregir t errores, con esto se calculará el polinomio síndrome S(x), que permitirá determinar si la palabra código recibida tienen errores o no (Castiñeira y Farrell, 2012). El cálculo del polinomio síndrome se lo puede expresar matemáticamente de la siguiente manera:

$$S_{i-1} = c(\alpha^i), \qquad i = 1, 2, 3, \dots, 2t \quad (3)$$
$$S(x) = S_0 + S_1X + S_2X^2 + \cdots + S_{2t-1}X^{2t-1} \quad (4)$$

Si el polinomio síndrome es distinto de cero, esto indica que existen errores en la palabra recibida por lo que se deberá iniciar el proceso de detección y corrección de errores, detallado a continuación:

- Detección y corrección de errores

Para la detección de errores es necesario calcular el polinomio localizador de errores y el polinomio magnitud de errores, para esto se pueden usar varios algoritmos como el de Berlekamp, Berlekamp-Masey, Euclides extendido o Euclides modificado, debido a las ventajas que ofrece el algoritmo de Euclides modificado respecto a la facilidad de ejecución y uso de recursos en este proyecto se usa el algoritmo de Euclides modificado el cual se lo ejecuta de la siguiente manera (Serrano, 2004):

- Algoritmo de Euclides modificado

Este algoritmo está basado en el algoritmo de Euclides, el cual es usado para calcular el máximo común divisor de dos factores, sin embargo el algoritmo de Euclides modificado evita el cálculo de los cocientes en cada división, por lo que su ejecución es más rápida y consume menos recursos. A continuación se describen las ecuaciones y sus condiciones iniciales para el uso del Algoritmo de Euclides modificado (Serrano, 2004):

$\mu_{0(x)} = 1$
$\lambda_{0(x)} = 0$
$R_{0(x)} = X^{n-k}$, término de grado $n - k$
$Q_{0(x)} =$ Síndrome

El polinomio magnitud de errores se calcula así (Serrano, 2004):

$$R_i(x) = [\sigma_{i-1}b_{i-1}R_{i-1}(x) + \overline{\sigma_{i-1}}a_{i-1}Q_{i-1}(x)] - X^{|l_{i-1}|}[\sigma_{i-1}a_{i-1}Q_{i-1}(x) + \overline{\sigma_{i-1}}b_{i-1}R_{i-1}(x)] \quad (5)$$

El polinomio localizador de errores se determina de la siguiente manera (Serrano, 2004):

$$\lambda_i(x) = [\sigma_{i-1}b_{i-1}\lambda_{i-1}(x) + \overline{\sigma_{i-1}}a_{i-1}\mu_{i-1}(x)] - X^{|l_{i-1}|}[\sigma_{i-1}a_{i-1}\mu_{i-1}(x) + \overline{\sigma_{i-1}}b_{i-1}\lambda_{i-1}(x)] \quad (6)$$

Los términos $Q_i(x)$ y $\mu_i(x)$ son expresiones complementarias que ayudan al cálculo de $\lambda_i(x)$ y $R_i(x)$.

$$Q_i(x) = \sigma_{i-1}Q_{i-1}(x) + \overline{\sigma_{i-1}}R_{i-1}(x) \quad (7)$$
$$\mu_i(x) = \sigma_{i-1}\mu_{i-1}(x) + \overline{\sigma_{i-1}}\lambda_{i-1}(x) \quad (8)$$

Donde a_{i-1} y b_{i-1} son los coeficientes de mayor grado de los polinomios $R_{i-1}(x)$ y $Q_{i-1}(x)$ respectivamente.

$l_{i-1} = \text{grado}(R_{i-1}(x)) - \text{grado}(Q_{i-1}(x))$;
$\sigma_{i-1} = 1$ si $l_{i-1} \geq 0$,
$\sigma_{i-1} = 0$ si $l_{i-1} < 0$.

La iteración del algoritmo termina cuando $\text{grado}(\lambda_i(x)) > \text{grado}(R_i(x))$ o bien $\text{grado}(\mu_i(x)) > \text{grado}(Q_i(x))$, en ese

momento el polinomio localizador $\sigma(x)$ será $\lambda_i(x)$ y el polinomio magnitud de error $\omega(x)$ será $R_i(x)$ (Serrano, 2004).

Luego de haber ejecutado el algoritmo de Euclides Modificado, se procede con la ejecución del algoritmo de Chien cuya función principal es encontrar la posición del error en la palabra código recibida. Para esto se calculan las raíces del polinomio localizador de errores $\sigma(x)$. La manera más fácil de calcular sus raíces es reemplazando la variable x por todos los elementos del campo finito de Galois, excepto el 0. Si α^i es una raíz del polinomio $\sigma(x)$, la posición del error será $(2^m - 1) - i$, 2^m para este caso es 256, este valor dependerá del campo de Galois que se esté usando para el código (Serrano, 2004).

El último algoritmo a ejecutar es el algoritmo de Forney, este permite calcular el valor del error localizado, mediante la evaluación siguiente formula (Serrano, 2004):

$$e = \frac{\omega(x)}{\sigma'(x)} \qquad (9)$$

- Corrección de errores

Una vez ejecutados los algoritmos de Euclides modificado, Chien y Forney se ha encontrado el polinomio error e(x).

$$e(x) = (\text{valor hallado con Forney})X^{\text{Posicion hallada en Chien}} \qquad (10)$$

Si existe más de un error, cada uno tendrá su valor y posición correspondiente. La corrección de errores se lo hace realizando la operación XOR entre el polinomio error y la palabra recibida (Castiñeira y Farrell, 2012).

$$m(x) = c(x) + e(x) \qquad (11)$$

Este método de codificación es general y puede ser aplicado para códigos Reed Solomon de cualquier magnitud.

C. Descripción en VHDL

Para la descripción del código Reed Solomon (204,188) se usará el lenguaje VHDL, que es un acrónimo de *VHSIC Hardaware Description Languaje*, en donde VHSIC es otro acrónimo de *Very High Speed Integrated Circuit* (Chu, 2006). Este lenguaje es ampliamente usado en la industria y permite realizar una descripción circuitos síncronos y asíncronos (Sanchez, n.d.).

La descripción en VHDL inicia con el diseño un UART (Universal Asynchronous Receiver-Transmitter), el cual usará el estándar RS-232 para la comunicación entre el computador y la FPGA (Chu, 2006).

Para la descripción del codificador Reed Solomon se usará una arquitectura genérica que puede ser usada en cualquier codificador Reed Solomon sin importar los valores de n y k (Sandoval y Fedon, 2008). Esta arquitectura se muestra a continuación:

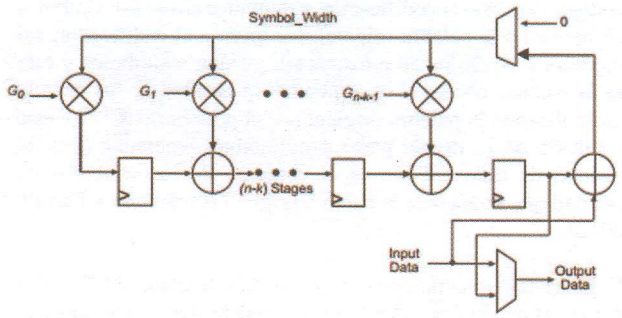

Figura 2 Arquitectura genérica de un codificador Reed Solomon (n, k).

A continuación, se definen los elementos más importantes en la arquitectura genérica y su funcionamiento.

Multiplicación en el campo de Galois.

Suma en el campo de Galois.

Registro de memoria de 8 bits (Flip flop tipo D)

Gn: Coeficientes del polinomio generador g(x)

El circuito inicia su proceso con el ingreso del primer símbolo de la palabra original el cual se sumará con el valor almacenado en el registro n-k-1 que será el valor 0, ya que es el valor inicial en cada registro, este resultado es multiplicado por cada coeficiente del polinomio generador g(x), $G_0, G_1, \ldots, G_{n-k-1}$. El resultado entre el primer elemento y el coeficiente G_0 es almacenado en su correspondiente registro de memoria, mientras que los demás resultados de cada multiplicación en cambio son sumados con el dato almacenado previamente en cada registro, que como se sabe tiene el valor inicial 0, realizada la suma el resultado se almacenará en su registro correspondiente.

Luego ingresará el segundo símbolo y se ejecutará el mismo proceso, el proceso se repetirá hasta que todos los símbolos de la palabra original hayan ingresado, una vez que se han procesado todos los símbolos, los últimos valores almacenados en cada registro se agregarán a la palabra original para obtener la palabra completa codificada.

Una vez descrito el funcionamiento de cada señal, en la Figura 3 se muestra la arquitectura del codificador Reed Solomon (204,188) con UART.

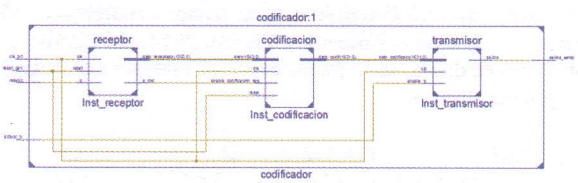

Figura 3 Esquemático del codificador con UART instanciado ISE

El diseño del decodificador presenta una alta complejidad, por lo que se ha dividido su descripción en 3 bloques, el primero

encargado de calcular el polinomio síndrome, el segundo ejecutará el algoritmo de Euclides modificado para hallar los polinomios localizador y magnitud de error, y el último bloque, será el encargado de hallar la magnitud y posición de los errores y realizar la corrección de los mismos, esta arquitectura se la muestra a continuación:

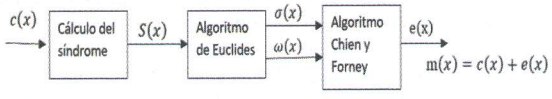

Figura 4 Arquitectura para el decodificador Reed Solomon (n, k)

Para el cálculo del síndrome se usa la arquitectura que se muestra en la Figura 5, que se encarga de evaluar la palabra recibida en el elemento α^i para hallar el coeficiente S_{i-1} del polinomio síndrome. Ya que el polinomio síndrome consta de 16 coeficientes, en VHDL se implementan 16 circuitos iguales cada uno con su respectivo α^i.

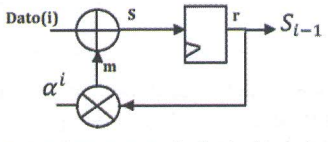

Figura 5 Arquitectura para el cálculo del síndrome en ISE

El bloque del algoritmo de Euclides modificado está implementado mediante la máquina de estados que se muestra en la Figura 6. En la máquina de estados se ejecutan las iteraciones necesarias para hallar los polinomios localizador y magnitud de errores.

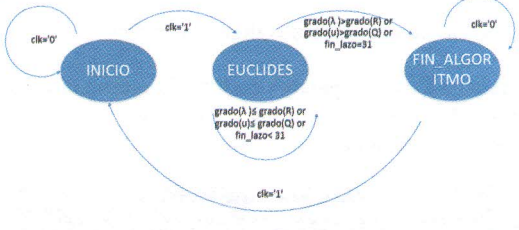

Figura 6 Máquina de estados para la ejecución del algoritmo de Euclides modificado.

Como se explicó anteriormente el algoritmo de Chien detecta la posición del error y esto se logra evaluando el polinomio localizador en un α^i, por lo que su arquitectura se ajusta a lo mostrado en la Figura 7 y será muy similar a la usada para el cálculo del polinomio síndrome.

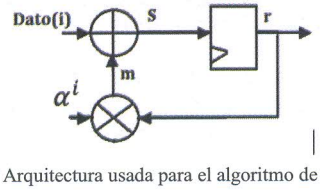

Figura 7 Arquitectura usada para el algoritmo de CHIEN.

Esta arquitectura se repetirá 3 veces en la descripción para evaluar el polinomio localizador de errores, su derivada y el polinomio magnitud de errores. Una vez evaluados los 3 polinomios, se toma el valor calculado después de evaluar el polinomio localizador de errores y si este es cero, se aplica el algoritmo de Forney para calcular la magnitud del error, si no es así entonces no existirá error en esa posición y no será necesario ejecutar el algoritmo, para este caso se asigna el valor 00000000 en esa posición. Este proceso se lo observa en la Figura 8, la cual describe la arquitectura usada para el algoritmo de Forney.

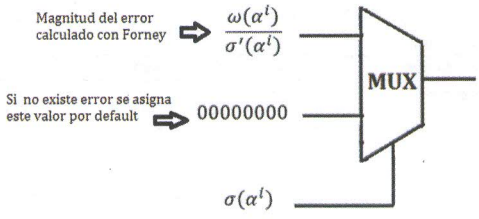

Figura 8 Arquitectura usada para el algoritmo de Forney.

Realizada la descripción en VHDL del decodificador, se instancia con su respectivo transmisor y receptor serial para obtener el decodificador Reed Solomon (204,188) con UART mostrado en la siguiente Figura.

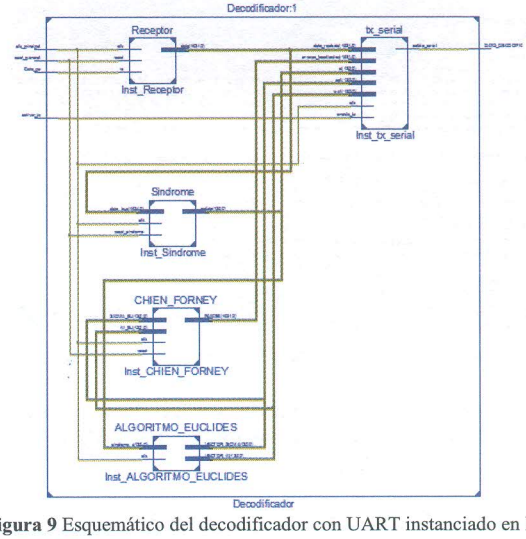

Figura 9 Esquemático del decodificador con UART instanciado en ISE

III. RESULTADOS Y DISCUSIÓN

En esta sección se muestran los resultados obtenidos luego de la implementación en la FPGA del código descrito en VHDL, tanto para el codificador como para el decodificador Reed Solomon (204, 188).

A. Evaluación y pruebas de la implementación del codificador Reed Solomon (204, 188) con UART

En la siguiente tabla se muestran los recursos utilizados para la implementación del codificador Reed Solomon (204,188) en la tarjeta FPGA empleada.

Tabla 1 Recursos utilizados en la tarjeta FPGA XUPV5-LX110T de Xilinx para el codificador RS (204,188).

Resumen de la utilización del dispositivo (valores estimados)			
Utilización lógica	**Usado**	**Disponible**	**Utilización**
Número de registros de Slice	1839	69120	2%
Número de LUTs de Slice	2118	69120	3%
Número de pares de LUT-FF usados completamente	526	3431	15%
Número de IOBs	5	640	0%
Número de BUFG/BUFGCTRLs	1	32	3%

A continuación, se muestran los resultados obtenidos en la interfaz gráfica tras ejecutar las pruebas prácticas del codificador Reed Solomon (204,188) en la FPGA, en este ejemplo se enviaron 188 símbolos con valor 01_H, y se obtiene el dato codificado mostrado en la Figura 10.

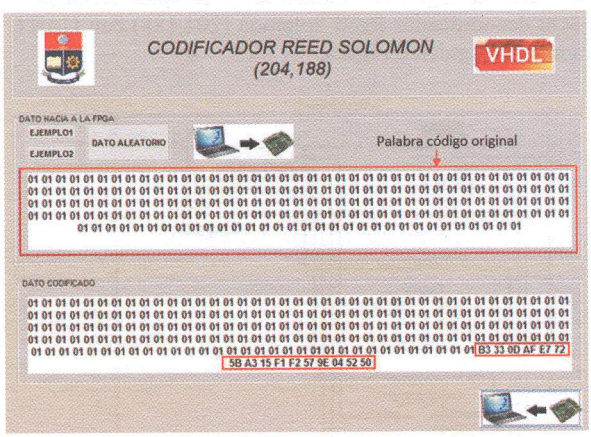

Figura 10 Pruebas del codificador Reed Solomon (204,188) en la FPGA.

En las siguientes figuras se puede ver el dato codificado obtenido en Matlab e ISE respectivamente luego de ingresar 188 símbolos con valor 01_H.

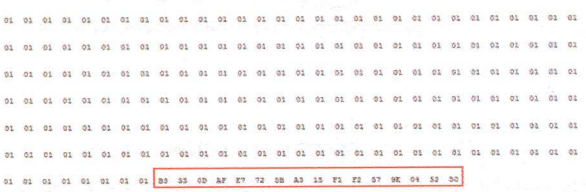

Figura 11 Palabra codificada en Matlab usando el código Reed Solomon (204,188).

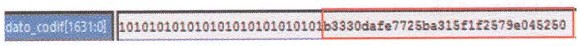

Figura 12 Simulación del codificador Reed Solomon (204,188) en ISE.

Se observa que los símbolos de paridad que se añaden a la palabra original luego de la codificación hecha en la FPGA coinciden con los datos obtenidos en Matlab e ISE, con esto se puede concluir que el código descrito en VHDL cumple con los parámetros de diseño establecidos y se ejecuta de manera correcta la codificación Reed Solomon (204, 188).

B. Evaluación y pruebas de la implementación del decodificador Reed Solomon (204, 188) con UART

Se presenta el cuadro resumido del porcentaje de utilización de la FPGA para el caso del decodificador.

Tabla 2 Recursos utilizados en la tarjeta FPGA XUPV5-LX110T de Xilinx para el decodificador RS (204,188).

Resumen de la utilización del dispositivo (valores estimados)			
Utilización lógica	**Usado**	**Disponible**	**Utilización**
Número de registros de Slice	6121	69120	8%
Número de LUTs de Slice	15531	69120	22%
Número de pares de LUT-FF usados completamente	2807	3431	14%
Número de IOBs	5	640	0%
Número de BUFG/BUFGCTRLs	1	32	3%

La correcta ejecución del decodificador se comprueba comparando la palabra decodificada enviada por la FPGA con la palabra codificada sin errores, retirando los 16 símbolos de paridad. Si las dos coinciden, se habrá comprobado el correcto funcionamiento del decodificador.

Adicional se verificarán los valores del síndrome, el polinomio localizador y magnitud de errores enviados desde la FPGA con los valores obtenidos en la simulación de ISE.

- Pruebas con la palabra código sin errores

Para esta prueba se usa la palabra codificada que se mostró en la Figura 11, la cual será enviada a la FPGA sin insertarle errores, como se muestra a continuación:

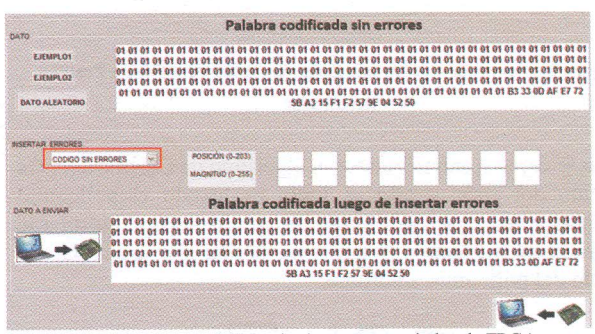

Figura 13 Palabra codificada sin errores enviada a la FPGA.

Como la palabra código enviada a la FPGA no tiene errores, es obvio que el polinomio síndrome y los errores localizados sean cero, lo cual se obtuvo en la decodificación de acuerdo a las figuras 13 y 14.

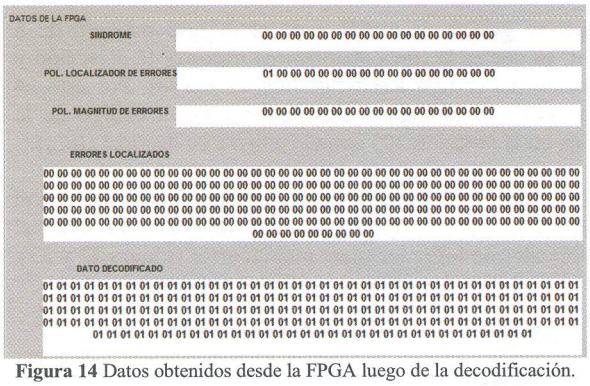

Figura 14 Datos obtenidos desde la FPGA luego de la decodificación.

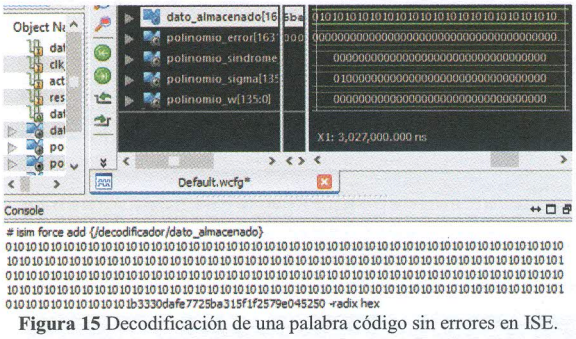

Figura 15 Decodificación de una palabra código sin errores en ISE.

- Pruebas con 1 error

Nuevamente se usa la palabra codificada de la Figura 11, pero esta vez se inserta un error en la posición cero, cambiando el valor 50 por el 35 en hexadecimal, como se muestra en la Figura 15.

La detección y corrección de errores se resume en la tabla 1, el error se localiza efectivamente en la posición cero, pero su valor es 65, esto es porque se realiza la operación XOR entre 65 y 35, para recuperar el valor original 50, esta operación se muestra en la tabla 3 y se la puede también verificar observando las Figuras 16 y 17.

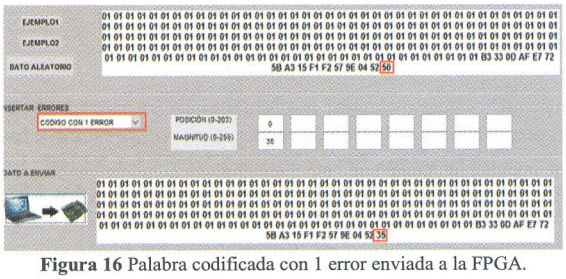

Figura 16 Palabra codificada con 1 error enviada a la FPGA.

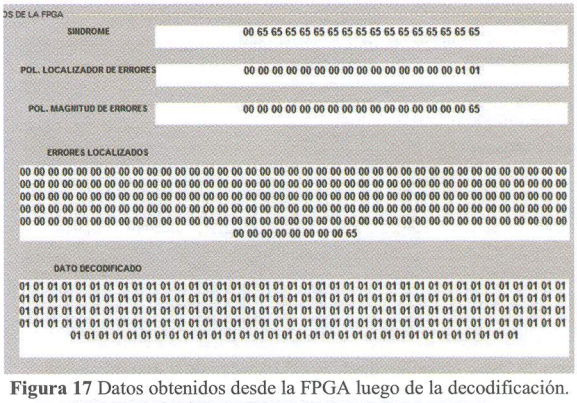

Figura 17 Datos obtenidos desde la FPGA luego de la decodificación.

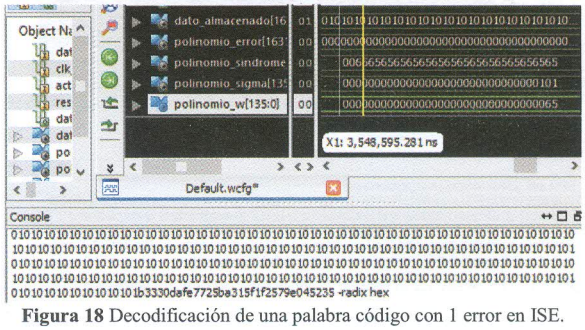

Figura 18 Decodificación de una palabra código con 1 error en ISE.

Tabla 3 Corrección de 1 error

Posición	Valor del dato original	Valor del error ingresado	Valor del error detectado	Valor del dato corregido
0	50	35	65	65 XOR 35=50

- Pruebas con 8 errores

Ahora se ingresan 8 errores para comprobar la capacidad máxima de corrección del código Reed Solomon (204,188). Los procesos de detección y corrección de errores se muestran en las siguientes figuras, mientras que en la tabla 4 se presentan las operaciones realizadas y se puede observar que fue posible la recuperación del dato original.

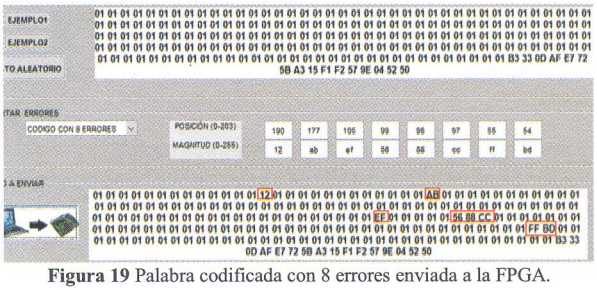

Figura 19 Palabra codificada con 8 errores enviada a la FPGA.

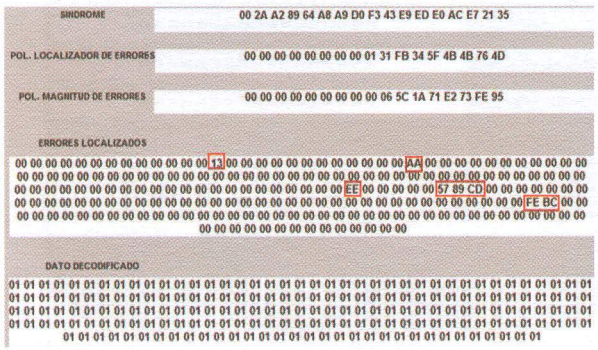

Figura 20 Datos obtenidos desde la FPGA luego de la decodificación.

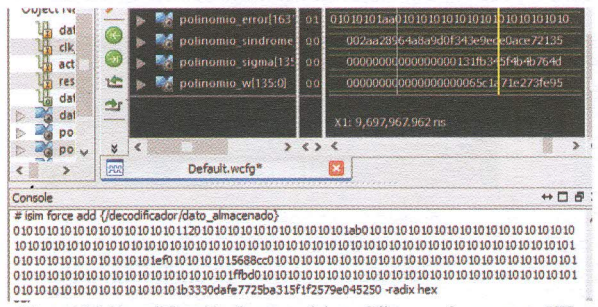

Figura 21 Decodificación de una palabra código con 8 errores en ISE.

Tabla 4 *Corrección de 8 errores.*

Posición	Valor del dato original	Valor del error ingresado	Valor del error detectado	Valor del dato corregido
54	01	BD	BC	BC XOR BD=01
55	01	FF	FE	FE XOR FF=01
97	01	CC	CD	CD XOR CC=01
98	01	88	89	89 XOR 88=01
99	01	56	57	57 XOR 56=01
105	01	EF	EE	EE XOR EF=01
177	01	AB	AA	AA XOR AB=01
190	01	12	13	13 XOR 12=01

- Pruebas con 9 errores

Se presenta la siguiente prueba para observar el comportamiento del decodificador ante la presencia de una cantidad de errores mayor a la capacidad de detección y corrección del codificador Reed Solomon (204,188). En la Figura 22, se muestran los 9 errores ingresados en la palabra codificada.

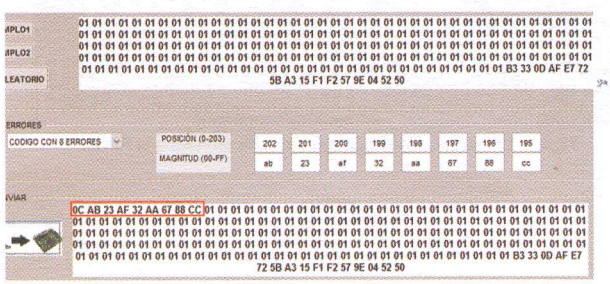

Figura 22 Palabra codificada con 9 errores enviada a la FPGA.

La interfaz gráfica se diseñó para el ingreso máximo de 8 errores, ya que es la capacidad teórica máxima de detección y corrección del código Reed Solomon (204,188). Por esto, el noveno error se lo ingresa directamente en el código de la interfaz, en la posición 203 con el valor 12 en decimal ($0C_H$).

Luego de ejecutar las pruebas con 9 errores se observa que el decodificador no recupera el dato original, ya que el decodificador no logra localizar ningún error (Figura 22 y 23), y por esto, el dato decodificado contiene los mismos errores.

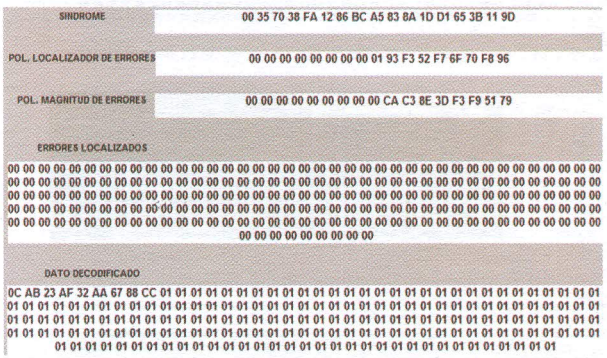

Figura 23 Datos obtenidos desde la FPGA luego de la decodificación

Figura 24 Decodificación de una palabra código con 9 errores en ISE.

Se observa que todas las pruebas realizadas son exitosas, el código diseñado corrige un máximo de 8 símbolos erróneos y recupera la palabra original enviada, con esto se puede concluir que el código descrito en VHDL cumple con los parámetros de diseño establecidos y ejecuta de manera correcta la decodificación Reed Solomon (204, 188).

IV. Conclusion

El codificador y decodificador implementados en este proyecto utilizan una cantidad reducida de recursos en comparación a los disponibles en una tarjeta FPGA Virtex 5 XUPV5-LX110T. En el caso del codificador, el recurso más utilizado son los FFs, alcanzando un porcentaje de utilización del 15%; en cambio, en el decodificador el recurso más utilizado son los LUTs con un 22%.

Con la aplicación del algoritmo de Euclides modificado se reduce la complejidad del decodificador, gracias a la ejecución de operaciones matemáticas más sencillas en los procesos de cálculo del polinomio localizador y polinomio magnitud de errores en comparación a otros algoritmos. Esto además, facilita su programación en VHDL.

El codificador y decodificador Reed Solomon (204,188) presentados en este trabajo, podrán ser usados en futuros

proyectos de mayor complejidad que requieran la detección y corrección de errores.

REFERENCES

[1] A. Bateman., "Comunicaciones Digitales Diseño para el mundo real", España: Marcombo, 2003.

[2] P. Chu, "RTL hardware design using VHDL", Hoboken, N.J.: Wiley-Interscience, 2006.

[3] S. Lujan, S. Almagro, A. Cabrera, J. Suardíaz, F. Cerdan, "Códigos RS y su aplicación a la capa física de 802.16 en FPGAs", Colombia: Universidad Politécnica de Cartagena. Obtenido de: http://repositorio.upct.es/bitstream/handle/10317/2041/crs.pdf;jsessionid=E2390EE86A7EABE6999707C8EDACF9F0?sequence=1.

[4] J. Castiñeira y P. Farrell, "Codificación para el control de errores", Mar del Plata, Argentina: Eudem, 2012.

[5] C. Sandoval, A. Fedon, "Descripción modular de un esquema de codificación concatenado para corrección de errores con programación de hardware", Ingeniare. Revista chilena de ingeniería, 16(3), pp. 310-317, 2008.

[6] M. Serrano, "Análisis de un codificador-decodificador Reed-Solomon.", Universidad de Sevilla. Obtenido de: http://encore.fama.us.es/iii/encore/search/C__SPT.%200912;jsessionid=D7F57127392E53B8338EF44A11E6BEE1?lang=spi, Julio 2004.

[7] M. Sánchez, "Introducción a la programación en VHDL", Universidad Complutense de Madrid. Obtenido de: http://eprints.ucm.es/26200/1/intro_VHDL.pdf,Julio, 2014.

Cristian Oñate Castillo, obtuvo el título de Bachiller en Ciencias especialidad Físico - Matemático en el colegio Nacional Eloy Alfaro en el año 2008, y obtiene el título de Ingeniero en Electrónica y Telecomunicaciones en la Escuela Politécnica Nacional en el año 2017.

Pablo Lupera Morillo, obtuvo el título de ingeniero en Electrónica y Telecomunicaciones en la Escuela Politécnica Nacional en el año 2002 y el título de Ph.D. en ciencias técnicas en la Universidad Estatal de Telecomunicaciones de San Petersburgo en Rusia en el año 2009. Sus áreas de investigación son el comportamiento del canal inalámbrico, técnicas de transmisión aplicadas a la capa física y el diseño de redes de comunicación móvil.