UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
DIRECCIÓN DE POSTGRADO
DOCTORADO EN INGENIERÍA ELÉCTRICA

Tesis Doctoral

MODELO OPTIMIZADO DEL CODIFICADOR REED-SOLOMON *(255,K)* EN VHDL A TRAVÉS DE UN LFSR PARALELIZADO

Cecilia E. Sandoval Ruiz
Ingeniero Electricista
Magister en Ingeniería Eléctrica

Tutor Dr. Antonio S. Fedón Rovira

Valencia, 2013

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
DIRECCIÓN DE POSTGRADO
DOCTORADO EN INGENIERÍA ELÉCTRICA

MODELO OPTIMIZADO DEL CODIFICADOR REED-SOLOMON *(255,K)* EN VHDL A TRAVÉS DE UN LFSR PARALELIZADO

AUTOR: Cecilia E. Sandoval Ruiz
Trabajo presentado ante la dirección de Postgrado
de la Universidad de Carabobo para
optar al título de Doctor en Ingeniería Eléctrica

Valencia, 2013

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERIA
DIRECCIÓN DE POSTGRADO

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERIA
DIRECCIÓN DE POSTGRADO

ACTA DE APROBACIÓN DEL PROYECTO DE TESIS DOCTORAL

Por medio de la presente hacemos constar que el Proyecto de Tesis Doctoral titulado: **"MODELO OPTIMIZADO DEL CODIFICADOR REED-SOLOMON (255,K) EN VHDL A TRAVÉS DE UN LFSR PARALELIZADO"**, presentado por la ciudadana **Cecilia E. Sandoval Ruiz, portadora de la cédula de identidad nro. V-14.470.830**, alumna regular del **PROGRAMA DEL DOCTORADO EN INGENIERÍA, ÁREA DE ELÉCTRICA**, reúne los requisitos exigidos para su aprobación.

El **Dr. Antonio S. Fedón Rovira**, aceptó la tutoría de esta Tesis Doctoral.

En Valencia, al catorce días del mes de marzo del año Dos mil Doce.

Por la Comisión Coordinadora:

REPUBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD DE CARABOBO
DOCTORADO EN INGENIERIA
AREA DE ESTUDIOS DE POSTGRADO DE INGENIERIA

Dr. Francisco Arteaga
Coordinador del Programa

Dr. Carlos Villanueva
Miembro

Dr. Demetrio Rey Lago
Miembro

Dr. Alfonso Zozaya
Miembro

Dr. Antonio Fedón
Miembro

FACULTAD DE INGENIERÍA NAGUANAGUA SECTOR BARBULA - Teléfonos Directos: (0241) 8672829 / 8674268 - 8678885 EXT 102. FAX - (0241) 8671655 http://postgrado.ing.uc.edu.ve

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
DIRECCIÓN DE POSTGRADO
DOCTORADO EN INGENIERÍA ELÉCTRICA

VEREDICTO

Nosotros, Miembros del Jurado designado para la evaluación del Trabajo de Grado titulado: MODELO OPTIMIZADO DEL CODIFICADOR REED-SOLOMON *(255,K)* EN VHDL A TRAVÉS DE UN LFSR PARALELIZADO, presentado por: Ing. Cecilia E. Sandoval Ruiz para optar al Título de Doctor en Ingeniería Eléctrica estimamos que el mismo reúne los requisitos para ser considerado como: Aprobado

Dr. Alfonso J. Zozaya S.
C.I.: 8066442

Dr. Demetrio J. Rey Lago
C.I.: 7127552

Dr. Dimas Mavares Terán
C.I.: 9618754

Valencia, 2013

A mi familia por construir en mí,
Los principios que han sido la base para el logro de cada una de mis metas

AGREDECIMIENTOS

A DIOS todopoderoso, que con su infinita luz ha guiado mi camino en cada momento, orientando mis pensamientos y bendiciendo nuestras vidas. Gracias DIOS, por estar siempre conmigo.

A la Santísima Virgen y a San Judas Tadeo, que interceden ante DIOS nuestro Señor por el feliz éxito de nuestros proyectos y nos enseñan el canal de la oración, para concientizar la sabiduría infinita.

A mi familia, que me han enseñado el poder de la Fe, para alcanzar con nobleza las metas, que han sido motivación, enseñanza, ejemplo y apoyo incondicional. Cada uno de ustedes le ha aportado un valor incalculable a la filosofía con la cual he abordado este maravilloso reto.

A mi tutor, el Dr. Antonio Fedón por aceptar ser parte de este proyecto, su apoyo y paciencia, orientando esta investigación.

Al comité de mi tesis doctoral y jurados: Dr. Alfonso Zozaya, Dr. Demetrio Rey Lago y Dr. Dimas Mavares, gracias por su asesoría y valiosas recomendaciones, y a todos los profesores de la Universidad de Carabobo que han sido parte de mi formación y han compartido sus conocimientos y enseñanzas conmigo.

A todos quienes han aportado en mi crecimiento humano, intelectual y profesional, estoy segura que DIOS ha colocado en mi camino a los seres maravillosos de quienes puedo aprender cada día para evolucionar en el especial diseño que construimos todos llamado vida.

Gracias….

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

MODELO OPTIMIZADO DEL CODIFICADOR REED-SOLOMON *(255,K)* EN VHDL A TRAVÉS DE UN LFSR PARALELIZADO

Autor: Cecilia E. Sandoval Ruiz
Tutor: Antonio S. Fedón Rovira
Fecha: 2013

En esta investigación se presenta un modelo eficiente de codificador Reed Solomon RS(*n,k*), para su descripción usando VHDL (*VHSIC hardware description language)*, bajo la filosofía de sistemas reconfigurables. El principal componente, el multiplicador sobre campos finitos de Galois GF(2^m) en base polinómica, ha sido modelado a través de un estructura concurrente de realimentación lineal LFCS (del inglés *Linear Feedback Concurrent Structure)*, cuyas ecuaciones son generadas a partir de la interpretación del circuito LFSR - *Linear Feedback Shift Register*. La metodología empleada inicia con la descripción VHDL de los componentes, aplicando técnicas de optimización de diseño. Seguidamente, se han simulado el comportamiento de los diseños usando ModelSim XE III 6.3c, y estudiado los reportes generados con la herramienta de desarrollo IDE Xilinx 11, para una interpretación de eficiencia del diseño. Una vez alcanzado un diseño con buenas prestaciones, se han obtenido las ecuaciones del modelo optimizado. Entre los resultados más destacados, se tienen las ecuaciones que soportan el modelo del CESR-Codificador RS(255,k) Eficiente para Sistemas Reconfigurables, a través de un procesamiento concurrente del componente multiplicador y un ahorro de recursos de hardware en el sistema, así como un menor consumo de potencia reportado a través del *XPower Analizer*. Finalmente, se reconoció la auto-similitud entre el componente de reducción modular del multiplicador y el generador de símbolos de redundancia del codificador Reed Solomon, la cual fue aplicada para la propuesta del modelo de codificador RS concurrente. Esta investigación generó un aporte científico por la interpretación del modelo a nivel de estructuras circuitales, un aporte tecnológico por una descripción eficiente orientada a sistemas reconfigurables de hardware y una propuesta de expansión de los resultados acá alcanzados, para concatenación de códigos.

Palabras Clave*: Multiplicador GF(2^m), LFSR, Codificador Reed Solomon, modelo concurrente, optimización de recursos.*

CAPÍTULO I

INTRODUCCIÓN

En este capítulo se presenta un breve diagnóstico de las tendencias actuales que permiten contextualizar la investigación, la perspectiva del problema a solucionar y los argumentos que sustentan el desarrollo de codificadores Reed Solomon, bajo el presente enfoque. En base a todo esto se plantean los objetivos y aportes de la investigación.

CONTEXTO DE LA TESIS DOCTORAL

En las investigaciones en el área de sistemas de comunicación de datos, se ha encontrando, en los últimos años, el desarrollo de conceptos como Radio Cognitivo (Fette, 2009) y Sistemas de Radio definido por Software (SDR, del inglés *Software Defined Radio*). Estos novedosos conceptos tienen un punto en común, al proponer la reconfiguración de la arquitectura y parámetros del sistema, para adaptar las funciones de los módulos de forma dinámica. Lo que ha motivado la indagación en sistemas de comunicación adaptativos y en la tecnología de soporte para diseños parametrizables y reconfigurables.

Resulta de interés los avances en sistema de comunicaciones, con características adaptativas en los esquemas de modulación y codificación, que pueden ser ajustados dinámicamente a las condiciones del canal para mejorar el rendimiento del sistema (Noordin, Ali, Ismail, & Jamuar, 2004). Donde se observa que estas implementaciones de sistemas con características adaptativas están basadas en arquitecturas *Very Large Scale Integration* – VLSI (Mora, 2008), con una tendencia hacia la ingeniería basada en dispositivo reconfigurables.

Los actuales diseños están siendo implementados sobre estos dispositivos de nueva tecnología por su flexibilidad (Atieno, Allen, Goeckel, & Tessier, 2006), esto facilita la adaptabilidad de los productos finales, aspecto importante en los procesos continuos de actualización y optimización de diseños. Motivo por el cual, el diseñador debe cumplir con la sintaxis del lenguaje estándar de descripción de hardware VHDL (Mora, 2008), lo que a su vez ofrece portabilidad del diseño a diversas plataformas o gamas de dispositivos. Así pues se han tomado en cuenta los SoPC (System on Programmable Chip), siendo los FPGA (Field Programmable Gate Arrays) aquellos con mayor escala de integración, representada por su equivalente en compuertas.

Adicionalmente, una característica de interés de los FPGA, es que los más recientes de estos dispositivos admiten modificaciones en tiempo de ejecución, como lo destacan las casas fabricantes. Esta capacidad denominada *Reconfiguración Parcial Dinámica* (Astarloa, 2005), soporta cambios en la configuración de componentes y resulta idónea para el diseño de los módulos configurables de los sistemas de comunicaciones. Por lo que los FPGAs se perfilan como dispositivos de soporte válidos para las aplicaciones avanzadas con parámetros ajustables.

Un módulo que por sus características, resulta de interés, para el diseño con esta tecnología, corresponde a los codificadores Reed Solomon, tal como lo presentan investigaciones previas (Mursanto, 2006). Así mismo, los diseños de codificadores RS encontrados en la literatura presentan sus avances en reducción del consumo de recursos

de hardware, tanto los presentados por parte de fabricantes de dispositivos (Altera, 2011; Lattice, 2005; Xilinx, 2011), bajo la modalidad de módulos de propiedad intelectucal – *IPCores*, como aquellas investigaciones en el área de codificación de canal (Jinzhou, Xianfeng, Zhugang, & Weiming, 2012). Lo que ha estimulado a la comunidad científica y tecnológica a investigar en el desarrollo de metodologías de diseño acordes con las tecnologías disponibles, para proponer codificadores RS cada vez más eficientes.

Estos trabajos demuestran que la optimización del codificador RS está sustentada sobre el diseño eficiente de sus componentes. En este orden de ideas, se encontraron diseños de multiplicadores de base polinomial sobre campos finitos de Galois – GF (Peter & Langend, 1962), orientados a su implementación paralelizada sobre dispositivos de hardware reconfigurable. Destacando que este componente es el más importante al momento de obtener aceleración de hardware en los códigos RS, por lo que implementaciones rápidas de la aritmética de los cuerpos GF(2^m) han sido intensamente estudiadas a lo largo de estos últimos años (Climent, Crespí, & Grediaga, 2008).

Todo este panorama, permite evidenciar una tendencia en el campo de investigación de diseños de codificadores RS y sus componentes usando FPGAs como dispositivos de soporte para la implementación, donde se exponen las alternativas de desarrollo que se han abordado hasta el momento para optimizar su eficiencia, bajo la descripción del hardware a través de VHDL.

1.1. PERSPECTIVA DEL PROBLEMA

Luego de revisar las tendencias actuales en el diseño de codificadores RS sobre hardware, se logra observar la vigencia de esta línea de investigación. Enfatizando que si bien los desarrollos previos han producido avances en el proceso de optimización, no se ha encontrado un método de diseño basado en un modelo matemático-lógico, el cual se encuentre orientado hacia la plataforma disponible. Lo que se traduce en la necesidad de un modelo en VHDL para la descripción de un Codificador RS Eficiente orientados a Sistemas Reconfigurables – CESR.

En el marco de este problema, en sentido general, se han desarrollado investigaciones donde se establecen técnicas y métodos para optimizar los diseños en VHDL (Anderson, 2005; Huang, 2003; Gustavo Sutter, 2005), todas estas orientadas hacia la implementación de módulos para procesamiento de datos, usando tecnología FPGA. Entre las técnicas propuestas se presenta la paralelización de algoritmos, distribución eficiente de las señales, entre otras (G Sutter & Boemo, 2007), las cuales se consideraron como alternativa de solución en la presente investigación.

En el caso específico del diseño de codificadores Reed Solomon, dada su complejidad de implementación es apropiada la utilización de estas técnicas de optimización para descripción de hardware. Siendo una alternativa de solución para cubrir la demanda en cuanto a capacidad de cómputo y velocidad de los codificadores RS (Sandoval, 2007). Igualmente, se observa que aun cuando se vienen implementando estos diseños sobre dispositivos de hardware reconfigurable, generalmente son descritos bajo algoritmos secuenciales, estando su salida siempre en función del número de ciclos requeridos, lo que demuestra la falta de adaptación del método de diseño a la tecnología actual.

Por otra parte, las optimizaciones alcanzadas en las diversas versiones de codificadores RS son presentadas como casos particularizados, lo que se puede interpretar como esfuerzos aislados, desde diferentes enfoques. La propuesta de un modelo generalizado que integre diversas técnicas para optimización de hardware resulta un aporte importante para generar el código de configuración, resultando un novedoso método de diseño.

Para atender a esta problemática se requiere proponer avances desde diversos aspectos en el diseño, a nivel de filosofía de diseño, compilación de técnicas de optimización, interpretación y generalizaciones pertinentes. En la tabla 1.1 se resumen los aspectos que componen el problema, la alternativa de solución propuesta y la justificación correspondiente.

Tabla 1.1. Problemas detectados en el área de investigación

Problema	Alternativa de Solución	Justificación
No se cuenta con un modelo para códigos en VHDL de codificadores RS(n,k) en hardware libre, los existentes son bajo la modalidad de IPCores ó bloques del System Generator de Matlab, como es el caso del Reed-Solomon Encoder 8.1 presente en el LogiCORE IP.	Se propone el diseño y generación de los códigos VHDL para el codificador RS genérico, bajo un modelo descriptivo de hardware. Con lo que se permita actualizar los módulos.	Permitirá desarrollar modelos en VHDL del codificador RS, que podrán tener optimizaciones e hibridaciones en sistemas de comunicaciones, partiendo de los códigos VHDL generados.
Contestar a la interrogante: ¿Es posible a través de un nuevo modelo obtener menor consumo de recursos hardware en aplicaciones de alta complejidad computacional?	Realizar un estudio de los reportes de consumo de hardware del diseño propuesto, para establecer el contraste de eficiencia del codificador RS(n,k) aplicando el modelo concurrente.	Alcanzará mayores niveles de eficiencia en el codificador RS, lo que se puede traducir en módulos con mayor rendimiento en su sistema de alimentación.
La necesidad de un modelo para la descripción en VHDL de los multiplicadores concurrentes en campos finitos de Galois y del codificador Reed Solomon.	Desarrollar un modelo en VHDL de un multiplicador GF, basado en la paralelización del algoritmo, a través de un circuito concurrente.	Soporte para un multiplicador GF que rinda prestaciones en la implementación del codificador RS y otras aplicaciones.
No se ha encontrado un modelo en VHDL del codificador Reed Solomon Paralelizado, para aplicaciones RS-PC y concatenadas	Sistematizar el modelo desarrollado para la arquitectura del multiplicador, aplicándolo como modelo alternativo para el codificador RS.	Se contará con el modelo matemático para un codificador RS concurrente, generado a partir de la aplicación del modelo de la estructura LFSR concurrente.

Definición del Objeto de Estudio y Formulación de la Problemática

De todo lo anterior, se ha definido como objeto de estudio al codificador Reed Solomon, el cual está basado en una arquitectura circuital para la generación de símbolos de redundancia, dada por una estructura de registro desplazamiento con realimentación lineal – LFSR (del inglés, *Linear FeedBack Shift Register*) y su componente principal, que es el multiplicador en campos finitos de *Galois* GF(2^m). Así pues, considerando que la naturaleza del codificador Reed Solomon es hardware, resulta pertinente su estudio bajo la filosofía de configuración de hardware.

Una vez definido el objeto de estudio, se encontró que no existe un modelo que describa de forma concurrente el circuito LFSR orientado a sistemas reconfigurables, es decir, un

modelo que esté adaptado a la tecnología disponible actualmente. Las recientes investigaciones apuntan hacia paralelización de las aplicaciones que usan LFSR, o bien hacia el desarrollo de modelos de NLFSR (del inglés, *No Linear Feedback Shift Register*), que son aplicados a criptografía, mas no se encontró un modelo del NLFSR completamente paralelo desde el enfoque acá requerido.

Siendo el circuito LFSR el componente básico del multiplicador en algebra de Galois y del codificador Reed Solomon, desarrollar un modelo concurrente de éste, como una 'Estructura Concurrente de Realimentación Lineal', permite la paralelización del componente y optimización del diseño, a fin de ofrecer una solución a la situación expuesta.

MOTIVACIÓN Y JUSTIFICACIÓN

Al momento de seleccionar el objeto de estudio para la presente investigación se han considerado factores de relevancia y vigencia tecnológica. Teniendo presente que si bien existen actualmente alternativas de codificación con buenas prestaciones, es el código Reed Solomon uno de los más competitivos (Gianni, Claudio, & Corteggiano, 2007). Esto en contraste con los Turbo Códigos, los cuales presentan ventajas en el dominio de transmisión de datos, pero se ven limitados para transmisión de voz y video, por la demanda de cómputo en las etapas del intercalador/des-intercalador en tiempo real.

Es en tal sentido, que el codificador Reed Solomon encuentra su vigencia tecnológica de acuerdo a las prestaciones y grado de complejidad del decodificador, característica que lo hace competitivo entre las alternativas de codificación de canal, encontrando necesaria una investigación acerca del modelado y optimización de éste, que permita describir el codificador RS a nivel de hardware.

Igualmente es importante destacar que actualmente, se adelantan proyectos de comunicaciones satelitales y modelaje matemático, en los cuales se requiere el diseño de

codificadores con alta eficiencia. Siendo los codificadores Reed Solomon aplicados en comunicaciones satelitales y estando el diseño de dispositivos enmarcado entre las necesidades de investigación recientes (Oncti, 2012), se puede definir esta propuesta como oportuna.

Por otra parte, se han considerado los avances en hibridación de los códigos RS en estructuras de códigos Turbo Producto (Sandoval, 2008) ó concatenación con otros en arreglos paralelos y seriales (Sandoval & Fedón, 2008a), que conlleva a la mejora en el desempeño del código final. Siendo necesaria la optimización del diseño de los componentes constitutivos del código compuesto, a fin de obtener una mayor velocidad de procesamiento. De allí la importancia del estudio y optimización del codificador RS(n,k), componente básico de estos esquemas híbridos.

Así lo señala el análisis presentado en el informe BM087 del ITU titulado; G.gen: Comparison of simulation results for different Coding Techniques (Uncoded, Reed-Solomon, Reed-Solomon plus Trellis and Reed-Solomon plus Parallel Concatenated Convolutional Codes) for G.992.1.bis and G.992.2.bis (Torres, 1999), donde se establece una relación de códigos concatenados y su desempeño, siendo esta técnica de codificación la más cercana al límite establecido por C.E.Shannon, a partir de lo cual se puede evidenciar la importancia y vigencia de los RS(*n*,*k*) en el área de codificación de canal para transmisión de datos.

Una de las aplicaciones de los codificadores RS(*n*,*k*) más interesantes por su arquitectura, corresponde a los *Reed–Solomon Product-Code* (RS-PC) (Chang, Shung, & Lee, 2001), (C. Kim, Rhee, Kim, & Jee, 2010). Un ejemplo de éste, puede ser un arreglo compuesto por un codificador horizontal RS(182,172) para el procesamiento de las filas y un codificador vertical RS(208,192) para el procesamiento de las columnas, como el presentado en la figura 1.1.

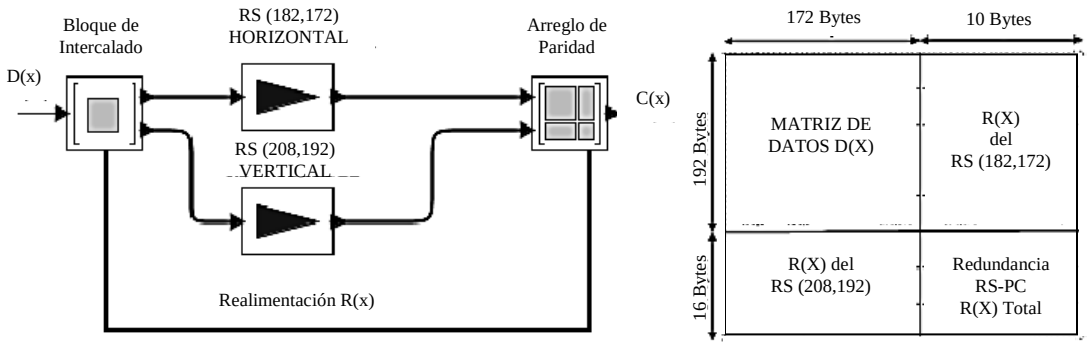


Figura 1.1. Esquema y Matriz de Códigos Producto Reed Solomon (RS-PC)

Este esquema permite observar que la codificación realimentada depende de los resultados de los símbolos de redundancia generados por la primera codificación. Esto demanda la implementación de un codificador altamente eficiente en cuanto a velocidad de procesamiento, siendo una arquitectura válida la aplicación de un LFSR concurrente. El modelo de codificador acá propuesto será útil en estas aplicaciones, ya que permite mejorar el desempeño del RS-PC, lo que justifica la optimización de los codificadores RS que lo componen.

De modo que, este desarrollo podrá servir de insumo para futuras aplicaciones de codificadores híbridos con características adaptativas, basados en la capacidad de reconfiguración dinámica de los FPGAs y las características que estos dispositivos presentan (Astarloa, 2005). Resultando altamente recomendable su tratamiento en forma modular. Todo esto con el fin de reducir el tiempo de desarrollo y optimizar el desempeño de las etapas de los sistemas compuestos por codificadores concatenados en sus diversas combinaciones y aportando un cambio de paradigma en el diseño.

Así mismo es válido señalar, que si bien el objetivo de generar modelos concurrentes de aplicaciones como el codificador RS ha sido un problema que se ha mantenido en el tiempo, también es cierto que las tecnologías disponibles representaban una limitación para su desarrollo, sin embargo, en este momento se ha encontrado una tecnología de

soporte para diseños de alta demanda de cómputo, la adopción de la tecnología FPGA continúa creciendo y la convierte en la solución pertinente para el desarrollo propuesto. Especialmente por las operaciones soportadas en la configuración de hardware en VHDL, tal es el caso del operador denominado '*concatenación*', el cual permite ordenar estructuras sin necesidad de una señal de reloj para su manejo

Por todo lo anterior, resulta lógico proponer el desarrollo de un modelo de codificador Reed Solomon para sistemas reconfigurables, dentro de la línea de codificación del área de teoría de la información, estando plenamente justificada en la necesidad de ofrecer ecuaciones de soporte para la generación del código VHDL del codificador RS, para su implementación sobre hardware con altos niveles de paralelismo y eficiencia de utilización de recursos de hardware. Esto por medio de una combinación estratégica de conocimientos de teoría de codificación, fundamentos y modelos matemáticos, con los métodos en el área de desarrollo de hardware.

Se ha considerado la hipótesis, que un codificador RS(*n*,*k*) basado en un modelo de componentes LFSR concurrente, aplicado en los multiplicadores GF(2^m), puede alcanzar prestaciones iguales o superiores a los sistemas diseñado sobre los modelos actuales.

1.2. OBJETIVOS DE LA TESIS DOCTORAL

El objetivo general de la tesis es la obtención de un MODELO OPTIMIZADO DEL CODIFICADOR REED-SOLOMON (*n*,*k*) EN VHDL, A TRAVÉS DE UN LFSR PARALELIZADO, esto a través del estudio de eficiencia de los codificadores RS diseñados a partir de multiplicadores GF(2^m) concurrentes, a fin de proponer una solución al consumo de recursos, desde una filosofía de diseño orientado a sistemas reconfigurables sobre dispositivos FPGA.

1.2.1. OBJETIVOS ESPECÍFICOS

El modelo se desarrolló a partir del cumplimiento de los siguientes objetivos:

- ✓ Diseñar un codificador RS(255,*k*) modular con componentes concurrentes en VHDL, para su optimización.
- ✓ Interpretar los reportes de síntesis del codificador RS (*255,k*) diseñado, a través de la herramienta ISE 11 de Xilinx, para análisis de eficiencia.
- ✓ Obtener el modelo concurrente del multiplicador en campos finitos de *Galois* GF (2^m), a partir del circuito LFSR paralelizado.
- ✓ Sistematizar el modelo concurrente para el codificador Reed Solomon, a partir de la correspondencia entre las estructuras circuitales.

1.3. ALCANCE DE LA INVESTIGACIÓN

La investigación comprende una fase teórica, donde se analizan los desarrollos científicos en el área de optimización de codificadores, multiplicadores y módulos LFSR, modelos matemáticos, arquitectura y comportamiento, generando un soporte para el tratamiento en la etapa de diseño y modelado.

Así como una fase de investigación aplicada, la cual consta de la descripción en VHDL del codificador RS(*255,k*), con el parámetro *k* ajustable y el desarrollo del modelo basado en multiplicadores concurrentes, validado por el análisis de eficiencia del diseño. Es importante destacar que solo se abordan las configuraciones de 8, 16 y 32 símbolos de redundancia, por tratarse de los protocolos más empleados en aplicaciones de comunicación y contar con referencias para la comparación del desempeño de estos con trabajos previos. El tamaño máximo de los símbolos del codificador será de 8 bits, cuyos resultados pueden extenderse a campos de valores $m>8$.

Destacando que en esta investigación, se mantiene el interés en ofrecer a la comunidad científica un modelo de codificador Reed Solomon con prestaciones competitivas. Para lo cual se deben considerar diversos aspectos, como la velocidad de respuesta, el consumo de recursos hardware y consumo de potencia. Empleando para ello el análisis de la relación entre área y desempeño de los multiplicadores GF(2^m) ofrecido por (Morales-Sandoval, Feregrino-Uribe, Cumplido, & Algredo-Badillo, 2009).

De esta manera se presentan como productos finales: los códigos en VHDL de los componentes diseñados, la simulación del comportamiento de estos, el análisis de utilización de recursos, las ecuaciones para la descripción en VHDL de los componentes de interés y una propuesta de codificador paralelizado, probado por medio de un caso de estudio particular.

ESTRUCTURA DEL DOCUMENTO

Esta tesis doctoral ha sido estructurada iniciando con una perspectiva general del problema, en el cual se introducen las tendencias en el área de investigación, se plantea la necesidad de un modelo orientado a sistemas reconfigurables para la descripción de un codificador Reed Solomon, con altas prestaciones, definiendo a partir de allí los objetivos y aportes que se asocian a la presente.

Seguidamente, se presentan una serie de antecedentes en el área, que sirven de soporte para la etapa de diseño, donde se desarrollan conceptos asociados a los componentes del codificador Reed Solomon y los métodos de configuración del hardware.

La etapa de diseño, modelado y desarrollo es documentada, a través de la descripción de los métodos, técnicas y procedimientos empleados. En el tercer capítulo, se presentan la operacionalización de variables, al igual que las fases de investigación y desarrollo, con especial énfasis en los criterios para la obtención de ecuaciones del modelo y configuración del hardware en lenguaje VHDL.

Posterior a ello, en el cuarto capítulo se presentan los resultados de los diseños, basados estos en el modelo propuesto, acompañados de las simulaciones de comportamiento, el análisis de consumo de recursos de hardware, velocidad de respuesta y consumo de potencia dinámica asociada al diseño, cumpliendo así con la validación de éste. Para realizar la disertación en los tópicos de interés con respecto a las variables definidas como indicadores de eficiencia del modelo propuesto, se contrastan los resultados con previos diseños y optimizaciones en los codificadores estudiados, que permite enunciar las ecuaciones del modelo desarrollado, para de esta manera arribar a los postulados.

Finalmente, se presentan las conclusiones obtenidas de la investigación y evaluación del modelo desarrollado, planteando a su vez las recomendaciones para futuros trabajos que pueden encontrar un fundamento importante en este desarrollo, a fin de avanzar sobre la línea de investigación de optimización de codificadores Reed Solomon y sus versiones para hardware configurable.

CAPÍTULO II

CODIFICADOR RS (N,K) ORIENTADO A HARDWARE RECONFIGURABLE

En este capítulo se desarrollan los conceptos asociados al modelo del codificador Reed Solomon, para su descripción en VHDL; a través de las características estructurales y de comportamiento de sus componentes, con base en trabajos previos, a la vez que se enuncian los modelos matemáticos que sirven de soporte para el diseño.

2.1. INTRODUCCIÓN

Entre los antecedentes consultados se encuentran un conjunto de tesis doctorales convergentes con el tema de estudio, estos han demostrado en todo momento la vigencia y el interés científico en las optimizaciones de codificadores RS, orientadas hacia el análisis de incidencia de los parámetros *n,k*-característicos del código (Allen, 2008), de allí nace el interés por desarrollar el código VHDL abierto y genérico para la descripción de codificadores RS(*n,k*), con parámetros ajustables. Donde se profundiza en los aspectos que determinan el consumo de potencia dinámico, asociados a la metodología de diseño para dispositivos FPGAs (Gustavo Sutter, 2005), (Todorovich, 2006), a fin de aplicar las técnicas correspondientes.

En cuanto al multiplicador en campos finitos, la comunidad científica presenta un conjunto de modelos existentes, considerando tanto los secuenciales como combinacionales, tomando las optimizaciones de multiplicadores GF (G. C. Ahlquist, Nelson, & Rice, 2002) como referencia para el presente diseño, con especial interés en el diseño de los módulos LFSR, para la generación de secuencias pseudo-aleatorias (Dubrova, 2008), (Pérez, 2009), la importancia de estos módulos y sus aplicaciones (Peralta, 2005), (Alvarez, 2005).

FUNDAMENTOS CIENTÍFICOS

La presente investigación se encuentra fundamentada en trabajos que relacionan principios, modelos y optimizaciones, de donde se pueden identificar ejes teóricos asociados al codificador RS y al multiplicador en campos finitos; los cuales han sido estudiados aplicando el principio de correspondencia de Bohr, dada la similitud entre la arquitectura de estos módulos. El esquema 2.1 presenta la relación entre los trabajos previos, definidos por autor, que soportan el enfoque teórico del codificador RS.

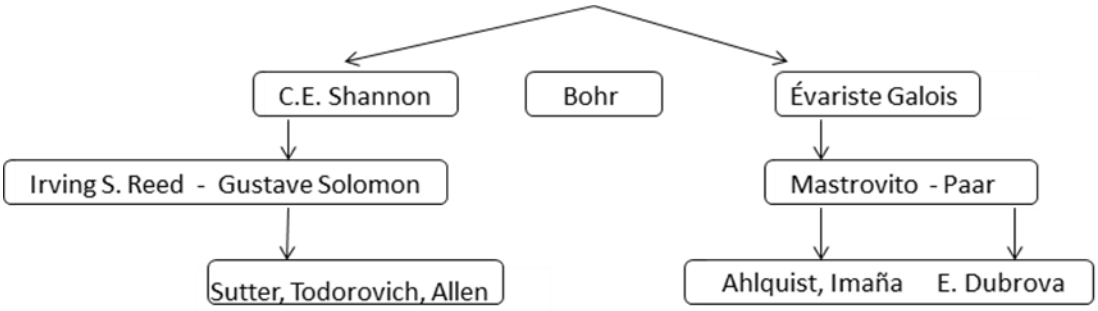


Figura 2.1. Fundamentos de Soporte del Modelo del codificador RS

El codificador RS(*n*,*k*) cumple con objetivos de desempeño que son establecidos a la luz de la teoría de información y codificación, estudiada por *C.E. Shannon*, sobre estos se fundamenta el diseño matemático - conceptual desarrollado por *I. Reed y G. Salomon*, a partir del cual se han propuesto implementaciones prácticas y ha sido aplicado por

investigadores en el área, presentando avances en implementaciones con optimización en el consumo de potencia (Allen, 2008; Gustavo Sutter, 2005; Todorovich, 2006).

De forma paralela, el multiplicador GF se encuentra sustentado en la teoría de campos finitos desarrollada por *Evariste Galois*, de donde se deriva el concepto de los multiplicadores en campos finitos GF y los modelos propuestos por *Mastrovito-Paar* (Halbutogullari & Koc, 2000; Paar, 1996), en base a los que se presentan aportes en el diseño de multiplicadores GF para tecnología FPGA (G. Ahlquist, Nelson, & Rice, 1999), (Deschamps, Imaña, & Sutter, 2009; Dubrova, 2008).

2.2. CONCEPTUALIZACIÓN DEL CODIFICADOR RS CONFIGURABLE

En esta sección se abordarán los conceptos de soporte para el codificador RS configurable, a fin de interpretar los puntos de interés para el diseño propuesto, los cuales se ilustran a través de la figura 2.2.

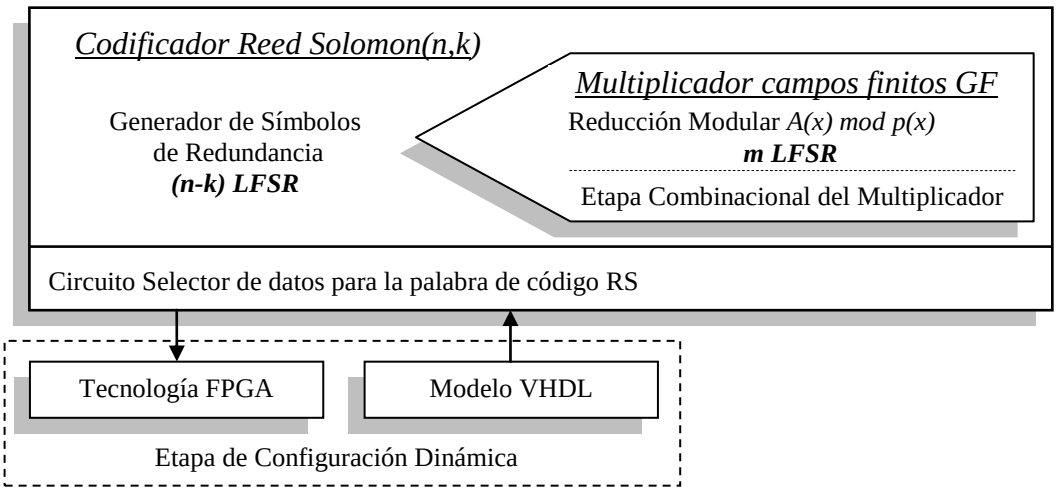


Figura 2.2. Esquema Conceptual del Modelo del Codificador RS(*n,k*)

Se presenta así el esquema conceptual del objeto de estudio correspondiente al codificador RS(*n,k*) compuesto por un circuito generador de símbolos de redundancia y un circuito selector de datos para la generación de la palabra de código RS, a su vez el

generador de símbolos de redundancia está constituido por multiplicadores GF, que a su vez cuenta con un circuito de reducción modular y una etapa combinacional, todos estos componentes son tratados bajo un enfoque de diseño para tecnología FPGA y con una filosofía de modelado para descripción de hardware, por medio de lenguaje VHDL.

CÓDIGOS REED SOLOMON

Los códigos Reed-Solomon, desarrollados por los ingenieros – matemáticos Irving S. Reed y Gustave Solomon, son códigos no binarios, de *n* símbolos de *m* bits cada uno, que están clasificados como códigos BCH (*por las siglas de sus autores: Bose, Chaudhuri y Hocquenghem*), estos son una variante de códigos lineales y cíclicos, cuya particularidad consiste en el tratamiento de los datos a través de bloques de longitud fija. Los códigos RS(*n,k*), son definidos por sus parámetros: *n*, que corresponde al número de símbolos del código*; y k*, el número de símbolos de datos.

Los RS(*n,k*) presentan una capacidad de corrección de errores, dada por: $t = (n-k)/2$, logrando la corrección de hasta *t* símbolos errados por bloque y permitiendo un diseño ajustado a la tasa de error estimada del canal. El proceso de decodificación debe cumplir que la palabra de código sea divisible de forma exacta entre el polinomio generador, para su comprobación se cuenta con técnicas de decodificación altamente eficientes, incluso con características adaptativas (Allen, 2008; Sandoval & Fedón, 2008b).

La eficiencia del código RS(*n,k*) en una transmisión estará dada por la relación entre: la tasa de código *k/n*, la cual permite calcular el ancho de banda y la velocidad efectiva de la transmisión, y la relación entre la potencia de la señal y el ruido del canal E_b/N_o, para el cálculo de la ganancia del código (en dB). Diversos trabajos (Kumar & Gupta, 2011; Torres, 1999) de análisis de eficiencia de los códigos RS(*n,k*) presentan las curvas de desempeño de estos, en función de la relación E_b/N_o vs. BER para cada codificador. Todos estos datos sirven de referencia para la selección apropiada de los parámetros del codificador según las condiciones del canal.

2.2.1. CODIFICADORES REED SOLOMON

El codificador RS es un elemento encargado de construir una palabra de código sistemática, en la que se concatenan los *k* símbolos de la palabra de datos (sin alterar) con los *n-k* símbolos de redundancia. Estos últimos resultan de procesar los símbolos de datos a través del generador de redundancia, que presenta una función con realimentación lineal. La función del generador de redundancia está asociada a un polinomio característico del código *G(x)*. Este polinomio generador se presenta estandarizado en (Xilinx, 2012), de acuerdo a los parámetros *n* y *k*.

Expresando en forma polinómica la palabra de datos *D(x)*, cuyos coeficientes corresponden a los símbolos de datos y procesándola por medio del polinomio generador *G(x)*, se obtiene la representación polinómica de la palabra de código, dada por: *C(x)=G(x)*D(x)*, que presenta características de ser una función lineal, invariante en el tiempo que puede ser expresada de forma recursiva. Este procesamiento puede implementarse de forma secuencial, como es el caso de los códigos convolucionales, los cuales se pueden interpretar como la convolución, entre los símbolos del mensaje con la respuesta impulsiva del codificador. Sin embargo, es importante recordar que el mensaje en el código RS no corresponde a un flujo continuo de símbolos, sino a un bloque de longitud definida, resultando apropiada la implementación paralela sobre hardware de estos codificadores, ya que su entrada puede ser tratada bajo procesamiento concurrente.

Dado que los codificadores RS realizan un procesado sobre bloques de datos, esto demanda una gran capacidad de cómputo para su implementación. Adicionalmente, sus operaciones se realizan sobre algebra de campos finitos, por tratarse de un algebra no convencional requiere hardware diseñado para tal fin, todo esto hace que la velocidad de respuesta en la salida del codificador sea un factor crítico, que estará limitada por el hardware sobre el cual se implementa el circuito. Para modelar el sistema del codificador RS(*n*,*k*), resulta importante estudiar su comportamiento y su arquitectura específica.

2.2.2. ARQUITECTURA DEL CODIFICADOR REED SOLOMON

En el codificador Reed Solomon (Sandoval & Fedón, 2007), se puede identificar la arquitectura característica del generador de símbolos de redundancia, para la implementación de las ecuaciones que describen matemáticamente el codificador RS (Sandoval, 2007). Esta arquitectura corresponde a un elemento circuital generador de secuencias pseudo-aleatorias conocido como LFSR. La estructura *n,k*-LFSR, *del inglés Linear Feedback Shift Register*, es un circuito secuencial de *n-k* etapas de *m* bits cada una, donde los símbolos de datos ingresados ordenadamente son operados (en algebra de campos finitos) con los coeficientes del polinomio generador del codificador *G(x).*

Esto se lleva a cabo, a través de componentes multiplicadores en campos finitos de Galois GF(2^m), los cuales son los responsables de obtener los elementos a operarse en las compuertas XOR y almacenarse en los elementos de memoria, como se muestra en la figura 2.3.

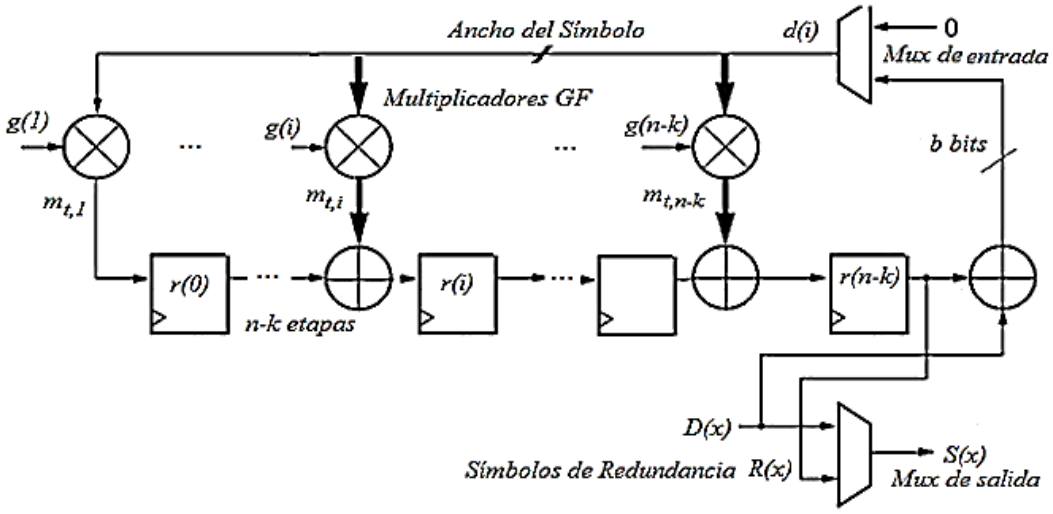


Figura 2.3. Arquitectura del Codificador RS(*n,k*) Genérico

El generador de símbolos de redundancia del codificador recibe el primer símbolo de la entrada de datos, este símbolo pasa a la salida sin alterar, a través del componente selector del símbolo de salida (*Mux de salida)*, a la vez es operado con el dato

almacenado en el registro menos significativo *r(0) a través de la xor de entrada*, generando un símbolo dado por: *D(x) xor r(0)*, el resultado es ingresado a través del siguiente multiplexor (*Mux de entrada)* a la realimentación del circuito LFSR y se realiza la operación de éste con cada uno de los coeficientes del polinomio generador *G(x)*, a través de la multiplicación en algebra de campos finitos de *Galois*.

El resultado del producto se opera a través de una *xor* de *m* bits, con la salida del registro precedente, al recibir la señal de reloj, el dato es almacenado en el registro a la salida de la *xor*. Este proceso se realiza para los *k* símbolos de datos, donde los resultados dependen de la secuencia de símbolos de entrada. Alcanzados los *k* símbolos, el *mux de entrada* conmuta a la selección de la entrada cero y el multiplexor de salida selecciona el símbolo del *r(0)* durante *(n-k)* pulsos de reloj, los cuales corresponden a los símbolos de redundancia generados del proceso, bajo esta secuencia de dos etapas se genera la palabra de código Reed Solomon.

2.2.3. INTERPRETACIÓN DEL CIRCUITOS LFSR

Ya se han mencionado las características del generador de redundancia, el cual ha sido identificado como un circuito LFSR, por su capacidad de generar secuencias con buenas propiedades estadísticas (Ekdahl, 2003). Sin embargo, resulta pertinente un análisis de la estructura LFSR generalizada. Estos circuitos pueden ser lineales o no lineales – NLFSR, de acuerdo a los operadores en la función de realimentación, tomada de cualquier etapa. Estos pueden operar sobre bits, símbolos o arreglos. La generalización de la arquitectura del componente permite observar su versatilidad para modelar diversas funciones matemático-lógicas, propias de los componentes de hardware.

Para el modelado del LFSR es necesario expresar el comportamiento del circuito de forma genérica. Siendo necesario definir los parámetros de escalabilidad del circuito en función del número de etapas y tamaño del bus. En base a esta necesidad, se han encontrado desarrollos en técnicas de factorización de las funciones y operaciones

comunes, que dan paso al concepto de LFSR reconfigurables (Alaus, Oguet, & Alicot, 2008), lo que permite la reutilización de hardware del circuito R_LFSR (ver figura 2.4).

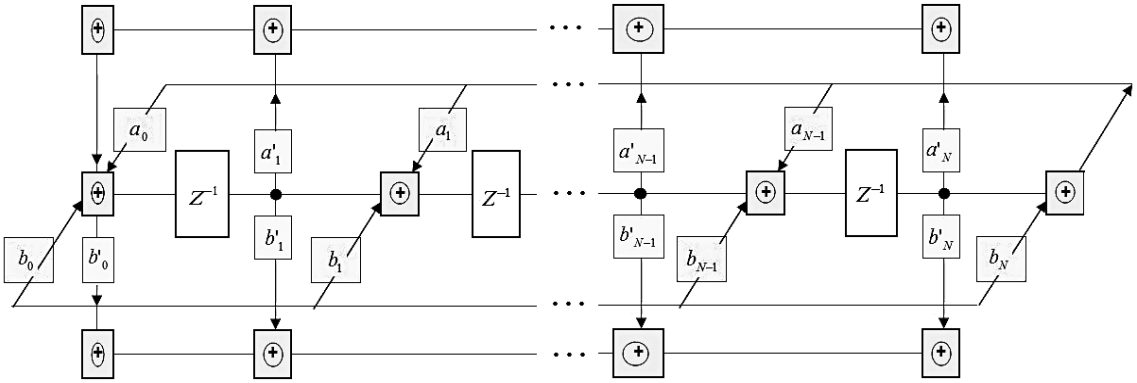


Figura 2.4. Arquitectura del R-LFSR con operadores comunes
Fuente: (Alaus et al., 2008)

La parametrización del LFSR será aplicada en la configuración de hardware en VHDL, con un tratamiento genérico del vector de memoria, el tamaño del bus, la capacidad de los operadores y los coeficientes del polinomio generador. De tal manera, que realizando la concatenación de componentes del circuito se puede tratar el diseño de forma modular con habilitadores de etapas, a fin de emplear el LFSR reconfigurable como modelo base, en este caso se emplea una generalización para manejar la estructura y los coeficientes. En la figura 2.5, se representa el tratamiento propuesto.

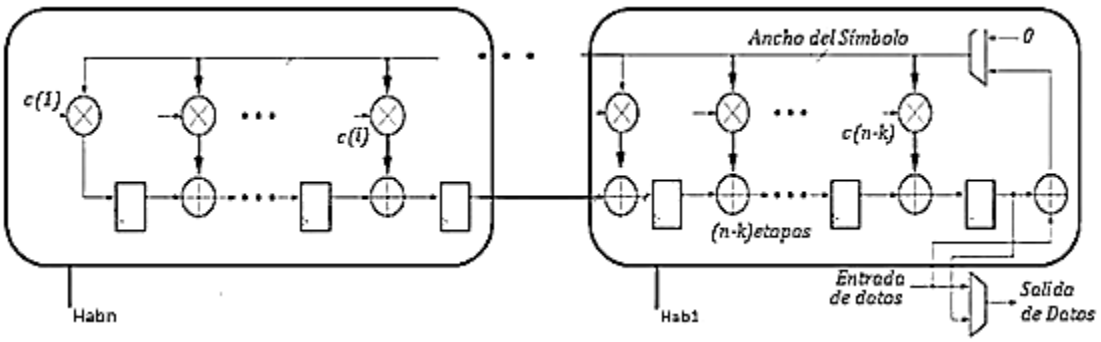


Figura 2.5. Arquitectura del Codificador RS(*n*,*k*) con habilitación de etapas

2.2.4. TENDENCIAS HACIA CODIFICADORES RS PARALELOS

Los avances en implementación de codificadores RS paralelos se proponen como alternativa al uso secuencial de las etapas de hardware. En esta línea se encontró un codificador y decodificador de 4 etapas paralelas (Choi, Ahn, & Lee, 2011), una modificación al algoritmo de Euclides para paralelizar etapas del decodificador (Lee, 2003), y el concepto de codificación a través de unidades de procesamiento paralelas basadas en multi-núcleos (Sobe, 2010). En vista que la paralelización es uno de los objetivos en el procesamiento de los datos para corrección de errores, se ha estudiado el diseño de codificadores RS paralelos para casos particulares (C. Sandoval-Ruiz, 2012), que pueden generalizarse, a fin de avanzar en un modelo que soporte de codificadores RS(n,k) concurrentes.

2.2.5. FUNDAMENTOS DE CAMPOS FINITOS

Profundizando en el estudio del codificador RS para su implementación eficiente, se encuentra que estos emplean aritmética de campos finitos de *Galois* GF(2^m), por producir resultados con longitud fija y que sus operaciones pueden ser implementadas con circuitería relativamente simples (Morelos-Zaragoza, 2002). Donde se debe tener en cuenta que la eficiencia computacional de las operaciones aritméticas en campos finitos está estrechamente relacionada con la forma particular en la cual los elementos del campo son presentados (C. H. Kim, Oh, & Lim, 2002). Por lo que se ha detectado la necesidad de estudiar alternativas de solución para la implementación de operaciones aritméticas GF(2^m), basadas en los fundamentos matemáticos de los campos finitos.

Los *Campos Finitos de de Galois* (GF) constituyen un área específica de la matemática desarrollada por E. Galois. Donde el campo es especificado a través de un elemento primo p, base del campo; y un entero positivo m, longitud del elemento del campo. Se cumple que p^m corresponde al número de elementos del campo, y las operaciones aritméticas sobre el campo finito da como resultado un elemento que pertenece al mismo (Saqib Nazar, 2004). Estas propiedades son utilizadas en aplicaciones de codificación,

decodificación Reed-Solomon (Sandoval & Fedón, 2007; Xilinx, 2011) y criptografía. Una presentación ampliamente utilizada es la forma polinomial, en la cual se define un polinomio generador del campo, conocido como polinomio irreducible *p(x)*, el cual se operará módulo con los resultados de las operaciones para llevar el resultado a la longitud fija definida para el campo.

2.2.6. MULTIPLICADORES DE CAMPOS FINITOS

El multiplicador en campos finitos es generalmente más complejo que un multiplicador convencional (Tejeda-calderón, García-martínez, & Posada-gómez, n.d.), estos han sido un tópico de investigación desde 1960 hasta la actualidad, por ello la relevancia de interpretar la lógica circuital de soporte para estos módulos y establecer un modelo eficiente para su diseño sobre dispositivos de hardware FPGA.

Los multiplicadores aritméticos sobre campos finitos de *Galois*, tienen como base la multiplicación de dos elementos del cuerpo finito y la reducción del resultado mediante un polinomio irreducible *p(x)* de grado *m*. Estos pueden ser implementados a través de un modelo algorítmico, el cual reproduce el comportamiento del multiplicador de forma secuencial (Cruz, 2005) o modelos paralelos, según la necesidad de que estos módulos aritméticos operen a frecuencias elevadas y ocupen el menor área posible, lo que demanda un equilibrio entre estos factores. Este compromiso ha dado lugar a la búsqueda de algoritmos y arquitecturas eficientes, tal como lo señalan las investigaciones en el área (J. Imaña, 2004).

Entre los modelos encontrados destacan: el multiplicador propuesto por Karatsuba-Ofman (Machhout et al., 2009), éste consiste en una técnica de sub-división modular del componente multiplicador, modelos combinacionales (G. C. Ahlquist et al., 2002), a través de tablas (Sandoval & Fedón, 2008c), (Song, Kuo, & Lan, 2007) y combinaciones tablas - algoritmos (Marchesan Almeida, Bezerra, Cargnini, Fagundes, & Mesquita, 2007). Este último presenta un aporte en reducción de recursos en base a la

implementación original de tablas de 255x255, equivalente a 65025 símbolos de 8 bits, a una reducción a 256 símbolos de 8 bits para el multiplicador, sin embargo, esta reducción sigue demandando 512LUTs de 4 entradas al hacer la expansión para cubrir la tabla mencionada.

A nivel de optimización de las implementaciones se tienen el Multiplicador *Mastrovito* (combinacional – matricial), Multiplicador *Massey-Omura*, Multiplicador *Hasan-Bhargava*, Multiplicador *Paar-Rosner*, Multiplicador *Morii-Berlekamp*, Multiplicador *Combinacional Pipelined*, Multiplicador de Registro Desplazamiento con Realimentación Lineal – *LFSR* todos estos descritos en (G. Ahlquist et al., 1999). Igualmente, para conseguir escalabilidad, se han propuesto varios algoritmos todos ellos implementados en hardware (Climent et al., 2008).

Del estudio de los modelos de multiplicadores GF desarrollados previamente, se encontró que los multiplicadores GF(16) *Mastrovito*, *Paar-Rosner* y *Pipelined Combinatorial* presentan el menor consumo y mayor velocidad en comparación con multiplicadores alternativos. Por tal motivo, se han considerado como referencia para la evaluación de los resultados, aun cuando algunos de estos multiplicadores presentan optimización bajo un enfoque particularizado de la arquitectura, han sido un importante aporte para la interpretación de las alternativas de optimización. De los modelos estudiados se ha considerado el multiplicador basado en LFSR estándar, como base para la optimización, éste presenta una latencia de *m* pulsos de reloj, que puede ser paralelizado bajo un análisis temporal del circuito.

Este análisis parte del estudio del modelo matemático de la representación polinomial, en el que se enuncia: Si *p(x)* es el polinomio irreducible, entonces la multiplicación de dos elementos del campo, representados como los polinomios *A(x)* y *B(x)* es el producto algebraico de los dos polinomios, y la operación módulo del polinomio *P(x)*, *también conocido* como reducción modular, es el mostrado en la ecuación 2.1.

$$C(x) = A(x).B(x) \leftrightarrow C(x) = A(X) \times B(x) \ (mod\ p(x)) \qquad (2.1)$$

La multiplicación de polinomios es asociativa, conmutativa y distributiva con respecto a la adicción por lo cual se obtienen la ecuación 2.2.

$$C(x)=B(x)(\Sigma_{i=0}^{m-1} A_i x^i) \ mod\ p(x) \rightarrow C(x)= \Sigma_{i=0}^{m-1} B_i(A(x)x^i mod\ p(x)) \qquad (2.2)$$

Donde *A(x)* y *B(x)* corresponden a la representación polinomial de los operandos. En el caso del codificador RS el multiplicando *A(x)* corresponde a un coeficiente del polinomio generador del código y el multiplicador *B(x)* a la entrada de datos, y *p(x)* es el polinomio irreducible del campo de *Galois*. Para realizar el cálculo de la reducción modular se emplea el concepto de la división de polinomios sobre campos finitos, cuya expresión matemática se presenta en la ecuación 2.3.

$$\begin{array}{l} a_{n-1}x^{m-i} + \ldots + a_1 x + a_0 \\ \quad r_{n-1}(x) \end{array} \quad \Big| \begin{array}{l} p_{r-1}x^{m-1} + \ldots + p_2 x^2 + p_1 x^1 + p_0 \\ \hline 1/p_{n-1}\, x^{m-1} + \ldots \end{array} \qquad (2.3)$$

Donde *r(x) = A(x) mod p(x)*, corresponde al residuo de la división entre el operando *A(x)* de la multiplicación y el polinomio irreducible del campo finito GF(2^m). Para la implementación circuital de las ecuaciones presentadas se deben identificar las etapas del multiplicador en campos finitos. En primer lugar se estudia el modelo propuesto por (Tejeda-calderón et al., n.d.), que comprende una etapa de reducción modular sobre los resultados, dividido en cuatro niveles de operación sobre los datos. En la figura 2.6 se presentan las etapas mencionadas de acuerdo al modelo inicial, donde se observan los operandos de *m* bits y el desplazamiento de los productos parciales que originan un acumulado de longitud *m+m-1*, lo que genera un elemento resultante que no pertenece al campo finito y debe incorporarse una conversión, a través de una reducción final para que el producto corresponda con un elemento del campo.

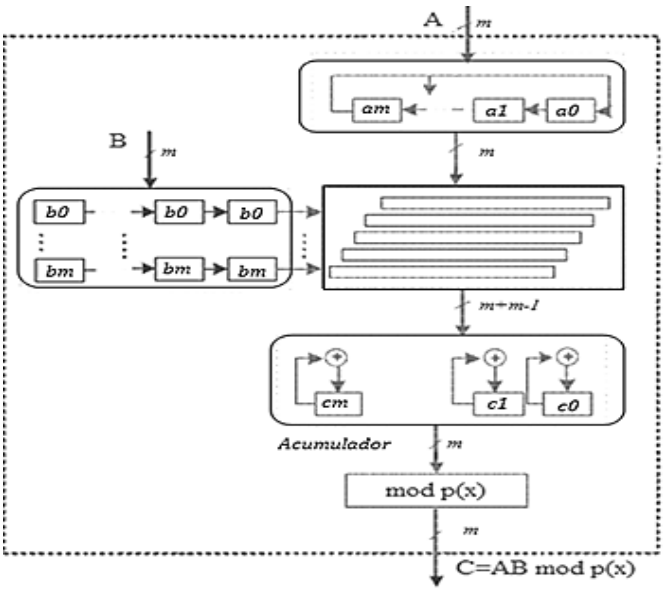


Figura 2.6. Multiplicador GF inicial
Fuente: (Tejeda-calderón et al., n.d.)

En el primer nivel se reconoce un *reductor Principal*, que opera el dato de entrada A(x), seguido del núcleo multiplicador, que opera a *A(x)* reducido con *B(x)*, donde se introducen desplazamientos, la etapa del acumulador y finalmente la *reducción final* a través del polinomio generador del campo *p(x)*. Con lo que se establece el resultado de la longitud definida por el campo.

Como alternativa a esta estructura se plantea el multiplicador presentado en la figura 2.7.

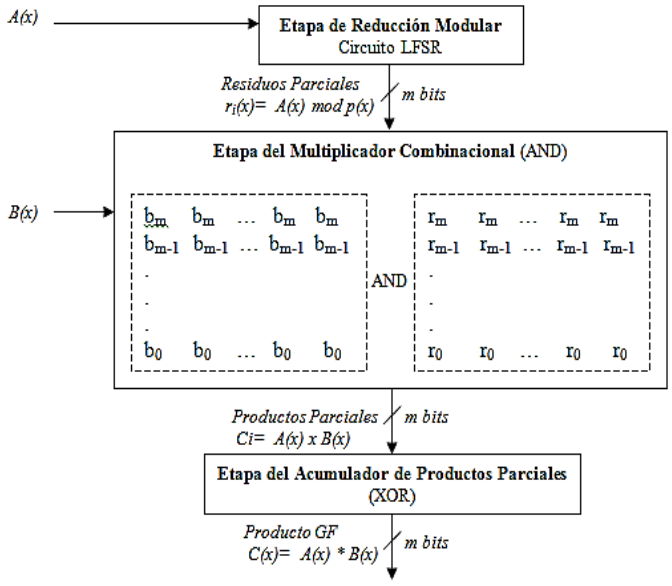


Figura 2.7. Multiplicador GF modificado

En el primer nivel presenta la etapa de *reducción modular* con el circuito LFSR que genera los *residuos parciales* de la división de *A(x)* entre *p(x)*, seguido de la multiplicación sin corrimiento de estos residuos parciales por el correspondiente bit de *B(x)*, para finalmente ser sumados en *mod-2*, lo que genera un acumulado de longitud *m* bits. Resultando un circuito más eficiente.

Partiendo del análisis de arquitecturas del multiplicador (García-Martínez, Morales-Luna, & Rodríguez-Henríquez, n.d.), podemos observar una etapa combinacional de multiplicación compuesta por un conjunto de *ANDs* y *XORs* y una etapa secuencial de *reducción modular*, que implementa una división de polinomios entre un elemento del campo finito $A(x) \in GF$ y el polinomio irreducible que define el campo $p(x)$. Si se logra definir el modelo del LFSR a través de una implementación concurrente, el área y la potencia consumida son similares e incluso menores comparadas con la versión secuencial (Yap, Khoo, & Poschmann, 2010). Para esto se analizaron optimizaciones sobre el mencionado circuito (Dubrova, 2008) y un modelo matemático para la generación de las sub-secuencias, considerando que aplicaciones de alto rendimiento requieren de implementaciones paralelas (Mucci et al., 2008).

2.2.7. CIRCUITO DE REDUCCIÓN MODULAR EN ALGEBRA DE GALOIS

La arquitectura de la etapa de reducción modular, está basada en generadores de secuencias de *Galois*, sus propiedades son estudiadas en (Dubrova, 2008). De allí se observa que originalmente es implementada a través de registros, para obtener los *n* vectores, correspondientes a los residuos parciales. Esto por medio de corrimiento de los bits sobre la estructura en *m* ciclos de reloj, donde *m* es igual al número de bits de la palabra. En el caso de un campo $GF(2^8)$, con 256 elementos de campos de 8 bits de longitud. La arquitectura del LFSR coincide con la presentada en la figura 2.8, donde se ilustra el circuito operador de los símbolos de 8 bits de la entrada $A(x)$ con el polinomio $p(x)$.

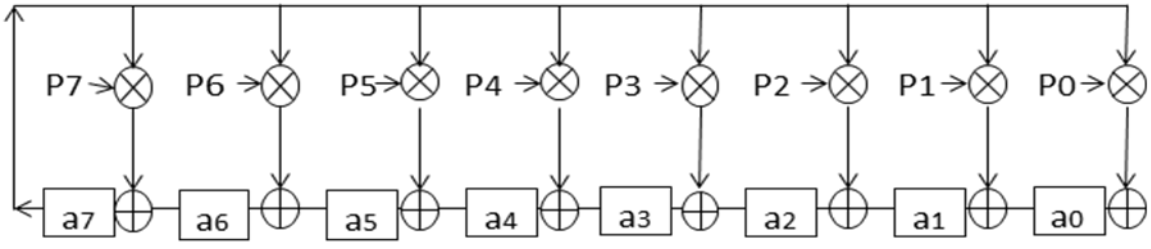


Figura 2.8. Estructura del módulo LFSR para $GF(2^8)$

El modelo de un LFSR configurable es de gran interés para el proceso de síntesis (Chen, 2001). Partiendo de esto, se propone el análisis del comportamiento de los resultados temporales generados por la estructura LFSR (Sandoval, 2010a), a fin de construir un modelo matemático, orientado a la descripción en VHDL.

2.2.8. ESTRUCTURA AUTOSIMILAR BASADA EN EL CIRCUITO LFSR

Llegado a este punto de la disertación, se han estudiado de forma separada las estructuras componentes de los elementos del codificador Reed Solomon. Identificando una arquitectura común entre el circuitos generador de símbolos de redundancia y el circuito reductor del multiplicador de campos finitos, lo que resulta de vital interés para establecer una descripción VHDL. En la figura 2.9 se presenta la mencionada estructura.

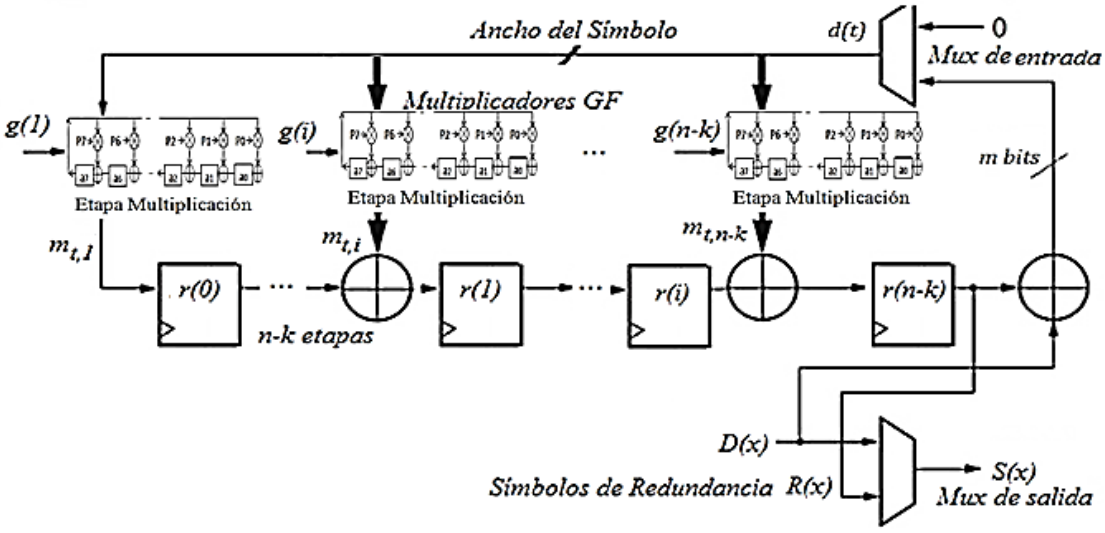


Figura 2.9. Arquitectura del Codificador RS(*n*,*k*) con multiplicadores GF

Se observa un LFSR cuya función es implementada a través de un LFSR interno, lo que permite observar el *Isomorfismo* estructural entre el módulo *A(x) mod p(x)* y el generador de símbolos de redundancia. Donde el polinomio generador del campo *p(x)* y

el polinomio generador del código *G(x)* definen los coeficientes de entrada de los multiplicadores en la función de realimentación para cada caso.

Así, considerando el *principio de correspondencia* los avances en el modelo VHDL que se realizan para el módulo reductor es aplicable para el generador de símbolos de redundancia del codificador. De esta manera, una vez que se cuenta con los fundamentos para la paralelización del circuito LFSR, se puede desarrollar un modelo concurrente tanto para los multiplicadores en campos finitos como para el codificador Reed Solomon.

2.2.9. ARQUITECTURA RECONFIGURABLE

Basados en los avances en sistemas con parámetros configurables por software, se han considerado alternativas de diseño, las cuales pueden ser arquitectura selectiva o configurable. En el primer caso, se define la arquitectura de soporte para los diversos modos y se conmuta para seleccionar el modo requerido. Donde se tendrían implementados componentes que no estarían siendo usados en determinado momento y consumirían recursos. Para el caso de módulos configurables, se tendría mayor eficiencia, ya que permite compartir módulos de hardware diseñados para funciones comunes y aplicando la técnica de parametrización, para los módulos de hardware que comparten funciones y operadores, con lo que un mismo componente se puede configurar para implementar funciones según los requerimientos específicos.

El enfoque de la presente investigación es hacia diseño con hardware reconfigurable, lo que permite que el diseñador pueda re-adaptar la arquitectura del sistema diseñado desde la configuración del dispositivo de hardware. Incluso el mismo diseño puede realizar ajustes en su arquitectura, lo que se ha definido bajo el concepto de *Hardware Evolutivo* – EHW (Kitano & Hendler, 1994). Éste relaciona la tecnología del dispositivo con una filosofía auto-reconfigurable, con la flexibilidad del hardware a adaptarse a requerimientos no previstos en la etapa de diseño inicial.

En este punto es oportuno señalar la diferencia entre "*Auto-Reconfiguración*" como la capacidad de un dispositivo para generar en tiempo de ejecución un nuevo *bitstream,* es decir, el código de configuración del hardware y usarlos para modificar su propia arquitectura, y "*Reconfiguración Parcial Dinámica",* en la cual se modifican etapas del sistema en tiempo de ejecución, en este tipo de reconfiguración el diseñador debe prever un módulo para esta tarea, capaz de generar los bits de configuración dentro del dispositivo, *bitstreams* parciales. Para lograr este objetivo, el primer requisito es contar con un modelo matemático-descriptivo, que cumpla con los principios de los sistemas adaptativos (Rodríguez & López, 1996).

2.2.10. TECNOLOGÍA FPGA

Todo el estudio de la arquitectura del codificador Reed Solomon ha sido orientado hacia una solución de hardware, por lo cual su descripción debe estar dado para un dispositivo de hardware configurable y de tecnología actualizada para el modelo que acá se plantea como solución, en este aspecto se selecciona la tecnología FPGAs (*Field Programmable Gate Arrays*), donde se puede configurar la arquitectura diseñada de acuerdo al modelo VHDL que describe su comportamiento.

Los FPGA se encuentran constituidos por *slices,* son bloques lógicos funcionales cuyos arreglos permiten la implementación en hardware del diseño, en función a estos elementos se realiza el análisis del consumo de recursos. Donde la eficiencia del diseño es medida en términos asociados a la complejidad espacial y temporal de la implementación, donde la complejidad espacial es definida por la ocupación de recursos sobre el dispositivo implementado, y la complejidad temporal, es medida a través de los retardos totales de la respuesta del circuito diseñado (Marchesan Almeida et al., 2007).

Es bastante el avance que se presenta en la optimización del consumo de potencia dinámica para dispositivos FPGA, asociada al proceso de diseño, la cual puede ser manejada con técnicas propias de la herramienta de desarrollo (Hussein, Klein, & Hart, 2011), o directamente por el diseñador, a través de métodos de diseño específicos, (G Sutter & Boemo, 2007; Gustavo Sutter, 2005). De las cuales en este desarrollo se han considerado diseño modular, paralelización, re-ordenamiento de señales, des-habilitación de etapas y simplificación de operaciones por coeficientes nulos, para alcanzar un mejor desempeño de los módulos diseñados.

Los diseñadores de sistemas basados en FPGAs cuentan con herramientas para optimizar los circuitos en área y velocidad, a través de los IDE (*Integrated Development Environment*) proporcionados por los fabricantes de FPGAs. Sin embargo, no proveen software de diseño para bajo consumo de potencia (Todorovich, Acosta, & Nacional, n.d.). En tal sentido, el presente diseño estará orientado fundamentalmente a la optimización del área del dispositivo y velocidad. La optimización de área dado que la potencia consumida por el dispositivo está directamente relacionada con la ocupación de área del diseño (Saqib Nazar, 2004) y la optimización de velocidad debido a mediciones obtenidas en trabajos previos, donde concluyen que el consumo total es menor en la versión paralela (Angarita, Marin-Roig, & Todorovich, 2005).

2.2.11. LENGUAJE DE CONFIGURACIÓN DE HARDWARE VHDL

VHDL (*VHSIC Hardware Language Descriptor*) es un lenguaje descriptor de hardware para circuitos integrados de muy alta velocidad y escala de integración (*VHSIC: Very High Speed Integrated Circuit*). Se encuentra normalizado bajo el estándar IEEE Std 1076-1987, lo que le brinda portabilidad entre dispositivos (Pérez L., Soto C., & Fernández G., 2002). Esto permite que los sistemas diseñados sean compatibilidad para diversos fabricantes y plataformas. Al mismo tiempo, soporta el diseño modular; lo que se traduce en una ventaja puesto que las optimizaciones sobre un módulo o componente

del diseño puede ser empleado para el sistema completo. Esta característica a su vez ofrece la posibilidad del diseño colaborativo, ya que se puede desarrollar un sistema por parte de diversos diseñadores en forma síncrona o asíncrona.

Por otra parte, se han estudiado conceptos y técnicas de optimización (Mora, 2008). Siendo, la filosofía de la reutilización de componentes uno de los conceptos que más se ha extendiendo para agilizar el desarrollo de un producto (Castillo, 2008), su metodología ofrece ventajas significativas, ya que su aplicación hace innecesario cubrir toda la curva de aprendizaje para la creación de cada producto, al reutilizar los bloques funcionales existentes. Para este proyecto se emplea una filosofía de diseño que comprende la aplicación de modelos para VHDL en lugar de los códigos fijos en VHDL, lo que agrega una variante de mayor flexibilidad al concepto de reutilización convencional. Teniendo en cuenta que los niveles de abstracción del diseño inciden en el consumo de potencia (Gustavo Sutter, 2005), el tratamiento propuesto puede permitir optimizaciones en el modelo de los componentes, lo que se puede definir como un nuevo paradigma y un cambio de filosofía.

La descripción de hardware se puede realizar mediante un proceso secuencial, el cual consiste en un conjunto de instrucciones ordenadas bajo una secuencia lógica de ejecución, donde una instrucción dependa del resultado de la instrucción anterior. Existe un elemento temporal a considerar que controla el orden de ejecución de cada instrucción. La instrucción de sintaxis *process* se emplea para describir la parte secuencial de un circuito. A efectos de la simulación, solo se ejecutará el *process* cuando se detecte variación de las señales declaradas en la lista de sensibilidad del proceso, en el proceso las sentencias seguirán una secuencia que depende de la ocurrencia de un cambio en las mencionadas señales, un ejemplo es la señal de reloj *clk*, para la sincronización de las instrucciones en el generador de redundancia del codificador RS.

A partir del diseño orientado a hardware, los componentes pueden ser descritos a través de expresiones booleanas, para circuitos combinacionales y asignaciones concurrentes,

con operadores propios en VHDL. Este tipo de asignaciones pueden definirse como operaciones sobre señales que coinciden en el tiempo. De esta manera, el diseño no requiere compartir componentes sino que se implementan múltiples etapas para el procesamiento de las señales. Este concepto es importante para la paralelización de componentes, permitiendo obtener las señales procesadas en un solo pulso de reloj, de forma simultánea. De esta manera se considera en la solución al codificador RS con salida paralela, lo que lo hace un código más eficiente y competitivo, mediante esta configuración.

2.2.12. CONCLUSIONES

Del estudio de las bases conceptuales, surgieron términos de interés, que conforman la taxonomía asociada a la tesis doctoral. Tal es el caso, del término *concatenación* corresponde a una disposición de las señales dentro de un circuito para definir un arreglo espacial de los elementos de una señal, en arquitecturas fijas de procesadores de señales no se encuentra soportada esta operación, VHDL presenta la alternativa de concatenar señales, lo que se puede tratar como un operador en la sintaxis de descripción. Permitiendo procesar los elementos de un arreglo, sin el uso de señal de reloj. Así la operación de concatenación reduce los retardos propios de la arquitectura en el hardware.

Partiendo de la arquitectura del circuito LFSR (*Linear FeedBack Shift Register*), de naturaleza secuencial, se propone un circuito concurrente que genere los resultados del circuito secuencial, al cual se le denominó *Estructura Concurrente de Realimentación Lineal*. Los conceptos revisados muestran que los fundamentos teóricos del codificador están desarrollados de forma algorítmica. La propuesta va orientada a definir un codificador RS Eficiente orientado a Sistemas Reconfigurables.

CAPÍTULO III

MODELADO DEL CODIFICADOR RS (N,K)

En este capítulo se proponen técnicas de optimización del costo computacional, en el modelado de los componentes del codificador Reed Solomon, orientado a sistemas reconfigurables y el diseño de la arquitectura hardware para el codificador RS. A lo largo del capítulo se describen los métodos y técnicas empleadas para el desarrollo del modelo optimizado del codificador Reed Solomon (255,k).

3.1. INTRODUCCIÓN

El modelado del codificador RS(n,k), comprende una investigación en el desarrollo de hardware en VHDL, aplicada en el área de codificación de canal. El diseño de la investigación contempla una revisión documental y pruebas experimentales para la estimación del consumo de recursos del codificador RS, usando la herramienta de desarrollo ISE 11 de Xilinx y contrastando los resultados con los trabajos disponibles en esta área de investigación.

En esta sección se abordará la descripción metodológica propuesta, para cada etapa de la investigación, mencionando las herramientas empleadas.

Fase I. Diseño de un codificador RS(255,k) modular con componentes concurrentes en VHDL, para su optimización

En esta fase se realizó un estudio inicial de los conceptos que definen el principio de la codificación Reed Solomon, se identificaron las características presentes en la arquitectura del codificador, encontrando un elemento común tanto en el generador de redundancia, como en el multiplicador sobre cuerpos finitos de *Galois*. Se diseñaron los codificadores de forma particularizada para diferentes parámetros *k*, para estos diseños se identificaron los componentes comunes y se realizó su descripción y declaración de parámetros ajustables, de acuerdo al protocolo de comunicación seleccionado. De este modo se parametrizó el codificador, a fin de su utilización en las siguientes etapas del diseño. La filosofía del diseño es modular basado en componentes, este diseño se validó obteniendo un comportamiento apropiado de acuerdo a la definición teórica, para luego presentar un análisis de cada versión del diseño.

Fase II. Interpretación de los reportes de síntesis del codificador RS(255,k), para estudio de Eficiencia

En esta fase se interpretaron los reportes generados por la plataforma de desarrollo ISE11 de Xilinx, correspondientes al consumo de recursos de hardware, consumo de potencia y velocidad, para el codificador diseñado. Los resultados fueron tabulados, comparándolos con trabajos de optimización presentados en investigaciones previas, a fin de evaluar el grado de optimización alcanzado y la eficiencia del modelo desarrollado. Para ello se aplicaron técnicas de diseño orientadas a hardware reconfigurable y se establecieron los indicadores de eficiencia, de esta manera se logra un diseño base para establecer el modelo.

Fase III. Obtención del Modelo del Multiplicador concurrente, en aritmética en campos de Galois GF(2^m), a partir del circuito LFSR paralelizado

En la obtención del modelo concurrente para el multiplicador en campos GF, se inició por la descripción y estudio del modelo matemático del multiplicador GF, los circuitos de implementación de la división de polinomios y su comportamiento secuencial. Para la paralelización, se realizó un análisis de arquitectura y generación de la secuencia de términos, a fin de caracterizarlo y así obtener las ecuaciones que describen el LFSR del multiplicador GF. El método comprendió la definición de los términos según los parámetros de espacio y tiempo, dejando este último expresado en forma combinacional, resultando un nuevo modelo concurrente. Igualmente, el proceso de diseño está basado en una metodología de programación modular de los componentes, que soportan el codificador en lenguaje descriptor de hardware VHDL, con lo que se logra una adaptación del modelo matemático inicial a un modelo para configuración de hardware.

Fase IV. Sistematización del modelo concurrente para el codificador Reed Solomon, a partir de la correspondencia entre las estructuras circuitales

En esta última fase, una vez alcanzada la optimización del codificador Reed Solomon, usando el modelo LFSR concurrente desarrollado para el componente multiplicador, se sistematiza el modelo, aplicando el LFCS para la generación de los símbolos de redundancia. En esta aplicación se requirió establecer ajustes sobre las ecuaciones para elementos de m símbolos y operaciones de orden superior, como la multiplicación GF en sustitución de la *and* lógica. Con lo que se obtiene el modelo de un codificador Reed Solomon concurrente, con mayor eficiencia respecto al modelo secuencial, éste se validó, a partir de la funcionalidad y resultados de implementación en un caso de estudio de un RS particular.

3.2. DISEÑO DEL RS(*N*,*K*) GENÉRICO

Al momento de diseñar componentes de sistemas reconfigurables resulta conveniente la descripción bajo una metodología de *abstracción-agrupación* (Mora, 2008), empleando un modelo con cierto nivel de abstracción y agrupando los componentes para lograr el objetivo final. Partiendo de una descripción en VHDL basada en un modelo particular y detallando sus componentes, para luego identificar patrones, que permitieron una generalización del sistema diseñado. Todo esto documentando el código fuente (código en VHDL y bitstream) y el diseño conceptual, orientado al desarrollo de hardware libre dinámico (Cenditel, 2012), a fin de facilitar el proceso de optimización y expansión.

DESCRIPCIÓN DEL CODIFICADOR RS PARAMETRIZABLE

Inicialmente, se partió del desarrollo de un codificador RS con la longitud del código fija tomando como referencia el modelo en VHDL de prueba (Sandoval & Fedón, 2007), se ha extendido el diseño, ampliando la longitud del código para 255 símbolos de 8 bits, donde se ha expresado el parámetros *k* como un valor genérico. Este parámetro puede ser definido en tiempo de configuración, de donde se obtiene un codificador RS(*255*,*k*) reconfigurable por software o estableciendo el valor de *k* apropiado, a través de una entrada de selección, que podría ser suministrada por la etapa de configuración del sistema.

De acuerdo al modelo conceptual, el codificador RS consta de un LFSR, con selectividad de la entrada de datos. De manera que su descripción será dada en función de los componentes del LFSR, (i) un registro desplazamiento y (ii) una función de realimentación (Peralta, 2005). Con base en esto, se realizó el diseño modular de las etapas del codificador RS(*n*,*k*) de longitud ajustable, a través del parámetro *k*. Donde se fijó el valor de *k* para cada caso particular y así establecer los coeficientes del polinomio generador.

Para la programación se empleó el software ISE 11 de Xilinx, se inició por la definición de la entidad para los módulos RS con longitud ajustable usando lenguaje descriptor de hardware VHDL, a través de la identificación de las características del código seleccionado. Las declaraciones de la entidad fueron realizadas, de forma genérica, con la longitud como un parámetro ajustable en función de *n-k* (*length*), y el número de bits por símbolo como (*width*). Seguidamente, en la arquitectura del codificador se definió una memoria tipo FIFO (*First Input First Output*) de *length* etapas de *width* bits.

Finalmente, se definió la entidad del componente *mult*, lo que comprende la definición de los puertos para la asignación de las señales a procesar: la entrada de datos *D_dato*, un vector de longitud *width-1* a 0; el coeficiente (*coef_i*), del polinomio generador *g(x)*, obteniéndose a la salida el producto en *datox*. Un breve fragmento de la sintaxis en VHDL se muestra en la tabla 3.1, donde se destacan los aspectos más relevantes del código. Es de hacer notar que la sintaxis – *generic* – permite definir el valor de un parámetro, así solo se debe reasignar éste y se modifica su valor en el código VHDL.

Tabla 3.1. Descripción en VHDL del Codificador_RS (n,k)

```
--parámetro ajustable para el RS (n,k), en función de n y k
generic(length: integer:= 16;  --length  n-k
--parámetro ajustable "m" correspondiente a la longitud del campo o
ancho del símbolo
width :integer:=8);   -- width of the symbol
Architecture Behavioral of Codificador_RS is
--Arquitectura del LFSR, donde el tamaño de este arreglo de memoria es
ajustable a través de los parámetros genéricos
Type memoria is array (0 to length-1)of std_logic_vector(width-1 downto 0);
component mult is
port(D_dato: in std_logic_vector (width-1  downto 0);
     coef :      in std_logic_vector (width-1  downto 0);
     datox:   out std_logic_vector (width-1  downto 0));
end component;
-- Asignación de los coef. y datos para el cálculo de productos
Ci:   mult   port map  (D_dato,coefi,datoi);

--al igual que el ancho de los registros memoria y datoi
```

Seguidamente, se describió en VHDL el circuito LFSR (*Linear Feedback Shift Register*) del codificador, con *n-k* etapas, estas etapas fueron representadas por elementos de

memoria *memoria_v(i),* accionados con un *clk*. Las salidas del circuito LFSR que serán transmitidas por el canal de comunicación, usará un multiplexor para seleccionar las *k* palabras de datos y las *n-k* palabras de redundancia, de acuerdo a la señal de control *hab*, como se muestra en la tabla 3.2.

Tabla 3.2. Descripción del LFSR del codificador RS(n,k)

```
process (clk)
variable memoria v:memoria:=(others=>"0000000");
begin
   if hab='1'  then
      D_dato<= memoria_v(0) xor D_in;
   else
      D_dato<="0000000";
   end if;
   if (clk'event and clk='1') then --sincronización del desplazamiento
      memoria_v(0):=memoria_v(1)xor dato1;
      ...
      memoria_v(n-k-1):= dato_n-k;
      -- La descripción del LFSR está dada desde i=0 hasta n-k-1 por:
      -- memoria_v(i-1):=memoria_v(i)xor datoi;
   end if;
   if hab='1' then
      salida<=D_in;
   else
      salida<=memoria_v(0);
   end if;
end process;
```

En la figura 3.1 se presenta el circuito descrito en las tablas 3.1 y 3.2, de manera de identificar cada uno de los componentes, la asignación de longitud del bus de datos y el registro desplazamiento como un arreglo de *n-k* vectores de *m* bits cada uno.

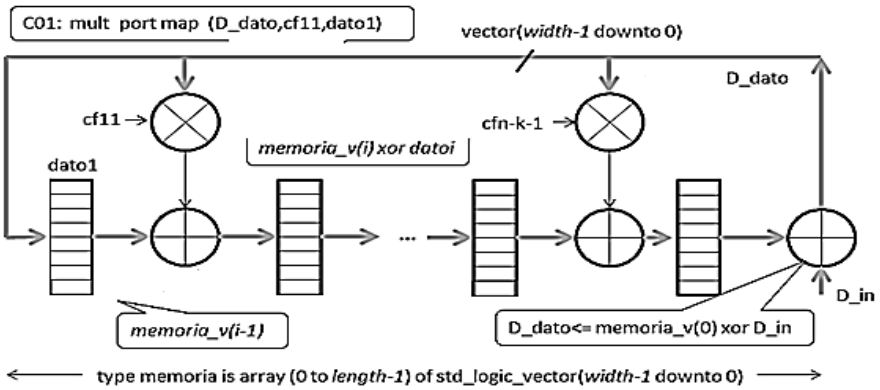


Figura 3.1. Esquema del Circuito del Codificador Reed Solomon (*n*,*k*) en VHDL

Para el diseño de la arquitectura del codificador se ha considerado la alternativa de utilizar habilitadores (tri-estados) para el manejo de las etapas, de esta manera obtenemos un codificador de arquitectura configurable, el cual es el insumo fundamental para el diseño bajo el enfoque adaptativo. Éste requiere de un componente para establecer los parámetros de operación, un módulo responsable de generar los habilitadores para configurar la arquitectura del codificador. En la tabla 3.3, se presenta el código en VHDL de la asignación de habilitadores del módulo de configuración.

Tabla 3.3. Descripción VHDL del Módulo de configuración del codificador

```
process (k)
begin
   case k is
   -- RS(255,247)
   when "11110111" => hab1<='1'; hab2<='0'; hab3<='0'; hab4<='0';
   -- RS(255,239)
   when "11101111" => hab1<='1'; hab2<='1'; hab3<='0'; hab4<='0';
   -- RS(255,231)
   when "11100111" => hab1<='1'; hab2<='1'; hab3<='1'; hab4<='0';
   -- RS(255,223)
   when "11011111" => hab1<='1'; hab2<='1'; hab3<='1'; hab4<='1';
   -- casos no válidos
   when others     => hab1<='0'; hab2<='0'; hab3<='0'; hab4<='0';
   end case;
end process;
```

A través de este módulo se establecen los parámetros que serán entrada para el codificador, según el estándar seleccionado por medio del número de símbolos de información *k*, así una vez establecidos los habilitadores, la arquitectura del codificador quedará ajustada a las etapas correspondientes. El polinomio generador de redundancia ha sido implementado sobre el LFSR configurable, los coeficientes asociados están fijados en la descripción del codificador al asignar los puertos del componente multiplicador. Es de hacer notar que se ha asignado la habilitación por grupos de multiplicadores, quedando los restantes sin implementar, para evitar la asignación paramétrica de los coeficientes por su efecto sobre el diseño al incorporar un mayor número de señales, lo cual fue el primer planteamiento que se estudió. En la tabla 3.4, se presenta el código en VHDL parametrizado para el *codificador_RS(255,k)*.

Tabla 3.4. Descripción Modular en VHDL del Codificador RS(255,k)

```
Begin
-- Módulo de multiplicadores para k=247
C01:  mult  port map  (hab1,D_dato,cf11,dato1);
…
C08:  mult  port map  (hab1,D_dato,cf18,dato8);
-- Módulo de multiplicadores para k=239
C01:  mult  port map  (hab2,D_dato,cf21, dato1);
…
C16:  mult  port map  (hab2,D_dato,cf216,dato16);
...
process (clk)
variable memoria_v:memoria:=(others=>"00000000");
begin
   if (hab='1')  then
      D_dato<=( memoria_v(0) xor D_in);
   else
      D_dato<="00000000";
   end if;
   if (clk'event and clk='1')then
   -- Primera etapa para el RS(255,247)
   if hab1='1' then
      memoria_v(0):= memoria_v(1)xor dato1;
      ...
      memoria_v(7):= memoria_v(8)xor dato8;
   end if;
   -- Segunda etapa para el RS(255,239)
   if hab2='1' then
      memoria_v(8) := memoria_v(9)xor dato9;
      ...
      memoria_v(15):= memoria_v(16)xor dato16;
   end if;
   -- Tercera etapa para el RS(255,231)
   if hab3='1' then
      memoria_v(16):= memoria_v(17)xor dato17;
      ...
      memoria_v(23):= memoria_v(24)xor dato24;
   end if;
   -- Cuarta etapa para el RS(255,223)
   if hab3='1' then
      memoria_v(24):= memoria_v(25)xor dato25;
      ...
      memoria_v(31):= memoria_v(32)xor dato32;
   end if;

   if hab='1' then
      salida<=D_in;
   else
      salida<=memoria_v(0);
   end if;
end process;
```

Una vez programados los módulos bajo sintaxis VHDL se procedió a la simulación, a través del software ModelSim XE III 6.3c, con lo que se validó el comportamiento del modelo con respecto al comportamiento teórico.

3.3. INTERPRETACIÓN DE EFICIENCIA DEL DISEÑO

En el diseño del codificador RS(*n*,*k*), se aplicó la técnica *clock_gating*, empleada para el control de la señal de reloj (Allen, 2008). Con el propósito de disminuir las transiciones innecesarias y así optimizar el consumo de potencia lo que corresponde a colocar la condición (*clk'event and clk='1'*) *and h_clk='1'*, con lo cual la señal de reloj depende de la habilitación de reloj *h_clk*.

Hasta esta etapa del diseño, se ha tratado al multiplicador de campos finitos como un componente, haciendo uso de la característica jerárquica propia del lenguaje VHDL, en el cual se describe el sistema de forma estructurada y luego se describe el comportamiento detallado de sus componentes, metodología conocida como *top-down*. Para el diseño del multiplicador concurrente (Sandoval, 2010a), se establece un conjunto de técnica de optimización, como lo son: modularidad para evaluar el efecto de cambios en los componentes de manera individual, paralelización del componente de reducción modular, re-organización de las señales de los operandos y particularización de operadores de acuerdo al polinomio generador específico del campo.

De todas estas técnicas, resulta un multiplicador basado en un modelo de LFSR concurrente, el cual emplea como base el operador concatenación. Con lo que se generan de forma combinacional, los arreglos binarios a la salida de la operación *mod_P(x)* y se logra una optimización significativa. Estos multiplicadores permiten obtener el resultado de la operación de forma instantánea, manejando un modelo de arreglos parciales que coexisten en el tiempo. Así mismo, el número de multiplicadores operativos está dado por el número de etapas activas, por tal razón son habilitados con un *hab_n*, logrando así que el dispositivo no consuma recursos en la tablas de búsqueda lógica, ni consuma potencia en cálculos que no son utilizados en el generador de redundancia.

OPERACIONALIZACIÓN DE LAS VARIABLES DE EFICIENCIA

Para la medición de la eficiencia del modelo propuesto, se plantea el análisis de los recursos utilizados por el diseño. Esto a través de las variables suministradas por las herramientas de desarrollo provistas en el paquete del IDE de Xilinx y la herramienta para el cálculo del consumo de potencia de los diseños, conocida como *XPower Analizer*, de la misma casa fabricante de dispositivos FPGAs. De esta manera se obtuvieron los reportes en el contexto de la presente investigación. La operacionalización de las variables consideradas se resume en la tabla 3.5.

Tabla 3.5. Operacionalización de las variables de la investigación

Variable	Consumo de recursos de Hardware en el diseño y Potencia
Tipo de Variable	Cuantitativa
Operacionalización	Forma en que se utilizan los recursos del dispositivo
Categorías	Consumo de los Slices
	Consumo de las LUTs
	Consumo de los Registros
Definición	Número de recursos utilizados del dispositivo - Oferta / Demanda
	Consumo de Potencia Dinámica (mW) asociada al diseño
Indicador	Porcentaje de utilización respecto a los recursos
Nivel de Medición	De razón
Unidad de Medida	% de utilización
	# de compuertas, # de elementos del FPGA de Slices, LUTs y Registros
	mW consumidos por el diseño en sus etapas
Índice	Índice de Consumo o Utilización
Valor	Los resultantes en el reporte de síntesis

3.4. MODELADO DEL LFCS

Paralelamente al diseño del codificador, se diseñó el multiplicador en campos finitos. Para ello, se estudió la representación matemática que describe la operación multiplicación sobre campos finitos, buscando su implementación con el menor consumo de hardware, para optimizar la descripción en VHDL y definir el comportamiento concurrente.

Los módulos para procesamiento de datos en algebra de campos finitos GF (2^m) pueden ser programados en software, a través de algoritmos para procesamiento secuencial o soluciones sobre hardware para modelos concurrentes, siendo este último el modelo que se plantea implementar. Una alternativa para la implementación de este modelo está basado en tablas de búsqueda, permitiendo cambiar la representación de los elementos del campo *GF(2^m)*, (Morelos-Zaragoza, 2002; Sandoval & Fedón, 2008c), o bien, a través de un modelo circuital, en función de lo cual se propone el diseño de los módulos basados en el modelo matemático e interpretación de comportamiento del circuito. Para tal fin, se analizó la arquitectura de la etapa de reducción modular con base en el LFSR, el circuito de implementación es el presentado en la figura 3.2.

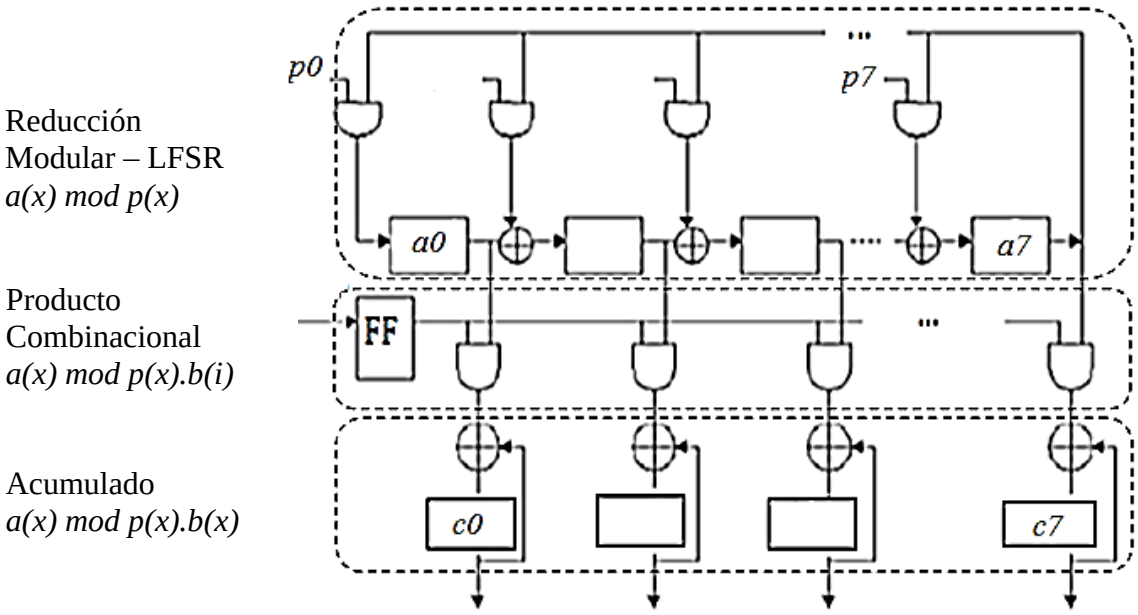


Figura 3.2. Esquema del Circuito de Multiplicador GF(2^m)
Fuente: (García-Martínez et al., n.d.)

A partir del modelo circuital y la construcción de términos en función del instante y la posición espacio, se realiza la descripción del comportamiento del módulo divisor de

polinomios, para obtener los vectores *a1….a8*, el resultado de *A(x) mod P(x)* usando concatenación en lugar del procesamiento secuencial. La implementación se reduce al uso de compuertas AND entre cada coeficiente de *p(x)*, y el uso de compuertas XOR que operan con los resultados de los coeficientes parciales, todo esto en el mismo pulso de reloj. De esta manera, se logra la sintaxis de configuración para una versión paralelizada de este multiplicador en VHDL para la cual se han eliminado los componentes secuenciales y estos han sido remplazados por variables que dependen de funciones combinacionales.

Para la interpretación y análisis del dispositivo LFSR bajo su estructura secuencial, se procedió a describir cada secuencia generada en un instante de tiempo *t*, con lo que se obtienen los estados asociados, de esta forma se logra la representación matemática del LFSR secuencial. En la figura 3.3, se presenta el circuito del LFSR a nivel de compuertas lógicas y elementos de memoria, de forma general para cualquier *p(x)*.

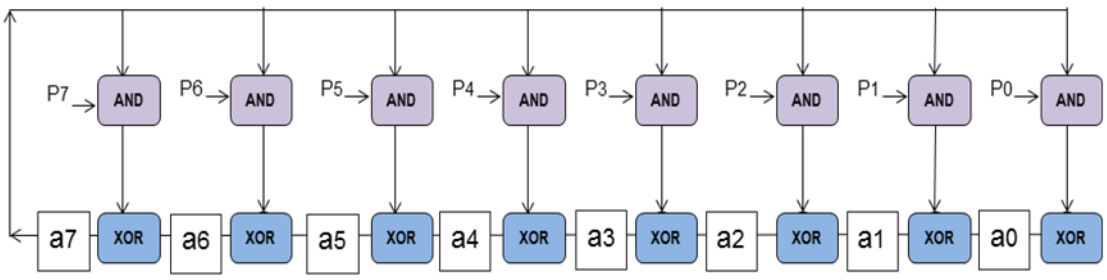


Figura 3.3. Arquitectura a nivel de compuerta del LFSR para el cálculo de la reducción modular con p(x) ajustable

En este punto, se procedió a describir el comportamiento del circuito para cada pulso de reloj, en función de los índices de posición correspondientes a cada elemento de memoria y las operaciones lógicas que se realizan para la construcción de los términos, de donde resulta la descripción dada en la tabla 3.6

Tabla 3.6. Descripción del circuito de reducción modular concurrente

```
u0: a1<= coef;

u1: a2<=(a1(6) xor (a1(7) and p(7)))
      & (a1(5) xor (a1(7) and p(6)))
      & (a1(4) xor (a1(7) and p(5)))
      & (a1(3) xor (a1(7) and p(4)))
      & (a1(2) xor (a1(7) and p(3)))
      & (a1(1) xor (a1(7) and p(2)))
      & (a1(0) xor (a1(7) and p(1)))
      &            (a1(7) and p(0));

...

-- La expresión del generador de código VHDL, para t desde 1 hasta m-1,
está dado por:
-- ut-1: at <= &((at-1(i-1)xor (at-1(m-1) and p(i)), para i=m-1 hasta 0
```

A partir de la descripción en VHDL, se logra reproducir el LFSR de forma paralela. Estos datos permiten generalizar las ecuaciones. En el instante *t=1* el vector resultante será la entrada de datos, *D_dato*, en el instante *t=2* se tiene el vector parcial de la operación del circuito, mostrado en la ecuación 3.1.

$$a_t = a_{t-1}(6) \text{ and } a_{t-1}(7) \text{ xor } p(7), \ldots, a_{t-1}(0) \text{ xor } a_{t-1}(7) \text{ and } p(0), a_{t-1}(7) \text{ and } p(8) \quad (3.1)$$

Donde las operaciones están definidas por las operaciones lógicas *AND y XOR*, debido a que los operandos son elementos binarios, que corresponden a los coeficientes de los polinomios *A(x)* y *p(x)* respectivamente. Analizando las salidas temporales del circuito durante su operación se obtiene una matriz de ecuaciones, las cuales dependen del ciclo de reloj y la posición del vector, éstas permiten el cálculo de los términos generados por el circuito LFSR. Así la relación matemático-lógica para cada uno de los elementos generados por el componente LFSR son presentados en la tabla 3.7.

Tabla 3.7. Cálculo de coeficientes de polinomios parciales $A(x)x^i mod\ p(x)$

t	*i=7*	*i*	*i=1*	*i=0*
1	$a_0(7)$	...	$a_0(1)$	$a_0(0)$
2	$a_1(6)$ xor $a_1(7)$ and $p(7)$	$a_1(i\text{-}1)$ xor $a_1(7)$ and $p(i\text{-}1)$	$a_1(0)$ xor $a_1(7)$ and $p(0)$	$a_1(7)$
3	$a_2(6)$ xor $a_2(7)$ and $p(7)$	$a_2(i\text{-}1)$ xor $a_2(7)$ and $p(i\text{-}1)$	$a_2(0)$ xor $a_2(7)$ and $p(0)$	$a_2(7)$
...	...	...	...	...
8	$a_7(6)$ xor $a_7(7)$ and $p(7)$	$a_7(i\text{-}1)$ xor $a_7(7)$ and $p(i\text{-}1)$	$a_7(0)$ xor $a_7(7)$ and $p(0)$	$a_7(7)$

En esta tabla se puede observar la relación entre los términos al aplicar la operación de cada rama del circuito LFSR sobre el polinomio *A(x)*. Donde cada rama está representada por el coeficiente del polinomio generador del campo *p(x)* en la posición *i*, correspondiente a *p(i)*, el cual será fijo en el tiempo. Dado que la operación de reducción modular genera un conjunto de *m* residuos parciales, en función de los cuales se puede describir su comportamiento, se plantea parametrizar la función generadora de cada elemento. De esta manera, el método a emplear consiste en asociar los sub-indices particulares con la relación de los parámetros dados por la posición *i* y el instante de tiempo *t*.

Basados en esta técnica de modelado se establecen los coeficientes del polinomio residuo en función del tiempo, como: $a_t\ (i)$; el coeficiente del polinomio en la posición más significativa, como: $a_{t-1}\ (m\text{-}1)$, que representa el bit más significativo realimentado en el circuito, en un instante de tiempo *t-1* y $a_{t-1}\ (i\text{-}1)$, que corresponde al bit precedente al elemento calculado. A partir de este procedimiento se desarrollaron las ecuaciones que describen el comportamiento del circuito LFSR para su descripción paralela, a fin de lograr la implementación de la estructura concurrente para VHDL.

Luego, se planteó la simplificación del circuito, definiendo éste en función del polinomio característico. De manera que el primer paso corresponde a la selección del polinomio irreducible o primitivo para la generación del campo, estos polinomios son propios de la longitud *m* del campo GF(2^m) y son presentados en detalle en (Delgado, 2010). Los criterios que deben cumplir es que divida los elementos del cuerpo finito y no puede ser escrito como el producto de dos polinomios del campo (Peralta, 2005).

Un polinomio que cumple con tal condición es el polinomio *P(x)* = 285, el cual ha sido seleccionado, por tratarse de un polinomio generador de campo GF ampliamente utilizado en codificadores RS estándar (Xilinx, 2012). En este punto se estudiaron los pasos del cálculo de la multiplicación en campos finitos, para describirlos de forma concurrente en lenguaje VHDL a fin de obtener el modelo para la implementación hardware del multiplicador. De allí que, tomando el polinomio generador del campo en su forma binaria, *p(x)=100011101*, que es un pentanomio de grado *m = 8*, se obtiene el modelo particularizado, mostrado en la figura 3.4.

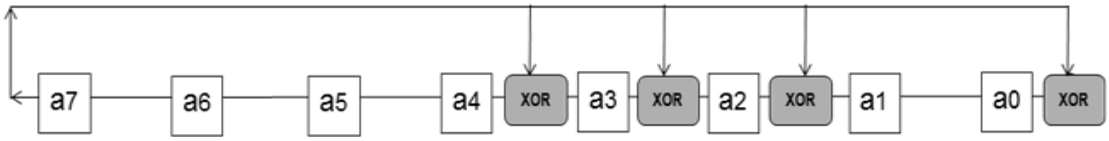


Figura 3.4. Modelo del LFSR para p(x)=285

Este circuito simplificado del componente de reducción modular, presenta un ahorro en el consumo de recursos del dispositivo. La descripción en VHDL del modelo circuital LFSR concurrente que emplea el multiplicador, se genera a partir de una simplificación de la tabla 3.7, por la cancelación directa de las ramas cuyos coeficientes son "0" en el polinomio irreducible seleccionado, resultando el código VHDL de la estructura simplificada, presentado en la tabla 3.8.

Tabla 3.8. Descripción VHDL concurrente del Componente Divisor basado en LFSR

```
p<="100011101"; -- Primitive polynomial = D^8+D^4+D^3+D^2+1 (285)
u0: a1<=D_dato
-- solo para los términos cuyo coeficiente de p(x) es distinto de cero:
u1:  a2<=a1(6  downto  4)&(a1(3)xor  a1(7))&(a1(2)xor  a1(7))&(a1(1)xor
a1(7))&a1(0)& a1(7);
...
u7:  a8<=a7(6  downto  4)&(a7(3)xor  a7(7))&(a7(2)xor  a7(7))&(a7(1)xor
a7(7))&a7(0)& a7(7);
```

En la tabla 3.9, se ilustra la representación lógica-temporal del comportamiento del circuito (3.2), en función de los términos generados en el multiplicador de campos finitos, los cuales cumplen el modelo matemático definido por la ecuación *A(x) mod*

P(x).B(x). Siendo su interpretación la que permite la descripción del comportamiento en VHDL, con lo que se puede re-estructurar del comportamiento secuencial conocido al comportamiento concurrente propuesto.

Tabla 3.9. Implementación matemática de C(x)=A(x) mod P(x).B(x)

LFSR	$a_{t,7}$	$a_{t,6}$	$a_{t,5}$	$a_{t,4}$	$a_{t,3}$	$a_{t,2}$	$a_{t,1}$	$a_{t,0}$
and	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	$c_{0,7}$	$c_{0,6}$	$c_{0,5}$	$c_{0,4}$	$c_{0,3}$	$c_{0,2}$	$c_{0,1}$	$c_{0,0}$
	$c_{1,7}$	$c_{1,6}$	$c_{1,5}$	$c_{1,4}$	$c_{1,3}$	$c_{1,2}$	$c_{1,1}$	$c_{1,0}$
	…							
	$c_{t,t}$	$c_{i,t}$	$c_{i,t}$	$c_{i,t}$	$c_{i,t}$	$c_{i,t}$	$c_{i,t}$	$c_{i,t}$
xor	$c_{7,7}$	$c_{7,6}$	$c_{7,5}$	$c_{7,4}$	$c_{7,3}$	$c_{7,2}$	$C_{7,1}$	$c_{7,0}$
	d7	*d6*	*d5*	*d4*	*d3*	*d2*	*d1*	*d0*

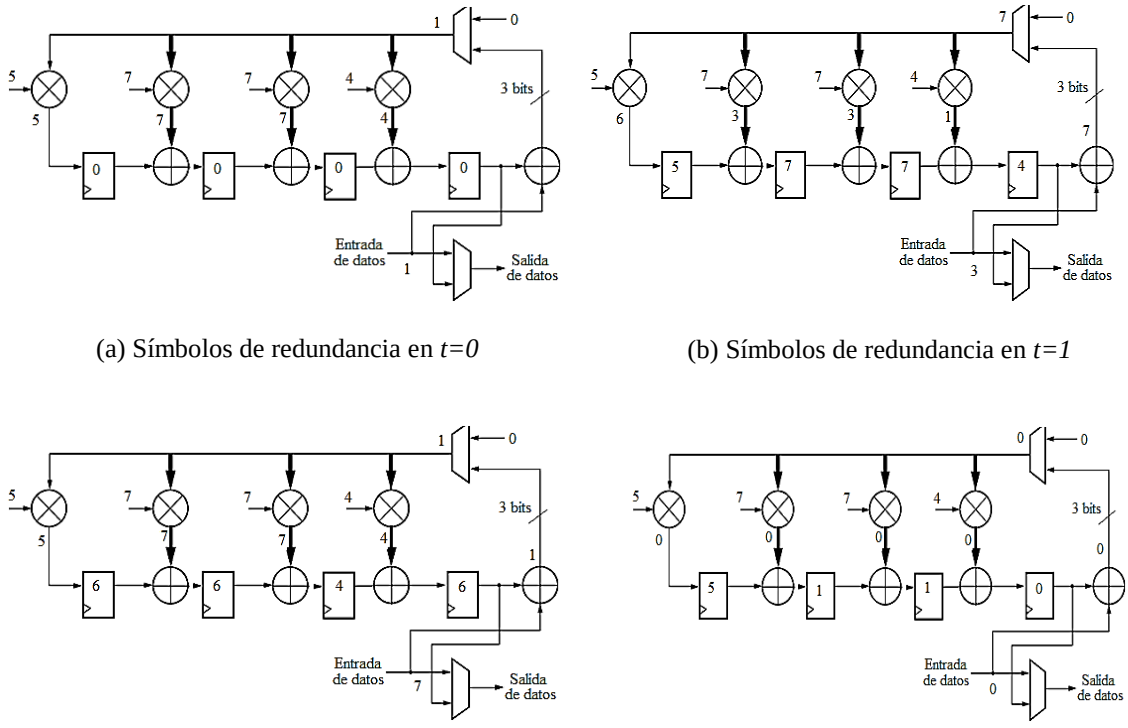
Los productos necesitan ser implementados con compuertas *AND* entre los elementos de las salidas parciales del LFSR, que han sido denominadas $a_{t,i}$, un vector de *m* elementos, con el elemento de datos b_t, siendo el bit en la posición *t* de la entrada *B*, en forma paralela bit a bit, a fin de obtener los vectores resultantes dados por $c_{i,j}$, que corresponde directamente a $a_{t,i}$, en caso de que el bit en la respectiva posición de b_t sea igual a '1', en caso contrario se anulará, por lo cual es definido un vector B_t como la concatenación del elemento b_t, *m* veces, los resultados parciales son operando a través de compuertas XOR los resultados para el producto final, como se describe en la tabla 3.10.

Tabla 3.10. Descripción del producto combinacional y acumulador en VHDL

```
-- De la descripción detallada se parametriza la generación de
descripción de los componentes como:
-- Generación de términos del LFSR:
-- ut:  at+1 <= at(i) xor (at(m-1) and p(i) & ... for i=m-1 to 0
-- Construcción del operador bt:
b1<= b(0) & b(0) & b(0) & b(0) & b(0) & b(0) & b(0) & b(0);
...
b8<= b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7);
-- bt <= b(t-1)& b(t-1)& b(t-1)& b(t-1)& b(t-1)& b(t-1)& b(t-1)& b(t-1)
-- Productos Parciales:
c1<=a1 and b1;
...
c8<=a8 and b8;
-- ct <= at and bt
-- Acumulador de Productos Parciales:
datox<=c1 xor c2 xor c3 xor c4 xor c5 xor c6 xor c7 xor c8;
```

3.5. APLICACIÓN DEL LFCS PARA EL GENERADOR DE REDUNDANCIA

A partir del análisis del circuito del RS(7,3) se estudió el comportamiento del codificador Reed Solomon secuencial, se generó cada vector de símbolos a través de los *n* pulsos de reloj, como se presenta en la figura 3.5.



(a) Símbolos de redundancia en *t=0*

(b) Símbolos de redundancia en *t=1*

(c) Símbolos de redundancia en *t=2*

(d) Símbolos de redundancia en *t=3*

Figura 3.5. Salidas del generador de Símbolos de Redundancia del Codificador RS(*7,3*)

De esta manera, con la observación del comportamiento del generador de símbolos de redundancia del codificador Reed Solomon, se obtienen las ecuaciones en el tiempo de cada símbolo de redundancia y se reconoció un arquitectura coincidente con el circuito de reducción modular, donde existe una correspondencia entre los coeficientes del polinomio generador que define el código Reed Solomon y los coeficientes del polinomio irreducible que define el campo de Galois, motivo por el cual se consideró

evaluar la extensión del modelo concurrente del LFCS desarrollado en la paralelización del multiplicador en campos de Galois (Sandoval, 2010a), al circuito generador del código. Así la relación lógica-matemática para cada uno de los elementos generados por el componente LFSR se describe en la tabla 3.11, donde el operador $\otimes$ corresponde a la operación producto y el operador $\oplus$ a la suma en GF.

Tabla 3.11. Cálculo de los símbolos del codificador RS(n,k)

t	*i=n-k*	*i*	...	*i=1*
0	*0*	*0*	...	*0*
1	$r_1(n-k-1) \oplus d(t) \otimes g(n-k)$	$r_1(i-1) \oplus (d(t) \otimes g(i))$	...	$d(t) \otimes g(1)$
2	$r_2(n-k-1) \oplus d(t) \otimes g(n-k)$	$r_2(i-1) \oplus (d(t) \otimes g(i))$	...	$d(t) \otimes g(1)$
...	...	...	...	...
n-k	$r_{n-k}(n-k-1) \oplus d(t) \otimes g(n-k)$	$r_{n-k}(i-1) \oplus (d(t) \otimes g(i))$	...	$d(t) \otimes g(1)$
...	...	...	...	...
n	$r_n(1)$	*0*	...	*0*

En la tabla 3.12 podemos observar la descripción VHDL del codificador diseñado.

Tabla 3.12. Descripción VHDL del codificador paralelo RS(7,3)

```
--LFSR CONCURRENTE para generación de símbolos de Redundancia

r1(3)<=m14;              r2(3)<=m24;              r3(3)<=m34;
r1(2)<=r0(3) xor m13;    r2(2)<=r1(3) xor m23;    r3(2)<=r2(3) xor m33;
r1(1)<=r0(2) xor m12;    r2(1)<=r1(2) xor m22;    r3(1)<=r2(2) xor m32;
r1(0)<=r0(1) xor m11;    r2(0)<=r1(1) xor m21;    r3(0)<=r2(1) xor m31;

--Salidas del Encoder RS
s <= e1 & e2 & e3 & r3(0) & r3(1) & r3(2) & r3(3);
-- se reordenaron R(x) de LSB … MSB
end Behavioral;
```

El método de descripción modular de los componentes permitió validar las etapas del diseño y depurar las ecuaciones para el modelo del multiplicador concurrente, la herramienta de desarrollo utilizada para la descripción corresponde al *ISE 11 webPack* de Xilinx, y el software para la simulación empleado fue el *ModelSim XE 6.1*, obteniendo las señales simuladas para la evaluación del comportamiento, así mismo se estudiaron los reportes de utilización de recursos y el análisis de consumo de potencia con la herramienta *PowerAnalizer* de Xilinx.

3.6 CONCLUSIONES EN FUNCIÓN A LOS MÉTODOS EMPLEADOS

A partir del diseño metodológico descrito, se planificó la relación entre los objetivos y resultados como se presenta en la tabla 3.13.

Tabla 3.13. Relación entre los objetivos y métodos planteados

OBJETIVO	Diseñar un codificador RS(255,k) para optimización	Interpretar la síntesis del diseño del codificador y los reportes de consumo de recursos	Obtener el modelo de ecuación descriptivas	Sistematizar el modelo del LFSR para las aplicaciones de los componentes
FUND.	Análisis de comportamiento y parametrización	Métodos de optimización de hardware	Operadores VHDL y concepto de concurrencia	Reconocimiento de estructuras auto-similares y principio de correspondencia
MÉTODOS	Formulación de sintaxis usando ISE11 de Xilinx para su	Ponderación de indicadores, a través de tablas y gráficos	Generalización de ecuaciones, a partir de tablas de ecuaciones instantáneas del LFSR	Aplicación de modelo LFSR paralelizado
RESULT.	Comportamiento del codificador RS, simulado con Modelsim XE	Eficiencia del diseño del codificador RS	Sistema modelado para descripción VHDL	Re-estructuración del codificador RS paralelizado

Esto permitió establecer conclusiones de análisis teórico – práctico, considerando que:

- Un diseño puede ser abordado desde diversos *niveles de abstracción*, se ha considerado importante en esta investigación partir de los modelos conocidos, estudiando su descripción matemática y diagramas circuitales para interpretar su arquitectura y comportamiento, a fin de identificar estructuras *optimizables* en el nivel de diseño seleccionado de configuración de hardware basado en las tendencias en implementación de hardware. Este procedimiento se realizó a través de casos de estudios simplificados de manera modular.

- La aplicación del operador "*Concetenación*" para el manejo de las señales parciales de la estructura de corrimientos secuenciales, con el fin de describir su comportamiento de forma concurrente, ha sido considerada como eje central en el proceso de modelado.

- Las ecuaciones de descripción VHDL del multiplicador GF y del codificador RS, usando la *parametrización*, como técnica para establecer un modelo único y con posibilidad de adaptación desde el componente jerárquico superior, es aplicada como técnica de generalización. En este caso, a través de la configuración en VHDL de las estructuras del multiplicador GF, se logró identificar instrucciones comunes en la descripción, que fueron definidas en función de su posición en el vector y el instante de tiempo, con lo que se tabularon las ecuaciones descriptivas a nivel matemático – lógico de los componentes.

- La identificación del isomorfismo entre la estructura LFSR del componente de reducción modular del multiplicador GF y del generador de redundancia en el codificador RS, permitió aplicando el principio de correspondencia, extender un modelo de codificador Reed Solomon concurrente para el caso RS(7,3), con lo que se prueba el funcionamiento del modelo concurrente.

- Dada la importancia de la optimización del consumo de potencia para el rendimiento de la fuente de alimentación, se estudiaron las técnicas de optimización en diseños electrónicos de arquitectura flexible, basadas en identificar los factores que permiten re-orientar el diseño hacia un procesamiento más rápido y menos complejos, lo que permitió aplicar algunas técnicas de paralelización y manejo de señales en el desarrollo del modelo, ya que un mismo componente puede ser descrito exitosamente bajo diversos operadores y parámetros, teniendo siempre presente que queda abierta la posibilidad de optimizaciones futuras, con base en los objetivos definidos.

- Del estudio del consumo recursos y potencia dinámica, en base a los indicadores seleccionados en la operacionalización de variables, para confirmar el impacto del modelo VHDL desarrollado, se empleó como primer método la estimaciones teóricas, a partir del modelo y comparado con los antecedentes en optimizaciones de componentes. Seguidamente, se realizó una estimación práctica fundamentada en la herramienta de desarrollo, lo que permitió contrastar los reportes generados.

CAPÍTULO VI

MODELO VHDL DEL CODIFICADOR RS

En este capítulo se presentan los resultados del modelo propuesto del codificador Reed Solomon sobre FPGA, evaluando el grado de eficiencia de los componentes del codificador, considerando la utilización de recursos de hardware y el consumo de potencia asociado al diseño, con lo que se logra validar éste, mostrando el enfoque concurrente del multiplicador GF, así como los resultados de la disertación doctoral.

RESULTADOS DEL MODELO OPTIMIZADO DEL CODIFICADOR RS(N,K) PARA SISTEMAS RECONFIGURABLES

En esta sección se presentan los resultados asociados a cada uno de los objetivos planteados hasta alcanzar el modelo optimizado del codificador Reed Solomon, bajo descripción de hardware para sistemas reconfigurables, para ello se realizó la descripción en VHDL de cada uno de los codificadores (Anexo A), observando su comportamiento y aplicando técnicas de optimización en la descripción de la sintaxis de los multiplicadores (Anexo B).

4.1. COMPORTAMIENTO DEL CODIFICADOR RS(*N,K*) DISEÑADO

Se crearon los bancos de prueba de los componentes para observar la respuesta de los módulos diseñados. Para validar el multiplicador GF, basado en LFSR fueron considerados: el parámetro *m*=8, con un polinomio generador de campo *p(x)=285*, y los coeficientes 104 y 13 respectivamente, la multiplicación de los primeros 16 elementos del campo finito *GF (256),* generaron los resultados esperados, de forma paralela, como se observa en la figura 4.1.

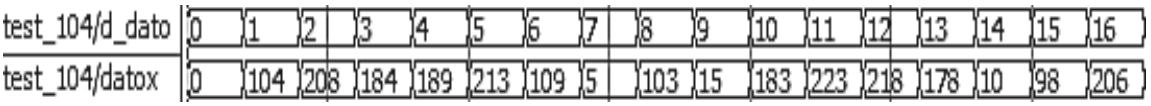


Figura 4.1.a. Multiplicación sobre GF(256) con el coeficiente 104

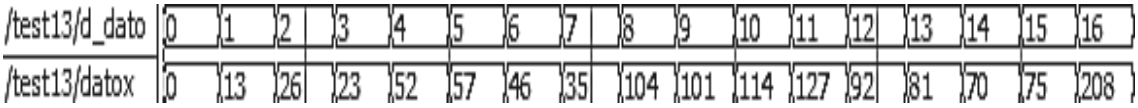


Figura 4.1.b. Multiplicación sobre GF(256) con el coeficiente 13

Probado el funcionamiento de los componentes multiplicadores a emplear en el codificador, se validó el funcionamiento del codificador RS(*255,k*), para los casos estudiados. Con este propósito, se estableció el parámetro *k* variable, al cual se le asignó los valores 223, 239 y 247 respectivamente, empleando el polinomio generador de código RS para cada caso. Los coeficientes del polinomio *G(x)* son asignados en la etapa del generador de redundancia del codificador, para su operación con los datos *D(x)*, dicha entrada de datos toma valores de 1 a *k* de forma creciente, establecidos estos en el *test_bench*. En cada caso se obtienen los *255-k* símbolos de redundancia correspondientes. La simulación fue realizada a través de la herramienta ModelSim XE III 6.3c, incorporada en el ISE de Xilinx, obteniendo que la palabra de código generada coindice con el resultado esperado de acuerdo al comportamiento teórico.

Se presentan la validación del codificador *RS*(*255,223*), en la figura 4.2.

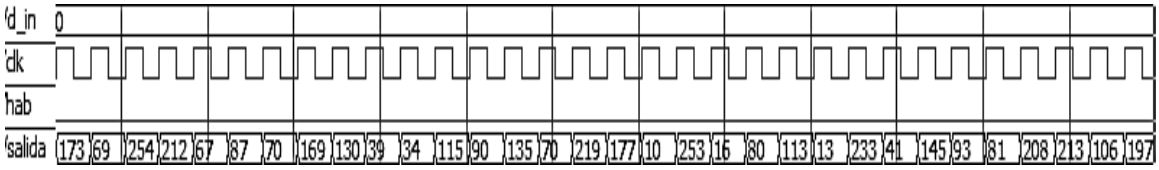


Figura 4.2. Resultados del codificador RS(255,223)

Para el codificador RS(*255,239*), se validó el comportamiento, donde se obtuvieron los 16 símbolos de redundancia, que se muestran en la figura 4.3.

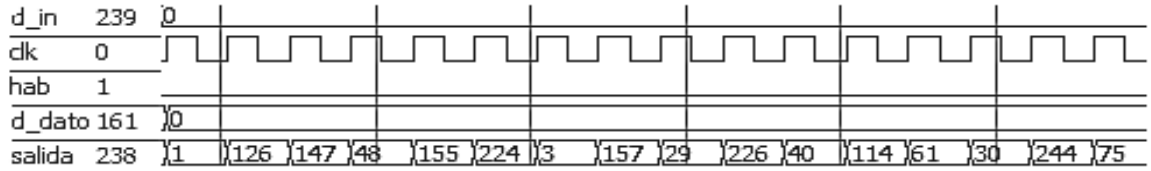


Figura 4.3. Resultados del codificador RS (255,239)

De la misma forma se realizó la simulación del codificador RS(*255,247*) donde se obtienen los resultados mostrados en la figura 4.4.

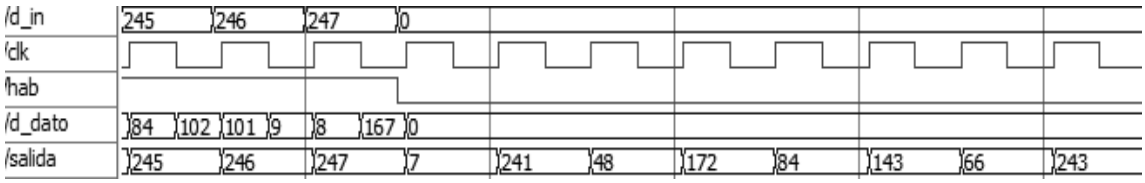


Figura 4.4. Resultados del codificador RS(255,247)

Para evaluar el comportamiento del modelo para un valor de m≠8, se realizó la simulación del RS(*127,111*), siendo para este caso *m*=7 bits por símbolos y el polinomio generador del campo *P(x)=10001001*, correspondiente a 137 en decimal. Este código presenta 16 símbolos de redundancia, los cuales corresponden teóricamente con el vector [38 81 22 83 107 48 90 67 40 100 33 92 27 110 3 55], en la simulación presentada en la figura 4.5 se observa que estos coinciden con los resultados obtenidos.

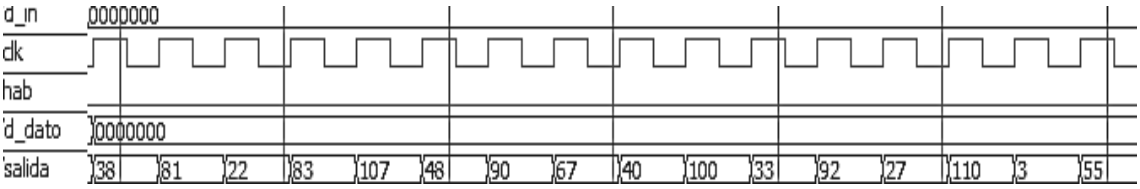


Figura 4.5. Resultados del codificador RS(127,111)

Una vez validado el funcionamiento del codificador Reed Solomon para *n*, *k* y *m* variables, se pudo observar que el codificador Reed Solomon parametrizable funciona de acuerdo a la descripción circuital en VHDL. La simulación permite observar que en la versión secuencial, el número de ciclos de reloj requeridos dependen del parámetro *n*. La arquitectura del generador de redundancia, implementado con el LFSR (*Linear Feedback Shift Register*) ha sido definida de forma modular, con lo que los recursos consumidos por el diseño dependen específicamente de las etapas implementadas.

4.2. EFICIENCIA DEL CODIFICADOR RS(*N*,*K*) OPTIMIZADO

Los primeros postulados que se consideraron en la optimización fueron; (i) los circuitos más rápidos son los que menos consumen potencia (Genser, Bachmann, Steger, Hulzink, & Berekovic, 2009; Todorovich, 2006), de manera tal que el número de pulsos de reloj es la variable de comparación, teniendo en cuenta que el *min clk period* es el inverso de la frecuencia máxima de operación que determina el tiempo de respuesta del circuito; (ii) los circuitos tendrán un consumo de potencia proporcional a la complejidad computacional o profundidad lógica del diseño (Biard & Noguet, 2008), de manera que la optimización en recursos de hardware permiten estimar la comparación del consumo de potencia del diseño. De manera que la medición de la eficiencia será dada por el número de compuertas y también por los retardos totales del circuito (Halbutogullari & Koc, 2000).

OPTIMIZACIÓN POR PROCESAMIENTO PARALELO

Para el cálculo del consumo de recursos del multiplicador paralelo, en primer lugar, se estimó el número de compuertas de acuerdo a la descripción del LFCS, esto contabilizando el número de operadores *AND* y *XOR,* utilizados en la tabla 3.9, y asociándolos con el número de bits del multiplicador *m*. Igualmente, se obtuvieron los reportes de síntesis del componente LFCS, a fin de evaluar el consumo de recursos de hardware del diseño. Para ello, se tomó como indicador el número de Slices, los cuales están basados en LUTs de 4 entradas para el dispositivo seleccionado. Estos datos son presentados en la tabla 4.1

Tabla 4.1. Reporte de Utilización para el LFCS con p(x) ajustable

Recursos del LFSR	AND	XOR	Slices	LUTs
Modelo para *m* genérico	$(m-1)^2+m$	$(m-1)^2$	-	-
LFCS con parámetro *m=8*	56	49	24	48

OPTIMIZACIÓN POR SIMPLIFICACIÓN DE OPERACIONES

De la simplificación desarrollada, donde se describió la implementación del divisor o componente de reducción modular: *A(x) mod p(x),* al definir el polinomio generador del campo *p(x),* se reduce significativamente la complejidad, quedando el número de compuertas, en función del número de bits del campo y la simplificación considerada. De allí se establecieron las ecuaciones para de estimación de compuertas, la ecuación 4.1 permite el cálculo del número de compuertas *AND* del multiplicador y la ecuación 4.2 para el cálculo de compuertas *XOR* tanto del circuito LFSR, como del arreglo producto.

$$\#AND_{mult} = m^2 \qquad (4.1)$$

$$\#XOR_{mult} = (p.m-p-2.m+2)_{LFSR} + (m^2-2.m+1)_{PROD} = m^2+p.m-4.m-p+3 \qquad (4.2)$$

Donde *m* corresponde al número de bits de cada palabra del campo y *p* corresponde al número de bits no nulos del polinomio irreducible *p(x).* En este caso la optimización

corresponde a la *simplificación de operaciones*, en función del número de coeficientes *p(x)* no nulos. La comparación entre el modelo LFCS general y simplificado es presentada en la tabla 4.2.

Tabla 4.2. Compuertas utilizadas para el LFCS con P(x) = 100011101

Compuertas del Multiplicador	*AND2*	*XOR2*	*XOR8*
LFCS general	$(m^2-2m+1) * m^2$	m^2-2m+1	*m-1*
LFCS simplificado con p(x)	m^2	*p.m-p-2m+2*	*m-1*

Los resultados obtenidos se calcularon a través de la suma de compuertas de cada uno de los tres módulos LFCS, ANDs y XORs, de la arquitectura del multiplicador GF, otras investigaciones obtienen el cálculo del número de compuertas *AND* y *XOR* aplicando algoritmos de segmentación para disminuir el número de compuertas requeridas (Saqib Nazar, 2004),(Kindap, 2004). Por otra parte, al implementar el multiplicador la herramienta de desarrollo reporta el consumo de recursos del diseño, donde se utilizaron 44 LUTs para la implementación. En la tabla 4.3 se presenta el consumo de recursos de los componentes del multiplicador.

Tabla 4.3. Reporte de Utilización del multiplicador GF con p(x) = 100011101

Recursos del Multiplicador GF	*AND2*	*and*	*XOR2*	*xor*	*Slices*	*LUTs*
R(x) = A(x) mod P(x) con m=8 y p=5	*0*	*0*	*p.m-p-2m+2*	*21*	*5*	*8*
C(x) = R(x) AND B(x)	m^2	*64*	*0*	*0*	*-*	*64*
D(x) = XOR C(x)	*0*	*0*	$m^2 - m$	*56*	*-*	*56*
Multiplicador basado en LFCS	m^2	*64*	$m^2 + p.m-p-3m+2$	*77*	*17*	*44*

Es importante destacar, que estos cálculos están basados en el procesamiento concurrente de los elementos del campo, lo cual pudiera llevar a pensar en un aumento de recursos por el procesado paralelo, en la tabla se puede observar que el presente diseño ha alcanzado una reducción del número de LUTs del multiplicador con respecto a sus componentes, a través de la simplificación propuesta. Por otra parte, estos resultados han sido comparados con la complejidad teórica reportada en trabajos previos (J. L. Imaña, Sánchez, & Fernández, 2002), que estaba en el orden de m^2-1 compuertas *xor*, al

sustituir m=8 se obtiene una optimización de un 54.05 %, a nivel de utilización de recursos hardware.

Para la evaluación del consumo de potencia del modelo propuesto, se realizó la comparación de los resultados, donde se consideraron las ecuaciones presentadas por (Biard & Noguet, 2008), calculando el consumo de potencia a partir de la complejidad computacional dada en compuertas, en tal sentido se ha realizado la estimación del diseño, como se muestra en la tabla 4.4.

Tabla 4.4. Complejidad Computacional y Consumo de Potencia

Operadores	Complejidad Computacional	Consumo de Potencia	(pJ)
[1]GFmult	*m^2 AND + 3(m-1)2/2 XOR*	*0.4 (m^2 + 3(m-1)2/2)*	55
[1]GF adder	*m XOR*	*0.4 m*	3.2
[1]GF inv	*m ROM read*	*8m*	64
[1]GF reg	*m REG write*	*2m*	16
[1]GF mem	*m RAM read and write*	*10m*	80
[2]LFSR CESR	*p.m-p-2.m+2 XOR*	*0.4 (p.m-p-2.m+2)*	8.4
[2]GFmult CESR	*m^2 AND + (p.m-p-2.m+2) (m-1) XOR*	*0.4(m^2 + (p.m-p-2.m+2)(m-1))*	39.2

Fuente: [1] Biard & Noguet, 2008; [2] Autora, 2013

La optimización del multiplicador ha sido comparada con los resultados presentados en trabajos previos, donde se reporta el consumo para diversos circuitos de multiplicadores de campos finitos de 8 bits, entre los cuales se pueden mencionar los basados en circuitos multiplica y reduce (*Multiply & Reduce*), desplazamiento y suma (*Shift & Add*) y Montgomery, resumidos en (G Sutter & Boemo, 2007), las versiones paralelas de multiplicadores Mastrovito, así como versiones serie estudiadas en (Sánchez Corrionero, 2011), de la comparación entre sus implementaciones combinacionales (Anexo C). Concluyendo que el diseño propuesto con el modelo concurrente LFCS para CESR presenta mejores resultados en eficiencia, como se muestra en la tabla 4.5.

Tabla 4.5. Comparación de Eficiencia de los modelos de multiplicadores

Modelo del Mult.	CLBs	Delay (ns)	Energía (nJ)
[1]Multiply & Reduce	85	186	96.1
[1]Shift & Add	157	201	186.4
[1]Montgomery	102	167	92.7
[1]Mastrovito	62	14.94	-
[1]Mastrovito AOP	68	17.05	-
[2]GFmult CESR (44 LUTs)	17	3.18	-

Fuente: [1] G Sutter & Boemo, 2007; [2] Autora, 2013

En (Allen, 2008) se presenta un análisis de los modelos más destacados, de donde se resume el consumo de recursos para los métodos de multiplicadores presentados, así mismo (Paar, 1996) presenta las ecuaciones para el cálculo de recursos, estos datos son presentados en la tabla 4.6.a, incluyendo los resultados de esta investigación.

Tabla 4.6.a. Compuertas empleadas por el multiplicador GF(2^m) con m=8

Compuertas	AND2	XOR2	8bit XOR8
[1]Mastrovito [A]	64	84	0
[1]Paar	48	62	0
[1]Karatsuba Clásicos	48	59	0
[2] GFmult CESR	64	21	7

Fuente: [1] Allen, 2008, [2] Autor, 2013

De donde se puede observar, que se logró optimizar el consumo de recursos hardware del multiplicador a nivel de compuertas. A nivel de recursos del dispositivo de hardware FPGA se obtiene una complejidad lógica de 3 etapas, para implementar el circuito a través de las 44 LUTs reportadas, el número de Slice está asociado al número de LUTs del FPGA como se mencionó antes, el dispositivo XC5VLX30 presenta 4 LUTs por Slice, sin embargo, el número de entradas requeridas por LUTs para el diseño son 6, 4 y 3, por lo que requieren 17 Slice, como se observa en la tabla 4.6.b. lo cual es una reducción significativa de componentes del FPGA, con respecto a los reportes previos.

Tabla 4.6.b. Recursos empleados por el multiplicador GF(2^m) con m=8

Componentes	Slice LUTs	LUTs	Reg. FF	Consumo de Potencia
[1]Ahlquist [A]	53	-	-	-
[1]Mastrovito[B]	64	58	-	2.75
[1]Paar[C]	48	53	-	3.54
[2]Multiplicador del CESR	17	44	-	0.39 (*)

Fuente: [1]Allen, 2008; [2]Autora, 2013.

Se tomó para la comparación el polinomio irreducible P(x)=100011101. (*) Consumo de Potencia Dinámica suministrada por el XPower del ISE11en mW para el FPGA XC5VLX30 [A] Mastrovito, E. D., VLSI Design for Multiplication over Finite Fields GF(2m). Proc. of Sixth International Applied Algebra, Algebraic Algorithms, and Error Correcting Codes, pp. 297-309 (July 1988) [B] Mastrovito, E.VLSI Architectures for Computation in Galois Fields. PhD thesis, Link¨oping University, Dept. Electr. Eng., Linköping, Sweden, 1991.[C] Paar, C. A New Architecture for a Parallel Finite Field Multiplier with Low Complexity based on Composite Fields. IEEE Transactions on Computers, 45(7):856- 861 (July 1996)

El siguiente paso consistió en analizar la implementación circuital del diseño, esto se realizó a través de los esquemas RTL (*Register*-Transfer *Level)* generados por la herramienta de desarrollo del ISE11, donde se pueden observar los componentes descritos en el diseño. En la figura 4.6 se presenta la arquitectura del LFSR reconfigurable resultante en el esquema sintetizado de la descripción VHDL, donde los elementos de memoria son habilitados por la señal *hab_i* dispuesta para desactivar etapas del diseño que no se encuentran en uso.

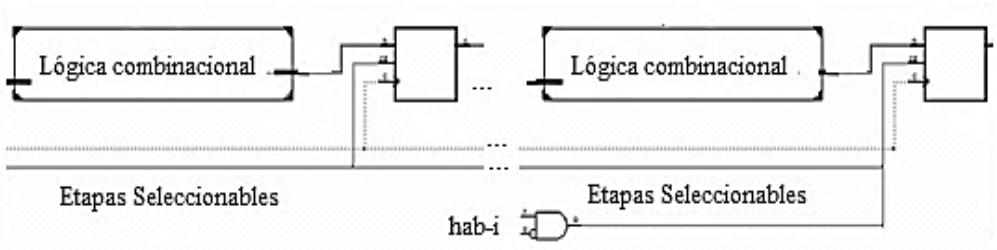


Figura 4.6. Arquitectura del LFSR configurable

Como se puede observar la señal de habilitación *hab_i* es conectada a los habilitadores de las etapas (figura 4.7), todo a nivel de hardware, aun cuando en la descripción se emplea la instrucción *IF*, su implementación no obedece a la comparación de una condición a nivel de software, por lo que ha sido tratada como una señal del circuito, permitiendo manejar la configuración del diseño.

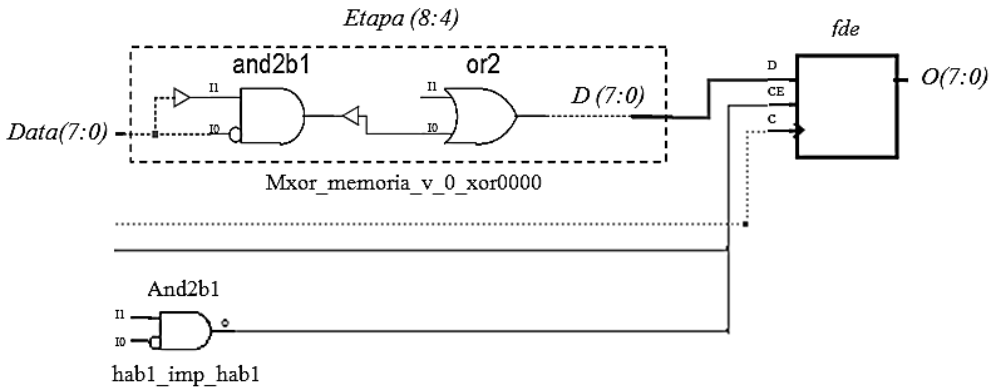


Figura 4.7. Esquemático RTL de una sección del LFSR configurable

El módulo de configuración genera las habilitaciones, con el propósito de que las etapas no implementadas para cierta configuración no consuman potencia, quedando deshabilitados los componentes que constituyen dichas etapas, dicha señal es generada de forma combinacional, como se puede observar en la figura 4.8, donde se presenta el circuito generador de la señal *hab4* a partir de la entrada *k*=239, correspondiente a la combinación binaria ”11101111”.

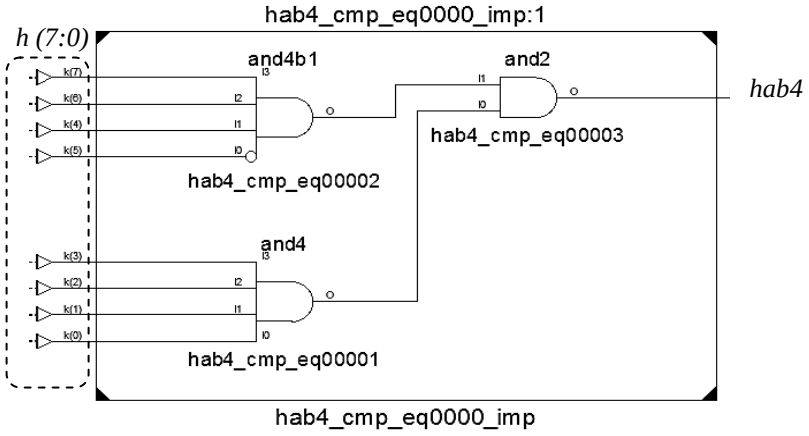


Figura 4.8. Esquemático RTL del módulo para configuración del *hab4*

Un punto no menos importante corresponde al estudio de la profundidad lógica de la implementación del diseño sobre el dispositivo FPGA, puesto que este factor incide sobre los retardos de las señales, al igual que sobre el consumo de potencia por

propagación de *glitches* (Sutter, 2007). La figura 4.9 presenta el esquemático de la tecnología para las primitivas asociadas a la salida *datox_0*, es decir los recursos de hardware en la construcción de la mencionada señal de salida, este esquema permite observar la profundidad lógica del circuito, donde se identifican 3 capas de LUTs.

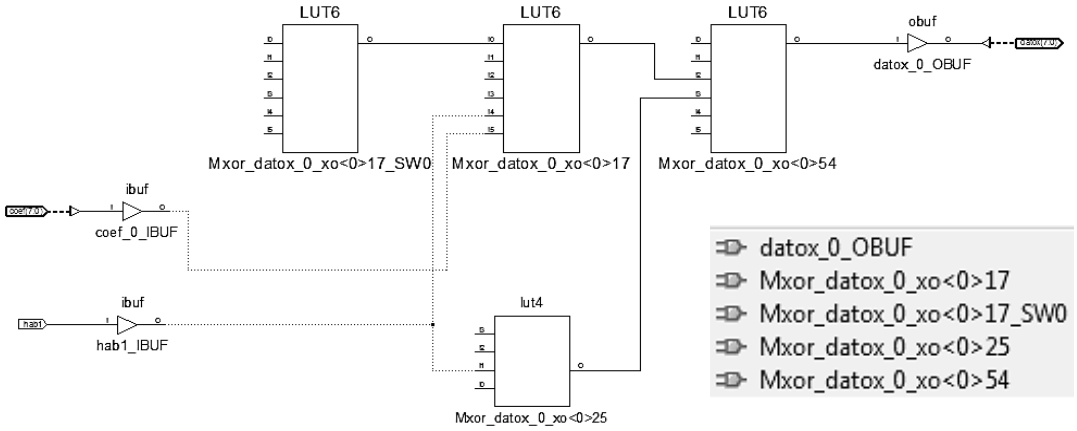


Figura 4.9. Esquemático RTL de una sección del LFSR configurable

OPTIMIZACIÓN POR ASIGNACIÓN DE SEÑALES

Luego del análisis teórico, se procedió a la obtención del consumo de potencia en mW de los módulos diseñados, usando la herramienta del *XPower Analyzer* del IDE de Xilinx, como se observa en la tabla 4.7, considerando como factor de comparación el orden de asignación de las señales para el circuito multiplicador.

Tabla 4.7. Comparación de Consumo de Potencia del multiplicador

Potencia (mW)	*Logic*	*Signal*	*Clk*	*IO*	*Dinámica*
*(A(x) mod P(x) * B(x))*	0,04	0,35	0,00	-	0,39
*(B(x) mod P(x) * A(x))*	0,04	0,38	0,00	-	0,42

En primer lugar, el consumo de potencia asociado a la lógica del circuito *Pot_Logic* en ambos casos fue de 0.04 mW, en tanto que la *Pot_Signal* presentó una variación, de acuerdo al orden de los operandos, consumiendo una potencia asociada a las señales de 0.38 mW en el caso de *B(x) mod P(x) * A(x)* y un consumo de potencia de 0.35 mW en

el caso de *A(x) mod P(x) * B(x)*, esta optimización se logró a través de la *Técnica de Reordenamiento de señales*, en este caso se probaron los resultados del multiplicador empleando la propiedad conmutativa de las operaciones. Por otra parte, no se presenta consumo asociado a la señal de reloj *Pot_clk*, ya que el multiplicador es concurrente y no se presenta consumo de *Pot_IO* debido a que es un componente interno y las señales del multiplicador no son implementadas en los pines externos del FPGA, encontrando que el orden de las entradas solo presenta un efecto en la *Pot_Signal* que corresponde a una técnica de disminución del consumo que se tomará para el diseño de los módulos del codificador.

Basados en la relación de potencia del diseño optimizado y la versión original se obtiene un 7.89% de ahorro del consumo de potencia asociada a la señal en el diseño del multiplicador. Una vez optimizado el multiplicador se aplicó el diseño en VHDL considerando el orden *A(x) mod P(x)*B(x)*, con lo que el efecto de esta disminución de potencia se multiplicará por el número de componentes optimizados, resultando de esta manera un 39.66% de disminución de potencia asociada a la señal en el codificador.

Efectos de la Optimización de Consumo de Hardware sobre el Codificador RS

En el mismo orden de ideas que para el componente multiplicador, se calcularon los recursos de hardware utilizados por el codificador RS(*n,k*), de acuerdo a los parámetros ajustables. De forma teórica, los elementos *flip-flops* requeridos depende del número de etapa de memoria del LFSR en el codificador *n-k* y el número de bits por símbolos *m*, como se presenta en la tabla 4.8.

Tabla 4.8. Estimación de los Reg. FF en función de los parámetros

n	*k*	*m*	Etapas del LFSR *n-k*	Reg. FF *(n-k)*m*
255	223	8	32	256
255	231	8	24	192
255	239	8	16	128
255	247	8	8	64
127	111	7	16	112

Los resultados de la síntesis del codificador RS -CESR, se resumen en la tabla 4.9 en contraste con codificadores previos (Anexo D).

Tabla 4.9. Recursos empleados por el codificador RS(n,k)

Componentes	*n*	*k*	*m*	CLB Slice	LUT	Reg. FF	$F_{MÁX}$ (MHz)	Delay (ns)	Pot. (mW)
[1] Altera 2.5G	255	239	8	374	40	186	>160	-	-
[2] Lattice DVB	-	-	8	129	254	201	400	-	-
[3] Xilinx G.709	255	239	8	195	192	186	441/596	-	-
[3] Xilinx ETSI	255	239	8	199	194	186	442/580	-	-
[3] XilinxCCSDS	255	223	8	337	329	316	245/316	-	-
[4] Gianni	255	239	8	-	353	170	173	5.766	-
[5] Jinzhou	255	223	8	-	460	278	395	-	-
[6] CESR G.709	255	239	8	128	159	128	426.457	3.897	0.585
[6] CESRCCSDS	255	223	8	256	305	256	418.712	3.811	0.586
[6] CESR -247	255	247	8	64	142	64	429.646	3.883	0.586
[6] CESR -111	127	111	7	112	129	112	462.591	3.895	0.417

Fuente: [1] Altera, 2011; [2] Lattice, 2005; [3] Xilinx, 2011; [4] Gianni et al., 2007); [5] Jinzhou, 2012; [6] Autora, 2013

De estos resultados podemos observar una optimización en consumo de recursos del hardware del FPGA. Tenemos el consumo de recursos menor al de los IPCores (Altera, 2011; Lattice, 2005; Xilinx, 2011), y otras investigaciones (Jinzhou et al., 2012) lo que se puede interpretar como un avance significativo en la optimización bajo el concepto de tratamiento concurrente y modelo desarrollado.

ANÁLISIS DE LOS REPORTES DEL CONSUMO DE POTENCIA DINÁMICA

El consumo de potencia dinámica de los codificadores optimizados – CESR, con parámetros de *n* y *k* variables, bajo el modelo del multiplicador concurrente, se ilustra en la tabla 4.10.

Tabla 4.10. Consumo de Potencia Dinámica de los Codificadores RS(n,k)

Potencia (mW)	Logic	Signal	Clk	IO	Dinámica
RS(255,247)	0,04	0,42	0,95	16,69	18,10
RS(255,239)	0,04	0,32	1,19	16,69	18,24
RS(255,223)	0,04	0,73	1,43	16,69	18,89

Para los tres codificadores el consumo de potencia por circuito lógico correspondió a 0.04 mW en todos los casos, la potencia asociada a las señales corresponde a 0.42, 0.32 y 0.73 mW respectivamente, la potencia asociada a la señal de reloj corresponde a 0.95, 1.19 y 1.43 mW respectivamente, lo que nos permite concluir que a mayor relación de redundancia mayor consumo de potencia, esto debido a que la señal de reloj y las señales del LFSR para la generación de los símbolos de redundancia serán mayor.

Una observación que resulta oportuno mencionar, es que siendo los coeficientes de los codificadores valores fijos, se consideró la configuración para los *n-k* multiplicadores de forma dedicada, se pudo observar que este enfoque además de eliminar la generalización trabajada en el diseño, no logra simplificar el consumo de recursos de hardware y por ende de potencia, aun cuando en la matriz de multiplicación se eliminan las compuertas *AND* y solo se requiere de un número de *t* compuertas *XOR*, siendo *t* el número de unos "1" del coeficiente a multiplicar, estas pruebas demostraron que la particularización, establece condiciones no flexibles en las compuertas a utilizar que no permite la optimización de los recursos hardware por parte del ISE11 empleado en el diseño.

En una primera aproximación, se colocaron en la entidad declarada en VHDL pines de entrada /salida para realimentación de los resultados de los multiplicadores, con el propósito de observar los resultados a través de la simulación. Al implementar el diseño se obtuvo una potencia dinámica en el orden de los 36.49 mW para el RS(255,223), siendo el componente de consumo de potencia mayor en *Pot_IO* en el orden de los 33.21 mW, teniendo en cuenta que el resultado de los productos puede ser manejado como una señal interna, se configuró como *signal*, eliminándola de los pines del dispositivo, obteniéndose una reducción del consumo de potencia a la mitad, es decir; la potencia Dinámica reportada pasó a ser de 18.89mW, lo que corresponde a un 50% de ahorro en el consumo de potencia del diseño.

Igualmente, se hicieron pruebas renombrando o reasignando señales, concatenando señales internas e intercambiando asignaciones de manera externa e interna a los

componentes, con el propósito de estudiar los efectos de estos cambios en la sintaxis sobre el consumo de potencia de la implementación, teniendo como resultado que los reportes no variaban, esto debido al proceso de optimización propio de la herramienta de diseño durante la síntesis del diseño.

Se analizó el reporte del consumo de potencia sobre el dispositivo xc5vlx50T-3ff1136, empleando para ello la herramienta *XPower Analyzer*, incorporada en el ISE11. En la figura 4.10, se presenta la comparación entre el codificador RS(255,223) particularizado y el optimizado a través de la generalización con habilitadores.

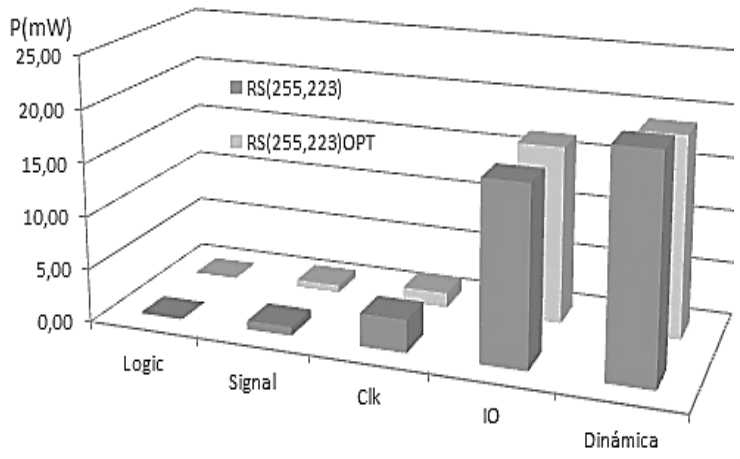


Figura 4.10. Consumo de potencia de los codificadores RS(255,223)

Al optimizar el RS(255,223), se obtiene un consumo de 18.89 mW vs 19.72 mW en el primer codificador diseñado. La potencia consumida en *Pot-Signal* aumenta en el codificador reconfigurable, ya que se ha requerido definir nuevas señales para el manejo de parámetros.

Cabe destacar, que el incremento en consumo por el componente *Pot-Signal* no es significativo en relación a la flexibilidad obtenida en el diseño paramétrico. Para este análisis no se consideró el caso del codificador RS(*255,k*) con *k* variable, puesto que aun cuando la optimización por des-habilitación de etapas es importante, el componente *pot_IO* se ve incrementado por el parámetro *k* que requiere de 8 pines para configurar el codificador *In-Circuit*.

EVALUACIÓN DE LA OPTIMIZACIÓN

Contrastando ahora los resultados obtenidos con los de investigaciones previas en esta área (Anexo D). En (Shih, 2000) se registra un consumo de 27.12mW para el codificador con *m=8* y *t=8*, RS(255,223), en tanto que en el presente diseño se ha obtenido un consumo (de Potencia Dinámica – asociada al diseño) para el mismo codificador de 18.89mW, lo que corresponde a un 30.34% de optimización en el consumo de potencia.

Por otro lado, al realizar la comparación correspondiente a la potencia consumida por el diseño del multiplicador, se han encontrado reportes de consumo de potencia de 35mW en el multiplicador (Marimuthu & Thangaraj, 2008) vs. 17.21mW en el diseño propuesto del multiplicador concurrente, lo que se traduce en un 50.82% de optimización en el consumo de potencia de los multiplicadores, es de hacer notar que estos multiplicadores han sido evaluados considerando las IO del componente para efectos de comparación, sin embargo; en el módulo codificador, esta factor desaparece puesto que pasan a ser señales internas del diseño.

Se ha analizado la simplicidad algorítmica del divisor polinómico $A(x)x^i \ mod \ P(x)$ secuencial, los cuales presentan resultados satisfactorios en área, pero con costo en tiempo de respuesta debido a las características secuenciales de éste, simultáneamente se han desarrollado multiplicadores paralelos basados en tablas o en circuitos combinacionales, siendo eficientes en tiempo, pero con un mayor consumo de hardware. Es en tal sentido que se ha propuesto avanzar en esta optimización considerando un modelo híbrido concurrente, a fin de crear un codificador altamente paralelo usando el menor número de recursos hardware, teniendo como referencia los reportes de síntesis de los modelos diseñados en las aplicaciones de propiedad intelectual de los fabricantes de dispositivos FPGA.

El estudio y pruebas funcionales permitió generar las ecuaciones para VHDL que describen el código, con parámetros simplificables y adaptables para los casos particulares, ya que se determinó que los parámetros una vez asignados permiten optimizar el diseño en gran manera, motivo por el cual los resultados reales estarán dados en función de los coeficientes que definen el polinomio generador del código, por tratarse de coeficientes constantes para un RS dado, podemos disminuir el consumo de recursos de hardware en el código RS, se logró, igualmente, la descripción hardware de módulos genéricos de interés y finalmente el código VHDL de los componentes del código CESR, que puede ser adaptado para nuevos diseños.

Por medio de la descripción VHDL del codificador Reed-Solomon, empleando el modelo circuital desarrollado del multiplicador concurrente, se logró obtener un modelo eficiente, que considera de forma equitativa la reducción de consumo de hardware, consumo de potencia y tiempo de procesamiento, a través del diseño del Código Eficiente sobre Sistema Reconfigurable y se han desarrollado los conceptos y estructuras de soporte para su óptima implementación.

4.3. MODELO DEL MULTIPLICADOR GF(2^M) CONCURRENTE

Luego de concretizar las ecuaciones del multiplicador en VHDL, se logró un modelo matemático-lógico para hardware reconfigurable, a partir del estudio de las ecuaciones de cada uno de los elementos del LFSR presentadas en la tabla 3.9. De esta manera, se obtuvo una generalización del comportamiento de los elementos $a_t\ (i)$. Dando como resultado la ecuación 4.3.

$$a_t\,(i) = a_{t-1}\,(i-1)\ xor\ (a_{t-1}\,(m-1)\ and\ p(i)) \qquad (4.3)$$

Siendo a_t, un elemento generado en un tiempo t de aplicar la reducción modular, $A(x)\ mod\ p(x)$, resultante de la operación *and* entre $p(i)$, el coeficiente del polinomio de la función de realimentación en la posición i, con el elemento a_{t-1} en la posición más

significativa del símbolo anterior, este término *XOR* con a_{t-1}, en la posición *i*. Es importante señalar que se ha considerado la optimización por *organización de señales*, en la cual el orden de los factores de la multiplicación afecta la eficiencia del diseño, siendo la presentada en el modelo la distribución más eficiente de las señales.

Seguidamente, se definió la ecuación general del vector de residuos de la división o reducción modular para cada instante de tiempo, empleando la variable *t* como una referencia de operación con los elementos de *B(x)*, pero empleando la operación concatenación, se ha logrado sustituir la señal de reloj para generar la secuencia de los bits dados por la ecuación 4.3, de donde se obtiene la ecuación 4.4.

$$a_t = \&_{i=0}^{m-1}\, a_{t-1}(i-1)\ xor\ (a_{t-1}(m-1) and\ p(i)) \qquad (4.4)$$

Donde se empleará el símbolo '&' para indicar la operación de concatenación a fin de coincidir con la descripción en VHDL, de esta manera se optimiza por *paralelización* del circuito, a través de la concatenación de señales para un tratamiento concurrente. Para implementar la etapa combinacional del producto de $(A(x)x^i \bmod P(x)).B(x)$, que corresponde a la operación *AND* de cada elemento de *B(x)* por cada a_t y la sumatoria *XOR* de estos, se describe la ecuación 4.5.

$$c = \oplus_{t=1}^{m}\, a_t\ and\ b_t \qquad (4.5)$$

Al sustituir la ecuación 4.4 en la ecuación 4.5, se obtiene el modelo del multiplicador dado por la ecuación 4.6.

$$c = \oplus_{t=1}^{m}\ [\&_{i=0}^{m-1}\, a_{t-1}(i-1)\ xor\ (a_{t-1}(m-1) and\ p(i))]\ and\ b_t \qquad (4.6)$$

Esta ecuación tiene la variable *i* para coeficientes no nulos del polinomio *p(x)*, con lo que se logra una optimización por la simplificación de las ecuaciones, como se mencionó en el análisis de eficiencia.

4.4. REESTRUCTURACIÓN DEL CODIFICADOR RS(*N,K)* CONCURRENTE

Se obtuvo la ecuación 4.7, del análisis de la tabla 3.11, para generar un símbolo de redundancia en un instante de tiempo dado, correspondiente a la suma módulo-2 del símbolo de redundancia, en la posición *i-1*, un instante de tiempo antes con el resultado del producto entre el coeficiente del polinomio generador *y* un término realimentado.

$$r_t\,(i) = r_{t-1}\,(i-1) \oplus\ (d_t \otimes g(i)) \qquad (4.7)$$

Donde $r_t(i)$ corresponde al símbolo *i* en el circuito generador de redundancia, en un instante de tiempo *t*, $r_{t-1}(i-1)$ corresponde al símbolo almacenado en el registro anterior, d_t corresponde al símbolo realimentado en un instante *t* y *g(i)* el coeficiente del polinomio generador que es fijo en el tiempo.

Seguidamente, se definió la ecuación general del vector de redundancia r_t, para un instante de tiempo *t*, el cual corresponde a la concatenación de los términos generados en la ecuación 4.7, se empleará el símbolo '&' para indicar la operación de concatenación a fin de coincidir con la descripción en VHDL, ,con *i* desde *0* hasta *n-k*, siendo el elemento $r_t\ (n-k)$ el resultado de la operación producto (término más significativo), de esta manera se obtiene la conformación del vector dado por la ecuación 4.8.

$$r_t = \&_{i=0}^{n-k}\, r_{t-1}\,(i-1) \oplus\ (d_t \otimes g(i)) \qquad (4.8)$$

Cabe recordar que *d(t)* se genera de la operación *xor-lógica* entre un símbolo de la entrada de datos *D(x)* en la posición *k* y el símbolo en la salida de memoria para *t-1*, en la localidad más significativa $r_{t-1}(n-k)$. La ecuación puede ser re-escrita en función de *D(k)*, como se expresa en la ecuación 4.9.

$$r_t = \&_{i=0}^{n-k}\, r_{t-1}\,(i) \oplus [(D(k) \oplus\ r_{t-1}\,(n-k)) \otimes g(i)] \qquad (4.9)$$

De esta manera, se tomó el modelo del circuito de estructura concurrente de realimentación lineal para la generación de los símbolos de redundancia del codificador Reed Solomon, partiendo del LFSR que forma parte de la arquitectura del codificador RS, cuya longitud está dada por los *n-k* registros, de *m* bits cada uno, se obtiene el modelo paralelizado, en el cual cada palabra corresponde a un elemento del campo finito GF(2^m), de esta manera se procedió a la generalización de la ecuación del LFCS, esta vez aplicable a registros de capacidad de *m* bits, de donde se obtiene la ecuación 4.10, correspondiente a la versión extendida de la ecuación de concatenación, esta representa la aplicación del modelo concurrente del LFSR para la descripción del codificador paralelo (C. Sandoval-Ruiz, 2012).

$$r_t = r_{i-1}(n-k-1) \oplus d(t) \otimes g(n-k) \,\&\, r_{i-1}(n-k) \oplus d(t) \otimes g(n-k-1)) \,\&\, \ldots \,\&\, d(t) \otimes g(1)) \qquad (4.10)$$

Se ha expresado el factor de la multiplicación en algebra finita de *Galois*, a través de la salida del componente, el producto se ha expresado en un único término, para lo cual la ecuación del modelo se re-escribe identificando al producto en el campo GF como $m_{t,i}$, tal como se observa en la ecuación 4.11.

$$r_t = r_{t-1}(n-k-1) \oplus m_{t,n-k} \,\&\, r_{t-1}(i-1) \oplus m_{t,i} \,\&\, \ldots \,\&\, m_{t,1} \qquad (4.11)$$

Esta ecuación permite generar las salidas del LFSR concurrente, siendo el vector de redundancia R(x) = r_t con $t=k$, se aplicará *k* veces en la descripción, de esta manera se obtiene la ecuación que sistematiza el modelo concurrente del codificador RS, a partir de la estructura de realimentación lineal concurrente desarrollada a lo largo de esta investigación. Al aplicar la ecuación 4.11 del modelo del codificador RS(*n*,*k*), se genera el código VHDL que describe la implementación en hardware, este código para el caso del codificador RS(*7*,*3*) se presenta en la tabla 4.11.

Tabla 4.11. Código VHDL generado a partir del modelo concurrente RS(n,k)

```
-- las entradas concurrentes D(x) corresponden a los símbolos  e1,e2,e3
de 3 bits cada uno
-- aplicación de la ecuación 4.14
r1<=m14 & r0(3) xor m13 & r0(2) xor m12 & r0(1) xor m11;     --para t=1
r2<=m24 & r1(3) xor m23 & r1(2) xor m22 & r1(1) xor m21;     --para t=2
r3<=m34 & r2(3) xor m33 & r2(2) xor m32 & r2(1) xor m31;     --para t=3
--Salidas del Encoder RS, se generó un vector s(x), con R(x)= r3
s1<=e1; s2<=e2; s3<=e3; s4<=r3(0); s5<=r3(1); s6<=r3(2); s7<=r3(3);
```

Igualmente, resulta oportuno destacar que para el modelo paralelizado del codificador RS, se puede aplicar la alternativa de optimización, a través de *habilitadores de etapas* del generador de secuencia LFSR, en casos de RS(*n*,*k*) con *k* variable.

Para efectos de la obtención de una ecuación total del circuito, se puede sustituir el modelo del multiplicador en el modelo del codificador RS (ecuación 4.11), resultado un sistema de funciones iterativas - SFI en el espacio, característica propia de las estructuras fractales, por tratarse en este caso de un circuito de implementación para la mencionada función del codificador concurrente. El modelo resultante puede expresarse, en forma resumida, a través de la ecuación 4.12.

$$r_t = \&_{i=0}^{n-k} \{ r_{t-1}(i-1) \oplus [\oplus_{t=1}^{m} [\&_{i=0}^{m-1} d_{t-1}(i-1) \text{ xor } (d_{t-1}(m-1) \text{and } p(i))] \text{ and}_m g(m-k)] \} \quad (4.12)$$

De esta manera, se presenta en una ecuación el modelo del codificador Reed Solomon, el cual comprende el componente generador de redundancia y el multiplicador de campos finitos GF.

En la figura 4.11, se puede observar la respuesta *concurrente*, es decir, la salida paralela en un solo ciclo el reloj, de la descripción VHDL (Anexo E) del codificador RS(7,3) modelado. Donde el polinomio generador del código corresponde a *G(x) = 5,7,7,4*, los símbolos de entrada de datos (*e1,e2,e3*) están dados por el vector *D(x) = 1,3,7*, los símbolos de redundancia generados (*s4,s5,s6,s7*) se presentan a través del vector *R(x) = 0,1,1,5*, de manera de obtener finalmente en la salida del codificador RS(*7,3*) paralelo, los símbolos de salida de datos (*s1 a s7*), dados por el vector *S(x) = 1,3,7,0,1,1,5*, donde

se valida el comportamiento del codificador RS(7,3) con el modelo paralelo desarrollado, en comparación al circuito secuencial estudiado.

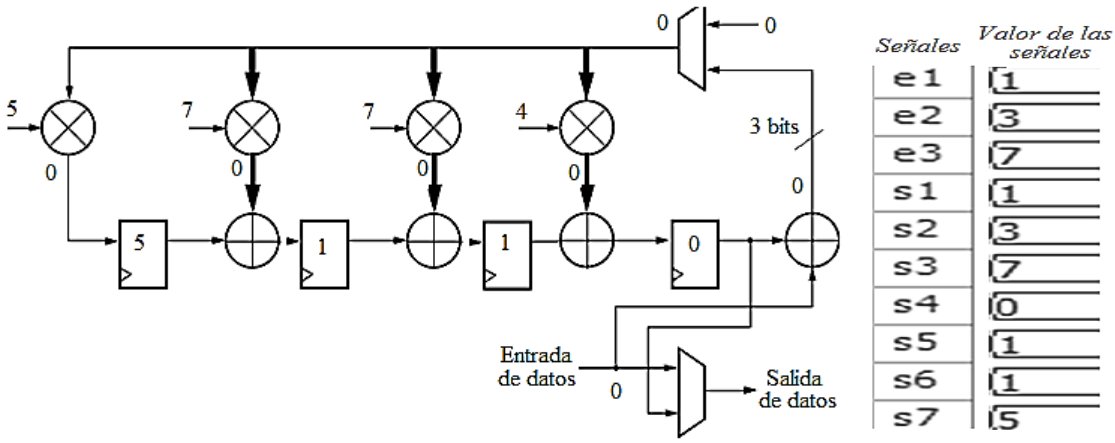


Figura 4.11. Resultados de la simulación del RS(7,3) paralelo

Uno de los análisis que resultó de interés consistió en el consumo de hardware entre la implementación secuencial y la implementación paralela tanto del multiplicador, como de la arquitectura del codificador, encontrando los reportes sintetizados en la tabla 4.12.

Tabla 4.12. Síntesis de los codificadores RS(7,3) diseñados

Codificador RS(7,4)	Secuencial	Paralelo
Clk	1	0
Slice	13	11
Slice FF	12	0
LUTs 4Input	23	19
Delay Máx.	7x12.235ns	10.621 ns

Se puede observar que para el codificador RS secuencial se requieren 8 *slices* del dispositivo XC5vlx50T-3FF1136, mientras que en el diseño paralelo se requieren 11 slice, tal como es de esperarse debido a que las funciones lógicas deben implementarse de forma concurrente y no se comparten recursos de hardware como en el caso secuencial, sin embargo, la utilización de *flipflops* (Slice FF) en la implementación secuencial demanda 12, correspondientes a los *n-k* registros de *b* bits cada uno, en tanto que el codificador RS paralelo no requiere de elementos de memoria, las tablas de búsqueda LUTs4 para el caso secuencial se requieren 20 en contraste con el modelo

paralelo que solo requiere 13, de manera que se evidencia un ahorro en componentes de hardware considerable.

Se observa que el codificador secuencial requiere de *n* pulsos de reloj para completar una codificación, en este caso *7*Delay Máx* que se traduce en 85,645ns, esto versus el procesamiento paralelo que no requiere señal de reloj y tiene un retardo máximo de 10,621 ns, con respecto a los cambios en las entradas del codificador RS. Es oportuno señalar que uno de los aspectos críticos corresponde a los retardos del circuito combinacional, debido a la profundidad lógica del diseño paralelo, sin embargo, esto ha sido solventado directamente por la herramienta de desarrollo ISE11, que cuenta con la capacidad de optimización de la distribución a través del proceso de síntesis, al tratarse el diseño a nivel de la tecnología del dispositivo, basada en LUTs en lugar de compuertas, lo que simplifica el número de etapas definido como complejidad lógica del circuito final.

En la figura 4.12, se reportan los resultados del consumo de potencia suministrada en el reporte de la herramienta de desarrollo ISE 11.

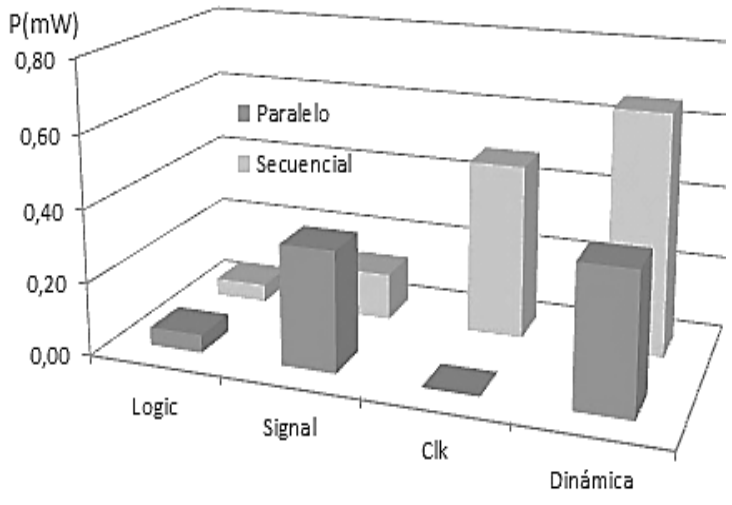


Figura 4.12. Consumo de Potencia de los RS(7,3)

En el modelo paralelizado se puede observar una disminución de 0,48 mW debido al componente *Pot-clk*. Una relación de potencia dinámica de 0,62 mW vs 0,40 mW para la versión RS(7,3) paralelo. De este modo, se presenta un ahorro de potencia dinámica de un 42,42%.

Vale destacar que el consumo de potencia dinámica ha sido calculado sin considerar la potencia disipada en los pines de entrada y salida, esto debido a que el diseño es un componente del sistema de comunicación donde las *IO* no son implementadas en los pines del dispositivo, en ese caso encontramos que el número de entradas /salidas es mayor para el diseño paralelo.

Se logró de esta manera, el desarrollo del modelo concurrente del codificador Reed Solomon, su validación y análisis de eficiencia de éste, con respecto a investigaciones previas con modelos secuenciales, siendo el modelo acá alcanzado idóneo para su implementación sobre hardware, sobre tecnología de dispositivos configurables a través del lenguaje VHDL, que ofrece soporte a tales niveles de paralelización de procesamientos complejos.

4.2. PRINCIPIOS APLICADOS EN LA INVESTIGACIÓN

Se ha identificado y caracterizado los bloques básicos del multiplicador, se describió un modelo genérico del comportamiento del divisor en GF(q) que soporta las operaciones de multiplicación en algebra de campos finitos. Éste ha sido diseñado con el *principio de modularidad*, integrando los componentes diseñados para solucionar el problema, presentando entre sus beneficios alto grado de paralelismo y escalabilidad, ya que la unidad aritmética es escalable porque ésta puede ser reusada en otras aplicaciones, independientemente de la unidad diseñada originalmente (Savaš, Tenca, & Koç, 2000). De esta manera se ha implementado en VHDL el módulo de soporte para el tratamiento de datos en algebra GF(2^m), un rápido multiplicador sobre campos finitos con una estructura interna re-diseñada de acuerdo a la tecnología disponible.

La descripción del modelo optimizado, en el cual se realizó el análisis temporal y se interpretó el comportamiento del LFSR para la obtención de las ecuaciones de una arquitectura concurrente, puede ser sustentada bajo un enfoque del *Principio Holográfico*, de coexistencia en el espacio de resultados secuenciales. De esta manera, la descripción circuital de modelos concurrentes, puede ser generada a través de la

operación de concatenación "&", fundamentada en las nuevas tecnologías de diseño de hardware, reenfocando un modelo secuencial.

Por otra parte, se presenta una novedosa aplicación del principio de *Diseño para Hardware Evolutivo*, provisto de capacidad para optimizar sus funciones, basados en el desempeño de sus objetivos. Donde el modelo VHDL permite configurar el hardware, sea generando un *bitstream* desde el software de aplicación, como un *sistema de configuración central* ó por módulos de configuración por jerarquía, residente en el FPGA, como un *módulo de configuración autónomo*.

La aplicación del *Principio de Correspondencia* (Bohr, 1923) permitió extender el diseño del multiplicador desarrollado, hacia la aplicación del codificador Reed Solomon, a partir del reconocimiento de características coincidentes entre los esquemas circuitales. En este caso se identificó que el generador de símbolos de redundancia del codificador Reed Solomon y el componente de reducción modular del multiplicador en el campo GF, presentan la misma estructura LFSR, como una estructura circuital fractal (Anexo F). En la tabla 4.13 se presenta la correspondencia entre los elementos del LFSR de los circuitos estudiados.

Tabla 4.13. Correspondencia entre circuitos con estructura LFSR

Componente LFSR	RS-PC	Codificador RS	Multiplicador GF
Elemento de memoria	*Bloque k símbolos*	*Símbolos de m bits*	*bit*
Operador de ramas	*RS_encoder*	*Mult_GF*	*and*
Expresión de función	*bloque(i) and f(i)*	*d(i) and g(i)*	*a(i) and p(i)*
Polinomio Generador	*f(x)*	*g(x)*	*p(x)*

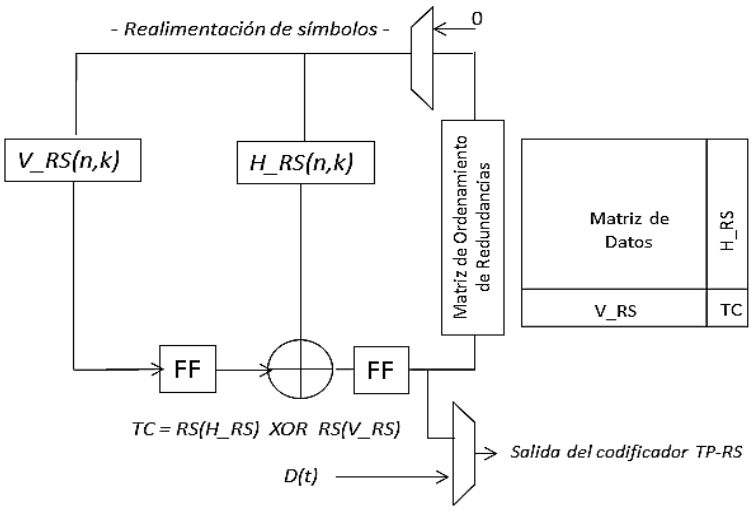


Figura 4.13. Estructura del RS-PC

En el circuito de la figura 4.13, se puede reconocer una estructura similar al LFSR estudiado. Donde la operación corresponde a la codificación RS(*n*,*k*), que procesa los bloques de entrada de *k* símbolos y se realimentan los símbolos de redundancia generados, en las etapas del arreglo.

4.3. POSTULADOS

El estudio desarrollado permitió postular un novedoso enfoque diseño de componentes (en forma modular), con capacidad de reconfiguración que integran los conceptos HW/SW. Esto definiendo las ecuaciones a través de la operación de "*concatenación*" para la implementación concurrente, con el propósito de obtener menor consumo de recursos, mayor velocidad de procesamiento y menor consumo de potencia. Con el objetivo de lograr diseños más eficientes y con mayor flexibilidad, por medio de una descripción matemática – lógica de soporte para la configuración, usando lenguaje descriptor de hardware. En relación al enfoque de diseño basado en modelos VHDL, se puede enunciar:

La configuración de hardware por estar basada en arquitectura flexible, permite definir modelos parametrizables, que serán definidos de acuerdo a objetivos y condiciones del sistema.

Del estudio del codificador Reed Solomon se encontró una correlación entre los datos, característica de los sistemas generadores de secuencia aleatoria, lo que permite abrir el

campo de aplicaciones de estos, desde la adaptación de los esquemas de codificación, teniendo presente que en la etapa de decodificación se puede hallar un patrón para la interpretación de los datos, por lo que se puede postular que:

Sea G(x) una estructura generadora de relación entre componentes de la señal tratada y D(x) la señal procesada por el generador de código, se obtendrá una señal codificada C(x) que permite establecer el modelo de la estructura generadora.

Finalmente, se encontró una correspondencia entre la estructura generadora del conjunto de Galois en el multiplicador, el generador de redundancia del codificador Reed Solomon y la estructura para la relación del codificador Turbo producto con concatenación paralela, lo cual permite afirmar que:

Cualquier función matemática puede hacerse corresponder con un generador de conjuntos LFSR GF, siempre que se considere la generalización de los operandos (sean estos: and, mult_GF, codificador RS, cualquier función de transferencia), donde la secuencia generada se obtiene de la combinación (xor, add, multiplexación) de aportes de cada una de sus ramas.

REFLEXIONES DE RESULTADOS ALCANZADOS

Gracias a la filosofía de diseño empleada se logra una optimización del consumo de potencia y eficiencia del componente fundamental para los codificadores RS(*n*,*k*), empleando la concatenación como herramienta en el diseño del registro desplazamiento para sustituir las reasignaciones dependientes de la señal de reloj (*clk*), con lo cual se paraleliza el procesamiento de los datos.

Del modelo matemático del multiplicado GF, se propuso un modelo concurrente, a partir de una estructura LFCS, para configuración bajo lenguaje descriptor de hardware VHDL, lo que hace más simple el proceso de descripción fundamentado en un análisis teórico, de los polinomios en campos finitos, logrando así garantizar la respuesta de los módulos diseñados, tal como se validó usando ModelSim XE III 6.3.c.

Importante resulta señalar que la síntesis del codificador RS, permiten concluir que el diseño es altamente competitivo, en comparación con los estudios comparativos de los desarrollos en el área, esto por el desempeño alcanzado en el modelo desarrollado.

Observaciones finales, permitieron reconocer un patrón característico en el circuito de reducción modular y el circuito de generación de símbolos de redundancia del codificador, donde existe una correspondencia entre ambas arquitecturas, motivo por el cual se consideró evaluar la extensión del modelo concurrente al circuito generador del código, este patrón permite asociar el modelo con funciones iterativas que repiten su estructura circuital, a través de una arquitectura fractal.

Se logró de esta manera, la propuesta del modelo concurrente del codificador Reed Solomon, así como la validación y análisis de eficiencia de éste, con respecto a investigaciones previas con modelos secuenciales, siendo el modelo acá alcanzado idóneo para su implementación sobre hardware.

CAPÍTULO V

CONCLUSIONES

En este capítulo se presentan las principales conclusiones, aportes, recomendaciones y propuestas para futuras investigación derivadas de la presente tesis doctoral.

5.1. DISCUSIÓN FINAL

El estudio presentado en la disertación doctoral se encuentra incluido en la línea de investigación de Hardware reconfigurable, al igual que en el área de codificación y teoría de la Información, se ha desarrollado dedicando especial atención a los aspectos de un modelo eficiente del codificador Reed Solomon, basado hacia las nuevas tecnologías en hardware reconfigurable. Este enfoque novedoso permite obtener excelentes prestaciones con respecto a otros diseños basados en modelos matemáticos convencionales y otros métodos empleados en descripción de hardware.

Adicionalmente, se presenta un compendio de investigaciones en el área, conformada por tesis doctorales y artículos científicos a nivel internacional, que han sido contrastadas con los resultados obtenidos y una descripción metodológica detallada de los procedimientos seguidos, el método de análisis y las herramientas empleadas, lo cual

permite reproducir los resultados acá documentados. La investigación teórica comprende conceptos y aportes desarrollados en el campo de codificación y los modelos de multiplicadores en campos finitos de Galois, lo cual ha sido empleado como soporte para el desarrollo del modelo propuesto, haciendo referencia a los trabajos en el área que son fuente para el contraste de los resultados.

El modelo acá presentado abre la puerta a un campo de investigación en codificación adaptativa, desde la perspectiva de radio cognitivo, basados en la parametrización realizada para el presente modelo. Así como a la línea de investigación de generadores de secuencia y criptografía, a partir de la utilización de los multiplicadores propuestos.

- ✓ De la descripción del codificador RS(*n,k*) parametrizable, orientado a sistemas adaptativos, se obtuvo el código VHDL de los componentes, a través de ecuaciones de comportamiento de hardware resulta más eficiente que a partir de tablas de búsqueda. A la vez que por ser tratado bajo la filosofía de hardware libre, ofrece a los diseñadores una base importante para desarrollos futuros.

- ✓ En cuanto a la optimización del codificador RS(*255,k*) a través de la estructura concurrente de realimentación lineal, se obtuvieron reportes de síntesis óptimos con respecto a los encontrados en la literatura desarrollada en investigaciones previas, e incluso superan en eficiencia los módulos *RSencoders v8.1, año 2012* desarrollados como módulos de propiedad intelectual (no libre) de Xilinx. Adicionalmente, se encontró que las ecuaciones que definen el modelo permite una estimación de los recursos hardware y del consumo de potencia.

- ✓ Por parte de la interpretación de la estructura concurrente para operaciones en aritmética en campos de Galois GF (2^m), se logró desarrollar un modelo, definido a través de ecuaciones que están pensadas de acuerdo a operadores lógicos que se emplean en la descripción de hardware, como es el caso de la concatenación empleada en VHDL. De donde se pudo reconocer que las funciones con

realimentación iterativa pueden ser interpretadas como circuitos fractales en los modelos para descripción de hardware.

- ✓ Finalmente, dados los resultados de las previas etapas del desarrollo del codificador RS, se sistematizó el modelo concurrente para paralelización del codificador Reed Solomon, lo que se traduce en la generalización del modelo del componente LFSR concurrente, para la estructura del generador de redundancia del codificador. Las pruebas demostraron que efectivamente las ecuaciones del modelo desarrollado permiten la implementación del codificador RS paralelo de forma óptima de acuerdo al modelo diseñado en esta investigación.

5.2. APORTES DEL MODELO DESARROLLADO

Entre los resultados de la tesis podemos encontrar un conjunto de aportes a la comunidad científica y para soporte de investigaciones en el área. El modelo concurrente generado para el componente LFSR, emplea la *concatenación* como un operador de hardware, para la descripción de comportamiento del componente. Esto en contraste a los métodos previos, basados en multiplicadores con estructura combinacional, secuencial o tablas de búsqueda, lo que ha resultado un aporte para el tratamiento innovador de codificadores concurrentes – concatenados bajo el mismo principio de diseño.

Hasta ahora resultan comunes los conceptos de modelado matemático, así como diseño y configuración de hardware, pero en esta investigación se ha abordado un paradigma que corresponde al modelado de hardware, con lo que se han obtenido ecuaciones detalladas que permiten describir el procesamiento de los datos en el codificador Reed Solomon. Este enfoque surge a través del proceso de concreción de las ecuaciones matemáticas del codificador RS (interpretando el modelo matemático de soporte) para casos particulares y la matematización del modelo con operadores VHDL, en el cual se adapta el modelo a

las operaciones sobre hardware. De esta manera a nivel científico, se logró una interpretación del modelo matemático – lógico para VHDL del componente LFSR.

El diseño desarrollado ofrece el código en VHDL para optimizaciones futuras, y la reconfiguración del hardware, dotando a éste de gran flexibilidad, simplicidad en la descripción y generalidad en el tratamiento de las estructuras estudiadas, para la reutilización de los componentes acá desarrollados en aplicaciones de procesamiento de datos con características similares y soportando el reciclado de tarjetas basadas en FPGA, lo que permite diseños sustentables y evolutivos. Por otra parte, el diseño está orientado a la optimización de consumo de recursos, consumo de potencia y velocidad de procesamiento, lo que se traduce en un ahorro de energía y mejores prestaciones en el codificador. Sin dejar de mencionar que, éste permitiría soportar la codificación adaptativa de acuerdo a las condiciones dinámicas del canal, con lo que se puede bajar la potencia de transmisión y por ende la contaminación electromagnética asociada a la comunicación, resultando así un aporte a nivel tecnológico.

5.3. TRABAJOS PUBLICADOS

La presente investigación ha generado un conjunto de artículos científicos, que comprenden el desarrollo de multiplicadores GF concurrentes bajo sintaxis en VHDL (Sandoval & Fedón, 2008c; Sandoval, 2010a), la descripción de codificadores y decodificadores RS secuenciales para casos particulares (Sandoval & Fedón, 2007), los que han sido la base del desarrollo acá presentado. Se ha presentado la aplicabilidad de los RS en códigos concatenados (Sandoval & Fedón, 2008a; Sandoval, 2008), donde surge el interés por los codificadores de alta velocidad de procesamiento, y finalmente, se ha llegado a la paralelización de un codificador Reed Solomon (C. Sandoval-Ruiz, 2012), con base en el modelo concurrente del LFSR desarrollado en el presente trabajo. Así mismo, se han presentado aplicaciones del modelo orientadas sistemas adaptativos (C. E. Sandoval-Ruiz & Fedón-Rovira, 2013) y algunos análisis de eficiencia en cuanto

al consumo de potencia (Sandoval, 2014), que han surgido de los resultados de la presente investigación.

5.4. PROYECTOS FUTUROS EN LA LÍNEA DE INVESTIGACIÓN

En primer lugar, es importante destacar que el modelo desarrollado se mantiene abierto a futuras optimizaciones de eficiencia y actualizaciones de acuerdo a la tecnología y aplicaciones, por lo que el diseño modular representa un avance y en función de la disertación doctoral se permite plantear recomendaciones para futuros trabajos.

Una de las propuestas consiste en diseñar una herramienta para la descripción de los componentes modelados en esta investigación, definiendo un algoritmo que genere el código VHDL a partir del modelo y los parámetros dados, a fin de obtener la sintaxis de manera automática para simplificar el proceso de diseño, con lo que podría extender la aplicación del codificador paralelo y el multiplicadores GF(2^m) a un mayor ancho de palabra, para m>8, todo esto en la línea de cómputo aplicado.

Por otra parte, se propone la aplicación del modelo hacia sistemas de comunicación adaptativos, teniendo en cuenta que en este campo se han desarrollado investigaciones, como es el caso de decodificadores basados en redes neuronales adaptativas (Sandoval, 2010b), decodificadores RS adaptativos para el uso eficiente de energía en el sistema (Allen, 2008) y la técnica de codificación con símbolos pilotos para estimar el modelo de referencia PSAC (*Pilot Symbol Assisted Coding*), el cual permite conocer las características del canal y tomar decisiones (Bonello, Chen, & Hanzo, 2009).

A través de la estimación del consumo de recursos hardware y consumo de potencia dinámica de los casos estudiados en la presente investigación y el cálculo de lógica utilizada a partir del modelo desarrollado, se puede generar un modelo matemático para la estimación de los recursos de hardware a emplear para diversos diseños basados en el modelo propuesto. Su relevancia está fundamentada en que actualmente un tema de

investigación de interés consiste en el cálculo del consumo de potencia, teniendo en cuenta que los modelos existentes como los empleados por la herramienta de desarrollo y otras investigaciones están basados en los recursos utilizados por el diseño y las capacitancias asociadas (Cheng, 2010), un estimador de recursos previo a la descripción en VHDL para los multiplicadores GF (2^m) y codificadores RS(n,k) generados bajo el modelo acá desarrollado permitiría realizar una evaluación de eficiencia de manera oportuna.

Por otra parte, el patrón reconocido en la estructura circuital asociado con funciones iterativas permite proponer un modelo de codificador compuestos, con codificadores Reed Solomon concatenados a través de una estructura LFSR, siendo una investigación de interés que puede resultar competitiva frente a los RS-PC que se implementan actualmente (C. Kim et al., 2010), resultando un modelo generalizado en el concepto de estructuras fractales de los esquemas de codificación.

Resulta oportuno dejar planteada la aplicación del modelo desarrollado en VHDL para diversos circuitos de procesamiento de datos, considerando la implementación de componentes basados en generadores de secuencia pseudo-aleatorios, en los cuales se pueden generalizar los modelos LFSR en su representación de Galois y Fibonacci (Anexo G), con el propósito de describir un modelo estandarizado para LFSR en VHDL que puedan ser paralelizados, a través de los avances acá desarrollados. Finalmente, se puede integrar los modelos de componentes en una descripción del sistema de codificador Reed Solomon genérico, como una alternativa para el modelo matemático, así el desarrollo del modelo se puede resumir a través de un compendio de ecuaciones, la expresión matricial del multiplicador y el arreglo LFSR generador de redundancia, y el procedimiento de generación del código en VHDL (Anexo H).

5.5. LEGADO DE LA TESIS DOCTORAL

El modelo descriptivo corresponde al principio integrador entre: el comportamiento de los componentes, para definir el hardware, y la re-configuración de la arquitectura, que genera el software.

La importancia de desarrollar las descripciones del hardware bajo un tratamiento modular, con el manejo de las señales como entradas – salidas entre los módulos, permite establecer una red de componentes, que soporte el diseño colaborativo o interdependiente entre componentes. En los LFSR del multiplicador GF se presenta una estructura anidada con relación al LFSR del codificador RS, donde se reconoce una arquitectura fractal. Por otra parte, el flujo de datos está asociado a un procesamiento concurrente. De todo lo anterior, el modelo permite el desarrollo del diseño aportando simplicidad al método de descripción de hardware, de esta manera puede soportar un ajuste dinámico con gran flexibilidad en su configuración, en un compromiso, para la optimización tanto a nivel de velocidad de procesamiento, como en cuanto a eficiencia en consumo de recursos y potencia.

Todos estos aspectos se deben considerar en la interpretación del comportamiento de un sistema basado en el diseño de su modelo descriptivo a nivel matemático – lógico, para la generación del código de configuración, por ello se debe partir del estudio de su arquitectura y señales. Esto con el propósito de reconocer patrones estructurales, siempre observando la arquitectura de sus módulos asociados, así como la correlación de las señales generadas a través de éste, con el objetivo de describir en una ecuación su comportamiento, lo que corresponde a un modelo de co-diseño para la configuración hardware/ software. El modelo generalizado permite extrapolar la descripción a sistemas más amplios, encontrando la correspondencia entre sus componentes a diversos niveles. *De manera de reconocer una expresión simplificada que describa el principio de comportamiento con capacidad de auto-configuración estructural para el diseño, cumpliendo con el principio establecido*.

Como punto final, el manejo de los diseños codificadores de bloque utilizando como principio su definición teórica, permite una descripción del hardware más eficiente. Al mismo tiempo que se deja planteada la posibilidad de optimizaciones futuras, empleando al máximo la flexibilidad característica de los sistemas reconfigurables.

REFERENCIAS

Ahlquist, G. C., Nelson, B. E., & Rice, M. D. (2002). Synthesis of small and fast finite field multipliers for field programmable gate arrays. Retrieved from http://klabs.org/richcontent/MAPLDCon02/papers/session_a/a4_ahlquist_p.pdf

Ahlquist, G., Nelson, B., & Rice, M. (1999). Optimal finite field multipliers for FPGAs. *Field Programmable Logic and Applications* (pp. 51–60). Springer. Retrieved from http://www.springerlink.com/index/C50WJ1GECE5UW7WF.pdf

Alaus, L. A., Oguet, D. N., & Alicot, J. P. (2008). Extended Reconfigurable Linear FeedBack Shift Register Operators for Software Defined Radio. *Gestion*, (1).

Allen, J. D. (2008). *Energy Efficient Adaptive Reed-Solomon Decoding System*. University of Massachusetts Amherst. Retrieved from http://scholarworks.umass.edu/theses/91/

Altera. (2011). AN 642: 2.5G Reed-Solomon II MegaCore Function Reference Design. *Design*, 1–22.

Alvarez, I. (2005). *Aplicaciones de las Matrices por Bloques a los Criptosistemas de Cifrado en Flujo, Tesis Doctoral*. Universidad de Alicante.

Anderson, J. H. (2005). *Power optimization and prediction techniques for FPGAs*. *Computer Engineering*. University of Toronto. Retrieved from http://www.eecg.utoronto.ca/~janders/anderson_thesis.pdf

Angarita, F., Marin-Roig, J., & Todorovich, E. (2005). Relación área-potencia en la implementación con aritmética distribuida de un Filtro FIR en FPGA. *Proc. JCRA 2005*, 59–63. Retrieved from http://arantxa.ii.uam.es/~ivan/jcra05-fir.pdf

Astarloa, A. (2005). *Reconfiguración dinámica de sistemas modulares multi-procesador en dispositivos SoPC*.

Atieno, L., Allen, J., Goeckel, D., & Tessier, R. (2006). An adaptive Reed-Solomon errors-and-erasures decoder. *Proceedings of the 2006 ACM/SIGDA 14th*

international symposium on Field programmable gate arrays (pp. 150–158). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=1117224

Biard, L., & Noguet, D. (2008). Reed-Solomon Codes for Low Power Communications, *3*(2), 13–21.

Bonello, N., Chen, S., & Hanzo, L. (2009). On the Design of Pilot Symbol Assisted Codes. *Vehicular Technology Conference Fall (VTC 2009-Fall), 2009 IEEE 70th* (pp. 1–5). IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5378698

Castillo, E. (2008). *PROTECCIÓN DE LA PROPIEDAD INTELECTUAL DE CIRCUITOS DIGITALES PARA LA SÍNTESIS DE DESCRIPCIONES DE ALTO NIVEL, Tesis doctoral*. UNIVERSIDAD DE GRANADA.

Cenditel. (2012). Proyecto Hardware Libre.

Chang, H. C., Shung, C. B., & Lee, C. Y. (2001). A Reed-Solomon product-code (RS-PC) decoder chip for DVD applications. *Solid-State Circuits, IEEE Journal of*, *36*(2), 229–238. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=902763

Chen, C. I. H. (2001). Synthesis of configurable linear feedback shifter registers for detecting random-pattern-resistant faults. *Proceedings of the 14th international symposium on Systems synthesis* (Vol. 1, pp. 203–208). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=500051

Cheng, L. (2010). *Statistical Analysis and Optimization for Timing and Power of VLSI Circuits*. *Evaluation*. Tesis Doctoral, Universiy of Califirnia.

Choi, C.-S., Ahn, H.-J., & Lee, H. (2011). High-Throughput Low-Complexity Four-Parallel Reed-Solomon Decoder Architecture for High-Rate WPAN Systems. *IEICE Transactions on Communications*, *E94-B*(5), 1332–1338. doi:10.1587/transcom.E94.B.1332

Climent, J. J., Crespí, F. G., & Grediaga, A. (2008). A Scalable Finite Field Multiplier. *IEEE Latin America Transactions*, *6*(7), 632–637.

Cruz, J. (2005). *Multiplicación Escalar en Curvas de Koblitz: Arquitectura en Hardware Reconfigurable*. *iberchip.net*. Instituto Técnico Nacional.

Delgado, O. (2010). *Nuevos Protocolos y Esquemas de Seguridad para Redes Ad-hoc Móviles Inalámbricas, TESIS DOCTORAL*. *Sierra*. UNIVERSIDAD CARLOS III DE MADRID.

Deschamps, J. P., Imaña, J. L., & Sutter, G. (2009). *Hardware implementation of finite-field arithmetic*. McGraw-Hill, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1550879

Dubrova, E. (2008). On Analysis and Synthesis of (n , k) -Non-Linear Feedback Shift Registers. *Transition*, 1286–1291.

Ekdahl, P. (2003). *On LFSR based Stream Ciphers-analysis and design*. Department of Information Technology,. Retrieved from http://swepub.kb.se/bib/swepub:oai:lup.lub.lu.se:21364?tab2=abs&language=en

Fette, B. A. (2009). *Cognitive radio technology*. *Engineering* (Second Edi.). Academic Press. Retrieved from http://books.google.com/books?hl=en&lr=&id=81q5AMSAMzgC&oi=fnd&pg=PP1&dq=Cognitive+Radio+Technology&ots=vhLkQSS6jV&sig=nopP6ykRxNdFos5mcF55_6Opzis

García-Martínez, M., Morales-Luna, G., & Rodríguez-Henríquez, F. (n.d.). IMPLEMENTACIÓN EN FPGA DE UN MULTIPLICADOR EFICIENTE PARA CAMPOS FINITOS GF(2m). Retrieved from http://delta.cs.cinvestav.mx/~francisco/arith/Mul_Iberchip_04_final.pdf

Genser, A., Bachmann, C., Steger, C., Hulzink, J., & Berekovic, M. (2009). Low-Power ASIP Architecture Exploration and Optimization for Reed-Solomon Processing. *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on* (pp. 177–182). IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5200026

Gianni, P., Claudio, G. Di, & Corteggiano, F. (2007). Implementación en FPGA de un Código Reed Solomon RS (255 , 239), (3), 77–81.

Halbutogullari, A., & Koc, C. K. (2000). Mastrovito multiplier for general irreducible polynomials. *Computers, IEEE Transactions on*, *49*(5), 503–518. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=859542

Huang, Z. (2003). *High-level optimization techniques for low-power multiplier design*. *Optimization*. University of California Los Angeles. Retrieved from http://arith.cs.ucla.edu/dissertations/dissertation_huang03.pdf

Hussein, B. J., Klein, M., & Hart, M. (2011). Lowering Power at 28 nm with Xilinx 7 Series FPGAs, *389*, 1–25.

Imaña, J. (2004). *Aplicación de Campos de Galois a la verificación probabilistica de Funciones Booleanas y Métodos de Multiplicación sobre campos de Extensión GF(2m)*. Universidad Complutense de Madrid.

Imaña, J. L., Sánchez, J., & Fernández, M. (2002). Método de multiplicación canónica sobre campos GF (2m) generados por AOPs orientado a hardware reconfigurable. *II Jornadas sobre Computacion Reconfigurable y Aplicaciones JCRA2002*.

Jinzhou, Z., Xianfeng, L., Zhugang, W., & Weiming, X. (2012). The Design of a RS Encoder. *Future Computing, Communication, Control and Management*, *144*, 87–91. doi:10.1007/978-3-642-27326-1_12

Kim, C. H., Oh, S., & Lim, J. (2002). A new hardware architecture for operations in GF (2m). *Computers, IEEE Transactions on*, *51*(1), 90–92. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=980019

Kim, C., Rhee, S., Kim, J., & Jee, Y. (2010). Product Reed-Solomon Codes for Implementing NAND Flash Controller on FPGA Chip. *2010 Second International Conference on Computer Engineering and Applications*, 281–285. doi:10.1109/ICCEA.2010.63

Kindap, N. (2004). *On an architecture for a parallel finite field multiplier with low complexity based on composite fields*. *Computers, IEEE Transactions on*. Middle East Technical University. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=508323

Kitano, H., & Hendler, J. (1994). Massively parallel artificial intelligence, 557–562. Retrieved from http://www.ijcai.org/Past Proceedings/IJCAI-91-VOL1/PDF/087.pdf

Kumar, S., & Gupta, R. (2011). Bit Error Rate Analysis of Reed-Solomon Code for Efficient Communication System. *International Journal of Computer Applications*, *30*(12), 11–15.

Lattice. (2005). Reed solomon encoder.

Lee, H. (2003). High-speed VLSI architecture for parallel Reed-Solomon decoder. *Very Large Scale Integration (VLSI) Systems, IEEE*, *11*(2), 288–294. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1210510

Machhout, M., Zeghid, M., El, W., Bouallegue, B., Baganne, A., & Tourki, R. (2009). Efficient Large Numbers Karatsuba-Ofman Multiplier Designs for Embedded Systems. *Computer Engineering*, 548–557.

Marchesan Almeida, G. M., Bezerra, E. A., Cargnini, L. V., Fagundes, R. D. R., & Mesquita, D. G. (2007). A Reed-Solomon algorithm for FPGA area optimization in space applications. *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on* (pp. 243–249). IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4291927

Marimuthu, C. N., & Thangaraj, P. (2008). Low Power High Performance Multiplier. *Vlsi Design*, *8*(1), 31–38.

Mora, A. (2008). *Estudio de Arquitecturas VLSI de la Etapa de Predicción de la Compensación de Movimiento, para Compresión de Imágenes y Video con Algoritmos. Aplicación al Estándar H.264/AVC Tesis doctoral*. Universidad Politécnica de Valencia.

Morales-Sandoval, M., Feregrino-Uribe, C., Cumplido, R., & Algredo-Badillo, I. (2009). An area/performance trade-off analysis of a GF(2m) multiplier architecture for elliptic curve cryptography. *Computers & Electrical Engineering*, *35*(1), 54–58. doi:10.1016/j.compeleceng.2008.05.008

Morelos-Zaragoza, R. H. (2002). *The art of error correcting coding*. (W. John & Sons, Eds.) (Vol. 2). England: Wiley. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/0470035706.fmatter/summary

Mucci, C., Vanzolini, L., Mirimin, I., Gazzola, D., Deledda, A., Goller, S., Knaeblein, J., et al. (2008). Implementation of parallel LFSR-based applications on an adaptive DSP featuring a pipelined configurable Gate Array. *Design, Automation and Test in Europe, 2008. DATE'08* (pp. 1444–1449). IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4484877

Mursanto, P. (2006). Generic reed solomon encoder. *New York*, *10*(2), 58–62.

Noordin, N. K., Ali, B. M., Ismail, N., & Jamuar, S. S. (2004). ADAPTIVE TECHNIQUES IN ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING IN MOBILE RADIO ENVIRONMENT. *International Journal of Engineering*, *1*(2), 115 – 123.

Oncti. (2011). Necesidades de Investigación. *Observatorio Nacional de Ciencia, Tecnología e Innovación*.

Paar, C. (1996). A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields 1 Introduction, *45*(7), 856–861.

Peralta, F. (2005). *DISEÑO DE ARQUITECTURAS DIGITALES PARA CRIPTOGRAFÍA*. Institulo Politécnico Nacional.

Pérez L., S. A., Soto C., E., & Fernández G., S. (2002). *DISEÑO DE SISTEMAS DIGITALES CON VHDL* (pp. 1–353). Madrid, España: Editorial Thomson.

Pérez, M. (2009). *Generación y correlación eficiente de códigos binarios derivados de conjuntos de secuencias complementarias para sistemas ultrasónicos, Tesis Doctoral*. Universidad de Alcalá.

Peter, S., & Langend, P. (1962). An Efficient Polynomial Multiplier in GF (2 m) and its Application to ECC Designs, 2–7.

Rodríguez, F., & López, M. (1996). *Control Adaptativo y Robusto*.

Sánchez Corrionero, F. (2011). *IMPLEMENTACIONES HARDWARE DE CIRCUITOS ARITMÉTICOS SOBRE CUERPOS FINITOS*. Universidad Complutense de Madrid.

Sandoval, C. (2007). *Diseño Modular de un Sistema para Procesamiento y Comunicación Digital en Banda Base usando Programación en VHDL*. Universidad de Carabobo. Retrieved from http://produccion-uc.bc.uc.edu.ve/documentos/trabajos/20001A55.pdf

Sandoval, C. (2008). Modular Programming of Functions for Turbo Product Codes on FPGA. *Rev.Téc.Ing.Zulia*, *31*(3), 294 – 301. Retrieved from http://www.scielo.org.ve/pdf/rtfiuz/v31n3/art10.pdf

Sandoval, C. (2010a). Multiplicador Paralelo en Campos Finitos de Galois GF (2m). *Congreso de Investigación UC* (pp. 1706–1711). Retrieved from http://www.cdch.uc.edu.ve/VIICongreso/noticias/noticia_27/memorias_VII_CIUC/TOMO_II.pdf

Sandoval, C. (2010b). FPGA prototyping of neuro-adaptive decoder. *Proceedings of the 9th WSEAS international*, 99–104. Retrieved from http://www.wseas.us/e-library/conferences/2010/Merida/CIMMACS/CIMMACS-13.pdf

Sandoval, C. (2014). POWER CONSUMPTION OPTIMIZATION IN REED SOLOMON ENCODERS OVER FPGA. *Latin American Applied Research*, *44*(1), 1–5.

Sandoval, C., & Fedón, A. (2007). Codificador y decodificador digital Reed-Solomon programados para hardware reconfigurable. *Ingeniería y universidad*, *11*(1), 17–32. Retrieved from http://www.redalyc.org/pdf/477/47711102.pdf

Sandoval, C., & Fedón, A. (2008a). MODULAR DESIGN OF SCHEME CODING CONCATENATED FOR CORRECTION ERROR WITH PROGRAMMING OF HARDWARE. *Ingeniare*, *16*(2), 310–317. Retrieved from http://www.scielo.cl/pdf/ingeniare/v16n2/art05.pdf

Sandoval, C., & Fedón, A. (2008b). Diseño de un codificador y decodificador digital Reed-Solomon usando programación en VHDL. *Nexo Revista Científica*, *21*(01), 2–10. Retrieved from http://lamjol.info/index.php/NEXO/article/view/393

Sandoval, C., & Fedón, A. (2008c). Programación VHDL de algoritmos de codificación para dispositivos de hardware reconfigurable. *Rev.Int.Mét.Num.Cálc.Dis.ing.*, *24*(1), 3–11. Retrieved from http://upcommons.upc.edu/revistes/bitstream/2099/10415/1/I - V24N1.pdf

Sandoval-Ruiz, C. (2012). Codificador RS (n,k) basado en LFCS : caso de estudio RS (7,3). *Rev. Fac. Ing. Univ. Antioquia*, *64*, 68–78.

Sandoval-Ruiz, C. E., & Fedón-Rovira, A. (2013). Codificador RS (255,k) en hardware reconfigurable orientado a radio. *Ingeniería y Universidad*, *17*(1), 77–91. Retrieved from http://revistas.javeriana.edu.co/index.php/iyu/article/view/1713

Saqib Nazar, A. (2004). *Implementación Eficiente de Algoritmos Criptográficos en Dispositivos de Hardware Reconfigurable*. *Electrical Engineering*. Tesis de Doctorado, CINVESTAV. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Implementación+Eficiente+de+Algoritmos+Criptográficos+en+Dispositivos+de+Hardware+Reconfigurable#0

Savaš, E., Tenca, A., & Koç, Ç. (2000). A scalable and unified multiplier architecture for finite fields GF (p) and GF (2 m). *Cryptographic Hardware and Embedded* (pp. 277–292). Springer. Retrieved from http://www.springerlink.com/index/fduvnn4nhl2w4uvb.pdf

Shih, C. (2000). *Soft IP Generator of Reed-Solomon Codec for Communication Systems*. *Electrical Engineering*.

Sobe, P. (2010). Parallel Reed/Solomon Coding on Multicore Processors. *2010 International Workshop on Storage Network Architecture and Parallel I/Os*, 71–80. doi:10.1109/SNAPI.2010.16

Song, M., Kuo, S., & Lan, I. (2007). A Low Complexity Design of Reed Solomon Code Algorithm for Advanced RAID System. *IEEE Transactions on Consumer Electronics*, *53*(2), 265–273. doi:10.1109/TCE.2007.381684

Sutter, G, & Boemo, E. (2007). EXPERIMENTS IN LOW POWER FPGA DESIGN. *Latin American Applied Research*, *104*, 99–104.

Sutter, Gustavo. (2005). *Aportes a la Reducción de Consumo en FPGAs Tesis Doctoral*. Universidad Autónoma de Madrid.

Tejeda-calderón, V. C., García-martínez, M. A., & Posada-gómez, R. (n.d.). IMPLEMENTACIÓN EN FPGA DE UN MULTIPLICADOR POR DIGITOS SOBRE CAMPOS FINITOS GF (2 m) División de Estudios de Postgrado Orizaba , Veracruz, 2–5.

Todorovich, E. (2006). *Estimación Estadística de Consumo en FPGAs*. Retrieved from http://www.ii.uam.es/~etodorov/Download/thesis_ETS.pdf

Todorovich, E., Acosta, G. S. N., & Nacional, U. (n.d.). Relación entre Velocidad y Consumo en FPGAs. *Potencia*, 1–6.

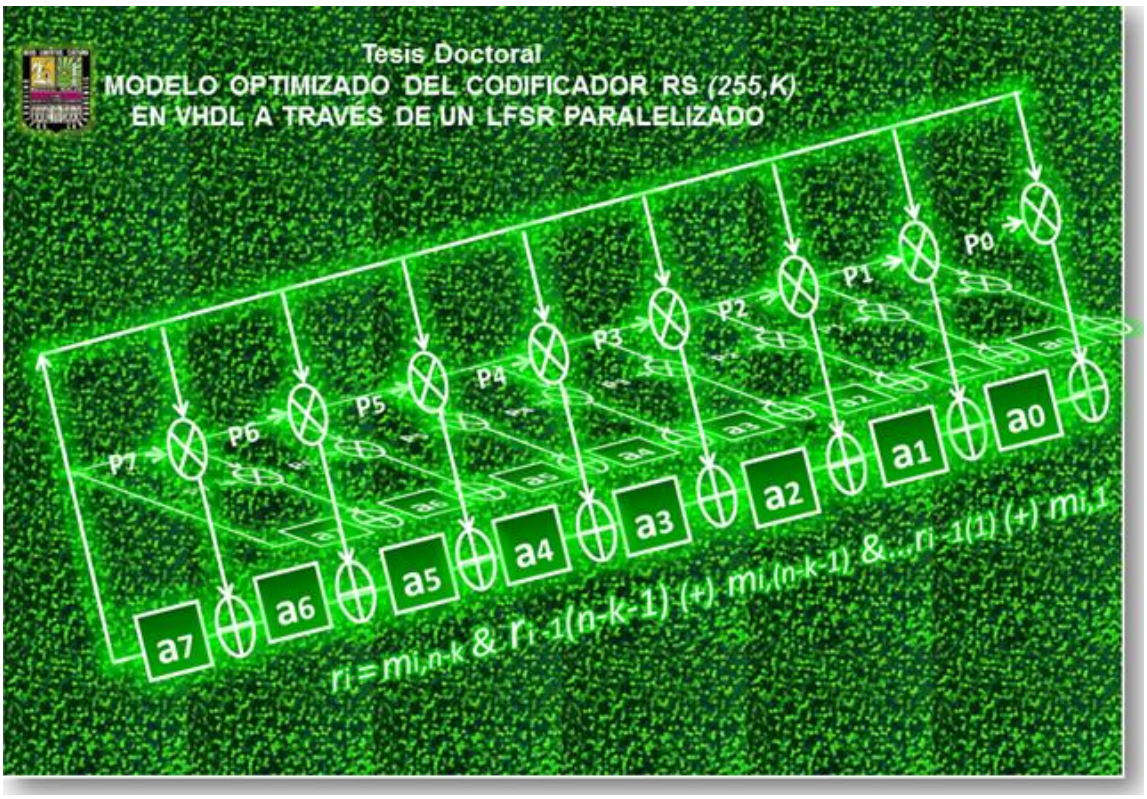
Torres. (1999). ITU - Telecommunication Standardization Sector STUDY GROUP 15 TITLE : Temporary Document BM-087 Original : English G . gen : Comparison of simulation results for different Coding Techniques (Uncoded , Reed- 3 . - Parallel Covolutional Concatenated Codes. *Reason*, *1*(May), 1–4.

Xilinx. (2011). Xilinx DS251, LogiCORE IP Reed-Solomon Encoder v7.1 Data Sheet. *Cadence*, 1–16.

Xilinx. (2012). *LogiCORE IP Reed-Solomon Encoder v8.0*. *Notes*.

Yap, H., Khoo, K., & Poschmann, A. (2010). *Parallelizing the Camellia and SMS4 Block Ciphers - Extended Version*. *Sciences-New York*. Retrieved from http://eprint.iacr.org/2010/426.pdf

ANEXOS


Tesis Doctoral
MODELO OPTIMIZADO DEL CODIFICADOR RS (255,K)
EN VHDL A TRAVÉS DE UN LFSR PARALELIZADO
P7
P6
P5
P4
P3
P2
P1
P0
a7
a6
a5
a4
a3
a2
a1
a0
ri=mi,n-k & ri-1(n-k-1) (+) mi,(n-k-1) &...ri-1(1) (+) mi,1

ANEXO A CÓDIGOS VHDL, SIMULACIÓN Y REPORTES DE SÍNTESIS

Tabla A.1.a Código VHDL para el codificador Reed Solomon (255,247)

```
entity Codificador_RS is
generic(length:integer:=9; --length corresponde a n-k+1
width :integer:=8);        --width m longitud del símbolo
Port ( D_in : in std_logic_vector(7 downto 0);
       clk,hab : in std_logic;
          D_dato: inout std_logic_vector(7 downto 0);
       salida : out std_logic_vector(7 downto 0));
end Codificador_RS;
architecture Behavioral of Codificador_RS is
--se define el tipo de variable a emplear como una matriz
type memoria is array (0 to length-1) of std_logic_vector(width-1 downto 0);
signal coef: std_logic_vector(7 downto 0);
signal dato1,dato2,dato3,dato4,dato5,dato6,dato7,dato8:
std_logic_vector(7 downto 0);
component mult is
port (D_dato: in std_logic_vector (7 downto 0);
      coef: in std_logic_vector (7 downto 0);
      datox: out std_logic_vector (7 downto 0));
end component;
begin
C1: mult  port map (D_dato,"11111111",dato1);--255
C2: mult  port map (D_dato,"00001011",dato2);--11
C3: mult  port map (D_dato,"01010001",dato3);--81
C4: mult  port map (D_dato,"00110110",dato4);--54
C5: mult  port map (D_dato,"11101111",dato5);--239
C6: mult  port map (D_dato,"10101101",dato6);--173
C7: mult  port map (D_dato,"11001000",dato7);--200
C8: mult  port map (D_dato,"00011000",dato8);--24
process (clk)
variable memoria_v:memoria:=(others=>"00000000");
begin
   if (hab='1')  then
       D_dato<=( memoria_v(0) xor D_in);
   else
       D_dato<="00000000";
   end if;
   if (clk'event and clk='1')then
      memoria_v(0):=memoria_v(1)xor dato1;
      memoria_v(1):=memoria_v(2)xor dato2;
      memoria_v(2):=memoria_v(3)xor dato3;
      memoria_v(3):=memoria_v(4)xor dato4;
      memoria_v(4):=memoria_v(5)xor dato5;
      memoria_v(5):=memoria_v(6)xor dato6;
      memoria_v(6):=memoria_v(7)xor dato7;
      memoria_v(7):= dato8;
   end if;
   if hab='1' then
      salida<=D_in;
   else
      salida<=memoria_v(0);
   end if;
end process;
end Behavioral;
```

Tabla A.1.b. Código en Matlab para validación del RS(255,247)

```
N=255; % TAMAÑO DEL MANSAJE (n SIMBOLOS)
K=247; % k SIMBOLOS DE DATOS
n=8; % número de bits por símbolo
B=0;
GENPOLY=RSGENPOLY(N,K,285,B);
%coeficientes=[1 255 11 81 54 239 173 200 24]
D=[1:1:247];
msg=gf(D,n);
code=rsenc(msg,N,K,GENPOLY);
redundancia=code(248:255)
%redundancia= [7 241 48 172 84 143 66 243]
```

rs255 Project Status (08/22/2011 - 12:28:53)			
Project File:	rs255.ise	**Implementation State:**	Synthesized
Module Name:	Codificador_RS	**• Errors:**	
Target Device:	xc5vlx50t-3ff1136	**• Warnings:**	
Product Version:	ISE 11.1	**• Routing Results:**	
Design Goal:	Balanced	**• Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	64	28800	0%
Number of Slice LUTs	142	28800	0%
Number of fully used LUT-FF pairs	64	142	45%
Number of bonded IOBs	26	480	5%
Number of BUFG/BUFGCTRLs	1	32	3%

Figura A.1.1 Resultados de síntesis (255,247)

Name	Value	Used	Total Available	Utilization (%)
Clocks	0.00095 (W)	1	—	—
Logic	0.00004 (W)	142	28800	0.5
Signals	0.00042 (W)	152	—	—
IOs	0.01669 (W)	18	540	3.3
Total Quiescent Power	**0.56766 (W)**			
Total Dynamic Power	**0.01810 (W)**			
Total Power	**0.58576 (W)**			
Junction Temp	**52.0 (degrees C)**			

Figura A.1.2 Consumo de Potencia del RS(255,247)

Tabla A.2.a. Descripción VHDL del codificador RS(255,239)

```
entity Codificador_RS is
generic(length:integer:= 17;    --length corresponde a n-k+1
width :integer:=8);             -- width corresponde a m longitud del simbolo
Port ( D_in : in std_logic_vector(7 downto 0);
       clk,hab : in std_logic;
       D_dato: inout std_logic_vector(7 downto 0);
       salida : out std_logic_vector(7 downto 0));
end Codificador_RS;
architecture Behavioral of Codificador_RS is --se define el tipo de variable matriz
type memoria is array (0 to length-1) of std_logic_vector(width-1 downto 0);
signal coef: std_logic_vector(7 downto 0);
signal dato1,dato2,dato3,dato4,dato5,dato6,dato7,dato8,dato9,dato10,dato11,dato12,dato13,
dato14,dato15,dato16:std_logic_vector(7 downto 0);
component mult is
port  (D_dato: in std_logic_vector (7 downto 0);
       coef: in std_logic_vector (7 downto 0);
       datox: out std_logic_vector (7 downto 0));
end component;
begin
C01:  mult  port map  (D_dato,"00111011",dato1);--59
C02:  mult  port map  (D_dato,"00001101",dato2);--13
C03:  mult  port map  (D_dato,"01101000",dato3);--104
C04:  mult  port map  (D_dato,"10111101",dato4);--189
C05:  mult  port map  (D_dato,"01000100",dato5);--68
C06:  mult  port map  (D_dato,"11010001",dato6);--209
C07:  mult  port map  (D_dato,"00011110",dato7);--30
C08:  mult  port map  (D_dato,"00001000",dato8);--8
C09:  mult  port map  (D_dato,"10100011",dato9);--163
C10:  mult  port map  (D_dato,"01000001",dato10);--65
C11:  mult  port map  (D_dato,"00101001",dato11);--41
C12:  mult  port map  (D_dato,"11100101",dato12);--229
C13:  mult  port map  (D_dato,"01100010",dato13);--98
C14:  mult  port map  (D_dato,"00110010",dato14);--50
C15:  mult  port map  (D_dato,"00100100",dato15);--36
C16:  mult  port map  (D_dato,"00111011",dato16);--59
process (clk)
variable memoria_v:memoria:=(others=>"00000000");
begin
   if (hab='1')  then
      D_dato<=( memoria_v(0) xor D_in);
   else
      D_dato<="00000000";
   end if;
   if (clk'event and clk='1')then
      memoria_v(0):=memoria_v(1)xor dato1;
      memoria_v(1):=memoria_v(2)xor dato2;
      memoria_v(2):=memoria_v(3)xor dato3;
      memoria_v(3):=memoria_v(4)xor dato4;
      memoria_v(4):=memoria_v(5)xor dato5;
      memoria_v(5):=memoria_v(6)xor dato6;
      ...
      -- memoria_v(i-1):= memoria_v(i) xor datoi;
      ...
      memoria_v(14):=memoria_v(15)xor dato15;
      memoria_v(15):=dato16;
   end if;
   if hab='1' then
      salida<=D_in;
   else
      salida<=memoria_v(0);
   end if;
end process;
end Behavioral;
```

Tabla A.2.b. Código en Matlab para validación del RS(255,239)

```
N=255; % TAMAÑO DEL MANSAJE (7 SIMBOLOS)
K=239; %NUMERO DE SIMBOLOS DE DATOS
n=8;   % número de bits por símbolo
B=0;
GENPOLY=RSGENPOLY(N,K,285,B);
D=[1:1:239];
msg=gf(D,n);
code=rsenc(msg,N,K,GENPOLY);
%GENPOLY = GF(2^8) array.
Primitive polynomial = D^8+D^4+D^3+D^2+1 (285)
%coeficientes = [1 59 13 104 189 68 209 30 8 163 65 41 229 98 50 36 59]
redundancia=code(240:255)
%redundancia=[ 1 126 147 48 155 224 3 157 29 226 40 114 61 30 244 75]
```

rs239 Project Status (08/11/2011 - 20:54:14)			
Project File:	rs239.ise	**Implementation State:**	Synthesized
Module Name:	Codificador_RS	**• Errors:**	
Target Device:	xc5vlx30t-3ff323	**• Warnings:**	
Product Version:	ISE 11.1	**• Routing Results:**	
Design Goal:	Balanced	**• Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	128	19200	0%
Number of Slice LUTs	159	19200	0%
Number of fully used LUT-FF pairs	128	159	80%
Number of bonded IOBs	26	172	15%
Number of BUFG/BUFGCTRLs	1	32	3%

Figura A.2.1. Resultados de la síntesis del RS (255,239)

Name	Value	Used	Total Available	Utilization (%)
Clocks	0.00119 (W)	1	—	—
Logic	0.00004 (W)	159	28800	0.6
Signals	0.00032 (W)	160	—	—
IOs	0.01669 (W)	18	540	3.3
Total Quiescent Power	**0.56767 (W)**			
Total Dynamic Power	**0.01824 (W)**			
Total Power	**0.58590 (W)**			
Junction Temp	**52.0 (degrees C)**			

Figura A.2.2. Consumo de potencia del RS (255,239)

Tabla A.3.a. Descripción VHDL del codificador RS(255,223)

```
entity Codificador_RS is
generic(length:integer:= 33;    --length corresponde a n-k+1
width :integer:=8);             -- width m longitud del símbolo
Port ( D_in : in std_logic_vector(7 downto 0);
       clk,hab : in std_logic;
          D_dato: inout std_logic_vector(7 downto 0);
          salida : out std_logic_vector(7 downto 0));
end Codificador_RS;

component mult is
port (D_dato: in std_logic_vector (7 downto 0);
      coef: in std_logic_vector (7 downto 0);
      datox: out std_logic_vector (7 downto 0));
end component;

architecture Behavioral of Codificador_RS is
--se define el tipo de variable a emplear como una matriz
type memoria is array (0 to length-1) of std_logic_vector(width-1 downto 0);
signal coef: std_logic_vector(7 downto 0);
signal
coef,dato1,dato2,dato3,dato4,dato5,dato6,dato7,dato8,dato9,dato10,dato11,dato12,dato13,da
to14,dato15,dato16,dato17,dato18,dato19,dato20,dato21,dato22,dato23,dato24,dato25,dato26,
dato27,dato28,dato29,dato30,dato31,dato32: std_logic_vector(7 downto 0);

begin
C01:  mult  port map  (D_dato,"01110100",dato1); --116
C02:  mult  port map  (D_dato,"01000000",dato2); --64
C03:  mult  port map  (D_dato,"00110100",dato3); --52
C04:  mult  port map  (D_dato,"10101110",dato4); --174
C05:  mult  port map  (D_dato,"00110110",dato5); --54
C06:  mult  port map  (D_dato,"01111110",dato6); --126
C07:  mult  port map  (D_dato,"00010000",dato7); --16
C08:  mult  port map  (D_dato,"11000010",dato8); --194
C09:  mult  port map  (D_dato,"10100001",dato9); --162
C10:  mult  port map  (D_dato,"00100001",dato10);--33
C11:  mult  port map  (D_dato,"00100001",dato11);--33
C12:  mult  port map  (D_dato,"10011101",dato12);--157
C13:  mult  port map  (D_dato,"10110000",dato13);--176
C14:  mult  port map  (D_dato,"11000101",dato14);--197
C15:  mult  port map  (D_dato,"11100001",dato15);--225
C16:  mult  port map  (D_dato,"00001100",dato16);--12
C17:  mult  port map  (D_dato,"00111011",dato17);--59
C18:  mult  port map  (D_dato,"00110111",dato18);--55
C19:  mult  port map  (D_dato,"11111101",dato19);--253
C20:  mult  port map  (D_dato,"11100100",dato20);--228
C21:  mult  port map  (D_dato,"10010100",dato21);--148
C22:  mult  port map  (D_dato,"00101111",dato22);--47
C23:  mult  port map  (D_dato,"10110011",dato23);--179
C24:  mult  port map  (D_dato,"10111001",dato24);--185
C25:  mult  port map  (D_dato,"00011000",dato25);--24
C26:  mult  port map  (D_dato,"10001010",dato26);--138
C27:  mult  port map  (D_dato,"11111101",dato27);--253
C28:  mult  port map  (D_dato,"00010100",dato28);--20
C29:  mult  port map  (D_dato,"10001110",dato29);--142
C30:  mult  port map  (D_dato,"00110111",dato30);--55
C31:  mult  port map  (D_dato,"10101100",dato31);--172
C32:  mult  port map  (D_dato,"01011000",dato32);--88
process (clk)
variable memoria_v:memoria:=(others=>"00000000");
begin
if (hab='1')  then
D_dato<=( memoria_v(0) xor D_in);
else
D_dato<="00000000";
end if;
```

```
if (clk'event and clk='1')then
memoria_v(0):=memoria_v(1)xor dato1;
memoria_v(1):=memoria_v(2)xor dato2;
memoria_v(2):=memoria_v(3)xor dato3;
memoria_v(3):=memoria_v(4)xor dato4;
memoria_v(4):=memoria_v(5)xor dato5;
memoria_v(5):=memoria_v(6)xor dato6;
memoria_v(6):=memoria_v(7)xor dato7;
memoria_v(7):=memoria_v(8)xor dato8;
memoria_v(8):=memoria_v(9)xor dato9;
memoria_v(9):=memoria_v(10)xor dato10;
memoria_v(10):=memoria_v(11)xor dato11;
memoria_v(11):=memoria_v(12)xor dato12;
memoria_v(12):=memoria_v(13)xor dato13;
memoria_v(13):=memoria_v(14)xor dato14;
memoria_v(14):=memoria_v(15)xor dato15;
memoria_v(15):=memoria_v(16)xor dato16;
memoria_v(16):=memoria_v(17)xor dato17;
memoria_v(17):=memoria_v(18)xor dato18;
memoria_v(18):=memoria_v(19)xor dato19;
memoria_v(19):=memoria_v(20)xor dato20;
memoria_v(20):=memoria_v(21)xor dato21;
memoria_v(21):=memoria_v(22)xor dato22;
memoria_v(22):=memoria_v(23)xor dato23;
memoria_v(23):=memoria_v(14)xor dato24;
memoria_v(24):=memoria_v(25)xor dato25;
memoria_v(25):=memoria_v(26)xor dato26;
memoria_v(26):=memoria_v(27)xor dato27;
memoria_v(27):=memoria_v(28)xor dato28;
memoria_v(28):=memoria_v(29)xor dato29;
memoria_v(29):=memoria_v(30)xor dato30;
memoria_v(30):=memoria_v(31)xor dato31;
memoria_v(31):=dato32;
end if;
if hab='1' then
salida<=D_in;
else
salida<=memoria_v(0);
end if;
end process;
end Behavioral;
```

Tabla A.3.b.Código en Matlab para validación del RS(255,223)

```
N=255; % TAMAÑO DEL MANSAJE (n SIMBOLOS)
K=223; % k SIMBOLOS DE DATOS
n=8;   % número de bits por símbolo
B=0;
GENPOLY=RSGENPOLY(N,K,285,B);
%GENPOLY=[116 64 52 174 54 126 16 194 162 33 33 157 176 197 225 12 59
55 253 228 148 47 179 185 24 138 253 20 142 55 172 88]
D=[1:1:223];
msg=gf(D,n);
code=rsenc(msg,N,K,GENPOLY);
%solo observar los simbolos de redundancia
redundancia_code=code(224:255)
%redundancia_code=[173 69 254 212 67 87 70 169 130 39 34 115 90 135 70
219 177 10 253 16 80 113 13 233 41 145 93 81 208 213 106 197]
```

RS223 Project Status (08/22/2011 - 12:23:50)			
Project File:	RS223.ise	**Implementation State:**	Synthesized
Module Name:	Codificador_RS	**• Errors:**	
Target Device:	xc5vlx50t-3ff1136	**• Warnings:**	
Product Version:	ISE 11.1	**• Routing Results:**	
Design Goal:	Balanced	**• Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	256	28800	0%
Number of Slice LUTs	305	28800	1%
Number of fully used LUT-FF pairs	256	305	83%
Number of bonded IOBs	18	480	3%
Number of BUFG/BUFGCTRLs	1	32	3%

Figura A.3.1 Resultados de síntesis (255,223)

Name	Value	Used	Total Available	Utilization (%)
Clocks	0.00143 (W)	1	—	—
Logic	0.00004 (W)	305	28800	1.1
Signals	0.00073 (W)	300	—	—
IOs	0.01669 (W)	18	540	3.3
Total Quiescent Power	**0.56768 (W)**			
Total Dynamic Power	**0.01889 (W)**			
Total Power	**0.58657 (W)**			
Junction Temp	**52.0 (degrees C)**			

Figura A.3.2. Consumo de potencia del RS (255,223)

Tabla A.4.a. Código VHDL del RS(127,111)

```
entity encoder is
    generic(length:integer:= 17;    --length corresponde a n-k+1
            width :integer:=7);     -- width corresponde a m longitud del simbolo
    Port ( D_in : in std_logic_vector(6 downto 0);
           clk,hab : in std_logic;
                --s1,s2,s3,s4 : out std_logic_vector(2 downto 0);
                --
dato1,dato2,dato3,dato4,dato5,dato6,dato7,dato8,dato9,dato10,dato11,dato12,dato13,dato14,
dato15,dato16:inout std_logic_vector(7 downto 0);
                D_dato: inout std_logic_vector(6 downto 0);
                salida : out std_logic_vector(6 downto 0));
end encoder;
architecture Behavioral of encoder is
--se define el tipo de variable a emplear como una matriz
type memoria is array (0 to length-1) of std_logic_vector(width-1 downto 0);   --
signal
coef,dato1,dato2,dato3,dato4,dato5,dato6,dato7,dato8,dato9,dato10,dato11,dato12,dato13,da
to14,dato15,dato16: std_logic_vector(6 downto 0);
component mult is
port     (D_dato: in std_logic_vector (6 downto 0);
               coef: in std_logic_vector (6 downto 0);
             datox: out std_logic_vector (6 downto 0));
end component;
--signal D_d,s1,s2,s3,s4 :  std_logic_vector(7 downto 0);
begin
--C01:mult  port map  (D_dato,"00000001",dato1);--1
C01:  mult  port map  (D_dato,"0001101",dato1);--13
C02:  mult  port map  (D_dato,"0110100",dato2);--52
C03:  mult  port map  (D_dato,"1101000",dato3);--104
C04:  mult  port map  (D_dato,"0110111",dato4);--55
C05:  mult  port map  (D_dato,"0001001",dato5);--9
C06:  mult  port map  (D_dato,"0000110",dato6);--6
C07:  mult  port map  (D_dato,"1111010",dato7);--122
C08:  mult  port map    (D_dato,"0000100",dato8);--4
C09:  mult  port map  (D_dato,"1101000",dato9);--104
C10:  mult  port map  (D_dato,"0000101",dato10);--5
C11:  mult  port map  (D_dato,"0111110",dato11);--62
C12:  mult  port map  (D_dato,"1011100",dato12);--92
C13:  mult  port map  (D_dato,"0110000",dato13);--48
C14:  mult  port map  (D_dato,"1100001",dato14);--97
C15:  mult  port map  (D_dato,"0000101",dato15);--5
C16:  mult  port map  (D_dato,"1001101",dato16);--77

process (clk)
variable memoria_v:memoria:=(others=>"0000000");
begin

   if (hab='1')  then
      D_dato<=( memoria_v(0) xor D_in);
   else
      D_dato<="0000000";
   end if;

if (clk'event and clk='1')then
  memoria_v(0) :=memoria_v(1)xor dato1;
  memoria_v(1) :=memoria_v(2)xor dato2;
  memoria_v(2) :=memoria_v(3)xor dato3;
  memoria_v(3) :=memoria_v(4)xor dato4;
  memoria_v(4) :=memoria_v(5)xor dato5;
  memoria_v(5) :=memoria_v(6)xor dato6;
  memoria_v(6) :=memoria_v(7)xor dato7;
  memoria_v(7) :=memoria_v(8)xor dato8;
  memoria_v(8) :=memoria_v(9)xor dato9;
  memoria_v(9) :=memoria_v(10)xor dato10;
  memoria_v(10):=memoria_v(11)xor dato11;
  memoria_v(11):=memoria_v(12)xor dato12;
  memoria_v(12):=memoria_v(13)xor dato13;
  memoria_v(13):=memoria_v(14)xor dato14;
  memoria_v(14):=memoria_v(15)xor dato15;
 memoria_v(15):=dato16;
end if;
if hab='1' then
  salida<=D_in;
else
  salida<=memoria_v(0);
end if;
end process;
end Behavioral;
```

Tabla A.4.b.Código en Matlab para validación del RS(127,111)

```
N=127; % TAMAÑO DEL MANSAJE (n SIMBOLOS)
K=111; % k SIMBOLOS DE DATOS
n=7; % número de bits por símbolo
B=0;
GENPOLY=RSGENPOLY(N,K,137,B);
%GENPOLY=[1 13 52 104 55 9 6 122 4 104 5 62 92 48 97 5 77]
D=[1:1:111];
msg=gf(D,n);
code=rsenc(msg,N,K,GENPOLY);
code(112:127)
; %%%%%%%%%%%%%%%%%solo observar los simbolos de redundancia
%redundancia_code=[38 81 22 83 107 48 90 67 40 100 33 92 27 110 3 55]
```

RS112 Project Status (08/23/2011 - 15:06:16)			
Project File:	RS112.ise	**Implementation State:**	Synthesized
Module Name:	encoder	**• Errors:**	
Target Device:	xc5vlx50t-3ff1136	**• Warnings:**	
Product Version:	ISE 11.1	**• Routing Results:**	
Design Goal:	Balanced	**• Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	112	28800	0%
Number of Slice LUTs	129	28800	0%
Number of fully used LUT-FF pairs	112	129	86%
Number of bonded IOBs	23	480	4%
Number of BUFG/BUFGCTRLs	1	32	3%

Figura A.4.1 Resultados de síntesis (127,111)

Name	Value	Used	Total Available	Utilization (%)
Clocks	0.00071 (W)	1	—	—
Logic	0.00006 (W)	129	19200	0.7
Signals	0.00086 (W)	134	—	—
IOs	0.02906 (W)	23	220	10.5
Total Quiescent Power	**0.38647 (W)**			
Total Dynamic Power	**0.03070 (W)**			
Total Power	**0.41716 (W)**			
Junction Temp	**52.7 (degrees C)**			

Figura A.4.2. Consumo de potencia del RS (127,111)

ANEXO B MULTIPLICADORES DE CAMPOS FINITOS DE GALOIS GF(2^M)

El esquema circuital del multiplicador GF(2m) comprende tres etapas: el circuito de operación mod p(x) implementado por un circuito LFSR, el módulo de productos parciales con el elemento Bi implementado por un conjunto de compuertas AND paralelas, el acumulador de productos parciales, implementado a través de compuertas XOR en caso de ser secuencial estás son realimentadas utilizando dispositivos de almacenamiento de memoria FlipFlop. Para el caso de *el polinomio generador del Campo Finito es:* $P(x)=x^8+x^4+x^3+x^2+1$, se observa en la figura B.1 la configuración del LFSR.

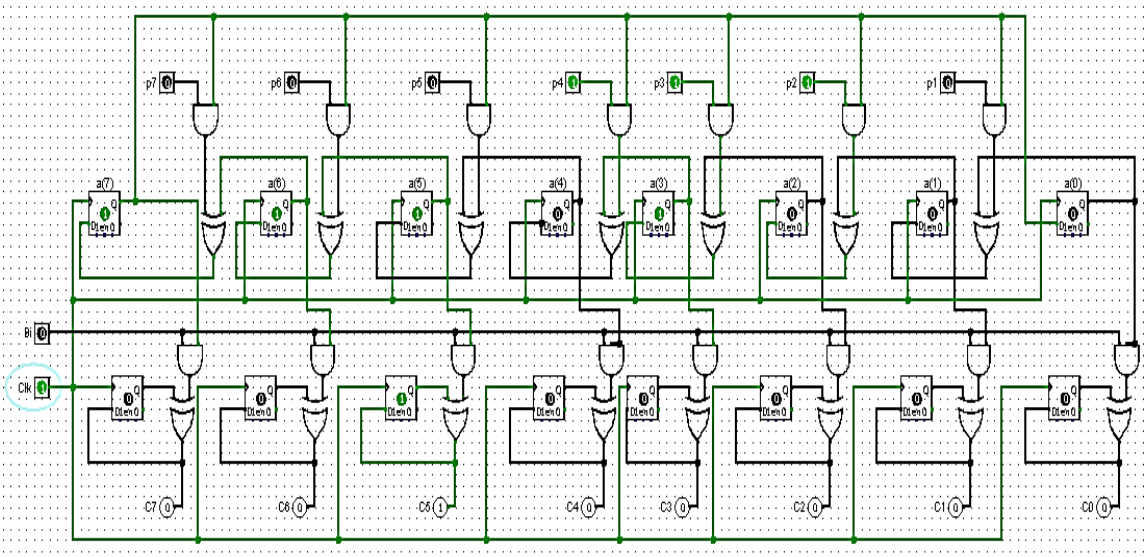


Figura B.1. Esquema Circuital del Multiplicador GF(2^8)

Tabla B.1. Multiplicador GF(128)

```
entity mult is
port     (D_dato: in std_logic_vector (6 downto 0);
         coef: in std_logic_vector (6 downto 0);
         datox: out std_logic_vector (6 downto 0));
end mult;

architecture Behavioral of mult is

signal            b,a2,a3,a4,a5,a6,a7,a8,b1,b2,b3,b4,b5,b6,b7,b8,c1,c2,c3,c4,c5,c6,c7,c8:
std_logic_vector (6 downto 0);
signal p: std_logic_vector (7 downto 0);
signal a1: std_logic_vector (6 downto 0);
begin
b<= D_dato;     -- dato de entrada para la multiplicación
-- c = a mod p(x) * b
p<="10001001"; -- Primitive polynomial = 137
a1<= coef;
u1: a2<=a1(5 downto 3)&(a1(2)xor a1(6))& a1(1 downto 0)& a1(6);
u2: a3<=a2(5 downto 3)&(a2(2)xor a2(6))& a2(1 downto 0)& a2(6);
u3: a4<=a3(5 downto 3)&(a3(2)xor a3(6))& a3(1 downto 0)& a3(6);
u4: a5<=a4(5 downto 3)&(a4(2)xor a4(6))& a4(1 downto 0)& a4(6);
u5: a6<=a5(5 downto 3)&(a5(2)xor a5(6))& a5(1 downto 0)& a5(6);
u6: a7<=a6(5 downto 3)&(a6(2)xor a6(6))& a6(1 downto 0)& a6(6);
--u1: a2<=a1(5 downto 1)&(a1(0)xor a1(6))& a1(6);
--u2: a3<=a2(5 downto 1)&(a2(0)xor a2(6))& a2(6);
--u3: a4<=a3(5 downto 1)&(a3(0)xor a3(6))& a3(6);
--u4: a5<=a4(5 downto 1)&(a4(0)xor a4(6))& a4(6);
--u5: a6<=a5(5 downto 1)&(a5(0)xor a5(6))& a5(6);
--u6: a7<=a6(5 downto 1)&(a6(0)xor a6(6))& a6(6);
b1<=  b(0) & b(0) & b(0) & b(0) & b(0) & b(0) & b(0);
b2<=  b(1) & b(1) & b(1) & b(1) & b(1) & b(1) & b(1);
b3<=  b(2) & b(2) & b(2) & b(2) & b(2) & b(2) & b(2);
b4<=  b(3) & b(3) & b(3) & b(3) & b(3) & b(3) & b(3);
b5<=  b(4) & b(4) & b(4) & b(4) & b(4) & b(4) & b(4);
b6<=  b(5) & b(5) & b(5) & b(5) & b(5) & b(5) & b(5);
b7<=  b(6) & b(6) & b(6) & b(6) & b(6) & b(6) & b(6);
c1<=a1 and b1; --c1<=coef and b1;
c2<=a2 and b2;
c3<=a3 and b3;
c4<=a4 and b4;
c5<=a5 and b5;
c6<=a6 and b6;
c7<=a7 and b7;
datox<=c1 xor c2 xor c3 xor c4 xor c5 xor c6 xor c7;
end Behavioral;
```

Tabla B.2. Multiplicador GF(256) Generalizado

```
entity mult is
port     (D_dato: in std_logic_vector (7 downto 0);
           coef: in std_logic_vector (7 downto 0);
           datox: out std_logic_vector (7 downto 0));
end mult;
architecture Behavioral of mult is
signal            b,a2,a3,a4,a5,a6,a7,a8,b1,b2,b3,b4,b5,b6,b7,b8,c1,c2,c3,c4,c5,c6,c7,c8:
std_logic_vector (7 downto 0);
signal p: std_logic_vector (8 downto 0);
begin
b<= D_dato; -- dato de entrada para la multiplicación
a1<= coef;
p<="100011101"; -- Primitive polynomial = D^8+D^4+D^3+D^2+1 (285)
u1: a2<=a1(6 downto 4)&(a1(3)xor a1(7))&(a1(2)xor a1(7))&(a1(1)xor a1(7))&a1(0)& a1(7);
u2: a3<=a2(6 downto 4)&(a2(3)xor a2(7))&(a2(2)xor a2(7))&(a2(1)xor a2(7))&a2(0)& a2(7);
u3: a4<=a3(6 downto 4)&(a3(3)xor a3(7))&(a3(2)xor a3(7))&(a3(1)xor a3(7))&a3(0)& a3(7);
u4: a5<=a4(6 downto 4)&(a4(3)xor a4(7))&(a4(2)xor a4(7))&(a4(1)xor a4(7))&a4(0)& a4(7);
u5: a6<=a5(6 downto 4)&(a5(3)xor a5(7))&(a5(2)xor a5(7))&(a5(1)xor a5(7))&a5(0)& a5(7);
u6: a7<=a6(6 downto 4)&(a6(3)xor a6(7))&(a6(2)xor a6(7))&(a6(1)xor a6(7))&a6(0)& a6(7);
u7: a8<=a7(6 downto 4)&(a7(3)xor a7(7))&(a7(2)xor a7(7))&(a7(1)xor a7(7))&a7(0)& a7(7);
b1<= b(0) & b(0) & b(0) & b(0) & b(0) & b(0) & b(0) & b(0);
b2<= b(1) & b(1) & b(1) & b(1) & b(1) & b(1) & b(1) & b(1);
b3<= b(2) & b(2) & b(2) & b(2) & b(2) & b(2) & b(2) & b(2);
b4<= b(3) & b(3) & b(3) & b(3) & b(3) & b(3) & b(3) & b(3);
b5<= b(4) & b(4) & b(4) & b(4) & b(4) & b(4) & b(4) & b(4);
b6<= b(5) & b(5) & b(5) & b(5) & b(5) & b(5) & b(5) & b(5);
b7<= b(6) & b(6) & b(6) & b(6) & b(6) & b(6) & b(6) & b(6);
b8<= b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7);
c1<=a1 and b1;
c2<=a2 and b2;
c3<=a3 and b3;
c4<=a4 and b4;
c5<=a5 and b5;
c6<=a6 and b6;
c7<=a7 and b7;
c8<=a8 and b8
datox<=c1 xor c2 xor c3 xor c4 xor c5 xor c6 xor c7 xor c8;
end Behavioral;
```

Tabla B.3. Multiplicador GF(256) particularizado: caso coef = 59

```
entity mult59 is
port     (D_dato: in std_logic_vector (7 downto 0);
       datox: out std_logic_vector (7 downto 0));
end mult59;
architecture Behavioral of mult59 is                                                --
donde c = a mod p(x) * b
signal a1,a2,a3,a4,a5,a6,a7,a8: std_logic_vector (7 downto 0);
begin                                        --p <="100011101"; -- Primitive polynomial =
D^8+D^4+D^3+D^2+1 (285)
a1<=D_dato;    -- dato de entrada para la multiplicación
u1: a2<=a1(6 downto 4)&(a1(3)xor a1(7))&(a1(2)xor a1(7))&(a1(1)xor a1(7))&a1(0)& a1(7);
u2: a3<=a2(6 downto 4)&(a2(3)xor a2(7))&(a2(2)xor a2(7))&(a2(1)xor a2(7))&a2(0)& a2(7);
u3: a4<=a3(6 downto 4)&(a3(3)xor a3(7))&(a3(2)xor a3(7))&(a3(1)xor a3(7))&a3(0)& a3(7);
u4: a5<=a4(6 downto 4)&(a4(3)xor a4(7))&(a4(2)xor a4(7))&(a4(1)xor a4(7))&a4(0)& a4(7);
u5: a6<=a5(6 downto 4)&(a5(3)xor a5(7))&(a5(2)xor a5(7))&(a5(1)xor a5(7))&a5(0)& a5(7);
u6: a7<=a6(6 downto 4)&(a6(3)xor a6(7))&(a6(2)xor a6(7))&(a6(1)xor a6(7))&a6(0)& a6(7);
u7: a8<=a7(6 downto 4)&(a7(3)xor a7(7))&(a7(2)xor a7(7))&(a7(1)xor a7(7))&a7(0)& a7(7);
-- los que son unos se implementan b<=coef<="00111011" – los unos b6,b5,b4,b2,b1
datox<=a6 xor a5 xor a4 xor a2 xor a1;
end Behavioral;
```

Tabla B.4.Código en Matlab para validación del Multiplicador GF(256)

```
% Definición de los elementos en el campo GF(2^m), viene dado por:
%( gf([operandos],m)
% A corresponde al operando de datos en GF desde 1…16, vector
traspuesto para operarse en columna
A= gf(1:16,8)';
% B corresponde al operando del coeficiente de G(x) en el campo GF(2^8)
B= gf([1 59 13 104 189 68 209 30 8 163 65 41 229 98 50  36 59],8);
% C la operación producto entre los elementos definidos en el campo
GF(2^m)
C= A*B;
```

Tabla B.5. Tabla del Multiplicador A*B en el campo GF(256)

A	B: Coeficientes del Generador de Redundancia RS(255,239)															
01	001	059	013	104	189	068	209	030	008	163	065	041	229	098	050	036
02	002	118	026	208	103	136	191	060	016	091	130	082	215	196	100	072
03	003	077	023	184	218	204	110	034	024	248	195	123	050	166	086	108
04	004	236	052	189	206	013	099	120	032	182	025	164	179	149	200	144
05	005	215	057	213	115	073	178	102	040	021	088	141	086	247	250	180
06	006	154	046	109	169	133	220	068	048	237	155	246	100	081	172	216
07	007	161	035	005	020	193	013	090	056	078	218	223	129	051	158	252
08	008	197	104	103	129	026	198	240	064	113	050	085	123	055	141	061
09	009	254	101	015	060	094	023	238	072	210	115	124	158	085	191	025
10	010	179	114	183	230	146	121	204	080	042	176	007	172	243	233	117
11	011	136	127	223	091	214	168	210	088	137	241	046	073	145	219	081
12	012	041	092	218	079	023	165	136	096	199	043	241	200	162	069	173
13	013	018	081	178	242	083	116	150	104	100	106	216	045	192	119	137
14	014	095	070	010	040	159	026	180	112	156	169	163	031	102	033	229
15	015	100	075	098	149	219	203	170	120	063	232	138	250	004	019	193
16	016	151	208	206	031	052	145	253	128	226	100	170	246	110	007	122

* Los coeficientes 13 y 104 han sido simulados para validar el comportamiento del multiplicador en campos finitos.

LFCS Project Status (07/31/2013 - 14:38:24)			
Project File:	LFCS.ise	**Implementation State:**	Placed and Routed
Module Name:	LFCS	**• Errors:**	
Target Device:	xc5vlx30t-3ff323	**• Warnings:**	
Product Version:	ISE 11.1	**• Routing Results:**	All Signals Completely Routed
Design Goal:	Balanced	**• Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	0 (Setup: 0, Hold: 0) (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice LUTs	44	19,200	1%	
Number used as logic	44	19,200	1%	
Number using O6 output only	44			
Number of occupied Slices	17	4,800	1%	
Number of occupied SLICEMs	0	1,280	0%	
Number of LUT Flip Flop pairs used	44			
Number with an unused Flip Flop	44	44	100%	
Number with an unused LUT	0	44	0%	
Number of fully used LUT-FF pairs	0	44	0%	
Number of slice register sites lost to control set restrictions	0	19,200	0%	
Number of bonded IOBs	24	172	13%	
Average Fanout of Non-Clock Nets	3.65			

Figura B.2. Resultados de síntesis del Multiplicador GF(256)

Name	Value	Used	Total Available	Utilization (%)
Logic	0.00004 (W)	44	19200	0.2
Signals	0.00035 (W)	60	—	—
IOs	0.01682 (W)	24	192	12.5
Total Quiescent Power	**0.40164 (W)**			
Total Dynamic Power	**0.01721 (W)**			
Total Power	**0.41885 (W)**			
Junction Temp	**50.0 (degrees C)**			

* NOTA: El consumo de potencia estimada de 0.40 mW no considera Pot_IO por ser un componente interno

Figura B.3. Consumo de potencia del Multiplicador GF(256)

```
Total                          6.442ns (3.182ns logic, 3.260ns route)
                                       (49.4% logic, 50.6% route)
```

Figura B.4. Reporte de Retardos del Multiplicador GF(256)

ANEXO C RESULTADOS DE MULTIPLICADORES GF(2^M) PREVIOS

Tabla C.1 Consumo de Compuertas del Multiplicador GF(2^n)

n	Q(y)	AND	XOR	$\mathcal{T}_{and}$	$\mathcal{T}_{xor}$
2	$2,1,0$	4	3	1	2
3	$3,1,0$	9	8	1	3
4	$4,1,0$	16	15	1	3
5	$5,2,0$	25	24	1	5
6	$6,1,0$	36	35	1	4
7	$7,1,0$	49	48	1	4
8	$8,5,3,2,0$	64	84	1	5
9	$9,4,0$	81	80	1	6
10	$10,3,0$	100	99	1	6
11	$11,2,0$	121	120	1	6
12	$12,8,5,1,0$	144	207	1	7
13	$13,7,6,1,0$	169	202	1	6
14	$14,9,7,2,0$	196	282	1	7
15	$15,1,0$	225	224	1	5
16	$16,116,5,0$	256	281	1	6

Tabla C.2 Consumo de Compuertas del Multiplicador GF(2^n)^m

k	n	m	P(x)	mod XOR	AB AND	modP XOR	k^2	AB $\mathcal{T}_{and}$	modP $\mathcal{T}_{xor}$
4	2	2	$1,1,w^2$	1	12	18	16	1	4
6	3	2	$1,1,w^6$	1	27	37	36	1	5
8	4	2	$1,1,w^{14}$	1	48	62	64	1	5
10	5	2	$1,1,w^3$	3	75	95	100	1	7
12	6	2	$1,1,w^{62}$	1	108	130	144	1	6
12	3	4	$1,0,0,1,w^6$	21	81	159	144	1	11
14	7	2	$1,1,w^{124}$	3	147	175	196	1	8
16	4	4	$1,1,1,0,w$	35	144	258	256	1	12
18	9	2	$1,1,w^5$	5	243	281	324	1	8
20	5	4	$1,0,0,w,w$	34	225	360	400	1	14
22	11	2	$1,1,w^{2036}$	11	363	415	484	1	12
24	6	4	$1,w^{62},w^{61},w^3,w^2$	60	324	507	576	1	14
26	13	2	$1,1,w^{8188}$	7	507	665	676	1	10
28	7	4	$1,0,0,w^{126},w^{126}$	46	441	632	784	1	13
30	15	2	$1,1,w^{32766}$	1	675	733	900	1	7
32	4	8	$1,0,0,1,0,0,1,0,w$	91	432	896	1024	1	15

Fuente: (Kindap, 2004)
http://www3.iam.metu.edu.tr/iam/images/8/8b/Nihalk%C4%B1ndapthesis.pdf

Tabla C.3. Consumo de Recursos de los Multiplicadores GF(2^m) comparación Mastrovito simplificado método Ahlquist

Pipelined Finite Field Multipliers										
Finite Field	Generator Polynomial	Mastrovito/Synplify				Our Method				% Imp.
		LB	reg	CT	FD	LB	reg	CT	FD	
GF(8)	1101	8	3	4.312	47.432	8	0	4.312	34.496	27.3
GF(16)	10011	12	11	4.312	99.176	12	0	4.312	51.744	47.8
GF(32)	100101	21	16	4.746	175.602	18	0	4.422	79.596	54.7
GF(32)	111101	20	30	4.318	215.900	21	3	4.422	106.128	50.8
GF(64)	1000011	28	17	4.795	215.775	27	4	4.312	133.672	38.1
GF(64)	1110011	30	31	4.827	294.447	29	4	4.312	142.296	51.7
GF(128)	10000011	37	22	4.795	282.905	39	3	4.727	198.534	29.8
GF(128)	10111111	50	61	5.129	569.319	42	1	4.627	198.961	65.1
GF(128)	11001011	57	42	5.398	534.402	41	1	4.627	194.334	63.6
GF(256)	100011101	54	34	4.896	430.848	52	1	4.312	228.536	47.0
GF(256)	110001101	71	56	5.656	718.312	54	1	4.312	237.160	67.0
GF(256)	111110101	52	50	4.727	482.154	53	1	4.312	232.848	48.3

Combinatorial Finite Field Multipliers								
Finite Field	Generator Polynomial	Mastrovito/Synplify			Our Method			Percent Improvement
		LB	CT	FD	LB	CT	FD	
GF(8)	1101	8	7.698	61.584	7	7.698	53.886	12.5
GF(16)	10011	12	9.316	111.790	12	6.210	74.528	33.3
GF(32)	100101	20	9.520	190.400	18	6.350	114.240	40.0
GF(32)	111101	20	11.569	231.380	20	8.677	173.535	25.0
GF(64)	1000011	28	9.520	266.560	26	9.520	247.520	7.1
GF(64)	1110011	30	11.569	347.070	28	8.673	242.949	30.0
GF(128)	10000011	37	9.621	355.977	36	9.621	346.356	2.7
GF(128)	10111111	50	13.659	682.950	39	8.195	319.621	53.2
GF(128)	11001011	57	12.181	694.317	39	9.136	356.294	48.7
GF(256)	100011101	54	12.124	654.696	51	9.093	463.743	29.2
GF(256)	110001101	71	13.775	978.025	54	8.265	446.310	54.4
GF(256)	110101001	61	14.204	866.444	50	8.522	426.120	50.8
GF(256)	111110101	52	13.659	710.268	51	8.195	417.965	41.2

Fuente: (G. C. Ahlquist et al., 2002)

Tabla C.4. Resultados de la Implementación de los multiplicadores GF(2^4)

Diseño	CLBs	T (ns)	CLB*ns	CLB*ns^2
Massey-Omura	15	40.016	600.240	24019.4
Hasan-Bhargava	19	24.952	474.088	11892.4
LFSR	13	26.092	339.196	8850.3
Morii-Berlekamp	13	23.780	309.140	7351.3
Para-Rosner	5	12.318	61.590	758.7
Mastrovito	6	9.784	58.704	574.4
Pipelined Combinatiorial	7	6.336	44.352	281.4

Fuente: (G. Ahlquist et al., 1999)

Tabla C.5. Comparación del Multiplicador GF (256) respecto a Área, retardos y consumo de potencia

Algorithm	Array Impl.			Fully Sequential					
	Energy (nJ)	Area (CLBs)	Delay (ns)	Total	Synchro	Data Path	Area (CLBs)	Area (FF)	Delay (ns)
m_r	96,1	85	186	71,5	46,8	24,7	57	67	320
s_a	186,4	157	201	104,8	52,5	52,3	33	37	465
mont.	92,7	102	167	38,6	27,2	11,1	34	31	249

Fuente: (G Sutter & Boemo, 2007)

Tabla C.6. Comparación de Complejidades Teóricas del Multiplicador GF

Método Mult	#XOR	#AND	Retardo
Itoh-Tsu.[5]	m^2+2m	m^2+2m+1	$T_{AND}+\lceil \log_2 m + \log_2(m+2) \rceil T_{XOR}$
Hasan [4]	m^2+m-2	m^2	$T_{AND}+(m+\lceil \log_2(m-1) \rceil) T_{XOR}$
Koç-Su.[6]	m^2-1	m^2	$T_{AND}+(2+\lceil \log_2(m-1) \rceil) T_{XOR}$
Halbut. [2]	m^2-1	m^2	$T_{AND}+(1+\lceil \log_2(m-1) \rceil) T_{XOR}$
Zhang [11]	m^2-1	m^2	$T_{AND}+(1+\lceil \log_2(m-1) \rceil) T_{XOR}$
Ntra.Aprox.	m^2-1	m^2	$T_{AND}+(1+\lceil \log_2(m-1) \rceil) T_{XOR}$

Fuente: (J. L. Imaña et al., 2002)

Tabla C.6. Slice para multiplicadores de 8 bits

Multiplicador	*Área (Slices)*	*% de ocupación en FPGA*
Mastrovito	32	16 %
MSB	14	7 %
LSB	20	10 %
Mastrovito AOP	34	17 %
Dígito Serie D=2	27	14 %
Dígito Serie D=3	26	13 %
Dígito Serie D=4	31	16 %

Fuente: (Sánchez Corrionero, 2011)

Tabla C.7. LUTs para multiplicadores de 8 bits

Multiplicador	*Área (LUTs)*	*% de ocupación en FPGA*
Mastrovito	62	16 %
MSB	27	7 %
LSB	38	9 %
Mastrovito AOP	68	17 %
Dígito Serie D=2	47	12 %
Dígito Serie D=3	50	13 %
Dígito Serie D=4	59	15 %

Fuente: (Sánchez Corrionero, 2011)

Tabla C.8. Retardos de los multiplicadores para GF(2^8)

Multiplicador	*Retado combinacional (ns)*	*Nº de ciclos*	*Retardo total (ns)*	Porcentaje detallado	
				% debido a lógica	*% debido a ruta*
Mastrovito	14,94	1	14,94	53	47
MSB	6,74	8	53,91	32,2	64,8
LSB	6,56	8	52,47	36,2	63,8
Mastrovito AOP	17,05	1	17,05	53,3	46,7
Dígito Serie D=2	7,29	4	29,16	34,6	65,4
Dígito Serie D=3	7,29	3	21,87	34,6	65,4
Dígito Serie D=4	7,38	2	14,76	34,1	65,9

Fuente: (Sánchez Corrionero, 2011)

ANEXO D RECURSOS DEL RS(*N*,*K*)

RS(223)	Usado
Slice	258
LUT	307
FF	258

RS(239)	Usado
Slice	128
LUT	162
FF	128

Tabla D.1 (a) Reporte de Síntesis de RS(*n*,*k*)de IPCores'11

	DVB 1	DVB 2	ATSC	G.709	ETSI-BRAN	CCSDS	ITU J.83 Annex B
Symbol Width	8	8	8	8	8	8	7
Generator Start	0	0	0	0	0	112	1
h	1	1	1	1	1	11	1
k	188	188	187	239	239	223	122
n	204	204	207	255	255	255	127[1]
Field Polynomial	285	285	285	285	285	391	137
Number of Channels	1	16	1	1	1	1	1
Variable Block Length	No	No	No	No	Yes	No	No
Xilinx Part	XC6VLX130T	XC6VLX130T	XC6VLX130T	XC6VLX130T	XC6VLX130T	XC6VLX130T	XC6VLX130T
LUT/FF Pairs[2]	197	338	243	195	199	337	97
LUTs[3]	190	337	237	192	194	329	87
FFs	186	326	218	186	186	316	109
Block RAMs (36k)	0	0	0	0	0	0	0
Block RAMs (18k)	0	0	0	0	0	1	0
Latency	3	18	3	3	3	5	3
Max Clock Freq[2][4]	436/583	448/562	450/589	441/596	442/580	245/316	450/600

Fuente:(Xilinx, 2011)
http://www.xilinx.com/support/documentation/ip_documentation/rs_encoder_ds251.pdf

Tabla D.1 (b) Reporte de Síntesis de RS(*n*,*k*)de IPCores'12

	DVB 1	DVB 2	ATSC	G.709	ETSI-BRAN	ITU J.83 Annex B	CCSDS	Var Check Symbs
Symbol Width	8	8	8	8	8	8	7	8
Generator Start	0	0	0	0	0	112	1	0
h	1	1	1	1	1	11	1	1
k	188	188	187	239	239	223	122	239
n	204	204	207	255	255	255	127[1]	255
Field Polynomial	285	285	285	285	285	391	137	285
Number of Channels	1	16	1	1	1	1	1	1
Variable Block Length	No	No	No	No	Yes	No	No	Yes
LUT/FF Pairs[2]	240	390	288	240	290	132	381	848
LUTs[3]	226	382	272	225	271	120	362	824
FFs	197	339	229	197	224	117	327	362
Block RAMs (36k)	0	0	0	0	0	0	0	0
Block RAMs (18k)	0	0	0	0	0	0	1	0
Max Clock Freq[2][4]	405/599	360/517	402/503	388/598	335/441	447/612	230/320	266/402

Fuente:(Xilinx, 2012)
http://www.xilinx.com/support/documentation/ip_documentation/rs_encoder/v8_0/pg025_rs_encoder.pdf

Tabla D.2 Reporte de Síntesis de RS(*n*,*k*)de IPCores de Altera P4SGX230KF40C2

Components	Combinational ALUTs	Memory ALUTs	Logic Registers	Memory Block (M9K/M144K)	f_{MAX} (MHz)
RS data generator	659	0	558	0/0	>160
2.5G RS II encoder	374	40	186	6/0	>160
RS error injection	143	0	152	0/0	>160
2.5G RS II decoder	3,051	96	1,978	40/0	>160
RS data checker	1,684	0	1,742	0/0	>160
Summary	6,148	136	4,657	46/0	>160

Fuente: (Altera, 2011) Disponible en: http://www.altera.com/literature/an/an642.pdf

Tabla D.3 Reporte de Síntesis de RS(*n*,*k*)de IPCores de Lattice

LatticeSC/M[1]						
IPexpress User-Configurable Mode	Slices	LUTs	Registers	sysMEM EBRs	I/Os	f_{MAX} (MHz)
OC-192	129	251	201	-	24	400
CCSDS	203	398	330	-	24	400
DVB	129	254	201	-	24	400
ATSC	150	293	233	-	24	400
IEEE 802.16-2004 WirelessMAN SCa	172	332	246	-	37	400
IEEE 802.16-2004 WirelessMAN SC	1276	2533	506	-	38	202

Fuente: (Lattice, 2005) Disponible en: http://www.latticesemi.com/products/intellectualproperty/ipcores/dynamicblockreedsolomonen.cfm

Tabla D.4. Comparación del codificador RS con (m=8, t=8)

Condition	Area(gate)	Power(mW)	Timing(ns)
Low complexity	38853	27.12	46.84
High speed	99993	92.06	7.25

Fuente: (Shih, 2000)

Tabla D.5. Resultados de la síntesis del codificador RS(255,223)

Slice	FF(Percentage Slice utilization)	LUT (Percentage Slice utilization)	Max frequency (unit:MHz)
246	278(56.5%)	460(93.5%)	395MHz

Fuente: (Jinzhou et al., 2012)

ANEXO E CODIFICADOR RS(7,3) PARALELO VS SECUENCIAL

Tabla E.1. Descripción VHDL del codificador paralelo RS(7,3)

```
entity RSParalelo is
generic(length:integer:=4;      --length corresponde a n-k+1
        width :integer:=3);   -- width corresponde a m longitud del simbolo
    Port ( e1 : in std_logic_vector(2 downto 0);
           e2 : in std_logic_vector(2 downto 0);
           e3 : in std_logic_vector(2 downto 0);
           s1 : out std_logic_vector(2 downto 0);
           s2 : out std_logic_vector(2 downto 0);
           s3 : out std_logic_vector(2 downto 0);
           s4 : out std_logic_vector(2 downto 0);
           s5 : out std_logic_vector(2 downto 0);
           s6 : out std_logic_vector(2 downto 0);
           s7 : out std_logic_vector(2 downto 0));
end RSParalelo;
architecture Behavioral of RSParalelo is
type r is array (0 to length-1) of std_logic_vector(width-1 downto 0);
component mult is
port     (D_dato: in std_logic_vector (2 downto 0);
               coef: in std_logic_vector (2 downto 0);
             datox: out std_logic_vector (2 downto 0));
end component;
signal d0,d1,d2,d3:std_logic_vector(2 downto 0);
signal r0,r1,r2,r3: r;
signal coef: std_logic_vector(2 downto 0);
signal m11,m12,m13,m14: std_logic_vector(2 downto 0);
signal m21,m22,m23,m24: std_logic_vector(2 downto 0);
signal m31,m32,m33,m34: std_logic_vector(2 downto 0);
begin
--todos los multiplicadores
UM11: mult  port map  (d1,"100",m11);
UM12: mult  port map  (d1,"111",m12);
UM13: mult  port map  (d1,"111",m13);
UM14: mult  port map  (d1,"101",m14);
UM21: mult  port map  (d2,"100",m21);
UM22: mult  port map  (d2,"111",m22);
UM23: mult  port map  (d2,"111",m23);
UM24: mult  port map  (d2,"101",m24);
UM31: mult  port map  (d3,"100",m31);
UM32: mult  port map  (d3,"111",m32);
UM33: mult  port map  (d3,"111",m33);
UM34: mult  port map  (d3,"101",m34);
--inicialización del LFSR
r0(3)<="000";
r0(2)<="000";
r0(1)<="000";
r0(0)<="000";
--realimentación xor entrada
d1<=e1 xor r0(0);
d2<=e2 xor r1(0);
d3<=e3 xor r2(0);
--LFSR     CONCURRENTE     para
generación   de   símbolos   de
Redundancia
r1(3)<=m14;
r1(2)<=r0(3) xor m13;
r1(1)<=r0(2) xor m12;
r1(0)<=r0(1) xor m11;
r2(3)<=m24;
r2(2)<=r1(3) xor m23;
r2(1)<=r1(2) xor m22;
r2(0)<=r1(1) xor m21;
r3(3)<=m34;
r3(2)<=r2(3) xor m33;
r3(1)<=r2(2) xor m32;
r3(0)<=r2(1) xor m31;
--Salidas del Encoder RS
s1<=e1;
s2<=e2;
s3<=e3;
s4<=r3(0);
s5<=r3(1);
s6<=r3(2);
s7<=r3(3);
end Behavioral;
```

Tabla E.2.Código en Matlab para validación del RS(7,3)

```
N=7; % TAMAÑO DEL MANSAJE (n SIMBOLOS)
K=3; % k SIMBOLOS DE DATOS
n=3; % número de bits por símbolo
B=0;
GENPOLY=RSGENPOLY(N,K,11,B);
%GENPOLY=[ 1     4     7     7     5 ]
msg= gf([1,3,7],m)

code=rsenc(msg,N,K,GENPOLY);
; %%%%%%%%%%%%%%%%%solo observar los símbolos de redundancia
%redundancia_code=[ 0     1     1     5 ]
```

RS7Paralelo Project Status (04/18/2013 - 12:21:37)			
Project File:	RS7Paralelo.ise	**Implementation State:**	Synthesized
Module Name:	RSParalelo	**• Errors:**	
Target Device:	xc5vlx50t-3ff1136	**• Warnings:**	
Product Version:	ISE 11.1	**• Routing Results:**	
Design Goal:	Balanced	**• Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	13	28800	0%
Number of fully used LUT-FF pairs	0	13	0%
Number of bonded IOBs	30	480	6%

Figura E.1 Resultados de síntesis del RS(7,3) Paralelo

Name	Value	Used	Total Available	Utilization (%)
Logic	0.00005 (W)	13	28800	0.0
Signals	0.00033 (W)	21	—	—
IOs	0.04147 (W)	30	540	5.6
Total Quiescent Power	**0.56821 (W)**			
Total Dynamic Power	**0.04185 (W)**			
Total Power	**0.61006 (W)**			
Junction Temp	**52.1 (degrees C)**			

Figura E.2. Consumo de potencia del RS(7,3) Paralelo

RS7Secuencial Project Status (04/18/2013 - 12:28:07)			
Project File:	RS7Secuencial.ise	**Implementation State:**	Synthesized
Module Name:	encoder	**• Errors:**	
Target Device:	xc5vlx50t-3ff1136	**• Warnings:**	
Product Version:	ISE 11.1	**• Routing Results:**	
Design Goal:	Balanced	**• Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	12	28800	0%
Number of Slice LUTs	20	28800	0%
Number of fully used LUT-FF pairs	12	20	60%
Number of bonded IOBs	23	480	4%
Number of BUFG/BUFGCTRLs	1	32	3%

Figura E.3. Reportes del RS (7,3) Secuencial.

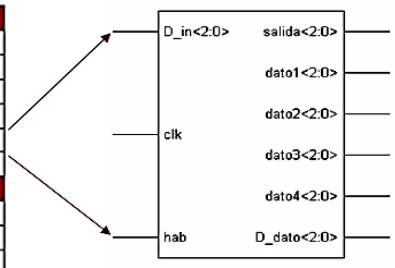
Name	Value	Used	Total Available	Utilization (%)
Clocks	0.00048 (W)	1	—	—
Logic	0.00005 (W)	20	28800	0.1
Signals	0.00013 (W)	27	—	—
IOs	0.03724 (W)	23	540	4.3
Total Quiescent Power	**0.56812 (W)**			
Total Dynamic Power	**0.03789 (W)**			
Total Power	**0.60601 (W)**			
Junction Temp	**52.1 (degrees C)**			

Figura E.4. Consumo de potencia del RS (7,3) Secuencial.

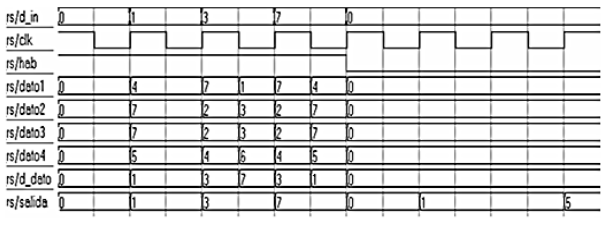
Codificador RS(7,3)

Análisis de recursos		
Recurso	**Cantidad utilizada**	**% de los disponibles**
Slices	13	0%
Registros	12	0%
LUTs	23	0%
IOBs	22	15%
Clk	1	25%
Análisis de retardos		
Rutas	12.235ns	
Tiempo de respuesta de la entrada	7.207ns	
Periodo mínimo	6.752ns	
Retardo máximo	11.780ns	
Frecuencia máxima	148.104MHz	

Tabla 1. Reporte de síntesis del codificador sobre el FPGA



La implementación cuenta con documentación acerca de las especificaciones técnicas del FPGA la simulación, verificación con VHDL Testbench, la Simulación donde se valida el funcionamiento y los código fuente del diseño.



Validación del diseño del Codificador RS(7,3)

```
N=7; % tamaño del mensaje (7 símbolos)
K=3; % numero de símbolos de datos
m=3; % numero de bits por símbolo
B=0;
% GENPOLY = RSGENPOLY(N,K,PRIM_POLY,B)= GF(2^3) array.
Primitive polynomial = D^3+D+1 (11 decimal)
B=0; %corresponde al valor de i
GENPOLY = RSGENPOLY(N,K,11,B) GENPOLY = GF(2^3) array.
Primitive polynomial = D^3+D+1 (11 decimal)
%Array elements =
%   1   4   7   7   5
% G(x) = 4x3+ 7x2+ 7x+ 5
msg= gf([1,3,7],m)
code = rsenc(msg,N,K,GENPOLY)
code=[1,3,7,0,1,1,5]  %palabra de código obtenida
```

Figura E.5. Datos Técnicos del Codificador RS(7,3) secuencial.

ANEXO F ESTRUCTURAS CIRCUITALES FRACTALES

La geometría fractal y la teoría de los sistemas dinámicos están íntimamente ligadas (Mangaña, 2011), en (Rubiano, 2007; Adame 2005) se presenta el estudio de los Sistemas de Funciones Iteradas (SFI) y la obtención del fractal asociado a un SFI. Es importante señalar que estos conceptos suelen emplearse para el modelado de sistemas complejos, pero no es común en modelado de circuitos, siendo en este caso su aplicación propicia por la naturaleza del diseño para la optimización en descripción de hardware. Luego de encontrar un patrón iterativo en las funciones que describen el modelo desarrollado, resultó de interés realizar un estudio acerca de las características identificadas en las estructuras que describe el modelo lógico – matemático, considerando el análisis fractal de las estructuras y los sistemas de funciones iterativos que se presentaron en la descripción espacio – tiempo.

En la figura F.1 se puede observar la relación entre los LFSR del multiplicador GF en una ***estructura anidada*** con relación al LFSR del codificador RS, y éste con el RS-PC donde se reconoce una arquitectura fractal.

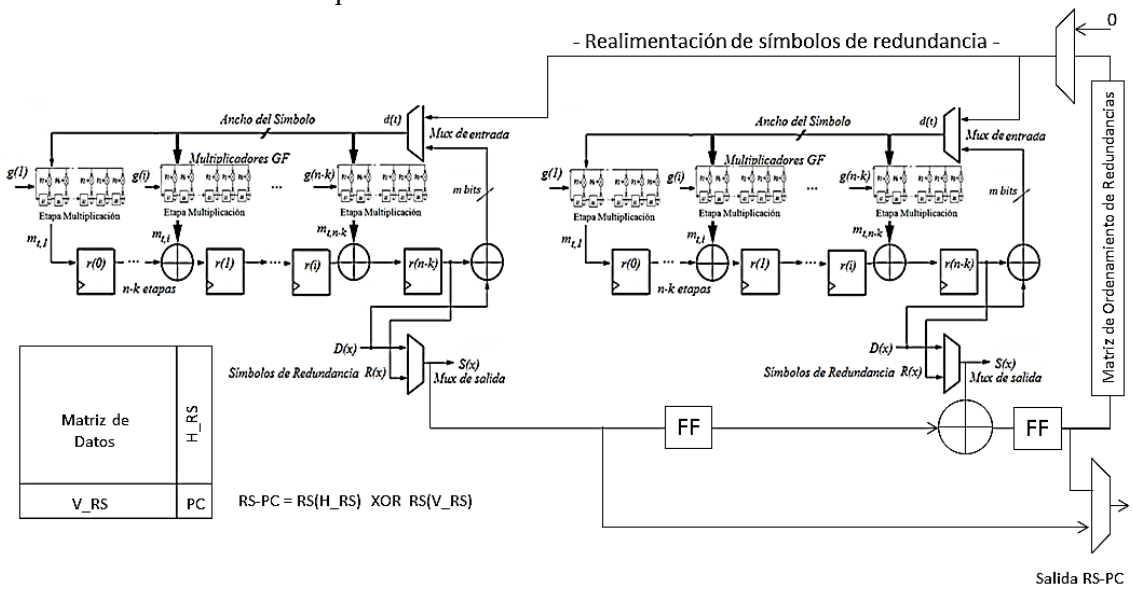


FIGURA F.1. ESTRUCTURA DEL CODIFICADOR CONCATENADO PARALELO REED SOLOMON

Adame, E., (2005), *Sistemas de Funciones Iteradas y Fractales*.

Magaña, R., *Análisis de Elementos Finitos y Remalleo Fractal en Geotecnía*, Ingeniería. Investigación y Tecnología, Vol. 12. N°1, pp. 103-118, 2011, disponible en: http://redalyc.uaemex.mx/redalyc/pdf/404/40419887002.pdf

Rubiano, G., (2007), *Método de Newton, matemática y fractales*, Boletín de matemáticas, Vol.14, N° 1, pp.44-63

De forma alternativa, la descripción del multiplicador (Tabla B.2) se puede particionar la implementación en componentes LFCS, logrando una representación RTL multinivel por elementos circuitales para describir sub-funciones reutilizables. Este concepto de lógica multinivel (J. L. Imaña et al., 2002), permite definir componentes circuitales interconectados, a través de capas, para generar las salidas (figura F.2). Las señales pueden atravesar un número arbitrario de compuertas en el proceso combinacional, pero la reutilización de los elementos circuitales es una característica que permite disminuir la complejidad lógica, donde se presenta una ***red de componentes*** para representar los *ciclos iterados*, de manera que se obtiene a través de un SFI el modelo concurrente.

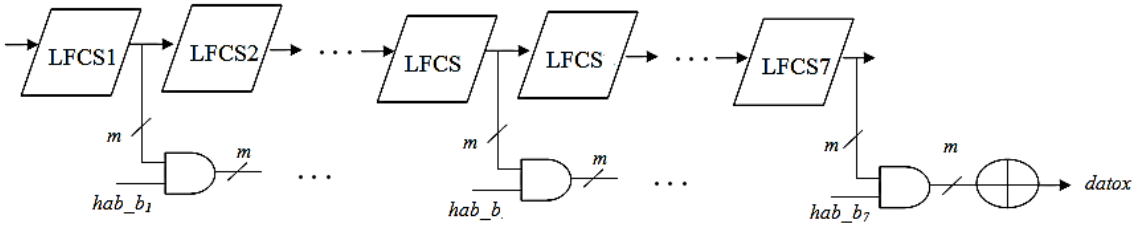


Figura F.2. Lógica Multinivel del Multiplicador GF

Tabla F.1. Declaración de lógica multinivel por componentes circuitales

```
-- entity mult (Las declaraciones idénticas al circuito multiplicador)
begin
b<= D_dato; -- dato de entrada para la multiplicación
a1<= coef; --a1<= coef;
p<="100011101"; -- Primitive polynomial = D^8+D^4+D^3+D^2+1 (285)
u1: LFCS  port map  (a1,a2);
u2: LFCS  port map  (a2,a3);   u4: LFCS  port map  (a4,a5);  u6: LFCS  port map  (a6,a7);
u3: LFCS  port map  (a3,a4);   u5: LFCS  port map  (a5,a6);  u7: LFCS  port map  (a7,a8);
--Lógica combinacional AND del multiplicador ci = ai * bi
datox<=c1 xor c2 xor c3 xor c4 xor c5 xor c6 xor c7 xor c8;
end Behavioral;
```

```
entity LFCS is
port     (a: in std_logic_vector (7 downto 0);
          r: out std_logic_vector (7 downto 0));
end LFCS;
architecture Behavioral of LFCS is
signal p: std_logic_vector (8 downto 0);
begin
a<= coef;
p<="100011101"; -- Primitive polynomial = D^8+D^4+D^3+D^2+1 (285)
u: r  <=a(6 downto 4)&(a(3)xor a(7))&(a(2)xor a(7))&(a(1)xor a(7))&a(0)& a(7);
end Behavioral;
```

A partir del estudio de las características de la operación producto en campos finitos de *Galois* GF(2^m), se tiene que la operación corresponde al producto de polinomios entre dos elementos del campo, *mod* el polinomio irreducible del campo (Saqib Nazar, 2004), operación que genera elementos del campo de longitud fija, con características iguales a los elementos que se operan sobre éste. La implementación de la operación *mod* puede ser definida como un sistema con funciones iteradas SFI con realimentación. De manera similar, debido a la naturaleza del código RS "*cíclico de bloques*", se analizó la realimentación en *ciclos iterados* para la paralelización del componente circuital LFSR (elemento de estructura común), definiendo el modelo como un SFI, a la vez de aplicar el procesamiento por bloques para un tratamiento concurrente del conjunto de datos.

Igualmente, se puede identificar la correspondencia entre la ecuación descriptiva del arreglo de términos r_t y el arreglo de términos a_t, como se observa en la ecuación F.1 y F2.

$$a_t = \&_{i=0}^{m-1}\, a_{t-1}(i-1)\ xor\ (a_{t-1}(m-1) and\ p(i)) \qquad (F.1)$$

$$r_t = \&_{i=0}^{n-k}\, r_{t-1}\,(i-1) \oplus [(d(i) \oplus r_{t-1}\,(n-k-1)) \otimes g(i)] \qquad (F.2)$$

La expresión del producto en campo finito GF se puede sustituir como se muestra en la ecuación F.3.

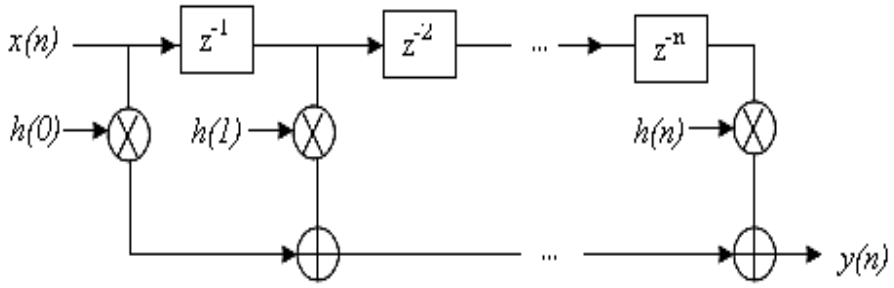
$$r_t = \&_{i=0}^{n-k}\, r_{t-1}\,(i-1) \oplus\ (\oplus_{t=1}^{m}\ a_t\ and_m\ b_t) \qquad (F.3)$$

Luego, sustituyendo a_t y b_t, *en función de la realimentación* d_{t-1} *y los coeficientes g(i)* desarrollando la ecuación, se obtiene la ecuación F.4. Se puede notar que de la generalización se debe operar el término $r_{t-1}\,(n-k)$, pero de acuerdo a la estructura este término es nulo.

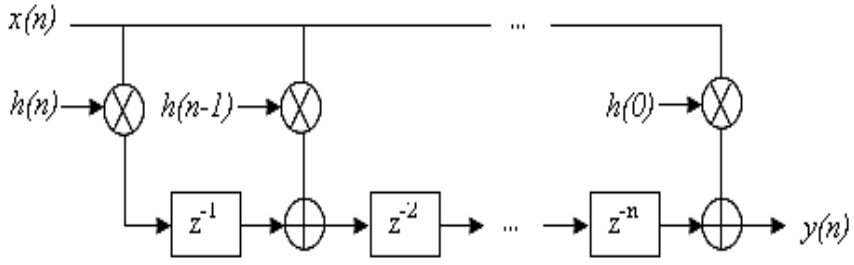
$$\begin{aligned} r_t = {} & r_{t-1}\,(n-k-1) \oplus [\ \oplus_{t=1}^{m}\ [\&_{i=0}^{m-1}\, d_{t-1}(i-1)\ xor\ (d_{t-1}(m-1) and\ p(i))]\ and_m\ g(n-k)\ ]\ \& \\ & r_{t-1}\,(i-1) \oplus [\ \oplus_{t=1}^{m}\ [\&_{i=0}^{m-1}\, d_{t-1}(i-1)\ xor\ (d_{t-1}(m-1) and\ p(i))]\ and_m\ g(n-k-1)\ ]\ \& \ \ldots \\ & r_{t-1}\,(0) \oplus [\ \oplus_{t=1}^{m}\ [\&_{i=0}^{m-1}\, d_{t-1}(i-1)\ xor\ (d_{t-1}(m-1) and\ p(i))]\ and_m\ g(1)\ ] \end{aligned} \qquad (F.4)$$

ANEXO G GENERALIZACIÓN DE ESTRUCTURAS LFSR

En la figura G.1 se presentan las representaciones *Galois* y *Fibonacci* de los circuitos generadores de secuencia (Goresky,2002), los cuales han sido estudiados para su descripción en VHDL (Tabla G.1) y lograr un modelo generalizado en la figura G.2 como un D-LFSR circuito de realimentación lineal dinámico, con coeficientes variables, lo que permite observar la diversas configuraciones con realimentación selectiva, para los casos particulares de aplicaciones de procesamiento de datos (Tabla G.2).



(a)



(b)

Figura G.1. Esquema de la operación de convolución representación
(a) LFSR Fibonacci (b) LFSR Galois

Goresky, M., Member, A., & Klapper, A. M. (2002). Fibonacci and Galois Representations of Feedback-With-Carry Shift Registers. *IEEE Transactions on Information Theory*, *48*(11), 2826–2836. http://www.math.ias.edu/~goresky/pdf/Fib.jour.pdf

Tabla G.1. Descripción VHDL de la operación de convolución

```
begin
process (clk)
variable x_v:x:=(others=>"00");
begin
-- modelo 1: y(n) = ∑ x(n -k) .h(k)
if hab ='1' then
    yn<= "0000";
else
-- Desplazamiento de la entrada x(n - k)
    if (clk'event and clk='1')then
         x_v(n):= x_v(n-1);
        …
         x_v(2):= x_v(1);
         x_v(1):= x_v(0);
         x_v(0):=xn;
--Producto de la entrada retardada con h(k)
         v0<= x_v(0) * h(0); -- ejemplo h(0)="01"
         v1<= x_v(1) * h(1); -- ejemplo h(1)="10"
        …
         vn<= x_v(n) * h(n); -- ejemplo h(3)="01"
    end if;
end if;
--Sumatoria de los productos parciales
yn<= v0 + v1 + … + vn;
-- en el ejemplo yn= v0 + v1 + v2- v3
end process;
end Behavioral;
```

```
variable y_v:y:=(others=>"00");
variable r_v:y:=(others=>"00");
begin
-- modelo 2: y(n) = ∑ x(k) .h(n-k)
if hab ='1' then
    yn<= "0000";
else
-- Producto de x(k) * h(i)
mk0<= x(k) * h(0); -- ejemplo
h(0)="01"
mk1<= x(k) * h(1); -- ejemplo
h(0)="01"
…
mkn<= x(k) * h(n); -- ejemplo
h(3)="01"
--  Desplazamiento  de  los  registros
r(n-k)
    if (clk'event and clk='1')then
-- ejemplo h(2)="01"r_v(n):= r_v(n-1);
        …
         r_v(1):= r_v(n-1)+mk1;
         r_v(0):= r_v(n-1)+mk0;
         y_v(k):= r_v(0);
    end if;
end if;
--Concatenación de yk(0)
yn<=   y_v7(0)& y_v6(0)&…& y_v0(0);
end process;
end Behavioral;
```

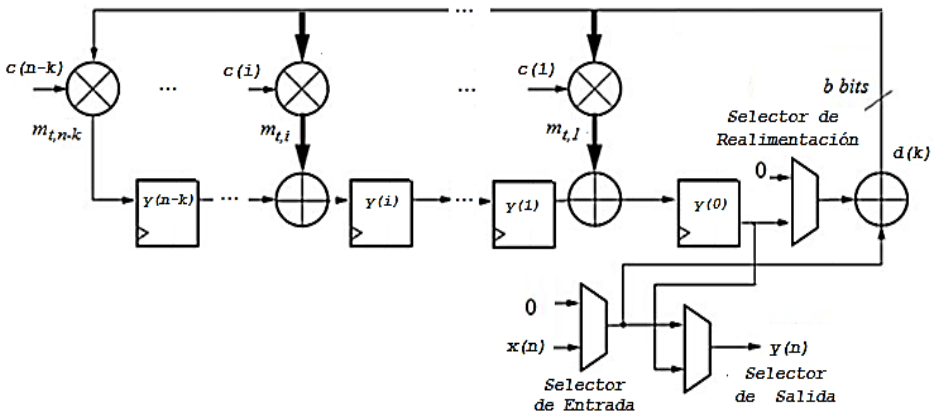
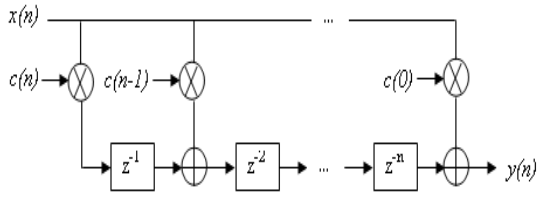
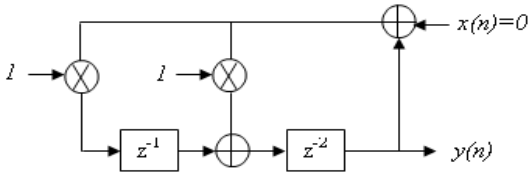
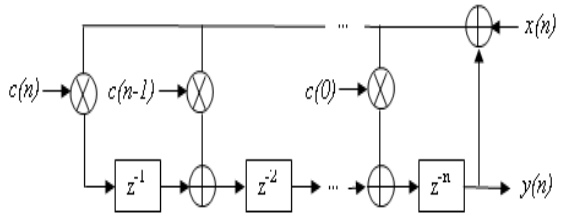
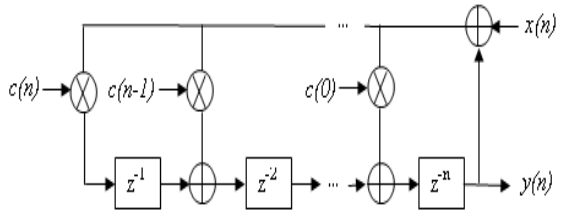


Figura G.2. Esquema generalizado para operaciones de generación de secuencias

Tabla G.2. Circuitos estudiados para su descripción en VHDL

Circuito basado en generador de secuencia	*Descripción VHDL del circuito concurrente*
Operación de Convolución: *y(n)= ∑ h(n-k).x(k), con k el elemento de la muestra actual* 	*--y(i)=y(i-1)+h(i).x(k)* *--con c(i)=h(k) y fb=0* ... UM31: mult port map (x3, h1,m31); UM32: mult port map (x3, h2,m32); UM33: mult port map (x3, h3,m33); UM34: mult port map (x3, h4,m34); y1(3)<=m14; y1(2)<=y0(3) xor m13; y1(1)<=y0(2) xor m12; y1(0)<=y0(1) xor m11; ... yn<= yk(i) & … & yk(1) & yk(0);
Generador_Fibonacci: y(n)=y(n-1)+y(n-2) *Condición inicial de los estados de memoria en 1* 	*-- f(i)=fk(i)+fk-1(i-1) con ci=1, fb=1* f1(0)<= 1; f1(1)<= 1; *-- dado que: y(n)<= fk(0) xor fk(1);* f2(n)<= y0(0) xor f1(1); f3(n)<= y1(0) xor f2(1); ... fn(n)<= fk-1(0) xor fk(1); fn<= fk(0) & … & f1(0) & f0(0);
Codificador_RS(n,k): *C(x)=G(x)*D(x), d(x)=x(n)+y(n) realimentación* 	*--r(i)=r(i-1)+g(i).(x(k)+r(n-1))* - *--con c(i)=g(i)* UM11: mult port map (d1,"100",m11); ... UM31: mult port map (d3,"100",m31); UM32: mult port map (d3,"111",m32); UM33: mult port map (d3,"111",m33); UM34: mult port map (d3,"101",m34); r1(3)<=m14; r1(2)<=r0(3) xor m13; r1(1)<=r0(2) xor m12; r1(0)<=r0(1) xor m11; ... cn<= rk(i) & … & rk(1) & rk(0);
ModGF_A(x): R(x)=A(x) mod P(x), A(x)=x(n) 	*--a(i)=a(i-1)+p(i).(x(k)+a(n-1)* *--con ci=p(i)* p<="100011101"; -- p(x)=285 u1:a2<=a1(6downto4)&(a1(3)xor a1(7))&(a1(2)xor a1(7))&(a1(1)xor a1(7))&a1(0)& a1(7); … UMki: mult port map (dk,pi,mki); … rk(i)<=rk-1(i+1) xor mk(i+1); ak<= ak(i) & … & ak(1) & ak(0);

ANEXO H INTERPRETACIÓN MATRICIAL DEL MODELO PARA GENERACIÓN DEL CODIFICADOR EN VHDL

En el desarrollo del modelo se consideraron métodos asociados de Análisis Fractal de las estructuras, Método de los Sistemas de Funciones Iterativos, Método de Estimación por descripción espacio – tiempo. Es importante señalar que estos métodos suelen emplearse para el modelado de sistemas complejos, pero no es común en modelado de circuitos, siendo en este caso su aplicación propicia por la naturaleza del diseño para descripción de hardware con optimización. Una vez hallado el modelo para VHDL, resultó interesante plantear una expresión bajo la interpretación holográfica –matricial del sistema de codificador, a través de operadores matemáticos.

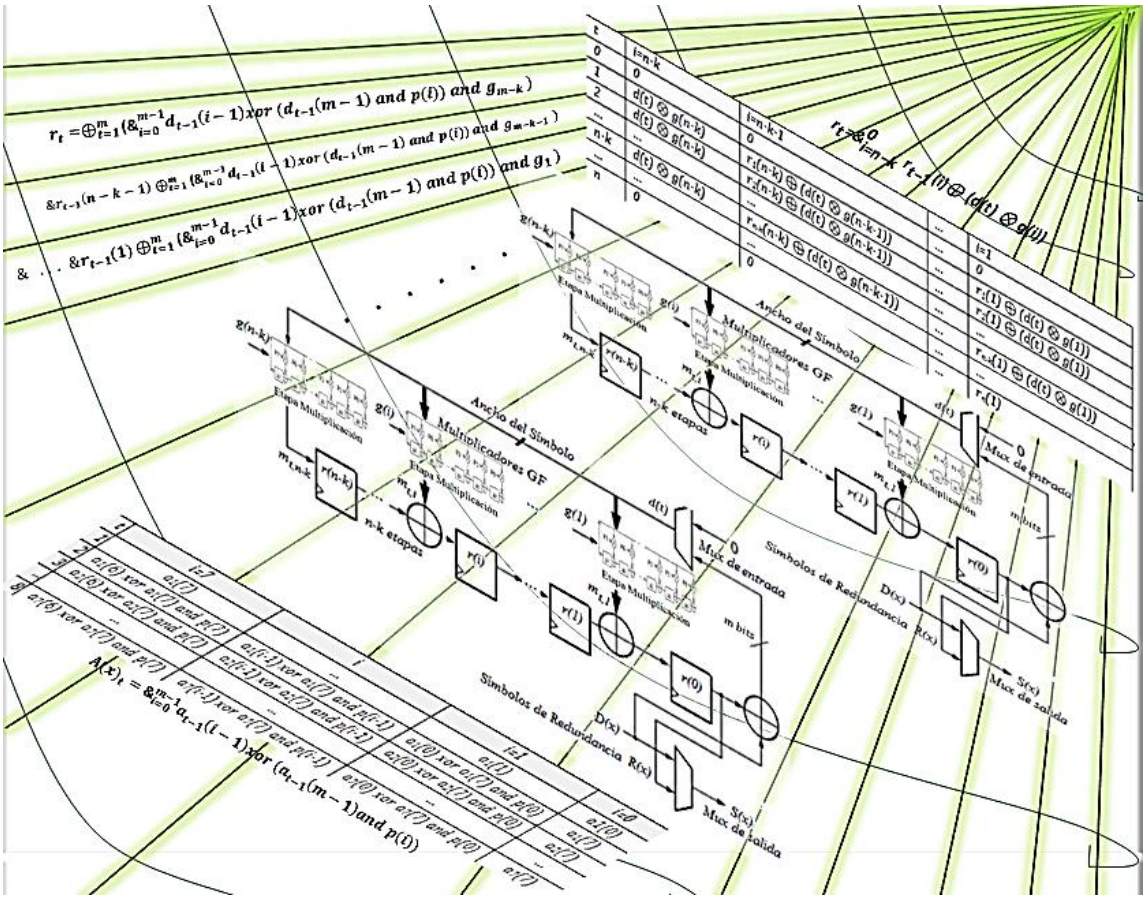


Figura H.1. Modelo Matricial del Generador de Código Reed Solomon (*n*,*k*)

Partiendo de la definición de la palabra de código Reed Solomon, donde cada palabra debe ser múltiplo del polinomio generador *G(x)*, expresada ésta en su forma sistemática corresponde al bloque de información *D(x)* adicionando los símbolos de redundancia calculados sobre el bloque de información. Este cálculo es el bloque resultante como residuo de la operación de división entre el polinomio generador G(x), dada como $R_{g(x)}[.]$ aplicada sobre los símbolos de datos (Moon, 2005).

$$C(x) = x^{n-k}\, D(x) + R_{g(x)}\,[\, D(x)\, x^{n-k}\,] \qquad (\text{H.1})$$

La expresión matemática corresponde a ensamblar dos polinomios con desplazamiento, definido como: *c = (D << (n-k)) + (D << (n-k)) % g*; donde se desplaza el polinomio de datos de información *n-k* posiciones a la izquierda, y los *n-k* símbolos menos significativos son completados con el residuo de la operación *mod* del polinomio *G(x)*.De manera tal que la expresión polinomial de la palabra de código queda definida como la suma de los polinomios mencionados.

$$C(x) = x^{n-k}\, D(x) + x^{n-k}\, D(x) \bmod g(x) \qquad (\text{H.2})$$

Encontrando así la expresión matemática del generador de símbolos de redundancia

$$R(x) = x^{n-k}\, D(x) \bmod g(x) \qquad (\text{H.3})$$

Similar a la expresión de los polinomios generados como residuos parciales en la operación de multiplicación en campos finitos, que se expresó en función de los *i* desplazamientos x^i, que se realizan en el proceso de multiplicación para la operación con el coeficiente B_i correspondiente.

$$a(x) = x^i\, A(x) \bmod p(x) \qquad (\text{H.4})$$

Lo que evidencia un tratamiento matemático similar, entre ambas estructuras.

La expresión matricial para el producto de símbolos en campos finitos de Galois, puede ser expresada como:

$$A(x) * B(x) = \begin{bmatrix} a_{1,0} & \cdots & a_{1,m-1} \\ \vdots & \ddots & \vdots \\ a_{m,0} & \cdots & a_{m,m-1} \end{bmatrix} . \begin{bmatrix} b_{1,0} & \cdots & b_{1,m,-1} \\ \vdots & \ddots & \vdots \\ b_{m,0} & \cdots & b_{m,m-1} \end{bmatrix} = [c_0 \quad \cdots \quad c_{m-1}], \qquad (\text{H.5})$$

$$con\ a_{t,i} = at-1(i-1) xor \left(at-1(m-1) and\ p(i)\right) y\ b_{t,i} = b(i)$$

Seguidamente se analiza la implementación de la ecuación H.2, la cual puede ser interpretada como la concatenación de los *k* símbolos de datos, con los *n-k* símbolos de redundancia. Estos últimos corresponden a la salida de un generador de secuencia LFSR en su representación de *Galois*.

$$c(x) = \&_{i=0}^{k} D(i) \,\&\, R(x), \text{ siendo } R(x) = \&_{i=k}^{n} R_i(0) \qquad (H.6)$$

Partiendo de la ecuación de convolución (anexo G), basado en el LFSR de la representación de *Galois*, se sustituye la salida de datos *y(n)* por el vector de símbolos de redundancia *R(x)*, la entrada de datos *x(k)* por los datos a codificar (compuesto con la realimentación) *d(t)*, y los coeficientes de la función de transferencia *h(n-k)* por los coeficientes del polinomio generador del código *g(n-k)*, obteniendo así una expresión de la forma:

$$R(x) = \sum_{k=0}^{n} d(t).g(n-k), \text{ para los } n\text{-}k \text{ símbolos generados} \qquad (H.7)$$

Para dicha expresión se ha empleado un término *d(t)* que corresponde a un arreglo compuesto entre *d(k)* y la realimentación del residuo en la posición menos significativa del polinomio $r_{k-1}(0)$, esto con el propósito de conservar la similitud de la expresión matemática (sin realimentación), al sustituir en función de la entrada del codificador *d(k)*, correspondiente a una muestra *k* del vector de datos, se obtiene:

$$R(x) = \sum_{k=0}^{n} (d(k) \oplus r_{k-1}(n-k)).g(n-k) \qquad (H.8)$$

Desarrollando el producto en campos finitos de *Galois*, se tiene:

$$R(x) = \sum_{k=0}^{n} [(d(k) \oplus r_{k-1}(n-k)) \bmod p(x)]. [\, g(n-k)] \qquad (H.9)$$

La ecuación define el producto de *a(x)* correspondiente al primer operando *mod* el polinomio generador del campo *p(x)* y el vector b(x) correspondiente al segundo operando. Siendo: $a(x) = d(i) \oplus r_{i-1}(0)$ Se puede expresar la operación *mod* como la convolución realimentada.

$$a(x) \bmod p(x) = \sum_{k=0}^{m} a(k).p(n-k) \qquad (H.10)$$

Donde se puede expresar en correspondencia con:

$$a_m(x) = \sum_{i=0}^{m} (a(i) \oplus a(m)).p(n-i) \qquad (H.11)$$

De esta manera, el tratamiento de la operación *mod* será dada como la convolución con *p(x)* del elemento *a(m)* realimentado. Luego se sustituye en la ecuación del producto en campos finitos GF, en la ecuación del codificador, de forma iterativa, con lo que se genera la expresión general:

$$R(x) = \sum_{i=0}^{n} (\sum_{x=0}^{m} (d_k(x) \text{ xor } (d_k(m-1)) \text{ and } p(x))] \text{ and}_m g(n-i)) \qquad (H.12)$$

Para finalmente, expresar la ecuación matemática del modelo matricial del codificador RS, como un arreglo de símbolos de redundancia.

$$R(x) = \begin{bmatrix} r_{1,0} & \cdots & r_{n-k} \\ \vdots & \ddots & \vdots \\ r_{n,0} & \cdots & r_{n,n-k} \end{bmatrix}; \qquad \text{(H.13)}$$

con $r_{t,i} = r_{t-1}\,(i\text{-}1) \oplus [(d(i) \oplus r_{t-1}\,(n\text{-}k\text{-}1)) \otimes g(i)]$

En función del análisis y dada la característica de arreglos de términos, en función de los parámetros de posición *i*, en la estructura circuital LFSR, y la posición *t*, en el vector que es procesado por la estructura, se obtiene un esquema para la generación del código de descripción del codificador RS(*n*,*k*) en VHDL, el cual puede ser aplicado para construir el código de descripción de hardware en VHDL, basado en las ecuaciones del modelo desarrollado, según el procedimiento:

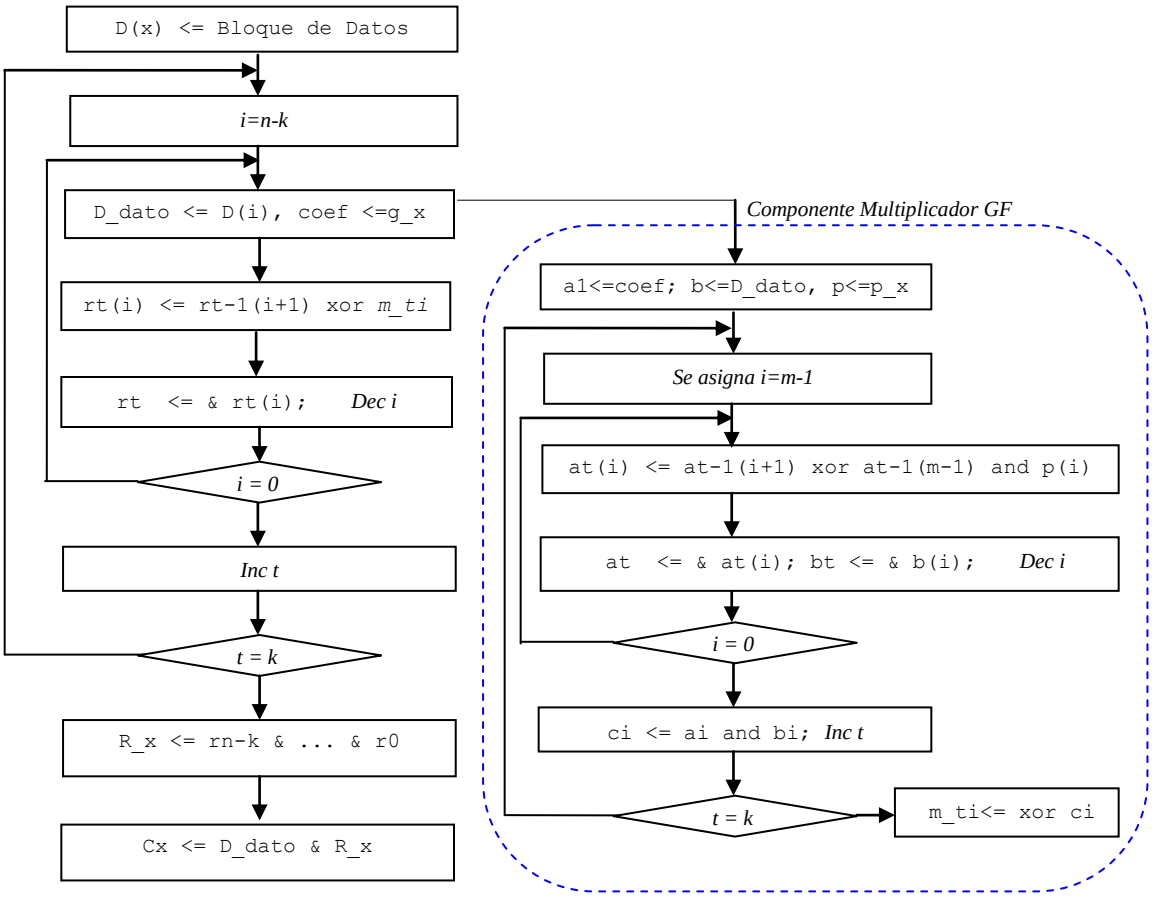


Diagrama H.1. Esquema del procedimiento para Generación de Código VHDL de un Codificador Reed Solomon (*n*,*k*)