Rev. Téc. Ing. Univ. Zulia. Vol. 31, Nº 3, 294 - 301, 2008

Modular Programming of functions for turbo product codes on FPGA

Cecilia Esperanza Sandoval Ruiz

Dirección de Postgrado, Facultad de Ingeniería, Universidad de Carabobo.
Naguanagua sector Bárbula. Telf. 0251-8672829/4268 et. 102. Fax 0241-8671655.
csandoval1@uc.edu.ve

Abstract

This paper present a review of the concepts turbo product codes, with the aim of designing an alternative based on the high degree of parallelism available in the reconfigurable hardware devices such as the FPGA, using these devices arrangements composed by field programmable; for designing functional modules such encoders. The selected modules have been described in language descriptor hardware, synthesized and simulated, using the design tool Xilinx ISE 9.2i, which was conducted with the programming of components, and sets out the findings on the basis of the alternatives raised.

Key word: VHDL, re-configurable hardware, coding, digital communications.

Programación modular de funciones para codificación turbo producto sobre FPGA

Resumen

En este artículo se realiza una breve revisión de los conceptos de códigos turbo producto, con el propósito de diseñar una alternativa basada en el alto grado de paralelismo disponible en los dispositivos de hardware reconfigurables, como es el caso de los FPGA, usando estos dispositivos de arreglos de compuertas programables por campo; para el diseño de módulos funcionales de dichos codificadores. Los módulos seleccionados han sido descriptos en lenguaje descriptor de hardware, sintetizados y simulados; usando la herramienta de diseño Xilinx ISE 9.2i, con la cual se realizó la programación de los componentes, y se establecen las conclusiones en función de las alternativas planteadas.

Palabras clave: VHDL, hardware re-configurable, codificadores, comunicación digital.

Introducción

Dadas las características de paralelismo de los códigos turbo producto, los dispositivos FPGAs (Field Programmable Gate Arrays) representan una alternativa para la implementación de éstos sobre hardware reconfigurable. Los FPGAs ofrecen un balance adecuado entre el espacio requerido por los circuitos y la rapidez con que se pueden realizar las operaciones de codificación y decodificación. La computación mediante el uso de FPGAs y/o hardware reprogramable a bajo nivel se suele denominar computación reconfigurable. Además de las ventajas inherentes cuando se pretende la reutilización del equipo, las FPGAs, bajo ciertas condiciones, pueden completar más tareas por unidad de tiempo que los procesadores debido básicamente a dos motivos; con menor exceso de instrucciones, pueden empaquetar mayor número de tareas dentro de la misma área de silicio y pueden controlar las operaciones a nivel de bit, mientras que el procesador sólo puede hacerlo a nivel de palabra.

Por ello, la tecnología de las FPGA está experimentando un amplio avance no sólo en cuanto al desarrollo de dispositivos cada vez más efi-

cientes sino también en cuanto a su capacidad de relacionarse entre ellos y con otros elementos [1].

La implementación de Códigos Turbo Producto (Turbo Product Codes ≡ TPC) pueden presentar la entidad con pines de control para la configuración del ancho de la palabra y del número de iteraciones en el tiempo, tal es el caso observado en el módulo de propiedad intelectual de Xilinx [2]. En este trabajo se ha tomado una alternativa basada en programación VHDL de algoritmos de codificación [3], donde se realiza un tratamiento de los módulos y algoritmos que comprende el código para soportar futuras aplicaciones, en este caso el propósito es la descripción en VHDL de los módulos principales que constituyen los codificadores TPC, para su implementación en hardware.

La selección de los TPC está basada en la eficiencia del código. Uno de los más comunes modelos de canal para la evaluación de tecnología para la corrección de errores es el de un canal con ruido Gaussiano, blanco y aditivo (AWGN, Additive White Gussian Noise), donde el TPC ofrece aproximadamente 3 dB de ganancia sobre Reed Solomon-Viterbi (para las condiciones del modelo con Rate 2/3, QPSK, AWGN), como lo enuncia Banister [4].

Estudio de los Códigos Turbo Productos

El procedimiento de codificación TPC dará como resultado una matriz general de dimensiones *(n+1,k)* por *(n+1,k); siendo n+1* bits la extensión de la palabra de código y *k* bits la de datos. En el caso, de un código turbo producto que concatena dos codificadores Hamming (*7,4*), se obtiene una matriz [64×64]. Los bits de datos están representados por D_{ij}, los bits de chequeo Hamming están representados por H_{ij}, y los bits de paridad por P_{ij}, de manera que se obtiene una matriz con bits de chequeo del arreglo de datos horizontal y vertical.

D_{11}	D_{21}	D_{31}	D_{41}	H_{51}	H_{61}	H_{71}	P_{81}
D_{12}	D_{22}	D_{32}	D_{42}	H_{52}	H_{62}	H_{72}	P_{82}
D_{13}	D_{23}	D_{33}	D_{43}	H_{53}	H_{63}	H_{73}	P_{83}
D_{14}	D_{24}	D_{34}	D_{44}	H_{54}	H_{64}	H_{74}	P_{84}
H_{15}	H_{25}	H_{35}	H_{45}	H_{55}	H_{65}	H_{75}	H_{85}
H_{16}	H_{26}	H_{36}	H_{46}	H_{56}	H_{66}	H_{76}	H_{86}
H_{17}	H_{27}	H_{37}	H_{47}	H_{57}	H_{67}	H_{77}	H_{87}
P_{18}	P_{28}	P_{38}	P_{48}	P_{58}	P_{68}	P_{78}	P_{88}

Figura 1. Estructura matricial de los datos. Fuente: IP Xilinx, 2002 [2].

En la Figura 1, se muestra una estructura matricial de datos para un TPC (8,4) x (8,4).

Codificador Turbo Producto (TPC)

El codificador TPC consta de un arreglo de memoria que alimentará a un par de codificadores Hamming, uno recibe los datos organizados en filas y el otro procesará los datos organizados en columnas, estos datos serán organizados en un arreglo matricial de memoria y se obtiene la paridad de los elementos de paridad del mismo modo, aplicando xor a los bits de paridad resultantes por filas y columnas. La Figura 2, presenta el diagrama de bloques del codificador TPC.

Decodificador TPC (Turbo Product Code)

El decodificador Turbo Producto (*TPC Turbo Producto Code*) realiza un procesamiento iterati-

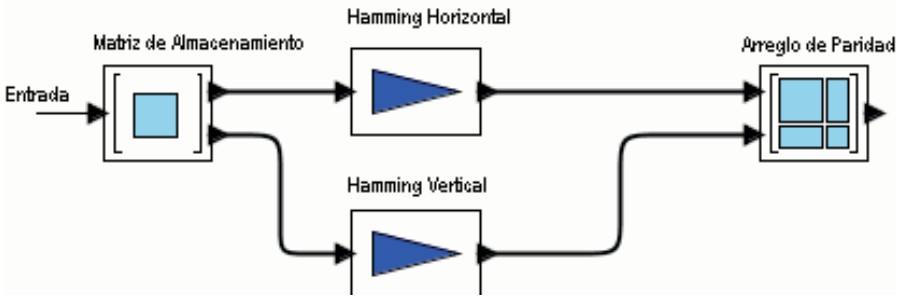

Figura 2. Diagrama de bloques del Codificador Turbo Producto.

vo de las señales de entrada del bloque de código con la ayuda de uno o cuatro decodificadores suave (*SISO soft-in/soft-out*). Los 4 bits de datos llegan al decodificador suave TPC, éstos se almacenan en una matriz de entrada. Una vez que todo un bloque de código se almacena, los decodificadores SISO empiezan a procesar los datos de la matriz, realizando el almacenamiento de los productos en un arreglo. La salida de la última iteración se combina con la entrada y se realimenta. El nivel superior del diagrama de bloques del decodificador TPC se muestra en la Figura 3.

La decodificación SISO se realiza sobre un eje del bloque de código en una instante de tiempo. La salida de la decodificación horizontal, junto con la matriz original de entrada horizontal, se incorpora a la decodificación SISO vertical. La salida de la decodificación SISO, junto con la serie original de entrada vertical, es realimentado para otra iteración horizontal, y así sucesivamente [5]. La realización de una decodificación de pasar a lo largo de una dimensión (*X* o *Y*) de la matriz se conoce como un eje de iteración. La finalización de la decodificación se denomina una iteración completa, el proceso se ilustra en la Figura 4.

Esta decodificación iterativa puede ser implementada a través de varios decodificadores SISO paralelos, tomando de la matriz de entrada el vector correspondiente a la muestra, siendo n muestras igual al número de iteraciones y por ende igual al número de SISOs paralelas. Cada una de las muestras es cuantizada con 4 bits, utilizando una tabla de acuerdo a la modulación empleada [2], y cada una de las combinaciones tendrá asociada una probabilidad para la decodificación.

Los algoritmos para decodificación SISO, basados en el algoritmo MAP que es una versión del algoritmo BCJR propuesto por Bahl, Cocke, Jelinek y Raviv [6], para estimar las probabilidades a posteriori de los estados y transiciones de una fuente de Markov (como por ejemplo un codificador convolucional) a partir de las salidas observadas provenientes de un canal discreto sin memoria, el cual tiene aplicación a los códigos de bloques y es basados en ellos que se realiza la decodificación SISO.

La literatura propone en la decodificación SISO (Soft Input Soft Output) algoritmos tales como Algoritmo de Viterbi salida suave (SOVA) y modificaciones del algoritmo (Máximo A Posteriori) o BCJR, donde se presentan versiones como Log MAP Max [7-11]. El estudio de estos algoritmos permite identificar módulos funcionales

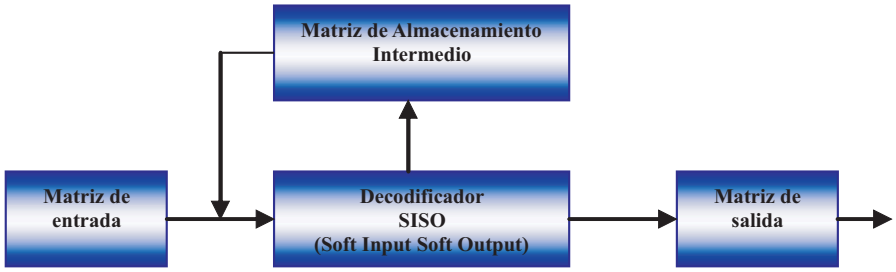

Figura 3. Diagrama de Bloques de un Decodificador TPC. Fuente: IP Xilinx, 2002 [2].

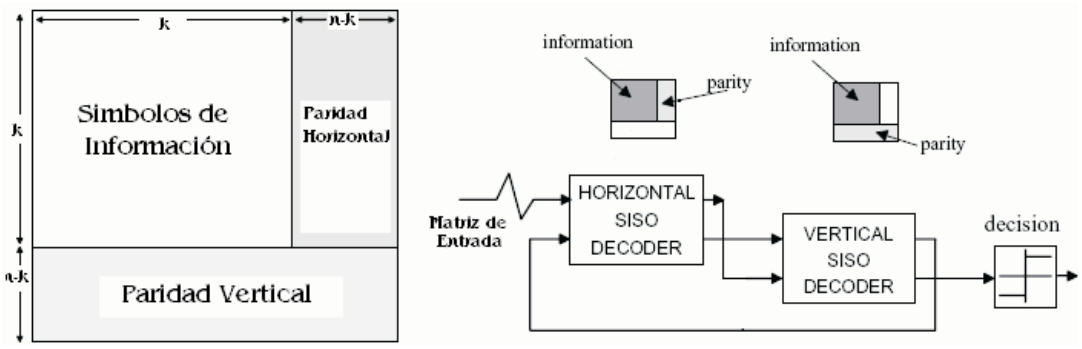

Figura 4. Estructura del de código TPC y esquema del decodificador Iterativo.

simplificados para la implementación en hardware de las funciones requeridas para estimar la salida decodificada.

Para aplicar la decodificación se requiere calcular los LLR's a posteriori de la secuencia de bits uk, cada uno de los cuales viene dado por la ecuación 1.

$$L(\hat{u}_k) = \ln \frac{P(u_k = +1, r)}{P(u_k = -1, r)} \Rightarrow$$
$$L(\hat{u}_k) = \ln \frac{\sum_{u_k=+1} P(S_{k-1}, S_k, r)}{\sum_{u_k=-1} P(S_{k-1}, S_k, r)} \quad (1)$$

Empleando las simplificaciones [12], manejadas a través del logaritmo natural, con lo que se pasan los productos a suma, y las sumas a selección del máximo, lo que se puede expresar como en la ecuación 2.

$$L(\hat{u}_k) = \max_{u_k=+1}\left(M_{\alpha k}(s) + M_{\beta k}(s)\right) - \max_{u_k=-1}\left(M_{\alpha k}(s) + M_{\beta k}(s)\right) \quad (2)$$

Aproximar cada operación máx(·) da lugar al algoritmo Máx logMAP. Se debe reseñar que la aproximación es tanto mejor cuanto mayor sea la relación señal a ruido.

El proceso de decodificación se basa en un enfoque pragmático de la Trellis Coded Modulation (TCM) se ha aplicado también a los códigos de bloque y los códigos de Turbo Producto (TPCs). Un simple mapa combinacional de código se utiliza para obtener las salidas, lo cual se debe describir a través de una tabla de búsqueda (Lookup table).

La aplicación de las funciones del algoritmo Log MAP, se logra estableciendo la selección con los SCS (Sumador Comparador Selector), siendo esta salida final, la que es realimentada a la siguiente iteración, con la próxima muestra. La Figura 5, representa la estructura de decodificación iterativa, basada en el algoritmo Log MAP simplificado.

Cada uno de los bloques componentes del decodificador SISO han sido estudiados. La función gamma representa la métrica de la rama [8] con base en los datos de salida, la paridad y la información extrínseca, ésta será dada por una tabla que según las entradas ofrecerá la salida respectiva.

Los módulos alpha y beta están constituidos por unidades básicas de suma, comparación y selección (SCS), las estimaciones hacia delante y hacia atrás de la métrica de estados se realizan con base en la métrica de la rama y del estado anterior, identificados por las funciones descritas en las ecuaciones 3 y 4.

$$\alpha_k(s) = \max\left(\alpha_{k-1}(s') + \gamma(s', s)\right) \quad (3)$$

$$\beta_{k-1}(s) = \max\left(\beta_k(s') + \gamma(s', s)\right) \quad (4)$$

La función Lamda calcula la razón de verosimilitud logarítmica (LLR, Log Likelihood Rate), cuyo comportamiento lo describe la ecuación 5.

$$\lambda = L(\hat{u}_k / \vec{y}) = \max_{u_k=+1}\left(\alpha_{k-1}(s') + \gamma(s', s) + \beta_k(s)\right) - \max_{u_k=-1}\left(\alpha_{k-1}(s') + \gamma(s', s) + \beta_k(s)\right) \quad (5)$$

La información extrínseca se obtiene de la diferencia de la LLR y la información extrínseca del otro decodificador, ésta se utiliza para realimentarla al otro decodificador en la siguiente iteración a fin del cálculo de la métrica de la rama, será dada por la ecuación 6.

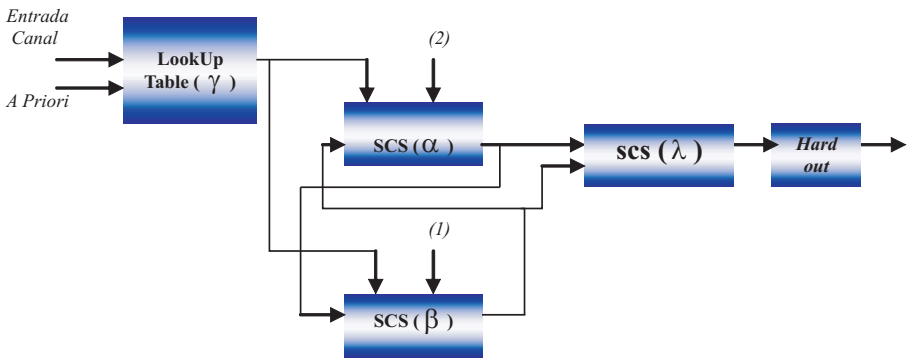

Figura 5. Diagrama de Bloques del Decodificador SISO.

$$L_e(u_k) = L(\hat{u}_k \,/\, \vec{y}) - L(\hat{u}_k \,/\, y_k) \quad (6)$$

Finalmente es importante señalar que la salida es una versión suave de todo el bloque de código TPC después de la decodificación, y no sólo la información binaria. Para el caso de decisiones finales se selecciona el bit más significativo (MSB) de la muestra como criterio de decisión (hard), estará dada por la ecuación 7.

$$bit = \sin g\left(L(\hat{u}_k \,/\, \vec{y})\right) \quad (7)$$

Estos modelos de los componentes del decodificador SISO, serán considerados para establecer su descripción en VHDL.

Metodología

Módulo Hamming

En el codificador TPC el principal componente corresponde al módulo de codificación de Hamming, su eficiencia estará dada en función de la relación de k/n Peebles [13] para efectos del diseño del codificador turbo producto se definió un código Hamming (7,4).

El vector de código resultante estará dado en función de la estructura de la trama mostrada en la Figura 3. Donde *C3, C2, C1*; corresponde a los bits de chequeo de paridad para la protección de la trama y *S(3), S(2), S(1), S(0)* los bits de datos. En base al modelo de la trama codificada mostrado en la Figura 6, se establece el comportamiento del codificador a configurar, tomando como base el codificador Hamming en donde se intercalan los bits de chequeo o paridad con las posiciones de los datos.

Para el diseño del codificador se selecciona la concatenación de los cuatro bits de datos con los tres bits resultantes del chequeo de paridad sobre los bits en las posiciones dadas, se realiza la descripción en VHDL para su programación, la cual se presenta a continuación.

```
******************** codificador Hamming**********************
architecture Behavioral of b_canal_encoder is
signal c1,c2,c3:std_logic;
begin
c1<=s(0) xor s(1) xor s(3); --comprueba paridad de datos d1, d2 y d4
c2<=s(0) xor s(2) xor s(3); --comprueba paridad de datos d1, d3 y d4
c3<=s(1) xor s(2) xor s(3); --comprueba paridad de datos d2, d3 y d4
code<= s(3) & s(2) & s(1) & c3 & s(0) & c2 & c1;
end Behavioral;
************************************************************
```

S(3)	S(2)	S(1)	S(0)	C3	C2	C1

Figura 6. Estructura de la trama codificada por Hamming.

Una vez identificados los componentes del decodificador SISO se define una tabla de búsqueda a la cual se le suministra el dato y los bits de chequeo correspondientes a su corrección, así como la información a priori, para la segunda iteración, a partir de lo que se obtendrá el correspondiente valor estimado.

El comportamiento del decodificador Hamming (hard) se describe a continuación.

```
********************* Decodificador Hamming*********************
architecture Behavioral of e_canal_decoder is
signal c1,c2,c3,c1r,c2r,c3r,a,b,c:s td_logic;
signal d:std_logic_vector (6 downto 0);
begin --calcula del lado del receptor
c1<=deco(2) xor deco(4) xor deco(6);--comprueba paridad de datos d1, d2 y d4
c2<=deco(2) xor deco(5) xor deco(6);--comprueba paridad de datos d1, d3 y d4
c3<=deco(4) xor deco(5) xor deco(6);--comprueba paridad de datos d2, d3 y d4
--compara con los transmitidos en la trama
a<=c1 xor deco(0);
b<=c2 xor deco(1);
c<=c3 xor deco(3);
process (a,b,c,clk)
begin
var<= c&b&a;
if clk'event and clk='1' then
case var is    --corrige el error detectado
when "011"  => d(2)<=not deco(2);
                        d(4)<= deco(4);
                        d(5)<= deco(5);
                        d(6)<= deco(6);
when "101"  => d(2)<= deco(2);
                        d(4)<=not deco(4);
                        d(5)<= deco(5);
                        d(6)<= deco(6);
when "110"  => d(2)<= deco(2);
                        d(4)<= deco(4);
                        d(5)<=not deco(5);
                        d(6)<= deco(6);
when "111"  => d(2)<= deco(2);
                        d(4)<= deco(4);
                        d(5)<= deco(5);
                        d(6)<=not deco(6);
when others => d(2)<= deco(2);
                        d(4)<= deco(4);
                        d(5)<= deco(5);
d(6)<= deco(6);
end case;
end if;
end process;
sr<= d(6) & d(5) & d(4) & d(2);
end Behavioral;
********************* ******************************************
```

Luego se ha seleccionado la descripción del módulo SCS, el cual es una arquitectura común para los demás componentes.

Módulo Sumador Comparador Selector (SCS)

El bloque SCS estará definido en su comportamiento por la estructura de la Figura 7.

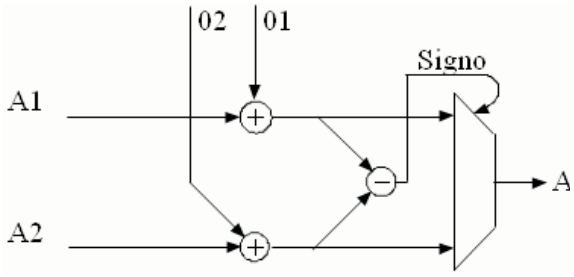

Figura 7. Unidad SCS Fuente: Chávez, 2006 [8].

```
********************* Módulo CSC*********************************
architecture Behavioral of CSC is
signal B1, B2, C: STD_LOGIC_VECTOR (3 downto 0);
begin
B1<=A1 + O1; B2<=A2 + O2;
C <= B1 + not (B2) + '1'; -- C<= B1-B2
Process (C(3)) begin If C(3)='1' then  A<= B1 ; else A<= B2; End If; End Process;
end Behavioral;
*****************************************************************
```

Resultados

Las simulaciones han sido realizadas con ModelSim XE II 5.7g, éste permite observar el comportamiento de los módulos programados en VHDL. En la Figura 8, la señal *encoder/s* corresponde a los datos donde cada paquete comprende de 4 bits de información, este componente produce a la salida la señal *encoder/code* de 7 bits que contiene los 3 bits de redundancia codificados a partir de la descripción programada.

En la Figura 9 se puede observar la decodificación de la data y la corrección de un bit de error que se ha introducido intencionalmente para comprobar la eficiencia del código. (comparar con la salida del codificador), la señal *var* se ha dejado accesible desde el puerto de la entidad solo para fines didácticos ya que esta señal identifica la posición del bit dañado, según la codificación realizada cuando la posición indica un elemento de dato (como se observa en la Figura 1) sea posición 6 ó 7, el decodificador negará el bit de dicha posición con lo que la trama quedará corregida, en tanto que si la posición corresponde a un bit de protección para el caso posición 1, el decodificador no realizará ninguna acción pues la salida será finalmente la data recibida, la misma se muestra a través del puerto como fuente recuperada (*source receive_ sr*).

Así mismo, se presenta la simulación del módulo SCS, en la Figura 10, para las entradas *a1*,*o1 y a2*, *o2* se obtiene la suma *b1* y *b2*, respectivamente; cuya diferencia permite seleccionar el máximo a través del signo o MSB (bit mas significativo) de la señal *c*, así se obtiene a la salida la señal *a* del máximo, de acuerdo a la programación del SCS.

Finalmente, en la Tabla 1 se presenta el reporte de síntesis del módulo SCS, en la cual se puede notar el porcentaje de utilización sobre el FPGA Spartan 2E.

Estos resultados se han presentado a través de las funciones especificas que integran el comportamiento del TPC, el módulo encargado de integrar las funciones debe contar con una tabla de demodulación para el manejo del decodificador SISO, el cual adaptará la señal de entrada y salida del decodificador y de los módulos SCS del diseño.

Conclusiones

Basado en el estudio de los TCP, se seleccionaron unos módulos generales de interés para realizar su programación en VHDL, con el propósito de proponerlos como herramientas para próximos diseños más complejos.

Interpretando los porcentajes de utilización de los módulos diseñados sobre el dispositivo

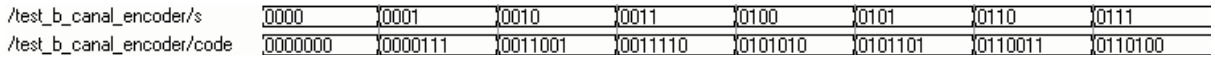

Figura 8. Simulación del codificador Hamming.

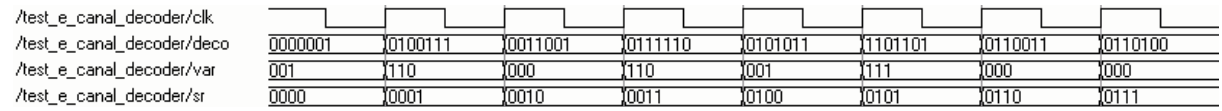

Figura 9. Simulación del Decodificador Hamming.

Tabla 1
Reporte de Síntesis de los módulos diseñados

Utilización	Numero de Slices (2352)	LUTs (4704)	IOBs (142)
Hamming Encoder	2	3	11
TPC Encoder	40	72	57
Hamming Decoder	4	7	14
SCS	10	19	20
Matriz 4 Decoder	172	360	432

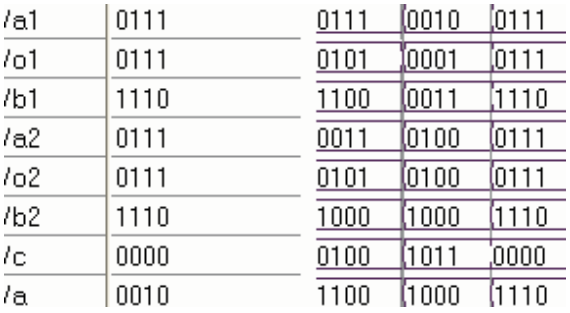

Figura 10. Simulación del SCS.

FPGA, caso particular del Spartan 2E (xc2s200e-6pq208), obtenidos en el proceso de síntesis de los componentes descritos, con la herramienta de diseño ISE 9.2i, se puede establecer que los actuales dispositivos FPGA cuentas con suficientes recursos como para implementar en ellos, los algoritmos de decodificación SISO estudiados, con un alto grado de paralelismo, programando sobre el dispositivo las LUTs (LookUp Tables) que describan la relación del primer componente, la función gamma, y las operaciones comunes en las siguientes etapas, a través de la reutilización del componente diseñado.

Esta propuesta de diseño modular para las etapas de codificación y decodificación, de acuerdo a las características de modularidad del lenguaje descriptor de hardware VHDL [14], permite reutilizar componentes diseñados en forma paralela, siendo escalable el ancho de la palabra, de esta se logra el tratamiento paralelo y se puede avanzar hacia la reducción del número de iteraciones, sin afectar la eficiencia del nuevo código.

Naturalmente, la eficiencia de los diseños se obtuvo mediante la aplicación de técnicas de modelación del comportamiento, aplicando algunas transformaciones a los algoritmos originalmente planteados [11-12], con el propósito de optimizar el consumo de recursos del FPGA. De esta manera se ofrece una herramienta basada en la descripción de los componentes, dejando la libertad de sustituir los codificacidores internos, por módulos paralelos Reed Solomon [15], o arreglos Hamming de longitud ajustable.

Referencias Bibliográficas

1. Revés X., Gelonch, A., Casadeval, L F., Entorno de Desarrollo Software Radio, XVII Simposium Nacional de la Unión Científica Internacional de Radio, 2002. En línea: http://w3.iec.csic.es/ursi/articulos_alcalahenares_2002/sesiones/sesion7.pdf
2. IEEE 802.16 Compatible Turbo Product Code Encoder v1.0, 2002, en línea: http://www.xilinx.com/support/documentation/ip_documentation/tpc_encoder.pdf
3. Sandoval R., Cecilia, Fedón R., Antonio, Programación VHDL de algoritmos de codificación para dispositivos de Hardware Reconfigurable, Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería, Vol. 24 (1), pág. 3-11. En línea: http://www.cimne.upc.es/rimni/papers.asp
4. Banister, Brian A., Using Turbo Product Codes in Client Station Uplink for Reduced Power Consumption, s/f, disponible en: http://www.aha.com/show_pub.php?id=158
5. Ferrari M., Bellin S., Low Ber Turbo Codes for Satellite Communications, s/f, disponible en: http://conferences.esa.int/01C14/papers/5.3.pdf

6. L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate, s/f, en línea: http: //www.tcs.tifr.res.in/~onkar/teaching/bcjr .pdf
7. Jiménez P. Alberto, Cancelación de Interferencias Multiusuario en Sistemas MC-CDMA, 2006, en línea: http://ajimenezp. googlepages.com/ic-mccdma.pdf
8. Chávez, R; Decodificador Turbo MAP de varias Iteraciones, 2006, en línea: http:// redalyc.uaemex.mx/redalyc/pdf/852/8520 2008.pdf
9. Sánchez F., Matilde, Contribución al estudio de las prestaciones de esquemas de codificación basados en Turbo Códigos para Sistemas de Comunicaciones móviles de Tercera Generación, Madrid, 2001, en línea: http:// oa.upm.es/653/01/09200105.pdf
10. Fonseca, A., Um Esquema de Equalização Turbo Aplicando Decodicação Turbo de Códigos Produto de Paridade Simples Multidimensionais, INATEL, 2005, em línea: http:// cict.inatel.br/nova2/docentes/dayan/Publications/MScAndre.pdf
11. Castiñeira M., J., Farrell Patrick G., "Codificacion para el control de errores", Primera Edición, Editorial Wiley, England, 2006
12. Abrantes, S., Do algoritmo BCJR à descodificação turbo, Portugal, 2004, en línea: http://paginas.fe.up.pt/%7Esam/textos/De%20BCJR%20a%20turbo.pdf
13. Peebles, Peyton; Digital Communication Systems, Segunda Edición, Editorial Prentice Hall, New Jersey 1987
14. IEEE Computer Society. "IEEE Standard VHDL Language Reference Manual". The Institute of Electrical and Electronics Engineers, Inc. Mayo 2002
15. Sandoval Ruiz, Cecilia E.; Antonio Fedón. 2007. "Codificador y decodificador digital Reed-Solomon programados para hardware reconfigurable". Ingeniería y Universidad, vol. 11, num. 1, pp.17-31. Disponible en: http: //redalyc.uaemex.mx/redalyc/src/inicio / ArtPdfRed.jsp?iCve=47711102.

Recibido el 18 de Febrero de 2008
En forma revisada el 05 de Noviembre de 2008