Uso de Matlab para la Síntesis en Lenguaje VHDL de Decodificadores Viterbi

Luis Alberto Luna Espinosa[a], Juan de Dios López Sánchez[a], Juan Iván Nieto Hipólito[a], y Mabel Vázquez Briseño[a]

[a] *Universidad Autónoma de Baja California, Facultad de Ingeniería, Arquitectura y Diseño.*
Carr. Ensenada-Tijuana 3917, Col. Playitas, Ensenada, B.C., México, 22860.
kel.night.eyes@hotmail.com,{jddios,jnieto,mabel.vazquez}@uabc.edu.mx

2013 Published by $\mathcal{DIFU}_{100}ci@$ http://www2.uaz.edu.mx/web/www/publicaciones
Selection and peer-review under responsibility of the Organizing Committee of the CICOMP-2013, www.cicomp.org

Resumen

En el presente trabajo se muestra una forma más sencilla de programar decodificadores de Viterbi utilizando dispositivos programables FPGA(Field Programmable Gate Array. Estos decodificadores son creados a partir de los puntos de suma que describen a las salidas de un codificador convolucional no retroalimentado a una tasa de codificación de $\frac{1}{2}$. Esta relación ha sido utilizado para implementar un programa en Matlab, el cual, genera decodificadores en lenguaje VHDL para un dispositivo FPGA a partir de un conjunto de entidades básicas utilizados para formar células ACS.

Palabras clave: Células de suma-compara-selecciona, Código Convolucional, Decodificador Viterbi.

1. Introducción

La codificación de canal es una herramienta ampliamente utilizada en los sistemas de comunicaciones debido a las mejoras que esta puede aportar al sistema, tales como mejor desempeño del Bit Error Rate (BER), reducción de potencia, incluso incrementar la tasa de transmisión manteniendo la misma calidad del enlace. Además, este tipo de técnica supone una mejora considerable sin implicar altos costos de hardware especializado, siendo adaptable a todo tipo de esquemas de comunicación debido a sus diversas propiedades y capacidades correctivas, por lo que para cada canal existen una o varias opciones viables [1].

Un caso en particular de este tipo de codificaciones que a pesar de que ha transcurrido un largo tiempo desde su aparición, son los códigos convolucionales, mismos que en la actualidad siguen siendo ampliamente utilizados mediante diversas variantes, y que dependiendo de la necesidad de la aplicación tiene varias ventajas sobre otros tipos de codificaciones [2].
En la industria de las comunicaciones es común la utilización de arreglos lógicos programables como lo FPGA para la implementación de funciones u operaciones que requieren tiempo real tales como la sincronización, filtrado, ecualización, etc. debido a su estructura que permite implementar de funciones y operaciones lógicas de forma eficiente en términos de tiempo de operacio-

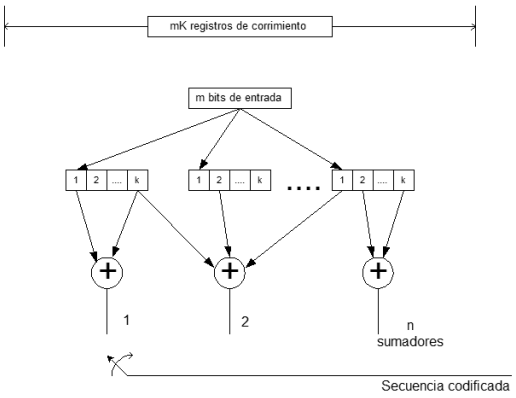

Figura 1. Diagrama de bloques de un codificador convolucional.

nes permitiendo realizar operaciones en pocos ciclos de reloj reduciendo sustancialmente el retardo [3]. Además, al ser programables permite actualización y la reconfiguración del receptor [4]. En el presente trabajo se trata el diseño e implementación de código de Matlab, el cual, a partir de los vectores que describen a los puntos de suma de un código convolucional, puede generar decodificadores convolucionales utilizando un conjunto de bloques básicos implementados como Entidades en VHDL. Dichos bloques son utilizados para crear células de Suma-Compara-Selección (ACS: Add-Compare-Select), mismas que serán interconectadas para dar origen al decodificador en cuestión. La complejidad de los decodificadores también es descrita.

2. Teoría de funcionamiento

2.1. Codificador Convolucional

Un código convolucional es generado cuando una secuencia de datos atraviesa una serie de registros de estados finitos, los cuales forman una secuencia nueva al hacer una combinación lineal de los datos en dichos registros. Los registros consisten en mK registros de corrimiento y n generadores de funciones lineales (puntos de suma), las conexiones entre los registros y los puntos de suma son generalmente descritos por n vectores V de longitud K. El codificador es alimentado con m bits a la vez, mientras que la salida está dada por los n generadores, de modo que la tasa del codificador puede definirse aproximadamente como m/n. El parámetro K es llamado constante de restricción y define la cantidad de registros a través de los cuales pasa un bit de información influyendo la salida del codificador antes de abandonar el mismo [5]. Un codificador convolucional genérico es ilustrado en la Figura 1.

Este diagrama puede ser también utilizado para conocer la salida generada por un codificador a partir de

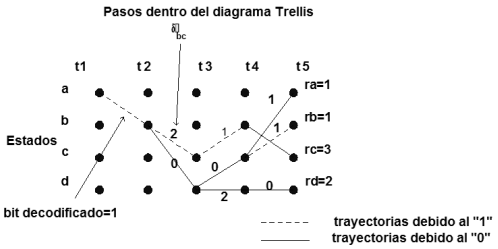

Figura 2. Pasos dentro del diagrama de enrejado.

la secuencia de datos entrantes, mediante observar los trayectos entre estados generados por la entrada.

2.2. Algoritmo del decodificador Viterbi

El algoritmo de Viterbi [6] se basa en un decodificador de máxima verosimilitud y en la recursividad que presenta el diagrama de Trellis. El algoritmo de esta decodificación consiste en hacer una comparación de la secuencia recibida en los instantes t_i (con $i = 1,\ 2,\ 3,\ \ldots$) con todos los caminos de Trellis entrando a los diversos estados en t_i, dicha comparación deja unas marcas sobre los caminos denominadas Distancia de Hamming, $\delta_{x,y}$, donde x es el estado de origen y y es el estado al que se llega, posteriormente se calcula la Distancia de Hamming Acumulada de cada estado para ese instante t_i que es la suma de los $\delta_{x,y}$ desde el tiempo t_1 hasta t_i. Una vez obtenida la Distancia de Hamming Acumulada para cada estado, se comparan las pertenecientes a un mismo estado, es decir se compara $\delta_{x,y}$ con $\delta_{z,y}$ para cada uno de los estados, eliminando aquellas Distancias de Hamming Acumuladas más grandes, de modo que solo quede un único camino entrando a cada estado, convirtiéndose la distancia de Hamming Acumulada en la Medida del Estado Γ_y, lo cual facilita el cálculo de las posteriores Distancias de Hamming Acumuladas. El proceso es repetido para los tiempos t_i posteriores y después de un número de comparaciones y avances dentro del diagrama de Trellis (en práctica 4 o 5 veces K), los caminos van convergiendo en un mismo origen del cual es posible extraer los datos codificados, como se muestra en la Figura 2 [6].

3. Implementación del decodificador Viterbi

Las células ACS son una implementación del algoritmo de Viterbi las cuales se encargan de obtener la diferencia entre la secuencia recibida y el código generado por cada transición entre los diversos estados, calcula las métricas de estado, y decide que trayectos almacenar dada la menor métrica de cada estado para cada momento t_i. Esta implementación permite proce-

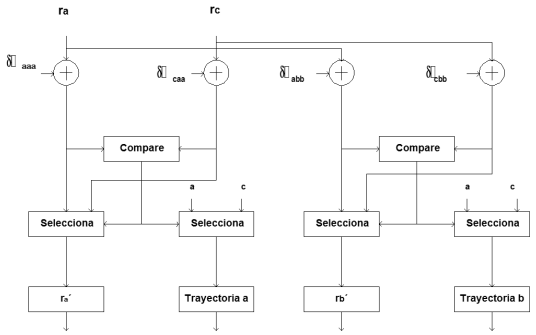

Figura 3. Diagrama de bloques de una implementación de células ACS en el decodificador Viterbi con $K = 3$.

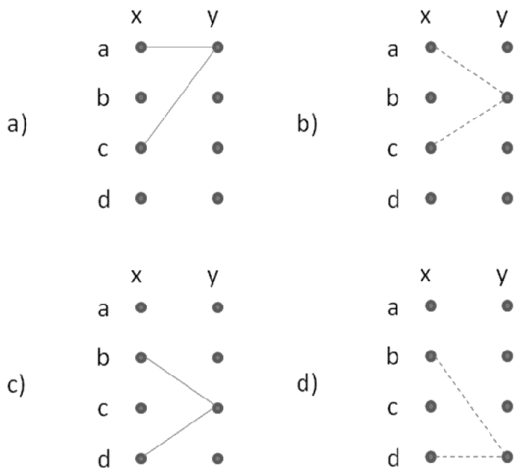

Figura 4. Relación entre los estados origen (x) y los estados (y) para los estados a, b, c y d respectivamente.

sar dos estados a la vez cuyos estados precedentes sean comunes a los estados procesados. Actualmente existen variantes de esta implementación en función del dispositivo en el cual se implementa, dichas variaciones permite mejorar ciertos aspectos tales como la latencia, minimizar la memoria requerida para almacenar las trayectorias de los estados, a cambio de algún parámetro no significativamente relevante para determinada aplicación [2],[7]. Un diagrama de bloques que ilustra el funcionamiento de las células ACS se muestra en la Figura 3.

Las células ACS son capaces de procesar dos estados consecutivos (estados destino), cuyos estados anteriores son comunes (estados origen). La relación entre estados origen y estados destino resulta de observar el diagrama de trellis y está dada por la cantidad de registros utilizados en el codificador, de modo que para un codificador con $K = 3$ los estados origen de cada estado destino quedan ilustrados en la Figura 4.

Una relación más precisa puede ser obtenida de generalizar la cantidad de registros que el codificador pueda tener. Para un determinado cantidad de registros K,

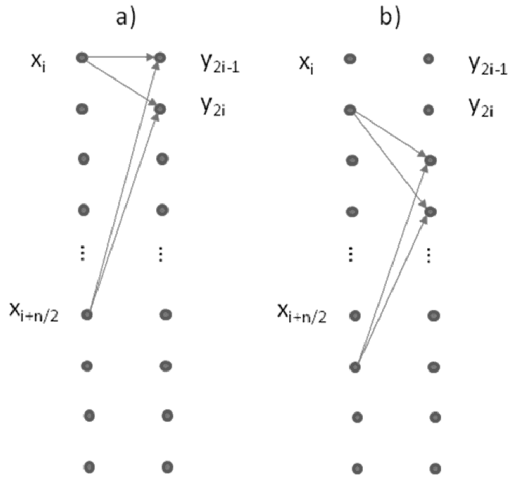

Figura 5. Relación genérica entre los estados x y los estados y para a) $i = 1$, b) $i = 2$.

se tienen 2^{K-1} estados. Dado que la transición entre estados está únicamente en función de la entrada tenemos para cada estado destino y_{2i-1} y y_{2i} le corresponden los estados origen x_i y $x_{i+\frac{n}{2}}$, con $n = 2^{K-1}$ e $i = 1,\ 2,\ \ldots,\ \frac{n}{2}$. En la Figura 5 observamos la representación de esta relación entre estados.

4. Implementación en el FPGA

Un primer decodificador fue hecho a partir de un código $K = 3$, $n = 2$, mismo que sirvió para definir las entidades de los bloques básicos requeridos para la implementación de las células ACS. Dichas entidades son:

DeltaXY Este componente se encarga de comparar la secuencia codificada contra la secuencia generada por los diversos trayectos en el diagrama de Trellis, dando como salida la cantidad de bits en los que difieren.

PathHistory Es una memoria que almacena los el camino con menor métrica de estado entrando a un determinado estado destino, mientras que el bit decodificado es el ultimo bit de la memoria asociada al estado con menor métrica de todos.

ACS Es el componente central del decodificador, ya que para cada estado realiza el cálculo de la trayectoria con menor métrica, esto en función de la información recibida del componente *DeltaXY*.

A partir de las entidades que describen a los bloques básicos y la relación general descrita anteriormente, se elaboró un programa (script) en Matlab, el cual, a partir de seleccionar los vectores V que describen a

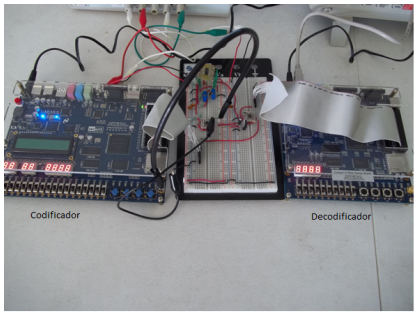

Figura 6. Implementación física.

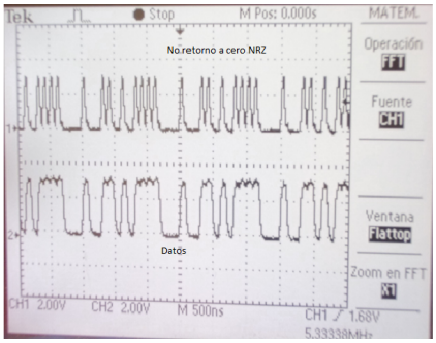

Figura 7. Formas de ondas requeridas para la transmisión de los datos codificados.

los puntos de suma, genera código en VHDL con las instancias e interconexiones necesarias para la descripción del decodificador especificado, siendo este código perfectamente portable a cualquier dispositivo FPGA, con el único requerimiento de las señales de reloj que indiquen los tiempos de bits.

5. Resultados

El programa en código Matlab realizado fue utilizado para generar un conjunto de codificadores óptimos [6]-[8] en cuanto a sus capacidades correctivas para diversos valores de K con la finalidad de evaluar la complejidad del mismo. Estos códigos fueron compilados en el IDE QUARTUS II^{TM} para un dispositivo FPGA Cyclone II^{TM}. El consumo de , unidades lógicas con respecto a K para valores de $K = 3,\ 4,\ 5,\ 6,\ 7,\ 8$, de la cual se observa que la complejidad, relacionada a la cantidad de unidades lógicas requeridas, incrementa exponencialmente de acuerdo a [1]. Para la verificación del correcto funcionamiento se implementó el código obtenido en los dispositivos de FPGAs, tanto para la codificación convolucional como la etapa de decodificación Viterbi. La implementación física del codificador y el decodificador se muestran en la Figura 6.

En la Figura 7 se muestran formas de onda No Retorno a Cero (NRZ) que se utilizaron para la transmisión por un canal físico entre el transmisor (codificador) y el receptor (decodificador).

6. Conclusiones

Se ha presentado una relación que permite predecir la forma en la cual estarán interconectados los estados para la implementación de un decodificador utilizando el algoritmo de Viterbi y células ACS. La utilización de un programa en código de Matlab que haga uso de estas entidades permite la generación de diversos tipos de decodificadores sin presentar ninguna dificultad en el desarrollo del mismo, lo cual permite realizar pruebas experimentales con rapidez, bajo diversos escenarios como variantes de canal, modulaciones, o sistemas en general, facilitando en diseño de los decodificadores. Este programa permite que el usuario no tenga la necesidad de implementar los decodificadores en código VHDL, sino de utilizar un script o código en Matlab que permite implementarlo rápido y fácilmente.

Agradecimientos

Los autores desean agradecer a la Administración de la Facultad de Ingeniería, Arquitectura y Diseño (FIAD) de la Universidad Autónoma de Baja California (UABC) y al CONACYT.

Referencias

[1] A. Neubauer, *Coding Theory: Algorithms, Architectures and applications*, England Wiley, 2010.
[2] S. V. Maiya, Daniel J. Costello and T. E. Fuja, "Low Latency Coding: Convolutional Codes vs. LDPC Codes," *IEEE Transactions on Communications*, Vol. 60, No. 5, May, 2012.
[3] S. Karris. *Digital Circuit Analysis and Design with Simulink Modeling and Introduction to CPLDs and FPGAs*. Orchard Publications, USA, 2007.
[4] W. Tuttlebee. *Software Defined Radio*. John Wiley and Sons, Sussex, Inglaterra, 2002.
[5] A. Viterbi. *Communication systems engineering*. Prentice Hall, Upper Saddle River, New Jersey, 2001.
[6] A. Viterbi. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, Vol. 13, pp. 260-269, Abril 1967.
[7] B. He, R. Schober. "Bit-Interleaved Coded Modulation for Hybrid RF/FSO Systems," *IEEE Transactions on Communications*, Vol. 57, pp. 3753-3763, Diciembre 2009.
[8] B. Sklar. *Digital Communications*. Prentice Hall, New Jersey, p.408, 2001.