Arquitectura HW para decodificador *Two-Step* SOVA con recorridos hacia atrás sistólicos

F. Javier Martín-Vega, F. Javier López-Martínez, J. Tomás Entrambasaguas
{fjmvega, fjlopezm, jtem}@ic.uma.es
Dpto. de Ingeniería de Comunicaciones. Universidad de Málaga.
Campus de Teatinos s/n, ETSI Telecomunicación, E-29071 Málaga.

***Abstract*- In this paper a novel SOVA (Soft-Output Viterbi Algorithm) decoder using systolic arrays to carry out trace back method is presented. Systolic arrays are associated with high data rates and small resource requirements, so they are especially attractive for efficient hardware implementations. The proposed architecture offers an excellent performance in terms of throughput, and has been used to implement a turbodecoder compliant to the 3GPP-LTE specification.**

I. INTRODUCCIÓN

En los sistemas de comunicaciones móviles actuales, los turbo códigos representan la pieza fundamental en la codificación de canal debido a la gran capacidad correctiva que ofrecen. En particular, se encuentran integrados dentro de estándares tales como 3GPP-LTE (*Long Term Evolution*) y LTE Advanced.

La decodificación de turbo códigos [1] consiste en un proceso iterativo en el que en cada iteración se produce una secuencia en forma de medidas LLR (*Log Likelihood Ratio*) que será usada como información a-priori en la siguiente iteración, permitiendo usar un algoritmo MAP (*Maximum a Posteriori*) que minimice la probabilidad de error de símbolo. No obstante, el algoritmo MAP disponible, el algoritmo BCJR (Bahl, Cocke, Raviv, Jelinek) [2] tiene asociado una latencia considerable. El algoritmo SOVA [3] se presenta como una alternativa a tener en cuenta, ya que presenta una menor latencia que [2]. Este algoritmo [3] se puede ver como una modificación del algoritmo de Viterbi para poder producir y consumir medidas LLR, y por tanto ser usado en un esquema de decodificación con turbo códigos. La versión adecuada a HW es conocida como Two-Step SOVA [4], y para producir su salida requiere hacer 3 recorridos hacia atrás en lugar de 1 como hacía falta en el algoritmo de Viterbi. Por tanto, la estructura que realice estos recorridos hacia atrás determinará en gran medida tanto el régimen binario alcanzado como el consumo en potencia del decodificador.

Una excelente solución para hacer los recorridos hacia atrás, ya que consigue decodificar un bit por ciclo de reloj, fue presentada por Truong [5] para el algoritmo de Viterbi y códigos no recursivos, haciendo uso de arrays sistólicos. Sin embargo, no es aplicable para códigos recursivos, ni tampoco es válida cuando se emplea un decodificador de tipo SOVA.

En este trabajo, se presenta una modificación de la arquitectura original propuesta en [5], que permite calcular los estados anteriores de forma eficiente en códigos recursivos. Asimismo, la arquitectura propuesta es adecuada para el algoritmo SOVA, con lo que puede emplearse para la implementación de turbodecodificadores.

El resto del artículo se estructura del siguiente modo. En la sección II se introduce la arquitectura Two-Step SOVA. Las secciones III y IV presentan las modificaciones realizadas para soportar el caso SOVA y códigos recursivos, respectivamente. Los resultados obtenidos en cuanto a régimen binario y capacidad correctiva se muestran en la sección V. Finalmente en el apartado VI se dan las conclusiones.

II. ARQUITECTURA TWO-STEP SOVA

El algoritmo de Viterbi utiliza los bits detectados que han pasado por el canal para decodificar. Para ello realiza la operación ACS (*Add, Compare, Select*) que consiste en calcular la métrica de los dos caminos competidores que llegan a cada nodo del trellis, compararlas y seleccionar como camino superviviente a aquel de mejor métrica, almacenando un bit que identifique dicho camino. Así se reduce la complejidad de la búsqueda del camino ML (*Maximum Likelihood*), ya que en todo momento el número de caminos posibles queda reducido a 2^K, siendo K la longitud del código. Cuando se llega al final de la secuencia se arranca el recorrido hacia atrás desde el estado S0 para obtener la secuencia decodificada.

El algoritmo SOVA por otro lado, en lugar de usar bits, utiliza las medidas LLR que vienen del canal y las medidas generadas en la iteración anterior, de forma que las medidas que sean más fiables tendrán mayor impacto en la métrica, y por tanto mayor impacto en el camino superviviente.

Al hacer la operación ACS, el algoritmo SOVA además calcula un peso asociado a la selección del camino superviviente en cada nodo como la diferencia de las métricas de los caminos competidores. Esto se hace para cuantificar la calidad de la decisión acerca de cada bit y obtener medidas LLR. Sin embargo, este cálculo representa valores demasiado optimistas con lo que se requiere un proceso de actualización de los pesos para obtener las medidas necesarias en la siguiente iteración.

La arquitectura Two-Step SOVA [4] es una arquitectura orientada a HW que propone calcular primero el camino ML, y actualizar después los pesos asociados a dicho camino, con objeto de reducir el número de operaciones. Al ser una arquitectura orientada a HW se basa en el uso de ventanas para decodificar. La idea que hay detrás de la decodificación con ventana, es que todos los caminos que hay supervivientes en un instante dado, acaban convergiendo en el camino ML transcurridos un número suficientemente grande de instantes de tiempo. Para ello se almacenan los bits de historia que identifican el camino superviviente en cada nodo del trellis para un intervalo de L muestras. Para encontrar el camino ML, se inicia un recorrido hacia atrás desde cualquier estado,

y se espera que el estado al que se llegue forme parte de dicho camino.

La Fig. 1 representa la arquitectura *Two-Step*, donde la unidad MLPU (*Maximun Likelihood Unit*) calcula los estados del camino ML, haciendo un recorrido hacia atrás a través de la ventana de decodificación, y calcula además los pesos asociados a dicho camino identificados como W_{k-L} y S_{k-L} respectivamente. Las entradas son las medidas sistemática y de paridad que vienen del canal en el instante k (Ls_k, Lp_k) y la medida a-priori La_k calculada en la iteración anterior.

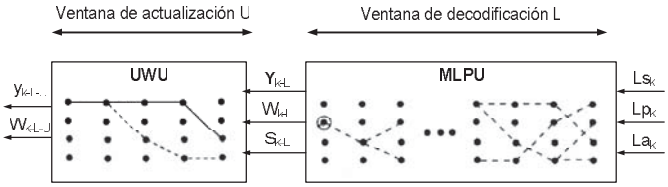

Fig. 1 Estructura Two-Step SOVA

La unidad UWU (*Updating Weigth Unit*) realiza un doble recorrido hacia atrás de los caminos superviviente y competidor a cada estado del camino ML, para obtener las secuencias asociadas a cada camino. Compara ambas secuencias, y actualiza los pesos donde ambas difieran tal y como se detalla en [4]. Por tanto, el objetivo de esta unidad es actualizar los pesos calculados por la unidad MLPU para que cuantifiquen de forma precisa la calidad de los bits decodificados, identificados como y_{k-L-U} en la fig. 1.

III. Recorrido hacia atrás con estructuras sistólicas

En la implementación clásica del algoritmo de Viterbi, un bloque lleva a cabo la operación ACS almacenando en memoria los bits que identifican las ramas supervivientes. Para decodificar cada bit, se inicia un recorrido hacia atrás a través de una ventana de longitud L haciendo uso de las decisiones almacenadas en memoria. El problema evidente de esta forma de operar es que la productividad es de un bit cada L ciclos de reloj.

En [5] se plantea una forma alternativa de proceder que consigue decodificar un bit por ciclo de reloj utilizando arrays sistólicos para hacer los recorridos hacia atrás. Estas estructuras formadas por elementos iguales, trabajan en cadena para conseguir un régimen binario elevado.

En la Fig. 2 se muestra la estructura propuesta en [5]. La unidad *SelectionUnit* lleva a cabo la operación ACS generando un vector de decisión cada ciclo de reloj que entrega al array sistólico formado por los bloques *TraceBackElement*. Este vector de decisión aparece identificado con la letra Y. Además entrega un estado que aparece en la figura con el símbolo S^0_m desde el que iniciar los recorridos hacia atrás.

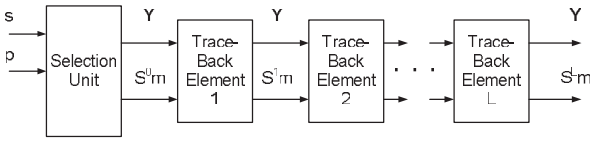

Fig. 2 Decodificador de Viterbi sistólico

Cada bloque *TraceBackElement* calcula el estado anterior a uno dado a partir del vector de decisión correspondiente en un ciclo de reloj. De esta forma un recorrido hacia atrás tardaría L ciclos en completarse. No obstante, puesto que la unidad de selección inicia un nuevo recorrido cada ciclo, las unidades *TraceBackElement* deben de ser capaces de almacenar dos vectores de decisión. Por tanto, la estructura del bloque *TraceBackElement* es la que se ilustra en la Fig. 3.

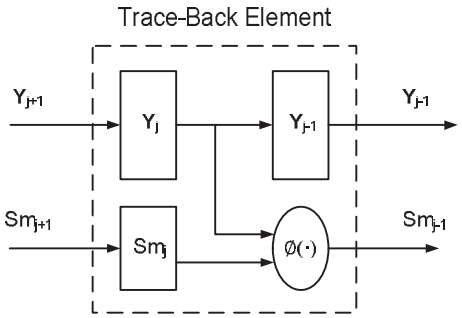

Fig. 3 Bloque TraceBackElement

El bloque que aparece en la figura representado con el símbolo $\emptyset(\cdot)$ se corresponde con la función que calcula el estado anterior a Sm_j utilizando las decisiones dadas por Y_j.

El hecho de que haya el doble de registros para almacenar las decisiones tiene una consecuencia evidente, y es que los recorridos hacia atrás viajan por el array sistólico en la mitad de ciclos. Esto implica que el primer recorrido hacia atrás se inicia cuando se hayan almacenado las decisiones de $L/2$ instantes de tiempo, con lo que la latencia del decodificador de Viterbi sistólico es de $2L$ ciclos.

Cuando se recibe la última palabra código se inicia el último recorrido hacia atrás, pero este recorrido avanza por la mitad de registros que los vectores de decisión. Por tanto, cuando este recorrido termina siguen quedando bits sin decodificar. La solución a este problema es sencilla siempre que para codificar el bloque se haya terminado en el estado S0, ya que este estado tiene un lazo a sí mismo. Por tanto, la solución consiste en que la unidad *SelectionUnit* continúe iniciando recorridos hacia atrás desde el estado S0, a la vez que entrega vectores de historia nulos.

En los siguientes apartados se van a comentar las modificaciones hechas sobre el array sistólico propuesto en [5] para adaptarlo al algoritmo Two-Step SOVA.

A. Bloque MLPU

La unidad que aparece en el diagrama del decodificador Two-Step SOVA (Fig. 1) con el acrónimo MLPU es la unidad encargada de encontrar el camino ML. Por tanto se corresponde esencialmente con un decodificador de Viterbi y el array sistólico de [5] se puede utilizar para este bloque. Sin embargo, en [5] sólo se tiene en cuenta el caso de códigos no recursivos, que son los usados normalmente con los códigos convolucionales. En el caso de los turbo códigos, los códigos usados son recursivos, con lo que la función que calcula el estado anterior será distinta.

La otra diferencia viene del hecho de que el algoritmo SOVA trabaja con medidas LLR en lugar de bits, con lo que el bloque *SelectionUnit* debe obtener la métrica en base a medidas, y debe calcular un peso asociado a cada decisión que entregará a la unidad UWU junto con los estados del camino ML.

B. Bloque UWU

Esta unidad se encarga de hacer la actualización de los pesos calculados por la unidad *SelectionUnit*. Para ello hace un doble recorrido de los caminos competidor y superviviente para cada estado del camino ML. El objetivo de hacer este doble recorrido es obtener la secuencia de bits

asociado a ambos caminos para poder compararlas y hacer la actualización de los pesos.

Nótese que en el camino ML hay una continuidad que no comparten los caminos competidores. Es decir, si se inicia un recorrido hacia atrás desde estados consecutivos del camino ML se obtiene la misma secuencia de bits retrasada. No ocurre lo mismo con los caminos competidores ya que cada uno es distinto de los demás. Esto requiere que el array sistólico que hace el doble recorrido hacia atrás calcule *U* bits de un camino competidor distinto cada ciclo de reloj, donde *U* es el tamaño de la ventana de actualización. Cuando un recorrido hacia atrás llega al final de la ventana se han calculado todos los estados anteriores de ese camino, con lo que es posible obtener los bits asociados a un camino retrasándolos de forma adecuada.

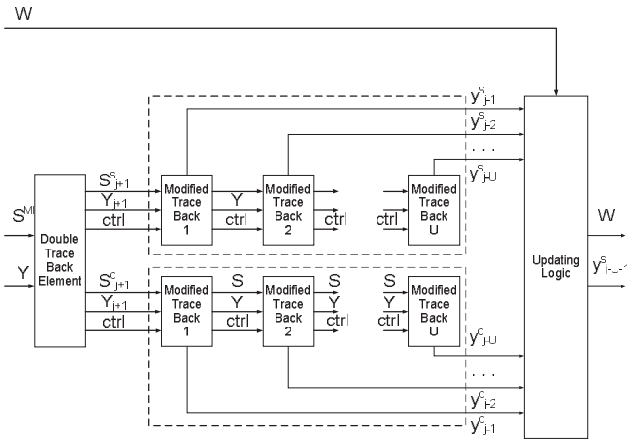

Fig. 4 Esquema unidad de actualización UWU

La Fig. 4 muestra el esquema de la unidad UWU, y los arrays sistólicos que hacen el doble recorrido hacia atrás. El bloque *DoubleTraceBackElement*, calcula los estados superviviente y competidor a los estados del camino ML que calcula la unidad MLPU. En un array sistólico distinto se inician los dos recorridos hacia atrás, que dan una salida paralela de *U* bits cada ciclo de reloj. El bloque *UpdatingLogic*, actualiza los pesos utilizando el circuito en pipe-line que se propone en [4] para la implementación del algoritmo Two-Step SOVA.

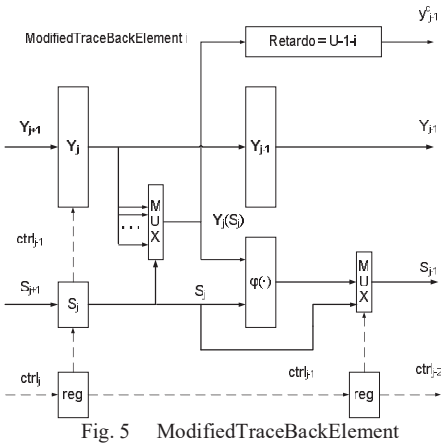

Fig. 5 ModifiedTraceBackElement

En el caso del camino competidor, el último recorrido hacia atrás se hace desde el estado *S1*. Por tanto, ahora hay que forzar ese último recorrido ya que este estado no tiene un lazo a sí mismo. Para ello se añade la señal de control *ctrl* que cuando está activa permite que el último estado avance por el array sin modificar, de forma que cada elemento de la cadena calcule el estado anterior a este estado.

De este modo se hacen los recorridos hacia atrás correctamente para los últimos bits del bloque. Para obtener la salida paralela se retrasan adecuadamente los bits decodificados para que salgan de la unidad todos los bits de una camino a la vez. La Fig. 5 muestra la estructura del elemento que forma el array sistólico con salida paralela de la unidad UWU.

IV. CÁLCULO DEL ESTADO ANTERIOR

El cálculo del estado anterior es la tarea principal que realiza cada elemento del array sistólico, y se corresponde con un bloque de lógica combinacional. En [5], este cálculo se hace con una operación muy sencilla que conduce a un retardo combinacional del array sistólico casi despreciable. Sin embargo, se hizo considerando códigos no recursivos. Así, en este apartado se hace la extensión a códigos recursivos.

La principal diferencia con los anteriores, está en que el bit de información no entra en el registro de desplazamiento, con lo que para decodificar no basta con conocer los estados del camino ML.

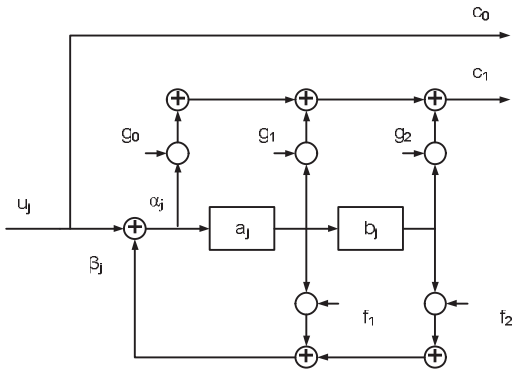

Fig. 6 Codificador Convolucional Recursivo

Un ejemplo de codificador sistemático recursivo, aparece en la Fig. 6, junto con los bits representativos. Las siguientes expresiones definen el bit que entra en el registro de desplazamiento α, para un código de longitud $K=2$.

$$\left.\begin{array}{c} \alpha = u \oplus \beta \\ \beta = F_{K=2}(a,b) = f_1 \cdot a \oplus f_2 \cdot b \end{array}\right\} \quad (1)$$

Al ser la función $F_K(\cdot)$ que define la realimentación un polinomio en GF(2) (*Galois Field*) se cumple la siguiente propiedad.

$$F_{K=2}\left(a,\bar{b}\right) = \overline{F_{K=2}(a,b)} \quad (2)$$

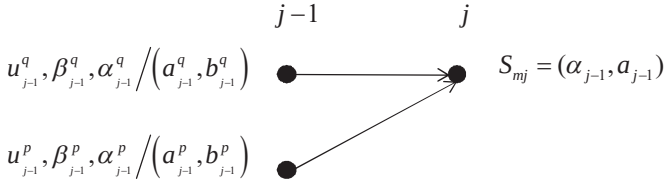

Fig. 7 Estados que convergen en códigos recursivos

La Fig. 7 ilustra los bits representativos para dos estados que convergen, en el caso de códigos recursivos. Los bits que

forman parte del estado al que se converge deben ser iguales para los estados competidores en el instante anterior, con lo que, el bit que diferencia a los estados es el bit que sale del codificador. Por tanto se cumplen las siguientes igualdades.

$$\left\{ \alpha_{j-1}^{p} = \alpha_{j-1}^{q},\ a_{j-1}^{p} = a_{j-1}^{q},\ b_{j-1}^{p} = \overline{b_{j-1}^{q}} \right\} \tag{3}$$

De (3) se aprecia que el bit b_{j-1} diferencia a las dos ramas que compiten, con lo que se podría usar para hacer los recorridos hacia atrás tal y como se hace en [5]. Sin embargo, el problema llega a la hora de decodificar, ya que en códigos recursivos, el bit de información no forma parte del estado. Conviene por tanto utilizar otra información para hacer los recorridos hacia atrás.

Esta información será el bit de información que entra al codificador, ya que en códigos recursivos este bit es distinto para los dos estados que convergen, como se demuestra fácilmente. Para ello se representa el bit que se realimenta en el codificador para los dos estados competidores. Usando las ecuaciones (1), (2) y (3) obtiene lo que sigue.

$$\beta_{j-1}^{p} = F_{K=2}\left(a_{j-1}^{q}, \overline{b_{j-1}^{q}}\right) = \overline{F_{K=2}\left(a_{j-1}^{q}, b_{j-1}^{q}\right)} = \overline{\beta_{j-1}^{q}} \tag{4}$$

Donde se aprecia que este bit realimentado es distinto para los dos estados. Como el bit α que entra en el registro de desplazamiento es el mismo para los estados competidores, y $\alpha = u \oplus \beta$, se obtiene la siguiente igualdad.

$$u_{j-1}^{p} \oplus \beta_{j-1}^{p} = u_{j-1}^{q} \oplus \overline{\beta_{j-1}^{q}} \rightarrow u_{j-1}^{q} = \overline{u_{j-1}^{p}} \tag{5}$$

Ya que en códigos recursivos el bit de información para los dos estados que convergen es distinto, puede usarse tanto para hacer los recorridos hacia atrás como para decodificar, con lo que éste será el bit que se almacena con este tipo de códigos $y(m_j)=u_{j-1}$. Queda pendiente por tanto obtener la expresión que calcula el estado anterior de un estado Sm_j con la información almacenada $y(m_j)$.

Conociendo el estado en j, se conoce el primer bit del estado anterior $b_j=a_{j-1}$. En el caso de códigos de longitud K, se conocen los $K-1$ primeros bits del estado anterior a partir de los $K-1$ bits del estado actual. Para calcular el último bit del estado anterior se parte del primer bit del estado actual a_j. Sabiendo que $a_j=\alpha_{j-1}$ y sustituyendo en (1) para $K=2$ se llega a la siguiente ecuación.

$$a_j = \alpha_{j-1} = u_{j-1} \oplus f_1 \cdot a_{j-1} \oplus f_2 \cdot b_{j-1} \tag{6}$$

Por tanto para calcular el último bit del estado anterior tan sólo hay que despejar b_{j-1} de (6). Esto requiere que $f_K=1$, lo cual se cumple siempre en la práctica, con lo que la función $\varphi(\cdot)$ que calcula el estado anterior al estado Sm_j es la que sigue para el ejemplo de $K=2$.

$$\varphi\left(S_{mj}, y\left(m_j\right)\right) = \left(b_j, a_j \oplus y\left(m_j\right) \oplus f_1 \cdot b_j\right) \tag{7}$$

V. RESULTADOS

El diseño realizado ha sido sintetizado en una FPGA Virtex-4 XC4VSX35-12ff668. Para evaluar sus prestaciones, se ha integrado la arquitectura SOVA dentro de un turbodecodificador para la tecnología 3GPP-LTE, y se ha atendido a dos parámetros como son el régimen binario y la capacidad correctora.

La Tabla 1 ilustra el régimen binario obtenido para el turbo decodificador si se ejecutan 3 o 5 iteraciones, considerando diferentes tamaños de mensaje. La frecuencia máxima de funcionamiento del turbodecodificador es de 188MHz, mientras que la del decodificador SOVA es de 250MHz.

Longitud del mensaje (bits)	Número de ciclos		Régimen Binario (Mbits/s)	
	3	5	3	5
128	983	1769	24.48	13.6
1024	5463	9833	35.24	19.58
2048	10583	19049	36.38	20.21

Tabla 1 Régimen binario Turbo Decodificador SOVA

La Fig. 8 ilustra la BER (*Bit Error Rate*) que se obtiene para distintas iteraciones para un tamaño de bloque de 3072 bits y canal AWGN (*Additive White Gaussian Noise*). Nótese la mejora que se obtiene con el paso de las iteraciones a medida que mejora la calidad de la información extrínseca que se intercambia entre iteración e iteración.

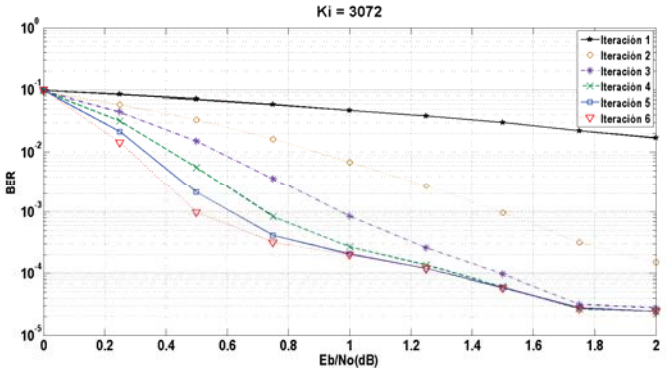

Fig. 8 BER Turbo Decodificador SOVA sistólico

VI. CONCLUSIONES

En este artículo se ha presentado un diseño de decodificador SOVA haciendo uso de arrays sistólicos para decodificar. Para ello se han obtenido las expresiones para el cálculo del estado anterior en el caso de códigos recursivos, y se han modificado los arrays sistólicos usados para el decodificador de Viterbi adaptándolos al decodificador Two-Step SOVA.

REFERENCIAS

[1] Berrou, C.; Glavieux, A.; Thitimajshima, P.; "*Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1,"* Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on , vol.2, no., pp.1064-1070 vol.2, 23-26 May 1993.

[2] Bahl, L.; Cocke, J.; Jelinek, F.; Raviv, J.; "*Optimal decoding of linear codes for minimizing symbol error rate,*" Information Theory, IEEE Transactions on , vol.20, no.2, pp. 284- 287, Mar 1974

[3] Hagenauer, J.; Hoeher, P.; "*A Viterbi algorithm with soft-decision outputs and its applications*," Global Telecommunications Conference, 1989, *GLOBECOM '89., IEEE* , pp.1680-1686 vol.3, 27-30 Nov 1989.

[4] Berrou, C.; Adde, P.; Angui, E.; Faudeil, S.; "*A low complexity soft-output Viterbi decoder architecture*," Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on , vol.2, no., pp.737-740 vol.2, 23-26 May 1993.

[5] Truong, T.K.; Shih, M.-T.; Reed, I.S.; Satorius, E.H.; "*A VLSI design for a trace-back Viterbi decoder*" Communications, IEEE Transactions on , vol.40, no.3, pp.616-624, Mar 1992.