(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 586 409**

(51) Int. CI.:

H04N 19/13 (2014.01)
H03M 7/40 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **19.09.2003 E 03756844 (1)**

(97) Fecha y número de publicación de la concesión europea: **11.05.2016 EP 1540962**

(54) Título: **Método y aparato para codificación y decodificación aritmética**

(30) Prioridad:

20.09.2002 US 412245 P
04.10.2002 US 415999 P
18.09.2003 US 666687
18.09.2003 US 665638
18.09.2003 US 666798

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
14.10.2016

(73) Titular/es:

NTT DOCOMO, INC. (100.0%)
11-1, Nagatacho 2-chome, Chiyoda-ku
Tokyo 100-6150, JP

(72) Inventor/es:

BOSSEN, FRANK JAN

(74) Agente/Representante:

LLAGOSTERA SOTO, María Del Carmen

Descripción

Método y aparato para codificación y decodificación aritmética

Descripción

PRIORIDAD

La presente solicitud de patente reivindica prioridad en la correspondiente solicitud de patente provisional número de serie 60/412.245 titulada "FINALIZACIÓN DE CODIFICACIÓN ARITMÉTICA Y RELLENO DE BYTES", presentada el 20 de septiembre de 2002 y la solicitud de patente provisional número de serie 60/ 415.999 titulada "LIMPIEZA DE CABAC Y REDUCCIÓN DE LA COMPLEJIDAD", presentada el 4 de octubre de 2002.

CAMPO DE LA INVENCIÓN

La presente invención se refiere en general a la teoría de la información, a la compresión de vídeo y a la codificación aritmética. Más en particular, la presente invención se refiere a un método y aparato para la finalización de la codificación aritmética y el relleno de bytes, así como la creación y la utilización de una máquina de estado durante la codificación aritmética

ANTECEDENTES

La compresión de datos es una herramienta extremadamente útil para almacenar y transmitir grandes cantidades de datos. Por ejemplo, el tiempo requerido para transmitir una imagen, como por ejemplo una transmisión a través de la red de un documento, se reduce drásticamente cuando se usa la compresión para reducir el número de bits requeridos para recrear la imagen.

Existen muchas técnicas de compresión de datos diferentes en la técnica anterior. Las técnicas de compresión se pueden dividir en dos grandes categorías, codificación con pérdida y codificación sin pérdida. La codificación con pérdida implica una codificación que tiene como resultado la pérdida de información, de manera que no existe ninguna garantía de reconstrucción perfecta de los datos originales. El objetivo de la compresión con pérdida es que los cambios en los datos originales se realicen de tal manera que no sean objetables ni detectables. En la compresión sin pérdida, se conserva toda la información y los datos se comprimen de una manera que permite la reconstrucción perfecta.

La codificación aritmética es una técnica de compresión bien conocida que se utiliza en algunos sistemas de codificación y de compresión de datos para reducir el número de bits o de símbolos requeridos para la transmisión. Un codificador aritmético recibe un dato de entrada, que incluye una secuencia de eventos (por ejemplo, eventos binarios) o símbolos. El codificador codifica la secuencia de entrada en una secuencia correspondiente de bits o bytes. En algunos casos, se produce un menor número de bits de datos en la salida del codificador de los que se reciben en la entrada de codificador, lo que da como resultado la compresión de datos. Un decodificador aritmético puede recibir o tener acceso a los datos codificados. El decodificador aritmético lee la secuencia de datos codificados y produce datos decodificados, que deben coincidir con los símbolos de entrada recibidos en el decodificador. La compresión se consigue mediante la generación de menos bits en las secuencias de información para los eventos que se codifican, en que las proporciones de los eventos por bits de información que se codifican pueden alcanzar 64:1 o incluso 128:1, dependiendo de la distribución de probabilidad de los eventos.

Preferiblemente, el funcionamiento del decodificador es simétrico con el funcionamiento del codificador. Si el codificador y el decodificador son simétricos en su funcionamiento, el número de bits de datos codificados leídos en el decodificador debe coincidir con el número de bits codificados producidos por el codificador.

En algunos decodificadores aritméticos, al iniciar el funcionamiento del decodificador, el decodificador lee por adelantado un grupo de bits. Sin embargo, dado que el decodificador lee por adelantado un grupo de bits, se puede producir una falta de coincidencia o asimetría.

Una solución convencional para compensar esta asimetría ha sido añadir bits adicionales a los datos codificados en el codificador. En otra solución de convención, no se generan bits codificados adicionales,

pero se permite que el decodificador lea por adelantado en el flujo de bits de los datos codificados, y que a continuación retroceda.

Estas dos soluciones convencionales introducen ineficiencias. Se desea una solución más eficiente para reducir la complejidad de los algoritmos de codificación y de decodificación, reducir los datos para la codificación, transmisión y decodificación y reducir los requisitos de almacenamiento.

RESUMEN DE LA INVENCIÓN

Se describen un método para codificar datos tal como se define por medio de la reivindicación 1, un codificador aritmético tal como se define por medio de la reivindicación 3, un artículo de fabricación tal como se define por medio de la reivindicación 5, un decodificador aritmético tal como se define por medio de la reivindicación 6, y un método de decodificación tal como se define a través de la reivindicación 9.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

La presente invención se entenderá con mayor plenitud a partir de la descripción detallada que se proporciona a continuación y de los dibujos adjuntos de varias realizaciones de la invención, que, sin embargo, no deben ser consideradas como que limitan la invención a las realizaciones específicas, sino que son para explicación y comprensión únicamente.

La **Figura 1** es un diagrama de bloques de una realización de un sistema de codificación y decodificación.

La **Figura 2** es un diagrama de flujo de un proceso de codificación para generar un flujo de bits.

La **Figura 3** ilustra un formato de datos a modo de ejemplo mediante el cual datos codificados pueden ser transmitidos en el sistema de la Figura 1.

La **Figura 4** ilustra un diagrama de bloques de una realización de un codificador aritmético.

La **Figura 5** es un diagrama de flujo de una realización para la codificación de un evento.

La **Figura 6** es un diagrama de flujo de una realización de un procedimiento de renormalización del codificador.

La **Figura 7** ilustra una realización del proceso para la realización de una forma de realización del procedimiento de introducción de bit.

La **Figura 8** es un diagrama de flujo de una realización de un proceso para la decodificación de un evento antes de la finalización.

La **Figura 9** ilustra un diagrama de flujo de una realización de un proceso para el volcado en el momento de la finalización.

La **Figura 10** es un diagrama de bloques de una realización de un decodificador aritmético.

La **Figura 11** es un diagrama de flujo de una realización de un proceso de inicialización de un decodificador aritmético.

La **Figura 12** es un diagrama de flujo de una realización de un proceso para decodificar un evento binario.

La **Figura 13** es un diagrama de flujo de un procedimiento de renormalización.

Las **Figuras 14A y 14B** ilustran diagramas de flujo para la decodificación de un evento binario con equi-probabilidad.

Las **Figuras 15A y 15B** son diagramas de flujo de realizaciones para decodificar una etiqueta de extremo de segmento u otros eventos binarios antes de la finalización.

Las **Figuras 16A y 16B** ilustran una tabla a modo de ejemplo para realizar una búsqueda de estimación de probabilidad.

La **Figura 17** es un diagrama de bloques de un ejemplo de sistema informático.

DESCRIPCIÓN DETALLADA DE LA PRESENTE INVENCIÓN

Se dan a conocer un método y aparato para la codificación y decodificación de información, en particular, datos de vídeo. Durante la codificación y la decodificación, se utiliza un indicador (por ejemplo, el extremo del segmento) para indicar el final de los eventos que son codificados aritméticamente. En una realización, también durante la codificación de la información, los bits o bytes de información de relleno se

añaden al flujo de bits de datos codificados generados por un codificador. En lugar de rellenar estos bits adicionales en medio del flujo de bits de datos codificados, se añaden bytes (o bits) de relleno al final de los datos codificados. Dicho relleno puede utilizarse para mantener una relación entre un número de eventos que se está codificando, una serie de bloques de datos de vídeo (por ejemplo, bloques de macros), y el tamaño de la secuencia de información que se está generando.

En la siguiente descripción, se exponen numerosos detalles para proporcionar una explicación más completa de la presente invención. Sin embargo, para un experto en la técnica será evidente que la presente invención puede ponerse en práctica sin estos detalles específicos. En otros casos, se muestran estructuras y dispositivos bien conocidos en forma de diagrama de bloques, en lugar de en detalle, con el fin de evitar ensombrecer la presente invención.

Algunas partes de las descripciones detalladas que siguen se presentan en términos de algoritmos y representaciones simbólicas de operaciones sobre bits de datos dentro de una memoria de ordenador. Estas descripciones y representaciones algorítmicas son los medios utilizados por los expertos en las técnicas de proceso de datos para transmitir más eficazmente la sustancia de su trabajo a otros expertos en la técnica. Aquí, y en general, un algoritmo se concibe para ser una secuencia de pasos auto-consistente que conduce a un resultado deseado. Los pasos son aquellos que requieren manipulaciones físicas de cantidades físicas. Por lo general, aunque no necesariamente, estas cantidades toman la forma de señales eléctricas o magnéticas capaces de ser almacenadas, transferidas, combinadas, comparadas y manipuladas de otro modo. En ocasiones ha demostrado ser conveniente, principalmente por razones de uso común, referirse a estas señales como bits, valores, elementos, símbolos, caracteres, términos, números o similares.

Debe tenerse en cuenta, sin embargo, que todos estos términos y otros similares han de estar asociados con las cantidades físicas apropiadas y son meramente etiquetas convenientes aplicadas a estas cantidades. A menos que se indique lo contrario de forma específica tal como se desprende de la siguiente discusión, se aprecia que a lo largo de toda la descripción, las descripciones que utilizan términos tales como "proceso" o "computación" o "calcular" o "determinación" o "muestra" o similares, se refieren a la acción y procesos de un sistema informático, o de un dispositivo informático electrónico similar, que manipula y transforma datos representados como cantidades físicas (electrónicas) dentro de los registros y memorias del sistema informático en otros datos representados de manera similar como cantidades físicas dentro de las memorias y registros del sistema informático u otros dispositivos de almacenamiento, transmisión o visualización de dicha información.

La presente invención también se refiere a un aparato para realizar las operaciones en el presente documento. Este aparato puede estar construido especialmente para los fines requeridos, o puede comprender un ordenador de tipo general activado o reconfigurado por un programa de ordenador almacenado en el ordenador de forma selectiva. Dicho programa de ordenador puede estar almacenado en un medio de almacenamiento legible por ordenador, como por ejemplo, pero sin limitarse a, cualquier tipo de disco incluyendo disquetes, discos ópticos, CD-ROM, y discos magneto-ópticos, memorias de sólo lectura (ROM), memorias de acceso aleatorio (RAM), EPROM, EEPROM, tarjetas magnéticas u ópticas, o cualquier tipo de medios adecuados para almacenar instrucciones electrónicas, y cada uno de ellos acoplado a un bus de sistema informático.

Los algoritmos y las pantallas presentados en el presente documento no están intrínsecamente relacionados con ningún equipo u otro aparato en particular. Se pueden utilizar diversos sistemas de finalidad general con programas de acuerdo con las enseñanzas que contiene el presente documento, o puede resultar conveniente construir un aparato más especializado para llevar a cabo los pasos del método requeridos. La estructura requerida para una variedad de estos sistemas resultará aparente a partir de la descripción que se proporciona a continuación. Además, la presente invención no se describe con referencia a ningún lenguaje de programación particular. Se apreciará que se puede utilizar una variedad de lenguajes de programación para implementar las enseñanzas de la invención tal como se describe en el presente documento.

Un medio legible por parte de una máquina incluye cualquier mecanismo para almacenar o transmitir información en un formato legible por parte de una máquina (por ejemplo, un ordenador). Por ejemplo, un medio legible por parte de una máquina incluye una memoria de sólo lectura ("ROM"); memoria de acceso aleatorio (“RAM"); medios de almacenamiento en disco magnético; medios de almacenamiento óptico; dispositivos de memoria flash; señales eléctricas, ópticas, acústicas u otra forma de señales propagadas (por ejemplo, ondas portadoras, señales de infrarrojos, señales digitales, etc...); etcétera

Vista General del Sistema de Codificación y Decodificación

La Figura 1 es un diagrama de bloques de una realización de un sistema de codificación y decodificación 100. Haciendo referencia a la Figura 1, el sistema 100 incluye un codificador 102 y un decodificador 104 que se encuentran en comunicación a través de un canal 120. Alternativamente, el sistema 100 puede incluir solamente codificador 102 o decodificador 104.

El canal 120 puede ser cualquier canal de comunicación de datos adecuado, incluyendo canales por cable e inalámbricos o combinaciones de los mismos. Se puede utilizar cualquier esquema de comunicación y modulación de datos apropiado en el canal 120. Un ejemplo de sistema 100 es un sistema para la codificación, la compresión y la decodificación de datos de vídeo que incluye una secuencia de imágenes. En una realización, cada una de las imágenes se divide en uno o más sectores.

El codificador 102 tiene una entrada 106 para recibir información de entrada como por ejemplo datos de entrada (por ejemplo, información de vídeo). En una realización, el codificador 102 codifica los datos utilizando codificación aritmética. De acuerdo con ello, el codificador 102 puede incluir el almacenamiento de datos, los registros de manipulación, y un motor de codificación aritmética. En una realización, el codificador 102 incluye un registro de intervalos, o registro R, y un registro bajo, o registro L. Además, en una realización, el codificador 102 incluye una máquina de estado de estimación de probabilidad. El algoritmo de codificación realizado por el codificador 102 puede ser codificación aritmética binaria adaptativa de contexto, referida aquí como CABAC, que es bien conocida en la técnica. Además, las técnicas y estructuras descritas en el presente documento pueden extenderse también a otros algoritmos y procedimientos de codificación y decodificación. El codificador 102 tiene una salida 108 para proporcionar datos codificados al canal 120.

En una realización, el codificador 102 genera un flujo de bits de datos codificados que incluye un evento codificado (por ejemplo, decisión) que indica la finalización de los datos aritméticos codificados. En una realización, el evento que indica la finalización de los datos aritméticos codificados comprende una etiqueta de extremo de segmento. El flujo de bits puede también incluir bytes (o bits) de relleno tal como se describe en mayor detalle a continuación.

El decodificador 104 tiene una entrada 110 para recibir los datos codificados del canal 120 y una salida 112 para proporcionar datos decodificados. En una realización, el funcionamiento del decodificador 104 para decodificar los datos codificados es generalmente simétrico con el funcionamiento de codificación del codificador 102. Debe tenerse en cuenta que el sistema 100 puede incluir más de un codificador y / o más de un decodificador.

El codificador 102 y el decodificador 104 pueden ser utilizados en el proceso de datos de vídeo, como, por ejemplo, datos de vídeo generados por un procesador de vídeo (por ejemplo, el códec de video). En una realización, se graba una imagen de vídeo, y se divide en bloques de muestras de datos que pueden representar muestras de 16x16, de 8x8, o de 4x4 de la imagen grabada. Los bloques son transformados a continuación por el procesador de vídeo (por ejemplo, utilizando una transformación discreta de coseno), y se cuantifican para proporcionar valores enteros que representan el bloque de muestra. Los valores enteros son convertidos en una secuencia de eventos (por ejemplo, eventos binarios) por el procesador de vídeo y se envían al codificador para su codificación. Alternativamente, el procesador de vídeo puede operar directamente en muestras individuales, incluyendo la transformación y la cuantificación de las muestras, y convirtiendo el valor entero cuantificado particular para la muestra en una secuencia de eventos.

La Figura 2 es un diagrama de flujo de un proceso de codificación para generar un flujo de bits. El proceso se lleva a cabo por medio de una lógica de proceso que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos.

Haciendo referencia a la Figura 2, la lógica de proceso codifica los eventos en una secuencia de eventos para producir datos codificados (bloque de proceso 201). Los eventos pueden ser decisiones binarias. Los eventos pueden ser también del mismo segmento. En una realización, uno de los eventos indica la finalización de la codificación aritmética (por ejemplo, un extremo del segmento). A continuación, la lógica de proceso genera un flujo de bits con los datos codificados para todos los eventos seguido de bytes (o bits) de relleno (lógica de proceso 202). Los bytes (o bits) de relleno pueden colocarse en el flujo de bits codificados después de un indicador que indica la finalización de la codificación aritmética.

La Figura 3 ilustra un formato de datos de ejemplo 300 mediante el cual datos codificados pueden ser transmitidos en un sistema como por ejemplo el sistema en la Figura 1. El formato 300 incluye una cabecera 302, un código aritmético 304, uno o más bits de parada 306, cero, uno, o más bits de alineación 308 y cero, uno o más bytes de relleno 310. En una realización alternativa, se pueden utilizar cero, uno o más bits de relleno en lugar de bytes.

Tal como se ha señalado anteriormente, el sistema de la Figura 1 y el formato de datos de la Figura 3 se pueden utilizar para la codificación y la transmisión de información de vídeo, incluyendo los datos relacionados con una secuencia de imágenes. En una realización, una imagen se divide en uno o más sectores, en los que un segmento contiene uno o más bloques de macros que son matrices de 16x16 píxeles. Cada segmento puede codificarse independientemente de otros segmentos dentro de la imagen. Los datos de imagen se codifican en el formato ilustrado en la Figura 3.

En una realización, la cabecera 302 comienza en un límite de byte y contiene datos codificados que utilizan códigos de longitud fija o códigos de longitud variable (por ejemplo, codificación de Huffman). La cabecera 302 puede ser una cabecera de segmento. Como cabecera de segmento, la cabecera 302 puede ir precedida por un código de inicio (SC) y un indicador que identifica el tipo de datos de segmento que viene a continuación.

El código aritmético 304 es una secuencia de bits generados por un motor de codificación aritmética de un codificador como por ejemplo el codificador 102 (Figura 1). En una realización, la secuencia de bits se inicia en un límite de byte. Uno o más bits de parada 303 van a continuación del código aritmético 304. En una realización alternativa, el bit de parada 303 puede estar incluido en el código aritmético 304. Un número (0 a 7) de los bits de alineación posteriores 308 va a continuación de los bits de parada 306 y, en una realización, garantiza la alineación de bytes de los bytes de relleno 310. El número de bytes de relleno 310 anexos a los datos puede ser cero bytes, un byte o más de un byte, en función del número de bytes necesarios para mantener la relación entre el número de eventos que se están codificando, el número de bloques de datos de vídeo (por ejemplo, bloques de macros), y el tamaño de la secuencia de información que se está generando.

Finalización de Flujo de Código

En una realización, el codificador codifica un evento (por ejemplo, decisión) que indica la finalización de datos aritméticos codificados a un decodificador. Esta finalización de los datos aritméticos codificados puede indicarse cuando se ha llegado a un extremo del segmento. La finalización de los datos aritméticos codificados puede producirse también cuando los datos aritméticos codificados en un flujo de bits se detienen y van seguidos de datos codificados no aritméticos.

Haciendo de nuevo referencia a la Figura 3, en una realización, para cada bloque de macro en un segmento, el código aritmético 204 habitualmente contiene los siguientes datos: un modo de bloque de macro, vectores de movimiento y coeficientes de transformación opcionalmente, así como una etiqueta de extremo del segmento (end_of_slice_flag). La etiqueta de extremo del segmento (end_of_slice_flag) permite al decodificador 104 (Figura 1) determinar cuándo se ha decodificado el último bloque de macro en un segmento. Esta etiqueta se utiliza dado que el último bit del código aritmético puede contener datos que describen más de un bloque de macro.

Los beneficios de la codificación de la finalización de los datos aritméticos codificados puede explicarse mediante el examen de las implementaciones convencionales. En las implementaciones convencionales, la finalización de un codificador aritmético se realiza en general de acuerdo una de dos alternativas. En un primer enfoque, se transmite todo el registro L. En un segundo enfoque, se añade una compensación a los contenidos del registro L y sólo se transmiten los bits más significativos del registro L. La ventaja del segundo enfoque es que el decodificador lee exactamente el mismo número de bits que han sido generados por el codificador. Sin embargo, esto se produce a expensas de enviar bits adicionales. En el segundo enfoque, los bits se guardan, pero el decodificador lee más bits de los que fueron generados por el codificador. Esto se puede solucionar protegiendo el flujo de bits en el decodificador.

Un enfoque descrito en este documento ofrece lo mejor de ambos mundos: el decodificador lee el mismo número de bits que han sido generados por el codificador sin que el codificador tenga que enviar más bits de los que son necesarios. Esto es posible por el hecho de que un evento, la etiqueta de extremo del segmento (end_of_slice_flag), se codifica para señalar el extremo de un segmento. Dada una probabilidad bien definida asignada a este evento, un decodificador puede decodificarlo, pero a continuación puede pasar por alto la renormalización si el resultado del evento señala la finalización. Es decir, normalmente, durante la codificación, para cada símbolo que se codifica, el valor R se multiplica por la probabilidad de obtener un sub-intervalo. A partir de entonces, se lleva a cabo la renormalización para llevar el valor de R de nuevo a un intervalo de valores. La renormalización es bien conocida por los expertos en la técnica de la codificación aritmética. Renunciar a la renormalización asegura que el número de bits leídos coincide con el número de bits generados por el codificador.

En una realización, la probabilidad asignada a una etiqueta de extremo del segmento (end_of_slice_flag) (u otros eventos que indican la finalización de la codificación aritmética) está definida por un número asignado al registro R durante la finalización de la codificación, antes de que se lleve a cabo cualquier nueva normalización. En una realización, para asegurar que el codificador y el decodificador están sincronizados, para la etiqueta extremo del segmento, el cálculo del subintervalo no se lleva a cabo multiplicando el valor almacenado en R por la probabilidad. Por el contrario, al subintervalo se le asigna un valor fijo, o constante. En una realización, se utiliza un valor fijo de 2. Más en general, el valor debería ser independiente del valor de los contenidos de registro R antes de la codificación de la etiqueta de extremo del segmento (end_of_slice_flag). Esto se hace para el último símbolo (bit) que se coloca en el flujo de bits. Al establecer el sub-intervalo en un valor de 2, el valor de 1 se puede añadir al valor del registro L sin afectar al funcionamiento del decodificador. Esto permite que el contenido de todo el registro bajo (L) sea enviado al flujo de bits. Debido a que se envía el contenido de todo el registro L, en este caso no es necesaria la renormalización.

En una realización, el bit menos significativo del registro L se establece en 1 antes de enviar el contenido de L. Establecer el bit menos significativo del registro L en 1 es equivalente a la adición de 1 a L si su bit menos significativo es cero. Por lo tanto, el último bit generado por el codificador aritmético es igual a 1 y el último byte del flujo de bits que contiene el código aritmético tiene un valor distinto de cero. En efecto, el bit menos significativo del registro L se convierte en un bit de parada.

Adición de Bytes de Relleno

En una realización, el codificador inserta bytes, o bits, de relleno en un flujo de bits de datos comprimidos. En una realización, los bytes de relleno se insertan después del código aritmético para un segmento, después de un bit de parada y cero, uno o más bits de alineación. Los bits de alineación se añaden para asegurar que cualquier byte de relleno añadido sea insertado en los límites de bytes. Uno de los beneficios de la colocación de los bytes de relleno después del bit de parada es que un decodificador no tendrá que decodificar los bytes de relleno. Por lo tanto, el decodificador decodifica en el mismo número de bits que el número de bits de datos codificados generados por el codificador.

En una realización, el número de bytes de relleno insertados en el flujo de bits comprimido se basa en el mantenimiento de una relación entre el número de eventos que se introducen en el codificador, el número de bloques de datos, y el número de bits que salen del codificador. La relación se describe con más detalle a continuación.

En una realización, los bytes de relleno tienen un patrón específico. El patrón puede ser exclusivo de tal manera que un decodificador sea capaz de determinar que los bytes de relleno están presentes mediante la identificación de los bits con este patrón particular después de un bit de parada y uno o más bits de alineación. Una vez que se realiza dicha determinación, el decodificador no tiene que decodificar los bytes de relleno. En una realización, el decodificador incluye la funcionalidad de demultiplexación que impide que los bytes de relleno sean enviados a un motor de decodificación aritmética en el decodificador, de una manera similar a los bits de cabecera (que no se envían a un motor de decodificación).

En una realización, el patrón de los bits de relleno es la secuencia de tres bytes 000003 Hex, que se adjunta al flujo de bits. Los dos primeros bytes representan una palabra cero (0000) y el tercer byte (03) es reconocido por el decodificador después de que un extremo del segmento identifique los bytes como bytes de relleno.

En una realización, el número de bytes de relleno 310 rellenados al final de un segmento garantiza que la relación entre el número de operaciones de decodificación aritmética y el número de bits es menor o igual a cuatro. Un codificador, como por ejemplo el codificador 102 de la Figura 1, puede utilizar un registro C para contar o de algún otro modo seguir la información de la relación de eventos (operaciones de decodificación) por bits (o bytes). Cada vez que se procesa un evento, el contador C se incrementa en 1, y cada vez que se produce un bit, el contador C se reduce en 4 (o en 32 para cada byte producido). En una realización, el recuento tiene en cuenta todos los bits en el segmento (u otro conjunto de eventos), incluyendo bits de parada y de alineación, de cabecera y de cola.

Debe tenerse en cuenta que en una realización las operaciones de decodificación de la etiqueta de extremo del segmento (end_of_slice_flag), no se cuentan con el contador C (aunque en una implementación alternativa, se pueden calcular) . Se sabe, sin embargo, que existe un evento por cada bloque de macro y que el número de este tipo de eventos está bien delimitado por el tamaño de la imagen. En este caso, no contar los eventos de la etiqueta de extremo del segmento (end_of_slice_flag) es equivalente a contarlos (por lo tanto incrementando C en 1 una vez por cada bloque de macro), pero al mismo tiempo reduciendo C en 1 cada 256 píxeles (una vez por bloque de macro). Alternativamente, C podría reducirse en cualquier valor para cada bloque de macro.

En una realización, añadir bytes de relleno en la forma descrita en este documento garantiza una longitud mínima para el segmento codificado. En relación con la técnica convencional de insertar bits de relleno en el centro de un segmento codificado, este avance simplifica las reglas por las que el codificador codifica los datos, en particular, la definición de la cantidad de datos a codificar.

El codificador puede limitar el número de eventos de la secuencia de eventos como una función del número de bits de información en la secuencia de bits de información, y un número de segmentos, o bloques, de los datos de entrada representados en la secuencia de eventos. Por ejemplo, la restricción puede tomar la forma de una combinación lineal:

$$e \leq \alpha B + \beta S,$$

donde

e es el número de eventos representados en la secuencia de bits de información (u otros elementos),

B es un número de bits de información en la secuencia de bits de información (u otros elementos),

S es un número de segmentos (por ejemplo, bloques de macro) representados en la secuencia de eventos, y

α y β representan un valor de decremento en un contador para mantener sustancialmente una limitación del número de eventos de la secuencia de eventos con respecto a un número de bits de información generados y a un número de segmentos procesados.

Los valores de α y β se proporcionan habitualmente a un controlador para un codificador aritmético, y la derivación de α y β se describirá más adelante. El valor de α puede representar un valor de decremento en, por ejemplo, un contador tras la generación de un bit de información en el codificador, en el que el valor β puede representar el valor de decremento en, por ejemplo, un contador en el momento de la finalización del proceso de un bloque de datos. Como alternativa, el valor β puede ser restado a partir de un valor de contador al inicio del proceso de un segmento, o en cualquier otro momento durante el proceso de un bloque de datos, tal como resultaría evidente para un experto en la técnica.

Dado que el número total de bloques, S, y el valor β son conocidos, el producto de β x S puede ser restado del número de eventos, e, para la secuencia de eventos después del proceso de los bloques (por ejemplo, bloques de macro) de los datos de entrada. Por ejemplo, cuando se utiliza un contador para limitar el número de eventos en respuesta al número de bits que se han generado, el contador puede ser restado inicialmente en un valor de β x S, y puede ser restado por un valor α para cada bit de información generada, mientras que el contador se incrementa en "1" para cada evento de la secuencia de eventos procesados por el codificador de entropía.

El valor de β puede ser cualquier valor, por lo general en el intervalo de 1 a 100, y puede ser determinado, por ejemplo, tal como se describe más adelante. El valor de α puede ser cualquier valor, por lo general en el intervalo de 1 a 10, y puede ser determinado, por ejemplo, tal como se describe más adelante.

En algunas circunstancias, una serie de bloques de los datos de entrada que van a ser procesados no son conocidos de antemano, por ejemplo, cuando el medio de comunicación limita el número de bits de información que pueden ser proporcionados en la secuencia de información. Esto puede ocurrir, por ejemplo, cuando la secuencia de información ha de ser transmitida a través de Internet, como un paquete de Protocolo de Internet (IP), en el que el paquete de IP tiene una limitación de tamaño máximo. En estas circunstancias, dependiendo de la complejidad de una imagen en particular, pueden ser necesarias una o más secuencias de bits de información para representar una sola imagen de los datos de entrada. Sin embargo, el número de bloques utilizados para la generación de una secuencia de bits de información no se puede saber de antemano, ya que no se puede saber después de cuántos segmentos procesados se alcanza el tamaño máximo de una secuencia de bits de información. Cuando no se conoce de antemano un número de segmentos de los datos de entrada que se van a procesar, el controlador puede dar cuenta de las secuencias de eventos mientras se codifica el uno o más bloques que representan una secuencia particular de eventos.

Por ejemplo, cuando se utiliza un contador para limitar el número de eventos que responden a la cantidad de bits que se han generado, el contador puede ser restado en un valor β para cada bloque procesado, y puede ser restado en un valor α para cada bit de información generado, mientras que el contador puede ser incrementado en "1" para cada evento de la secuencia de eventos procesados por el codificador de entropía.

Los valores de α y β se podrán determinar de antemano, a través de un diseñador del sistema del codificador que representa una o más de las limitaciones mencionadas anteriormente, y podrán ser proporcionados al controlador. Alternativamente, o asimismo, los valores de a y β pueden ser determinados por el controlador, o cualquier otro componente del codificador, de acuerdo con una o más de las limitaciones mencionadas anteriormente, o como valores por defecto del codificador. Cuando el controlador determina los valores para α y β utilizando una o las dos limitaciones impuestas por el estándar, o por un dispositivo de decodificación, la información con respecto a una o más de las limitaciones se puede almacenar en una memoria del controlador (que no se muestra), y ser utilizada por el controlador en la determinación de los valores de α y β. Además, o como alternativa, se puede proporcionar información con respecto a las limitaciones al controlador, por ejemplo, por medio de algún dispositivo externo, como por ejemplo una memoria externa (es decir, un disco de vídeo digital (DVD)), un dispositivo reproductor de DVD, o por parte de un ingeniero de sistemas, por ejemplo, que manipule algunas de las funciones en lo que respecta a la codificación de los datos de entrada particulares. En este último caso, el ingeniero de sistemas puede introducir en una consola u otro dispositivo de entrada (que no se muestra), o de algún otro modo especificar, la información con respecto a las limitaciones impuestas como resultado de un estándar de codificación y / o un dispositivo de decodificación, tal como sería apreciado por un experto en la técnica.

Además, cuando se determinan los valores para α y β, se pueden hacer consideraciones en cuanto a si la restricción de la complejidad es demasiado ajustada, por ejemplo, si los valores de α y / o β son demasiado bajos. Una alta proporción de bits de información de relleno al final de la secuencia de bits de

información (es decir, un número de bytes (o bits) de relleno mayor que aproximadamente un 1% o un 2% de los bits de información de la secuencia de información) puede indicar que la restricción es demasiado ajustada. Un experto apreciaría que otras proporciones pueden indicar una alta proporción de bits de información de relleno, por ejemplo, teniendo en cuenta el estándar y / o el decodificador particular que puede ser utilizado.

Cuando se determina, por ejemplo, que los valores de α y β son demasiado ajustados, se puede aumentar los valores de α y β para reducir la probabilidad de que se añadan bytes de relleno (es decir, la reducción de la probabilidad de una penalización de la calidad en la secuencia de información codificada). Al aumentar los valores de α y β, se pueden hacer consideraciones en cuanto al efecto sobre los límites de complejidad resultantes con respecto a un decodificador que se utilizará para decodificar la secuencia de información codificada. Estas consideraciones pueden incluir el coste de implementar el decodificador. Si el límite de complejidad es mayor, puede ser necesaria más potencia de proceso en el decodificador. Un aumento de la capacidad de proceso requerida es probable que pueda tener como resultado un mayor coste de implementación. Debe tenerse en cuenta que en una realización, los cambios en a y β se pueden hacer después de la codificación de datos de cada bloque de macro.

Los valores de α y β pueden determinarse experimentalmente, utilizando técnicas de regresión lineal. Se puede codificar una serie de secuencias de eventos, cada uno de los cuales representa segmentos S, sin imponer ninguna restricción de complejidad. Para cada secuencia z de eventos, se conoce el número de bits de información generada resultante B(z) para el número de eventos e(z). Se puede determinar, mediante regresión lineal, una línea $e + c * B + d$ que se aproxima a los pares de datos (e(z), B(z)). Un valor inicial de α y / o β puede entonces ser incrementado, por ejemplo para reducir, y potencialmente reducir al mínimo, el número de pares de datos (e(z), B(z)) que se encuentran por encima de la línea $e = \alpha*B + \beta*S$.

Utilizando los valores de α y β tal como se determina a través de una o más de las diversas técnicas descritas anteriormente, el codificador puede dar cuenta de un valor de α (es decir, disminuir un contador en el valor de α) para cada bit de información generado, y puede dar cuenta de un valor de β (es decir, disminuir un contador por el valor de β) sobre la finalización de un segmento de los datos de entrada. Por ejemplo, cuando α y β son valores enteros, dicha cuenta (es decir, decrementos en uno o más contadores) puede llevarse a cabo directamente.

Cuando, por ejemplo, uno de ellos o tanto α como β son valores fraccionarios, se puede determinar un denominador común para proporcionar valores no fraccionarios para α y β. En esta circunstancia, los valores nuevos y no fraccionarios para α y β pueden ser contabilizados tal como se ha descrito anteriormente, por ejemplo, restando un contador por los valores de α y β a partir de la generación de bits de información y la finalización del proceso del segmento, respectivamente. El denominador común determinado puede explicarse, por ejemplo, mediante la adición del valor del denominador común para el valor del contador en el proceso de cada evento de la secuencia de eventos. Por ejemplo, cuando los valores de α y β se determina que son 4/3 y 25, respectivamente, se puede determinar un denominador común como 3. De este modo se puede determinar que los valores no fraccionarios de α y β son 4 y 75, respectivamente, utilizando el denominador común. Así pues, cuando se utiliza un contador para dar cuenta de los valores de α y β, el contador puede ser disminuido en 4 para cada bit de información generado, restado en 75 a partir de la finalización del proceso de cada segmento, e incrementado en 3 para cada evento procesado.

Ejemplo de Funcionamiento del Codificador

La Figura 4 ilustra un diagrama de bloques de una realización de un codificador aritmético. Haciendo referencia a la Figura 4, el codificador aritmético 400 incluye un secuenciador 405, un estimador de probabilidad 410, y un motor de codificación 415, que están acoplados entre sí. Una o más líneas de datos de entrada 420 proporcionan un puerto de entrada para recibir una secuencia de eventos 425 (por ejemplo, una secuencia ordenada de eventos binarios) en el codificador 400. La secuencia de eventos es procesada por el codificador 400, tal como se describe a continuación, para generar una secuencia de información. En una realización, la secuencia de información es una secuencia ordenada que consta de al menos un elemento de información (por ejemplo, un bit). En una realización, el número de bits de información en la secuencia de información es menor que el número de eventos en la secuencia de eventos. La salida 430 proporciona un puerto de salida para el envío de información de secuencia 435 desde el codificador 400. La secuencia ordenada de bits de la secuencia de información incluye uno o más bits que tienen un valor de "0" o "1".

Al recibir la secuencia de eventos 425, el secuenciador 405 transmite eventos 425 secuencialmente tanto al estimador de probabilidad 410 como al motor de codificación 415. Para cada evento binario de la secuencia de evento 425, el secuenciador 405 también transmite información de contexto al estimador de probabilidad 410 para el evento binario. El estimador de probabilidad 410, utilizando la información de contexto recibida, genera una estimación de probabilidad P(A) que se transmite al motor de codificación 415. En una realización, el estimador de probabilidad 410 envía múltiples estimaciones de probabilidad al

motor de codificación 415 y el motor de codificación 415 selecciona una de las estimaciones de probabilidad basándose en el valor R. Alternativamente, el valor R puede ser enviado al estimador de probabilidad 410, que la utiliza para seleccionar una estimación de probabilidad que va a ser enviada. El estimador de probabilidad 410 actualiza entonces su estado interno basándose en el valor del evento binario recibido. El motor de codificación 415 produce 0 o más bits de información utilizando el evento binario recibido y la correspondiente estimación de probabilidad P(A).

En una realización, el motor de codificación 415 codifica un evento que indica una finalización de los datos aritméticos codificados. El evento puede ser una etiqueta de fin de segmento u otro indicador de datos codificados no aritméticos que irá a continuación, en su caso, en el flujo de bits.

En la producción de los cero o más bits de información, el motor de codificación 415 utiliza diversos registros incluyendo un registro de intervalos 465, un registro bajo 470, un registro de bits pendientes 475, y un registro contador 480. El funcionamiento del codificador 400 en la realización de la codificación aritmética es bien conocido en la técnica.

En una realización, el codificador 400 limita una relación de los eventos por los bits de información, que se describe en este documento. El codificador 400 realiza esta operación, en parte, mediante la inserción de bytes (o bits) de relleno en la secuencia de información, tal como se describe en el presente documento.

La Figura 5 es un diagrama de flujo de una realización para la codificación de un evento. El proceso se lleva a cabo a través de lógica de proceso, que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos. Las entradas en el proceso de codificación aritmética son los eventos binarios que son decodificados con la ID de contexto que identifica el contexto, y el valor R, L y symCnt, y escritos en las salidas se encuentran los bits que resultan de la codificación. En una realización, la codificación es simétrica con la decodificación y, el estado del motor de codificación aritmética, tal como se ha descrito anteriormente, está representado por el valor del valor de L que señala hacia el extremo inferior del sub-intervalo y el valor de la R que especifica el rango correspondiente del sub-intervalo.

En una realización, el proceso de codificación se invoca solamente después de que se inicialice el motor de codificación. En una realización, la inicialización se lleva a cabo mediante el envío del valor de L igual a cero y el valor de R igual a 0x01FE, estableciendo una primera etiqueta de bits en uno, y los contadores de valor de bits restantes (BO) y de symCnt (C) iguales a cero. La primera etiqueta de bits se utiliza durante la codificación para indicar cuando el codificador está pasando por el procedimiento de introducción de bit por primera vez. El contador de symCnt almacena un valor que indica el número de eventos que están siendo codificados.

Haciendo referencia a la Figura 5, el proceso comienza codificando un solo evento (por ejemplo, un bit) derivando el valor R_{LPS} tal como sigue (bloque de proceso 501). En una realización, la lógica de proceso deriva la variable R_{LPS} estableciendo el índice R (o R_{idx}) igual al valor de R desplazado seis posiciones a la derecha y añadiendo AND con el número 3 Hex. A continuación, la lógica de proceso establece el valor R_{LPS} igual a un valor determinado accediendo a una tabla de la máquina de la estación de estimación de probabilidad, como por ejemplo una tabla que se muestra en la Figura 16A utilizando el valor R_{idx} y el valor del estado para el contexto actual asociado con el contexto. El valor de R se sitúa entonces en el valor actual R menos R_{LPS}.

Después de calcular el intervalo de sub-rango para el recuento de los MPS, la lógica de proceso prueba si el valor del evento binario codificado no es igual al valor de los MPS (bloque de proceso 502). Si el valor del evento binario es igual al MPS, entonces la lógica de proceso toma la vía de los MPS y pasa el bloque de proceso 503, donde la lógica de proceso actualiza la máquina de estado al siguiente estado indicado en la máquina de estado para el contexto utilizando la tabla de la Figura 16B y el proceso pasa hacia el bloque de proceso 508. Si la lógica de proceso determina que el evento binario que está siendo codificado no es igual al valor de los MPS, entonces la lógica de proceso toma la vía de los LPS y pasa al bloque de proceso 504, donde la lógica de proceso establece el valor de L igual a el valor de L, más el valor de R y define el valor de R igual al valor de R_{LPS}.

A continuación, la lógica de proceso determina si el estado para el contexto particular no es igual a cero (bloque de proceso 505). En una realización, el estado cero es un estado correspondiente a una probabilidad de 50/50. Alternativamente, el estado cero es un estado que corresponde a otra probabilidad tal como, por ejemplo, alrededor de una probabilidad de 50/50. Si el estado para el contexto no es igual a cero, la lógica de proceso pasa al bloque de proceso 507. Si el estado para el contexto es igual a cero, la lógica de proceso cambia el significado de los MPS (bloque de proceso 506) y pasa el proceso al bloque 507, y la lógica de proceso actualiza el número de estado del contexto al siguiente estado utilizando la tabla de la Figura 16B (bloque de proceso 507).

Después de realizar los bloques de proceso 507 y 503, el proceso pasa el bloque de proceso 508, donde la lógica de proceso realiza el procedimiento de renormalización, como por ejemplo la renormalización en

la Figura 6. A continuación, la lógica de proceso incrementa el valor del contador de eventos en 1 (bloque de proceso 509) y finaliza el proceso.

La Figura 6 es un diagrama de flujo de una realización de un procedimiento de renormalización del codificador. El proceso se lleva a cabo mediante el la lógica de proceso, que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos.

Haciendo referencia a la Figura 6, la lógica de proceso prueba si el valor de R es de menos de 100 Hex (bloque de proceso 601). En caso contrario, se realiza el proceso. En caso afirmativo, el proceso pasa al bloque de proceso 602, donde la lógica de proceso prueba si el valor de L es de menos de 100 Hex. En ese caso, el bloque de proceso pasa al bloque de proceso 603, donde se realiza un procedimiento de introducción de bit con parámetro 0 y a continuación el proceso pasa al bloque de proceso 608. Si la lógica de proceso determina que el valor de L es mayor que o igual a 100 Hex, la lógica de proceso prueba si el valor de L es mayor que 200 Hex. En caso contrario, la lógica de proceso establece el valor de L en el resultado de restar 100 Hex del valor de L e incrementa el valor de los bits pendientes (BO) por uno con el parámetro 1 (bloque de proceso 605) y el proceso pasa al bloque de proceso 608. Si el valor de L es mayor que o igual a 200 Hex, el proceso pasa al bloque de proceso 606, donde la lógica de proceso establece el valor de L en el resultado de restar 200 Hex del valor L, realiza el procedimiento de introducción de bit (bloque de proceso 607) y pasa al bloque de proceso 608.

En el bloque de proceso 608, la lógica de proceso desplaza el valor de R a la izquierda en una posición y desplaza el valor de L en una posición. A continuación el proceso pasa al bloque de proceso 601 y el proceso se repite.

La Figura 7 ilustra una realización del proceso para la realización de una forma de realización del procedimiento de introducción de bit. El procedimiento de introducción de bit escribe cero o más bits en el flujo de bits. El proceso se lleva a cabo mediante lógica de proceso, que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos.

Haciendo referencia a la Figura 7, la lógica de proceso inicialmente comprueba si la primera etiqueta de bits no es igual a cero (bloque de proceso 701). Si la primera etiqueta de bit se establece en 1, entonces la lógica de proceso establece el primer indicador de bits igual a cero (bloque de proceso 702) y el proceso pasa al bloque de proceso 704. Si no es así, la lógica de proceso envía un bit con valor de B (bloque de proceso 703) y la lógica de proceso pasa al bloque de proceso 704).

En el bloque de proceso 704, la lógica de proceso prueba si el valor de los bits restantes (BO) es mayor que cero. Si no es así, el proceso termina. En ese caso, la lógica de proceso envía un bit con valor 1-B y disminuye el valor de BO en uno (bloque de proceso 705). A continuación la lógica de proceso pasa al bloque de proceso 704.

La Figura 8 es un diagrama de flujo de una realización de un proceso para la codificación de un evento antes de la finalización. Este proceso se puede utilizar para codificar el extremo del segmento, así como cualquier otro evento binario que indica la finalización de la codificación aritmética. El proceso se lleva a cabo por medio de lógica de proceso, que puede comprender hardware (por ejemplo, Circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos.

Haciendo referencia a la Figura 8, la lógica de proceso disminuye inicialmente el valor de R en 2 (bloque de proceso 801). A continuación, la lógica de proceso prueba si el valor del evento binario codificado no es igual a cero (bloque de proceso 802). Si el evento es igual a cero, la lógica de proceso lleva a cabo el procedimiento de renormalización tal como se muestra en la Figura 6 (bloque de proceso 803), y el proceso pasa al bloque de proceso 806. Si el valor del evento binario que va a ser codificado no es igual a cero, entonces la lógica de proceso establece el valor de L en el resultado de sumar el valor de L, más el valor de R (bloque de proceso 804), lleva a cabo un procedimiento de limpieza de codificador (bloque de proceso 805), y pasa al bloque de proceso 806. En el bloque de proceso 806, la lógica de proceso incrementa el valor del contador de eventos en 1 y finaliza el proceso de codificación.

Tal como se aprecia en el proceso anterior, en una realización, cuando el valor del evento binario es igual a 1, se finaliza la codificación aritmética y se aplica el procedimiento de limpieza después de la codificación del evento. Cuando se codifica este tipo de evento, el último bit escrito contiene un bit de parada igual a 1.

La Figura 9 ilustra un diagrama de flujo de una realización de un proceso para la limpieza en la finalización. El proceso se lleva a cabo mediante la lógica de proceso, que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o en una máquina dedicada), o una combinación de ambos.

Haciendo referencia a la Figura 9, la lógica de proceso establece inicialmente el valor de R en 2 (bloque de proceso 901). A continuación, la lógica de proceso realiza un procedimiento de renormalización como por ejemplo el procedimiento de renormalización que se muestra en la Figura 6 (bloque de proceso 902). La lógica de proceso realiza a continuación el procedimiento de introducción de bit que se muestra en la Figura 7 en un valor igual al valor de L desplazado nueve posiciones a la derecha y añadiendo AND con el valor de 1 Hex (bloque de proceso 903). Los resultados de la realización de la operación de añadir AND en el contenido desplazado del valor del registro L provocan que se genere el bit en la 10a posición (tal como se cuenta a partir del bit reciente significativo) y posteriormente se envía mediante el procedimiento de introducción de bit.

Por último, la lógica de proceso envía dos bits iguales al valor del registro L desplazado siete lugares a la derecha, añadiendo AND con un valor de 3 Hex, y a continuación se le añade OR con 1 Hex (bloque de proceso 904). La operación de añadir OR con 1 Hex se realiza para añadir el bit de parada.

Ejemplo de Funcionamiento del Decodificador

La Figura 10 es un diagrama de bloques de una realización de un decodificador aritmético 1000. Haciendo referencia a la Figura 10, el decodificador 1000 incluye un secuenciador 1005, un estimador de probabilidad 1010, y un motor de decodificación 1015 acoplados entre sí. Una entrada 1020 proporciona un puerto para una secuencia de información 1025 (por ejemplo, una secuencia ordenada de bits binarios) al decodificador 1000. Los bits binarios de la secuencia 1025 pueden tener un valor de "0" o "1". En una forma de realización, el decodificador 1000 procesa la secuencia de información para generar una secuencia de eventos 1035. La secuencia de eventos generada es una secuencia de eventos ordenada que comprende múltiples eventos (por ejemplo, eventos binarios), que pueden tener valores distintos de valores de bits individuales. La secuencia de eventos se proporciona a la salida 1030, que incluye al menos un puerto de salida desde el decodificador 1000.

Al recibir la secuencia de información 1025, el secuenciador 1005 transmite los uno o más bits al motor de decodificación 1015. El decodificador 1000 genera iterativamente los uno o más eventos de la secuencia de eventos tal como sigue. Para cada evento, el secuenciador 1005 transmite un contexto correspondiente al estimador de probabilidad 1010.

Basándose en el valor de contexto recibido, el estimador de probabilidad 1010 genera una estimación de probabilidad P (A) correspondiente, que se envía al motor de decodificación 1015, y es utilizado por parte del motor de decodificación 1015 en la generación del evento. En una realización, el estimador de probabilidad 1010 envía múltiples estimaciones de probabilidad al motor de decodificación 1015 y el motor de decodificación 1015 selecciona una de las estimaciones de probabilidad basándose en el valor R. Alternativamente, el valor R puede ser enviado al estimador de probabilidad 1010, que lo utiliza para seleccionar una estimación de probabilidad que va a ser enviada. A continuación, el estimador de probabilidad 1010 actualiza su estado interno basándose en el valor del evento binario recibido del motor de decodificación 1015.

El motor de decodificación 1015 envía cada evento binario generado al estimador de probabilidad 1010 y al secuenciador 1005. El motor de decodificación 1015 consume cero o más bits de información para cada evento binario generado. El secuenciador 1005 puede por lo tanto transmitir cero o más bits de la secuencia de información al motor de decodificación 1015 después de la generación de un evento. El motor de decodificación 1015 utiliza diversos registros en la generación de los eventos de la secuencia de eventos 1035, incluyendo un registro de intervalo 1065 y un registro de valor 1070. El funcionamiento del decodificador 1000 se muestra en el diagrama de flujo que se describe a continuación.

Los siguientes diagramas de flujo describen las operaciones de decodificación realizadas en un segmento por parte de una realización de un decodificador, como por ejemplo el decodificador 1000. En una realización, el decodificador realiza la decodificación de acuerdo con los diagramas de flujo representados en las Figuras 12,14A, 14B, 15A o 15B en función del valor de un contexto. Los procesos ilustrados se pueden incorporar en otros procesos, modificados o adaptados de otro modo para obtener los beneficios de las mejoras contenidos en los mismos. En una realización, el decodificador lee un byte cada vez. En una realización alternativa, el decodificador lee un bit cada vez.

La Figura 11 es un diagrama de flujo de una realización de un proceso de inicialización del decodificador aritmético. El proceso se lleva a cabo mediante lógica de proceso que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o en una máquina dedicada), o una combinación de ambos.

Haciendo referencia a la Figura 11, el proceso comienza con la lógica de proceso que establece el intervalo R en un número predeterminado (bloque de proceso 1101). En una realización, el número predeterminado es 0xff00. Después de inicializar el intervalo R, la lógica de proceso lee dos bytes de datos comprimidos en el registro V (bloque de proceso 1102). En una realización, el registro V almacena los bits comprimidos en forma de un byte cada vez. El registro V puede ser implementado para almacenar

los datos comprimidos en forma de un bit cada vez, pero las constantes utilizadas en el proceso descrito en el presente documento tendrían que ser cambiadas en consecuencia.

Más específicamente, tal como se muestra, la lógica de proceso lee en un byte y lo desplaza 8 posiciones a la izquierda y a continuación, obtiene otro byte y lo añade en el registro V con una operación aritmética de añadir OR. Una vez que los datos comprimidos se han leído en el registro V, la lógica de proceso establece el valor del registro B en un valor predeterminado. El registro B indica el número de bits adicionales en el registro V que están disponibles para su proceso. Cuando el valor del registro B es menor que 0, debe obtener otro byte de datos comprimidos. En una realización, el valor predeterminado es 7.

La Figura 12 es un diagrama de flujo de una realización de un proceso para descifrar un evento binario. El proceso se lleva a cabo por medio de la lógica de proceso que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la Figura 12, el proceso comienza calculando el tamaño del intervalo para el LPS (bloque de proceso 1202). En una realización, este cálculo se realiza por medio de una multiplicación. La multiplicación puede ser aproximada mediante el uso de una tabla de consulta que se basa en el estado asociado con el contexto (CTX). En una realización, una máquina de estados finitos se utiliza para indicar qué probabilidad existe en función del estado de la máquina. A continuación busca el valor del estado y los siguientes dos bits más significativos de R después del bit más significativo de R. Un ejemplo de tabla para realizar la búsqueda se muestra en la Figura 16A. A continuación también se proporciona un método ejemplar para la generación de la tabla.

El resultado de la tabla de consulta es desplazado 7 posiciones, ya que esta aplicación lee bytes de una vez en lugar de bits. El resultado desplazado de la búsqueda en la tabla es el rango de sub-intervalo de los LPS que se refiere como R_{LPS}.

También como parte del bloque de proceso 1202, la lógica de proceso calcula el rango de sub-intervalo para los MPS restando el rango de sub-intervalo de la LPS R_{LPS} del valor del registro R. La lógica de proceso establece el valor de R igual al resultado de la sustracción.

Después de calcular el intervalo de sub-rango para el MPS, la lógica de proceso prueba si el valor del registro V es mayor que o igual al sub-intervalo de MPS almacenado en el registro R, lo que indica que el bit actual que se está procesando se encuentra en el sub-intervalo de LPS (bloque de proceso 1203). En caso contrario, la lógica de proceso toma la vía MPS y pasa al bloque de proceso 1204 donde la lógica de proceso establece el valor que está siendo decodificado (es decir, el resultado que se retorna) S igual al valor que se define como el MPS para ese contexto determinado y actualiza la máquina de estado para el contexto al siguiente estado indicado en la máquina de estado para el contexto utilizando la tabla de la Figura 16B. En una realización, para un MPS, la actualización de la máquina de estado comprende incrementar el estado de la tabla de estado en uno.

Si la lógica de proceso determina que el valor V es mayor que o igual que el valor en el registro R, entonces la lógica de proceso toma la vía de LPS y pasa al bloque de proceso 1205 en el que el resultado S se establece igual al LPS (no MPS) para el CTX de contexto particular, el valor de V se establece igual al resultado de restar el valor del intervalo R del valor actual de V, y el intervalo R se establece igual al intervalo para el LPS, es decir, R_{LPS} (bloque de proceso 1205).

La lógica de proceso también comprueba si el estado para el contexto del evento binario es cero (bloque de proceso 1206). En una realización, el estado 0 es el estado correspondiente a una probabilidad de 50/50. Alternativamente, el estado cero es un estado que corresponde a otra probabilidad tal como, por ejemplo, alrededor de una probabilidad de 50/50. Si no es así, entonces el proceso pasa al bloque de proceso 1208. En caso afirmativo, la lógica de proceso conmuta el sentido de la MPS (bloque de proceso 1207).

A continuación, el número de estado del contexto se actualiza al estado siguiente utilizando la tabla en la Figura 16B (bloque de proceso 1208) y la lógica de proceso realiza un procedimiento de renormalización (bloque de proceso 1209), que se describe con mayor detalle a continuación.

La Figura 13 es un diagrama de flujo de un procedimiento de renormalización. El proceso se lleva a cabo por medio de la lógica de proceso que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos.

Haciendo referencia a la Figura 13, el proceso se inicia por medio de la lógica de proceso, que prueba si R es inferior a 8000 Hex (bloque de proceso 1301). Si R es mayor o igual a 8000 Hex, el proceso de renormalización termina. En caso contrario, la lógica de proceso duplica los valores de R y V (bloque de proceso 1302). En una realización, la lógica de proceso duplica los valores de R y V por medio del

desplazamiento de los bits de R y V una posición a la izquierda. El valor de B también se reduce en 1, ya que el desplazamiento ha causado que esté disponible un bit menos para su proceso. La lógica de proceso comprueba entonces si el valor de B es inferior a 0 (bloque de proceso 1303). En caso contrario, el proceso pasa al bloque de proceso 1301 y el proceso se repite. Si el valor de B es menor que 0, el proceso pasa al bloque de proceso 1304 en el que el valor de B se establece en 7 y el otro byte que se va a procesar se recupera y se le añade OR de manera lógica con el contenido actual del registro V. A continuación, el proceso pasa al bloque de proceso 1301 y el proceso se repite.

Las Figuras 14A y 14B ilustran diagramas de flujo para la decodificación de un evento con equi-probabilidad. La Figura 14A se puede utilizar cuando el tamaño del registro V es mayor de 16 bits, mientras que la Figura 14B se puede utilizar cuando el tamaño del registro V es de 16 bits. Estas implementaciones se pueden utilizar al recuperar un byte a la vez.

Los procesos se llevan a cabo mediante la lógica de proceso que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos.

Cuando las distribuciones se centran alrededor de cero y la probabilidad de obtener un valor positivo o un valor negativo es más o menos la misma, se pueden utilizar estos procesos. Por ejemplo, se pueden utilizar cuando se procesa un valor de signo de los coeficientes. En lugar de la estimación de probabilidad de que sea positivo o negativo, se utilizan estimaciones fijas que reconocen que la probabilidad es de 50/50. Por lo tanto, se es necesario realizar una búsqueda en la tabla para la multiplicación de R con una probabilidad. Debe tenerse en cuenta que ello no afecta a la finalización.

Haciendo referencia a la Figura 14A, el proceso se inicia mediante la lógica de proceso duplicando el valor de V y restando el valor de B en 1 (lógica de proceso 1401). La duplicación del valor de V puede llevarse a cabo desplazando los bits de V una posición a la izquierda.

A continuación, la lógica de proceso comprueba si el valor de B es inferior a 0 (bloque de proceso 1402). En caso contrario, el proceso pasa al bloque de proceso 1404. Si el valor de B es menor que 0, entonces el proceso pasa al bloque de proceso 1403 en el que el valor de B se establece en 7 y el otro byte que va a ser procesado se recupera y se le añade OR de forma lógica con el contenido actual del registro

En el bloque de proceso 1404, la lógica de proceso prueba si el valor de V es mayor que o igual que el valor de R. En ese caso, la lógica de proceso establece el resultado de S en 1 y establece el valor de V para el resultado de restar el valor R del valor V (bloque de proceso 1405), y el proceso finaliza. En caso contrario, la lógica de proceso establece el resultado S en 0 (bloque de proceso 1406) y el proceso finaliza.

Haciendo referencia a la Figura 14B, el proceso comienza mediante la lógica de proceso que establece el valor V' igual a V, duplicando el valor de V, y restando el valor de B en 1 (lógica de proceso 1411). La duplicación del valor de V puede llevarse a cabo desplazando los bits de V una posición a la izquierda.

A continuación, la lógica de proceso comprueba si el valor de B es inferior a 0 (bloque de proceso 1412). En caso contrario, el proceso pasa al bloque de proceso 1414. Si el valor de B es menor que 0, entonces el proceso pasa al bloque de proceso 1413 en el que el valor de B se establece en 7 y se recupera el otro byte que se va a procesar y se añade OR de forma lógica con el contenido actual del registro V.

En el bloque de proceso 1414, la lógica de proceso prueba si el valor de V es mayor que o igual que el valor de R o V' es mayor que o igual a 8000 Hex. En ese caso, la lógica de proceso establece el resultado S en 1 y establece el valor de V en el resultado de restar el valor R del valor V (bloque de proceso 915), y el proceso finaliza. En caso contrario, la lógica de proceso establece el resultado de S en 0 (bloque de proceso 916) y el proceso finaliza.

La Figura 15A es un diagrama de flujo de una realización para la decodificación de eventos codificados que indican la finalización de la codificación aritmética. Dicho evento puede comprender una etiqueta de extremo de segmento. Con respecto al extremo de la etiqueta del segmento, se puede utilizar la sintaxis para indicar a un decodificador la presencia de un extremo de etiqueta del segmento. En una realización, este proceso se realiza para cada bloque de macro; sin embargo, el resultado va a indicar un final de un segmento sólo para el último bloque de macro en el segmento (por ejemplo, la producción de un resultado que es 1).

Se puede utilizar un evento para señalar la finalización de la codificación aritmética (para un decodificador) cuando los datos van a ir a continuación de la codificación aritmética en el flujo de bits que está sin comprimir o comprimido dentro de otra técnica de codificación distinta de la codificación aritmética. Debe tenerse en cuenta que los datos adicionales aritméticos codificados pueden ir a continuación de los siguientes datos no comprimidos o datos comprimidos con una técnica de codificación no aritmética. De este modo, se puede utilizar el evento de señal de finalización en los casos en que datos aritméticos no codificados se intercalen en un flujo de bits con datos aritméticos codificados.

El proceso se lleva a cabo mediante la lógica de proceso que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos.

Haciendo referencia a la Figura 15A, la lógica de proceso prueba si el valor de V es inferior a 100Hex (bloque de proceso 1501), lo que indica que se ha alcanzado el último bloque de macro en el segmento. En ese caso, la lógica de proceso establece el resultado S, que representa el símbolo decodificado, en 1 (bloque de proceso 1502) y el proceso de decodificación para los extremos del segmento. En caso contrario, la lógica de proceso establece el resultado de salida de S en 0, establece el valor de R en el resultado de restar 100 Hex del valor de R, y establece el valor de V en el resultado de restar 100 Hex del valor de V (lógica de proceso 1503). A continuación, la lógica de proceso realiza el procedimiento de renormalización de la Figura 3 (bloque de proceso 1504) y el proceso termina.

Debe tenerse en cuenta que en una realización, la convención entre el MPS y LPS puede ser intercambiada. La Figura 15B es un diagrama de flujo de una realización de un proceso para la codificación de un evento antes de la finalización cuando se cambia una convención entre el MPS y el LPS. El proceso puede ser realizado por la lógica de proceso, que puede comprender hardware (por ejemplo, circuitos, lógica dedicada, etc.), software (tal como se ejecuta en un sistema de ordenador de uso general o una máquina dedicada), o una combinación de ambos.

Haciendo referencia a la Figura 15B, la lógica de proceso comienza restando 100 Hex del valor de R (bloque de proceso 1511). A continuación, la lógica de proceso comprueba si el valor de V es mayor que o igual que el valor de R (bloque de proceso 1512). En ese caso, la lógica de proceso establece el resultado de la salida S, que representa el símbolo decodificado en uno (bloque de proceso 1513) y el proceso de decodificación para decodificar el evento antes de la finalización termina. Por lo tanto, no se realiza ninguna nueva normalización. En caso contrario, la lógica de proceso establece el resultado de los datos de salida S en cero (bloque de proceso 1514) y lleva a cabo el procedimiento de renormalización de la Figura 13 (bloque de proceso 1515) y finaliza el proceso.

Construcción de Máquina de Estado para la Estimación de Probabilidad

Un ejemplo de proceso para la construcción de la máquina de estado en las Figuras 16A y 16B se expresa en código C más abajo.

```
C-code:
#define N 64
#define Pmax 0.5
#define Pmin 0.01875
#define regsize 9
#define ONE (1<<regsize)
double alpha;
double sum;
int i,j;
double q;
float prob64[N];
int next_state_MPS_64[N];
int next_state_LPS_64[N];
int switch_MPS_64[N];
int qLPS[N][4];
alpha = pow(Pmin/Pmax,1.0/(N-1));
sum = 0.5;
for (i=0; i<N; i++) {
  prob64[i] = Pmax*pow(alpha,i);
  next_state_MPS_64[i] = (i==N-1)?N-1:i+1;
  q = prob64[i]*alpha+(1-alpha);
  q = q/prob64[i];
  q = -log(q)/log(alpha);
  sum += q;
  k = (int)(sum);
  sum -= k;
  next_state_LPS_64[i] = (i-k<0)?0:i-k;
  for (j=0; j<4; j++) {
    RTAB[i][j] =
(int)(ONE/8*prob64[i]/log((j+5.0)/(j+4.0))+0.
5);
    if (j == 0 && RTAB[i][j] > ONE/4)
     RTAB[i][j] = ONE/4;
   }
 }
```

En el código anterior, N define el número de estados en una máquina de estados. En una realización, la máquina de estado es simétrica y el número total de estados es 2*N (128 en este ejemplo). Un estado puede estar representado por dos variables: estado (un número entre 0 y N-1, ambos inclusive) y una etiqueta de MPS (determina si 0 o 1 es el MPS).

En una realización, los estados están organizados de tal manera que los números superiores del estado corresponden a las probabilidades más bajas para el LPS. La máquina de estado se define de manera que se aproxime al siguiente procedimiento:

(a) p(LPS) <-P(LPS) * alfa, si se observa un MPS

(b) p(LPS) <-P(LPS) * alfa + (1-alfa), en caso contrario,

donde alfa define un coeficiente de adaptación. Alfa está habitualmente en el intervalo de 0,9 a 1, pero puede extenderse a o encontrarse en otros intervalos basados en la adaptación deseada.

En el código anterior, alfa se establece igual a

pow (0.01875/0.5,1.0/63) donde 0.01875 (Pmin) define la probabilidad de un LPS para el estado N-1, 0,5 (Pmax) define la probabilidad de un LPS para el estado 0, y 1,0 / 63 es 1 sobre N-1. Debe tenerse en cuenta que pow (a, b) es el número a para una potencia b.

La matriz denominada prob64 contiene valores de coma flotante que representan probabilidades de un LPS asociado con cada estado. Prob64 [i] está situado en Pmax*pow (alpha, i). Prob64 [0] es igual a Pmax y Prob64 [N-1] es igual a Pmin.

Next_state_MPS_64 [i], define la transición de estado en la observación de un MPS. Si i es diferente de N-1, el estado se incrementa en 1. De lo contrario, el estado se mantiene sin cambios. Dada la combinación de Prob64 [i] y Next_state_MPS_64 [i], la parte (a) del procedimiento de actualización que se ha definido anteriormente tiene una aproximación óptima.

Para aproximarse a la parte (b) del procedimiento de actualización, Next_state_MPS_64 [i] se debería establecer en i-(-log((prob64[i]*alpha + (1 - alpha))/prob64 [i])/log(alfa)). Este valor, sin embargo, no es un número entero y deben buscarse una aproximación de número entero. En una realización, el valor se redondea al número entero más próximo. Sin embargo, en una realización alternativa, para un mejor equilibrio entre redondeo al alza y redondeo a la baja, se utiliza una suma variable de tal manera que, como promedio, la diferencia introducida por el redondeo es próxima a cero.

El valor de RTAB[i][j] se calcula de manera tal para aproximar R*prob64[i]. La variable j se determina por el intervalo en el que se encuentra R. La variable j se establece igual a 0 para R en [256, 319], 1 para [320, 383], 2 para [384, 447], y 3 para [448, 511], donde, por ejemplo, (ONE*4)/8 es igual a 256, (ONE*5) / 8-1 es igual a 319, etc. El cálculo de (ONE/8)/log ((j + 5)/(j + 4)) representa el valor esperado de R dado el valor de j.

Para habilitar implementaciones más rápidas, es deseable garantizar que en la codificación de un MPS, se produzca a lo sumo una iteración de renormalización. Con este fin, RTAB [i][0] se asocia a ONE / 4. Por lo tanto R no puede ser menor que ONE / 4 antes de renormalización. De manera más general, en una realización, RTAB[i][j] se asocia a (ONE/4) + (ONE/8)*j pero este caso no se produce para un valor j distinto de 0 en el presente ejemplo.

Por lo tanto, utilizando la técnica descrita anteriormente, se puede generar la tabla de estado de las Figuras 16A y 16B, con la excepción de un estado en una realización. En la Figura 16A, el estado 63 incluye los valores R de 2,2, 2,2. En la Figura 16B, una vez en el estado 63, el siguiente estado es el estado 63. Por lo tanto, independientemente de si se produce un LPS o de si se produce un MPS, el estado no cambia. También en la Figura 16B, una vez en el estado 62, el estado permanece en el estado 62 después de la aparición de un MPS.

Ejemplos de Realización en Código Fuente

A continuación se proporcionan codificadores de ejemplo y un decodificador de ejemplo en código C. Estos métodos pueden implementarse utilizando cualquier dispositivo de proceso adecuado para la codificación y la decodificación de los datos (por ejemplo, datos de vídeo). En algunas realizaciones, el proceso puede ser realizado por una combinación de elementos de hardware y software. Pueden llevarse a cabo otras adaptaciones. Las Funciones para la codificación y decodificación se describen a continuación en formato C.

```
Encoder:
void start_encode() {
   encode_slice_header();
   while (!byte_aligned)
     send_bit(0);
   R = 0x1fe;
   L = 0;
   BO = 0;
   FB = 1;
 }

void finish_encode() {
   R = 2;
   renorm_encode();
   bit_plus_follow((L >> 9) & 1);
   send_bit((L >> 8) & 1);
   send_bit(1); / / stop_bit
   while (!byte_aligned())
      send_bit(0); / / alignment_bit
 }
 void bit_plus_follow(int b) {
   if (FB == 1)
      FB = 0;
   else
      send_bit(b);
   while (BO > 0) {
      BO--;
      send_bit(!b) ;
    }
 }
 void encode_renorm() {
   while (!(R&0x100) {
      if (L+R < 0x200)
        bit_plus_follow(0);
      else if (L >= 0x200) {
        bit_plus_follow(1);
        L -= 0x200;
      }
      else {
        BO++;
        L -= 0x100;
      }
      R <<= 1;
      L <<= 1;
    }
 }
 void encode_event(int ctx, int b) {
    rLPS = table[state[ctx]][(R>>6)-4];
    R -= rLPS;
    if (b == MPS[state[ctx]])
  state[ctx] = next_state_MPS[state[ctx]];
   else {
       L += R;
      R = rLPS;
      if (state[ctx] == 0)
      MPS[state[ctx]] = !MPS[state[ctx]];
         state[ctx] =
next_state_LPS[state[ctx]];
```

```
   }
   encode_renorm();
 }
void encode_equiprob_event(int b) {
   L <<= 1;
   if (b)
      L += R;
   if (L+R < 0x400)
      bit_plus_follow(0);
   else if (L >= 0x400) {
      bit_plus_follow(1);
      L -= 0x400;
   }
   else {
      BO++;
      L -= 0x200;
   }
 }
void encode_end_of_slice_flag(int b
   if (b == 0) {
      R-=2;
      L+=2;
      encode_renorm();
   }
 }
 Decoder (byte based):
 void start_decode() {
   decode_slice_header();
   while (!byte_aligned())
      get_bit();
   R = 0xff80;
   V = get_byte() << 8;
   V |= get_byte();
   B = 7;
 }

 void finish_decode() {
    while (more_bytes_in_slice())
      get_byte(); // stuffing byte
 }
 void decode_renorm() {
    while (R<0x8000) {
      R << = 1;
      V <<= 1;
      B--;
      if (B<0) {
          B = 7;
         V |= get_byte();
      }
    }
 }
 int decode_equiprob() {
   V = (V<<1) ;
    B--;
    if (B<0) {
      V | = get_byte();
      B = 7;
    }
    if (V >= R) {
      V -= R;
      bit = 1;
    }
    else
```

```
        bit = 0;
      return bit;
    }
    int decode_event(int ctx) {
     rLPS = table[state[ctx]][(R>>13)-4]<<7,
      R -= rLPS;
       if (V < R) {
    state[ctx] = next_state_MPS[state[ctx]],
        bit = MPS[state[ctx]];
      }
      else {
        bit = !MPS[state[ctx]];
        V -= R;
        R = rLPS;
        if (state[ctx] == 0)
        MPS[state[ctx]] = !MPS[state[ctx]],
    state[ctx] = next_state_LPS[state[ctx]],
      }
      decode_renorm();
      return bit;
    }
    int decode_end_of_slice_flag() {
      if (V < 0x100)
        bit = 1;
      else {
        bit = 0;
        R-=0x100;
        V-=0x100;
        decode_renorm();
       }
      return bit;
    }
  Alternative byte-based end of slice decoding
for use
    when the MPS/LPS convention is switched
    int decode_end_of_slice_flag(){
      R -= 0x100;
      if (V >= R)
        bit = 1;
      else {
        bit = 0
         decode_renorm();
      }
      return bit;
    }
    Decoder (bit_based):
    void start_decode() {
       decode_slice_header();
       while (!byte_aligned())
          get_bit();
       R = 0x1fe;
       V = 0;
      for (i=0; i<9; i++)
          V = (V<<1) | get_bit();
    }
    void finish_decode() {
      while (!byte_aligned())
           get_bit(); // alignement bit
      while (more_bytes_in_slice())
           get_byte(); / / stuffing byte
    }
    int decode_renorm() {
      while (R<0x100) {
```

```
            R <<= 1;
            V = (V<<1) | get_bit() ;
        }
      }
      int decode_equiprob() {
        V = (V<<1) | get_bit();
        if (V >= R) {
          V -= R;
          bit = 1;
        }
        else
           bit = 0;
        return bit;
      }
      int decode_event(int ctx) {
          rLPS = table[state[ctx]][(R>>6)-4];
          R -= rLPS;
          if (V < R) {
       state[ctx] = next_state_MPS[state[ctx]];
             bit = MPS[state[ctx]];
           }
          else {
             bit = !MPS[state[ctx]];
             V -= R; R = rLPS;
             if (state[ctx] == 0)
           MPS[state[ctx]] = !MPS[state[ctx]];
       state[ctx] = next_state_LPS[state[ctx]];
           }
          decode_renorm();
           return bit;
      }
      int decode_end_of_slice_flag() {
        if (V < 2)
           bit = 1;
        else {
           bit = 0;
           R-=2;
           V-=2;
           decode_renorm();
         }
        return bit;
      }
      Alternative bit-based end_of_slice flag
decoding for use when
      the MPS/LPT convention is switched
      int decode_end_of_slice_flag(){
           R -= 2;
           if (V >= R)
             bit = 1
           else {
             bit = 0;
             decode_renorm();
           }
           return bit;
        }
```

Debe tenerse en cuenta que en el codificador aritmético que se ha descrito anteriormente, existe un intervalo que se divide en dos, un intervalo superior y un intervalo inferior. Uno de los intervalos representa un MPS y el otro intervalo representa el LPS. En una realización, la asignación de los MPS y LPS a intervalos comprende la asignación de un 1 a un intervalo y un 0 al otro. En el código fuente indicado más arriba, cuando el intervalo se divide para la codificación de la etiqueta de extremo de segmento (end_of_slice_flag), al MPS (valor 0) se le asigna el sub-intervalo superior. También es posible asignar el MPS al sub-intervalo inferior.

El código siguiente ilustra otro codificador ejemplar. Debe tenerse en cuenta que en el código, S es el número mínimo de bytes del segmento para satisfacer la relación de límite descrita anteriormente.

```
      void start_encode() {
        send_NAL_first_byte();
        encode_slice_header();
        while (!byte_aligned())
          send_bit(0);
        R = 0x1fe; // range
        L = 0; / / low
        BO = 0; / / bits outstanding
        C = 0; / / event counter
        FB = 1; / / first bit flag
      }
      void finish_encode() {
        bit_plus_follow((L >> 9) & 1);
        for (i=8; i>=1; i--)
          send_bit((L >> i) & 1);
        send_bit(1); / / stop_bit
        while (!byte_aligned())
          send_bit(0); // alignment_bit
        RBSP_to_EBSP();
        S = min_bytes(C,
number_of_macroblocks_in_slice);
        while (S > bytes_in_NAL_unit())
         send_three_bytes(0x000003); / / write
bytes directly into
          NAL unit
      }
      void bit_plus_follow(int b) {
        if (FB == 1)
          FB = 0;
        else
          send_bit(b);
        while (BO > 0) {
          BO--;
          send_bit(!b);
        }
      }
      void encode_renorm() {
        while (!(R&0x100) {
          if (L+R < 0x200)
            bit_plus_follow(0);
          else if (L >= 0x200) {
            bit_plus_follow(1);
            L -= 0x200;
          }
          else {
            BO++;
            L -= 0x100;
          }
          R <<= 1;
          L <<= 1;
        }
      }
      void encode_event(int ctx, int b) {
```

```
  rLPS = table[state[ctx]][(R>>6)-4];
  R -= rLPS;
  if (b == MPS[state[ctx]])
state[ctx] = next_state_MPS[state[ctx]];
  else {
    L += R;
    R = rLPS;
    if (state[ctx] == 0)
     MPS[state[ctx]] = !MPS[state[ctx]];
state[ctx] = next_state_LPS[state[ctx]];
  }
  encode_renorm();
  C++;
}
void encode_equiprob_event(int b) {
  L «= 1;
  if (b)
    L += R;
  if (L+R < 0x400)
    bit_plus_follow(0) ;
  else if (L >= 0x400) {
    bit_plus_follow(1);
    L -= 0x400;
  }
 else {
    BO++;
    L -= 0x200;
  }
  C++;
}
void encode_end_of_slice_flag(int b) {
  if (b == 0) {
    R-=2;
    L+=2;
    encode_renorm();
  }
```

En el código indicado anteriormente, el envío del primer byte, que es parte de la cabecera, a una unidad NAL se realiza para indicar el tipo de datos que viene a continuación. La unidad NAL y su utilización son bien conocidos en la técnica.

La llamada de función RBSP_to_EBSP () hace que los datos sean insertados en el flujo de bits. Más preferiblemente, en una realización, un 03 Hex se inserta después de los bytes de 0000 Hex de los siguientes patrones, por ejemplo, 000000, 000001, 000002, 000003, como una forma de evitar que se produzca un número predeterminado de ceros consecutivos en el flujo de bits. El resultado es que los patrones de 000000 Hex, 000001 Hex y 000002 Hex no aparecen en los datos comprimidos, y pueden utilizarse como marcadores de resincronización. Cuando el patrón 000003 Hex es encontrado por un decodificador, un procedimiento inverso elimina el "03" del flujo de bits.

Aunque una utilización de este tipo para los codificadores y decodificadores que se han descrito en el presente documento se encuentra en la codificación y decodificación de datos de vídeo, un experto en la técnica se daría cuenta de que el codificador y el decodificador que se describen en el presente documento se pueden utilizar en cualquier situación en la que se comprime una secuencia de eventos a una secuencia de información en el caso del codificador, y en la que dicha secuencia de información se descomprime en el caso del decodificador. Además, aunque la descripción anterior del codificador se encuentra en el contexto del proceso de una secuencia de eventos que comprenden múltiples eventos binarios a una secuencia de información que comprende al menos un bit, y para el decodificador se encuentra en el contexto del proceso de una secuencia de información que comprende al menos un bit de una secuencia de eventos que comprenden múltiples eventos binarios, que el codificador y el decodificador podrían operar en secuencias de eventos y secuencias de información compuestas de eventos, que son M-arios en su naturaleza (es decir, cada evento M-ario representa más de un bit de

datos) utilizando las enseñanzas descritas en este documento, tal como sería apreciado por un experto en la técnica.

Un Ejemplo de Sistema Informático

La Figura 17 es un diagrama de bloques de un sistema de ordenador de ejemplo que puede realizar una o más de las operaciones descritas en el presente documento. Debe observarse que estos bloques o un subconjunto de estos bloques pueden estar integrados en un dispositivo tal como, por ejemplo, un teléfono móvil, con el fin de llevar a cabo las técnicas descritas en el presente documento.

Haciendo referencia a la Figura 17, el sistema de ordenador 1700 comprende un mecanismo de comunicación o bus 1711 para comunicar información, y un procesador 1712 acoplado con el bus 1711 para procesar información. El Procesador 1712 incluye un microprocesador, pero no se limita a un microprocesador, tal como, por ejemplo, Pentium™, PowerPC™, alfa™, etc.

El sistema 1700 comprende además una memoria de acceso aleatorio (RAM), u otro dispositivo de almacenamiento dinámico 1704 (al que se hace referencia como la memoria principal) acoplado al bus 1711 para almacenar información e instrucciones que van a ser ejecutadas por el procesador 1712. La memoria principal 1704 también puede ser utilizada para almacenar variables temporales u otra información intermedia durante la ejecución de las instrucciones por parte del procesador 1712.

El sistema de ordenador 1700 comprende también una memoria de sólo lectura (ROM) y / u otro dispositivo de almacenamiento estático 1706 acoplado al bus 1711 para almacenar información estática e instrucciones para el procesador 1712, y un dispositivo de almacenamiento de datos 1707, como por ejemplo un disco magnético o disco óptico, y su unidad de disco correspondiente. El dispositivo de almacenamiento de datos 1707 está acoplado al bus 1711 para almacenar información e instrucciones.

El sistema informático 1700 puede además estar acoplado a un dispositivo de pantalla 1721, como por ejemplo una pantalla de tubo de rayos catódicos (CRT) o una pantalla de cristal líquido (LCD), acoplado al bus 1711 para la visualización de información a un usuario de la computadora. Un dispositivo de entrada alfanumérico 1722, que incluye un teclado alfanumérico y otras teclas, también puede estar acoplado al bus 1711 para comunicar información y realizar comandos de selecciones al procesador 1712. Un dispositivo de entrada de usuario adicional es el control del cursor 1723, como por ejemplo un ratón, un trackball, un trackpad, un stylus o teclas de dirección del cursor, acoplados al bus 1711 para comunicar información de dirección y realizar comandos de selecciones al procesador 1712, y para controlar el movimiento del cursor en la pantalla 1721.

Otro dispositivo que puede estar acoplado al bus 1711 es un dispositivo de copia impresa 1724, que se puede utilizar para la impresión de instrucciones, datos, u otra información sobre un medio como por ejemplo papel, película, o medios de tipo similar. Además, un dispositivo de grabación y reproducción de sonido, como por ejemplo un altavoz y / o un micrófono puede estar opcionalmente acoplado al bus 1711 para funcionar como interfaz de audio con el sistema informático 1700. Otro dispositivo que puede estar acoplado al bus 1711 es una capacidad de comunicación por cable / inalámbrica 1725 para la comunicación con un teléfono, con un dispositivo sujeto en la palma de mano u otro dispositivo.

Debe tenerse en cuenta que en la presente invención puede utilizarse cualquiera o todos los componentes del sistema 1700 y el hardware asociado. Sin embargo, se puede apreciar que otras configuraciones del sistema de ordenador pueden incluir algunos o todos los dispositivos.

Si bien muchas alteraciones y modificaciones de la presente invención, sin duda, resultarán evidentes para una persona con una experiencia ordinaria en la técnica después de haber leído la descripción anterior, debe entenderse que cualquier realización particular mostrada y descrita a modo de ilustración no pretende de ninguna manera ser considerada como limitante. Por lo tanto, las referencias a los detalles de las diversas formas de realización no están destinadas a limitar el alcance de las reivindicaciones, que en sí mismas muestran sólo aquellas características consideradas como esenciales para la invención.

LOS SIGUIENTES ENUNCIADOS A-I2 NO SON REIVINDICACIONES, SINO QUE AFECTAN A ASPECTOS PREFERENTES DE LA PRESENTE INVENCIÓN

A. Un decodificador aritmético que comprende:

un secuenciador para generar un identificador de contexto para un evento de una secuencia de eventos; y
un estimador de probabilidad para determinar un valor para un LPS y una estimación de probabilidad para el LPS; y
un motor de decodificación que incluye un registro de intervalos para asignar un valor a un intervalo para el LPS, en que el valor se basa en la estimación de probabilidad, un valor almacenado en el registro de intervalos y el identificador de contexto para un intervalo para el LPS si el identificador de contexto no es igual a un índice y el valor no está basado en el valor

almacenado en el registro de intervalos si el identificador de contexto es igual al índice, y el motor de decodificación asimismo para determinar un valor de un evento binario basado en el valor del intervalo para el LPS y bits de una secuencia de información.

B. El decodificador aritmético definido en A en que el motor de decodificación deja de decodificar cuando el identificador de contexto es igual al índice y se decodifica un LPS.

C. El decodificador aritmético definido en B en que los datos codificados no-aritméticamente van a continuación de los datos codificados aritméticamente en la secuencia de información.

D. El decodificador aritmético definido en B en el que el índice representa un indicador de un extremo de segmento.

E. El decodificador aritmético definido en A en el que el motor de decodificación incluye un registro de valores y cuando el identificador de contexto es igual al índice decodifica el evento basándose en el valor en el registro de valores generando un evento en un primer estado si el valor en el registro de valores es inferior al número asignado al intervalo de LPS o generando un evento en un segundo estado si el valor en el registro de valores es mayor que o igual al número.

F. El decodificador aritmético definido en E en que el motor de decodificación realiza la renormalización en respuesta a decodificar el evento solamente cuando el valor en el registro de valores es mayor que o igual al número.

G. El decodificador aritmético definido en A en que el motor de decodificación incluye un registro de valores y cuando el identificador de contexto es igual al índice decodifica el evento restando en primer lugar el valor asignado al intervalo de LPS del registro de intervalos y, en que el evento es generado en un primer estado si el valor en el registro de valores es mayor que o igual a un valor en el registro de intervalos o es generado en un segundo estado si el valor en el registro de valores es inferior al valor en el registro de intervalos.

H. El decodificador aritmético definido en G en que el motor de decodificación realiza la renormalización en respuesta a la decodificación del evento solamente cuando el valor en el registro de valores es mayor que o igual al valor en el registro de intervalos.

I. El decodificador aritmético definido en G en que el valor asignado al intervalo del LPS es 2 si el identificador de contexto es igual al índice.

J. El decodificador aritmético definido en H en que el valor asignado al intervalo del LPS es 10 Hex si el identificador de contexto es igual al índice.

K. El decodificador aritmético definido en G en que el valor asignado al intervalo del LPS es 2 si el identificador de contexto es igual al índice y en que un último bit leído en un evento binario que contiene flujo de bits que está siendo decodificado por el motor de decodificación es igual a 1.

L. Un método de decodificación que comprende:

generar un identificador de contexto para un evento binario de una secuencia de eventos;
determinar un valor para un LPS y una estimación de probabilidad para el LPS;
asignar un valor a un intervalo para el LPS, en que el valor se basa en la estimación de probabilidad, un valor almacenado en el registro de intervalos y el identificador de contexto para un intervalo para el LPS si el identificador de contexto no es igual a un índice y el valor no está basado en el valor almacenado en el registro de intervalos si el identificador de contexto es igual al índice, y
determinar un valor de un evento binario basado en el valor del intervalo para el LPS y bits de una secuencia de información.

M. El método definido en L que comprende asimismo detener la decodificación cuando el identificador de contexto es igual al índice y se decodifica un LPS.

N. El método definido en L en que los datos codificados no-aritméticamente van a continuación de los datos codificados aritméticamente en la secuencia de información.

O. El método definido en L en el que el índice representa un extremo de un indicador de segmento.

P. El método definido en L en el que determinar un valor de un evento binario basado en el valor del intervalo para el LPS y bits a partir de una secuencia de información comprende decodificar un evento basándose en un valor en un registro de valores, cuando el identificador de contexto es igual al índice, generando un evento en un primer estado si el valor en el registro de valores es

inferior al valor asignado al intervalo de LPS o generando un evento en un segundo estado si el valor en el registro de valores es mayor que o igual al número.

Q. El método definido en L que también comprende realizar la renormalización en respuesta a decodificar el evento solamente cuando el valor en el registro de valores es mayor que o igual al número.

R. El método definido en L en que determinar un valor de un evento binario basado en el valor del intervalo para el LPS y bits de una secuencia de información comprende decodificar un evento, cuando el identificador de contexto es igual al índice restando en primer lugar el valor asignado al intervalo de LPS del registro de intervalos y, cuando el evento es generado en un primer estado si el valor en el registro de valores es mayor que o igual a un valor en el registro de intervalos o es generado en un segundo estado si el valor en el registro de valores es inferior al valor en el registro de intervalos.

S. El método definido en R que comprende además realizar la renormalización en respuesta a la decodificación del evento solamente cuando el valor en el registro de valores es mayor que o igual al valor en el registro de intervalos.

T. El método definido en R en que el valor asignado al intervalo del LPS es 2 si el identificador de contexto es igual al índice.

U. El método definido en S en que el valor asignado al intervalo del LPS es 10 Hex si el identificador de contexto es igual al índice.

V. El método definido en S en que el valor asignado al intervalo del LPS es 2 si el identificador de contexto es igual al índice y en que un último bit leído en un evento binario que contiene flujo de bits que está siendo decodificado por el motor de decodificación es igual a 1.

W. Un artículo de fabricación que tiene una o más instrucciones de almacenamiento de medios grabables en el mismo que, cuando son ejecutadas por un sistema, provocan que el sistema decodifique datos por medio de:

generar un identificador de contexto para un evento binario;
determinar un valor para un LPS y una estimación de probabilidad para el LPS;
asignar un valor a un intervalo para el LPS, en que el valor se basa en la estimación de probabilidad, un valor almacenado en el registro de intervalos y el identificador de contexto para un intervalo para el LPS si el identificador de contexto no es igual a un índice y el valor no está basado en el valor almacenado en el registro de intervalos si el identificador de contexto es igual al índice, y
determinar un valor de un evento binario basado en el valor del intervalo para el LPS y bits de una secuencia de información.

X. Un codificador aritmético que comprende:

un estimador de probabilidad para generar una estimación de probabilidad de que cada evento de una secuencia de eventos tenga un valor determinado, en que el estimador de probabilidad genera la estimación de probabilidad en respuesta a la información de contexto correspondiente para cada uno de dichos eventos; y
un motor de codificación acoplado con el estimador de probabilidad para generar cero o más bits de una secuencia de información en respuesta a cada evento y su estimación de probabilidad correspondiente, en que el motor de codificación codifica un evento para señalar el extremo de los datos codificados aritméticamente en la secuencia de información utilizando una constante para un rango de sub-intervalo que es independiente de un valor del registro de intervalos antes de codificar la señal del extremo de segmento.

Y. El codificador definido en X en que el motor de codificación que utiliza un constante para codificar el evento para señalar el extremo de los eventos en la secuencia de eventos permite la inclusión de cualquier contenido restante de un registro bajo en la secuencia de información.

Z. El codificador definido en Y, en que el motor de codificación vuelca cualquier elemento restante del registro bajo y establece un último bit escrito durante el volcado igual a 1.

A1. Un método para codificar datos, en que el método comprende:

codificar eventos en una secuencia de eventos para producir datos codificados; y

generar un flujo de bits utilizando los datos codificados, incluyendo la codificación de un indicador para su utilización cuando se decodifica para indicar un extremo de un dato codificado aritméticamente en el flujo de bits.

B1. El método en A1 en que datos codificados no-aritméticamente van a continuación de los datos codificados aritméticamente en el flujo de bits.

C1. El método definido en A1 en que codificar el indicador incluye codificar un evento para señalar el extremo del segmento.

D1. El método definido en C1 en que codificar el evento para señalar el extremo del segmento incluye utilizar una constate para un rango de sub-intervalo que es independiente de un valor del registro de intervalos antes de codificar la señal de extremo de segmento.

E1. El método definido en D1 en que codificar el evento para señalar el extremo de un segmento utilizando una constate permite volcar cualquier contenido restante de un registro bajo en la secuencia de información.

F1. El método definido en E1 en que volcar cualquier contenido restante de los registros bajos comprende establecer un último bit escrito durante el volcado igual a 1.

G1. Un artículo de fabricación que tiene una o más instrucciones de almacenamiento de medios grabables en el mismo que, cuando son ejecutadas por un sistema, provocan que el sistema decodifique datos por medio de:

codificar eventos en una primera secuencia de eventos para producir datos codificados; y
generar un flujo de bits utilizando los datos codificados, incluyendo la codificación de un evento para señalar un extremo de datos codificados aritméticamente en el flujo de bits utilizando una constante para un rango de sub-intervalo que es independiente de un valor del registro de intervalos antes de codificar la señal de extremo de segmento.

H1. El artículo de fabricación definido en G1 que también contiene instrucciones que, cuando son ejecutadas por el sistema provocan que el sistema vuelque contenidos de un registro bajo, incluyendo establecer un último bit escrito durante el volcado igual a 1.

I1. Un aparato para codificar datos, en que el aparato comprende:

medios para codificar un bloque de datos para producir datos codificados; y
medios para generar un flujo de bits utilizando los datos codificados, incluyendo medios para los cuales el motor de codificación codifica un evento para señalar un extremo de datos codificados aritméticamente en el flujo de bits utilizando una constante para un rango de sub-intervalo que es independiente de un valor del registro de intervalos antes de codificar la señal de extremo de segmento.

J1. Un método para crear una máquina de estado para estimación de probabilidad, en que el método comprende:

asignar probabilidades a estados de una tabla de consulta (LUT), incluyendo establecer una probabilidad para cada estado i de los estados igual a la máxima probabilidad del LPS multiplicada por el coeficiente de adaptación para la potencia i, donde i es un número para un estado determinado y el coeficiente de adaptación es menor de 1;
generar transiciones de estado para estados en la LUT a los que se va a pasar de acuerdo con la observación de un MPS y un LPS, en que el estado siguiente al cual pasa la máquina de estado desde un estado actual cuando se observa un MPS es un estado superior siguiente al estado actual si el estado actual no es el estado superior y es el estado actual si el estado actual es el estado superior, y en el que asimismo el estado siguiente al cual pasa la máquina de estado desde un estado actual cuando se observa un LPS para una pluralidad de estados es una versión redondeada de un resultado de calcular:

número de estado actual + log(probabilidad del estado actual*el coeficiente de adaptación + (1- el coeficiente de adaptación))/ probabilidad del estado actual)/ log (el coeficiente de adaptación).

K1. El método definido en J1 en el que la versión redondeada del resultado es tal que, como promedio, cualquier redondeo introducido cuando se generan los estados siguientes para cuando se observa un LPS es sustancialmente cero.

L1. El método definido en J1 en el que la LUT tiene una pluralidad de valores asociados con cada estado, en que cada uno de la pluralidad de valores se aproxima a un producto de un rango de intervalos previsto multiplicado por una probabilidad asociada con el estado.

M1. El método definido en L1 en que un valor asociado con un estado se obtiene por medio del producto de N/(2*M*log(j+M+1)/(j+M))) con una probabilidad asociada con el estado, redondeada a un número entero, en que j representa un índice de columnas en un conjunto, M es el número de columnas en el conjunto, y N es una constante.

N1. El método definido en L1 en que al menos uno de la pluralidad de valores para al menos uno de la pluralidad de estados está asociado a un número predeterminado.

O1. El método definido en N1 en que el número predeterminado permite tener al menos una iteración de renormalización durante la codificación de un MPS.

P1. El método definido en M1 en que

El número de estados en la LUT es 63,

El coeficiente de adaptación es igual a 0.5/0.1875 para la potencia 1.0/63, y

La máxima probabilidad de un LPS es 0.5, y

El número de columnas en el conjunto es 4, y

Los valores en una primera columna en el conjunto están asociados a N/4.

Q1. El método definido en P1 en que el número N es 512.

R1. Un codificador aritmético que comprende:

un estimador de probabilidad para generar una estimación de probabilidad de que cada evento de una secuencia de eventos tenga un valor determinado, en que el estimador de probabilidad genera la estimación de probabilidad en respuesta a la información de contexto correspondiente para cada uno de dichos eventos utilizando una máquina de estado de estimación de probabilidad creada por medio de
asignar probabilidades a estados de una tabla de consulta (LUT), incluyendo establecer una probabilidad para cada estado i de los estados igual a la máxima probabilidad del LPS multiplicada por el coeficiente de adaptación a la potencia i, en que i es un número para un estado determinado y el coeficiente de adaptación es inferior a 1;
generar transiciones de estado para estados en la LUT a los que se va a pasar de acuerdo con la observación de un MPS y un LPS, en que el estado siguiente al cual pasa la máquina de estado desde un estado actual cuando se observa un MPS es un estado superior siguiente al estado actual si el estado actual no es el estado superior y es el estado actual si el estado actual es el estado superior, y en el que asimismo el estado siguiente al cual pasa la máquina de estado desde un estado actual cuando se observa un LPS para una pluralidad de estados es una versión redondeada de un resultado de calcular:

número de estado actual + log(probabilidad del estado actual*el coeficiente de adaptación + (1- el coeficiente de adaptación))/ probabilidad del estado actual)/ log (el coeficiente de adaptación); y
un motor de codificación acoplado al estimador de probabilidad para generar cero o más bits de una secuencia de información en respuesta a cada evento y su estimación de probabilidad correspondiente.

S1. El codificador aritmético definido en R1 en el que la versión redondeada del resultado es tal que, como promedio, cualquier redondeo introducido cuando se generan los estados siguientes para cuando se observa un LPS es sustancialmente cero.

T1. El codificador aritmético definido en R1 en el que la LUT tiene una pluralidad de valores asociados con cada codificador aritmético, en que cada uno de la pluralidad de valores se aproxima

a un producto de un rango de intervalos previsto multiplicado por una probabilidad asociada con el estado.

U1. El codificador aritmético definido en T1 en que un valor asociado con un estado se obtiene por medio del producto de N/(2*M*log(j+M+!)/(j+M))) con la probabilidad asociada con el estado, redondeada a un número entero, en que j representa un índice de columnas en un conjunto, M es el número de columnas en el conjunto, y N es una constante.

V1. El codificador aritmético definido en T1 en que al menos uno de la pluralidad de valores para al menos uno de la pluralidad de estados está asociado a un número.

W1. El codificador aritmético definido en V1 en que el número permite tener al menos una iteración de renormalización durante la codificación de un MPS.

X1. El codificador aritmético definido en R1 en que

El número de estados en la LUT es 63,

El coeficiente de adaptación es igual a 0.5/0.1875 para la potencia 1.0/63, y

La máxima probabilidad de un LPS es 0.5, y

El número de columnas en el conjunto es 4, y

Los valores en una primera columna en el conjunto están asociados a N/4.

Y1. El codificador aritmético definido en X1 en que el número N es 512.

Z1. Un decodificador aritmético que comprende:

Un estimador de probabilidad para generar una estimación de probabilidad de que un evento de una secuencia de eventos tenga un valor determinado, en que el estimador de probabilidad genera la estimación de probabilidad en respuesta a la información de contexto correspondiente para dicho evento de la secuencia de eventos utilizando una máquina de estado de estimación de probabilidad creada por medio de
Asignar probabilidades a estados de una tabla de consulta (LUT), incluyendo establecer una probabilidad para cada estado i de los estados igual a la máxima probabilidad del LPS multiplicada por el coeficiente de adaptación a la potencia i, en que i es un número para un estado determinado y el coeficiente de adaptación es inferior a 1; y
Generar transiciones de estado para estados en la LUT a los que se va a pasar de acuerdo con la observación de un MPS y un LPS, en que el estado siguiente al cual pasa la máquina de estado desde un estado actual cuando se observa un MPS es un estado superior siguiente al estado actual si el estado actual no es el estado superior y es el estado actual si el estado actual es el estado superior, y en el que asimismo el estado siguiente al cual pasa la máquina de estado desde un estado actual cuando se observa un LPS para una pluralidad de estados es una versión redondeada de un resultado de calcular:

número de estado actual + log(probabilidad del estado actual*el coeficiente de adaptación + (1- el coeficiente de adaptación))/ probabilidad del estado actual)/ log (el coeficiente de adaptación); y
un motor de decodificación acoplado al estimador de probabilidad para generar un evento de una secuencia de eventos en respuesta a su estimación de probabilidad correspondiente y una secuencia de información.

A2. El decodificador aritmético definido en Z1 en el que la versión redondeada del resultado es tal que, como promedio, cualquier redondeo introducido cuando se generan los estados siguientes para cuando se observa un LPS es sustancialmente cero.

B2. El decodificador aritmético definido en Z1 en el que la LUT tiene una pluralidad de valores asociados con cada codificador aritmético, en que cada uno de la pluralidad de valores se aproxima a un producto de un rango de intervalos previsto multiplicado por una probabilidad asociada con el estado.

C2. El decodificador aritmético definido en B2 en que un valor asociado con un estado se obtiene por medio del producto de N/(2*M*log(j+M+!)/(j+M))) con la probabilidad asociada con el estado,

redondeada a un número entero, en que j representa un índice de columnas en un conjunto, M es el número de columnas en el conjunto, y N es una constante.

E2. El decodificador aritmético definido en B2 en que al menos uno de la pluralidad de valores para al menos uno de la pluralidad de estados está asociado a un número.

F2. El decodificador aritmético definido en E2 en que el número permite tener al menos una iteración de renormalización durante la codificación de un MPS.

G2. El decodificador aritmético definido en Z1 en que

El número de estados en la LUT es 63,

El coeficiente de adaptación es igual a 0.5/0.1875 para la potencia 1.0/63, y

La máxima probabilidad de un LPS es 0.5, y

El número de columnas en el conjunto es 4, y

Los valores en una primera columna en el conjunto están asociados a N/4.

H2. El decodificador aritmético definido en G2 en que el número N es 512.

I2. Un método de decodificación que comprende:

generar una estimación de probabilidad de que un evento de una secuencia de eventos tenga un valor determinado, en que la estimación de probabilidad que se genera en respuesta a la información de contexto correspondiente para dicho evento de la secuencia de eventos utilizando una máquina de estado de estimación de probabilidad creada por medio de

asignar probabilidades a estados de una tabla de consulta (LUT), incluyendo establecer una probabilidad para cada estado i de los estados igual a la máxima probabilidad del LPS multiplicada por el coeficiente de adaptación a la potencia i, en que i es un número para un estado determinado y el coeficiente de adaptación es inferior a 1; y
generar transiciones de estado para estados en la LUT a los que se va a pasar de acuerdo con la observación de un MPS y un LPS, en que el estado siguiente al cual pasa la máquina de estado desde un estado actual cuando se observa un MPS es un estado superior siguiente al estado actual si el estado actual no es el estado superior y es el estado actual si el estado actual es el estado superior, y en el que asimismo el estado siguiente al cual pasa la máquina de estado desde un estado actual cuando se observa un LPS para una pluralidad de estados es una versión redondeada de un resultado de calcular:

número de estado actual + log(probabilidad del estado actual*el coeficiente de adaptación + (1- el coeficiente de adaptación))/ probabilidad del estado actual)/ log (el coeficiente de adaptación); y
generar un evento de una secuencia de eventos en respuesta a su estimación de probabilidad correspondiente y una secuencia de información.

Reivindicaciones

1. Un método para la codificación de datos, en que el método comprende:

codificar aritméticamente una pluralidad de eventos en una secuencia de eventos para producir datos codificados, en que la pluralidad de eventos son eventos binarios convertidos a partir de valores enteros que representan los bloques de datos de vídeo de muestra transformados y cuantificados; y
generar un flujo de bits utilizando los datos codificados, incluyendo la adición de uno o más bits de relleno precedidos por cero o más bits de alineación al flujo de bits, en el que el uno o más bits de relleno son un patrón reconocible por un decodificador; en que el flujo de bits incluye datos codificados de un segmento incluyendo una indicación de extremo de segmento que indica la finalización de la codificación aritmética, y el uno o más bytes de relleno precedidos de los cero o más bits de alineación se añade después de los datos codificados que incluyen la indicación de extremo de segmento codificada,
en que el uno o más bytes de relleno se utilizan para limitar la cantidad de eventos codificados aritméticamente como una combinación lineal del número de bits en el flujo de bits y el número de bloques de muestra codificados aritméticamente, en que la limitación adquiere la forma de la siguiente combinación lineal:

$$e \leq \alpha B + \beta S,$$

donde

e es el número de eventos codificados aritméticamente en el flujo de bits,
B es el número de bits en el flujo de bits,
S es el número de bloques de muestra codificados aritméticamente en el flujo de bits, y
α y β son valores predeterminados

2. El método definido en la reivindicación 1 en el que el flujo de bits incluye datos de cabecera.

3. Un codificador aritmético (400) que comprende:

un estimador de probabilidad (140) para generar una estimación de la probabilidad de que cada evento de una secuencia de eventos tenga un valor particular, en el que el estimador de probabilidad (140) genera la estimación de probabilidad en respuesta a la correspondiente información de contexto para cada uno de dichos eventos, en que los eventos son eventos binarios convertidos a partir de valores enteros que representan los bloques de datos de video de muestra transformados y cuantificados; y

un motor de codificación (415) acoplado al estimador de probabilidad para generar cero o más bits de flujo de bits en respuesta a cada evento y su correspondiente estimación de probabilidad, en el que el motor de codificación (415) genera uno o más bits de relleno precedidos por cero o más bits de alineación en el flujo de bits, en que el uno o más bits de relleno son un patrón reconocible por parte de un decodificador, en que el flujo de bits incluye datos codificados de un segmento incluyendo una indicación de extremo de segmento que indica la finalización de la codificación aritmética, y el uno o más bits de relleno precedidos por los cero o más bits de alineación se añaden después de los datos codificados que incluyen la indicación de extremo de segmento codificado, en el que el uno o más bits de relleno se utilizan para limitar el número de eventos codificados aritméticamente como una combinación lineal del número de bits en el flujo de bits y el número de bloques de muestra codificados aritméticamente, en que la limitación adquiere la forma de la siguiente combinación lineal:

$$e \leq \alpha B + \beta S,$$

donde

e es el número de eventos codificados aritméticamente en el flujo de bits,
B es el número de bits en el flujo de bits,
S es el número de bloques de muestra codificados aritméticamente en el flujo de bits, y
α y β son valores predeterminados

4. El codificador definido en la reivindicación 3 en el que el flujo de bits incluye datos de cabecera.

5. Un artículo de fabricación que tiene uno o más medios grabables que almacena instrucciones en el mismo y que, cuando es ejecutado por un sistema, provoca que el sistema codifique los datos mediante:

la codificación aritmética de una pluralidad de eventos en una secuencia de eventos para producir datos codificados, en que la pluralidad de eventos son eventos binarios convertidos a partir de valores enteros que representan los bloques de datos de vídeo transformados y cuantificados; y
la generación de un flujo de bits utilizando los datos codificados, incluyendo añadir uno o más bits de relleno precedidos por cero o más bits de alineación en el flujo de bits, en que el uno o más bits de relleno son un patrón reconocible por parte de un decodificador, en que el flujo de bits incluye datos codificados de un segmento incluyendo una indicación de extremo de segmento que indica la finalización de la codificación aritmética, y el uno o más bits de relleno precedidos por los cero o más bits de alineación se añaden después de los datos codificados que incluyen la indicación de extremo de segmento codificado,
en el que el uno o más bits de relleno se utilizan para limitar la cantidad de eventos codificados aritméticamente como una combinación lineal del número de bits en el flujo de bits y el número de bloques de muestra codificados aritméticamente, en que la limitación adquiere la forma de la siguiente combinación lineal:

$$e \leq \alpha B + \beta S,$$

donde
e es el número de eventos codificados aritméticamente en el flujo de bits,
B es el número de bits en el flujo de bits,
S es el número de bloques de muestra codificados aritméticamente en el flujo de bits, y
α y β son valores predeterminados

6. Un decodificador aritmético (1000) que comprende:

un estimador de probabilidad (1010) para generar una estimación de probabilidad de que un evento de una secuencia de eventos tenga un valor determinado, en que el estimador de probabilidad (1010) genera la estimación de probabilidad en respuesta a la información de contexto correspondiente para dicho evento de la secuencia de eventos, en que los eventos son eventos binarios convertidos a partir de valores enteros que representan los bloques de datos de vídeo de muestra transformados y cuantificados; y
un motor de decodificación (1015) acoplado al estimador de probabilidad (1010) para generar el evento de la secuencia de eventos a partir de un flujo de bits en respuesta a la estimación de probabilidad correspondiente al evento, en que el flujo de bits incluye datos codificados de un segmento incluyendo la indicación de extremo de segmento que indica la finalización de la codificación aritmética,
en que el motor de decodificación (1015) reconoce uno o más bytes de relleno precedidos por cero o más bits de alineación añadidos después de los datos codificados que incluye el extremo codificado de la indicación de segmento en el flujo de bits identificando un patrón asociado con los bytes de relleno y no realizando ninguna decodificación aritmética en el uno o más bytes de relleno, en que el uno o más bytes de relleno se utilizan para limitar la cantidad de eventos codificados aritméticamente como una combinación lineal del número de bits en el flujo de bits y el número de bloques de muestra codificados aritméticamente, en que la limitación adquiere la forma de la siguiente combinación lineal:

$$e \leq \alpha B + \beta S,$$

donde

e es el número de eventos codificados aritméticamente en el flujo de bits,

B es el número de bits en el flujo de bits,

S es el número de bloques de muestra codificados aritméticamente en el flujo de bits, y

α y β son valores predeterminados.

7. El decodificador aritmético definido en la Reivindicación 6, en que el motor de decodificación (1015) decodifica una cabecera antes de decodificar aritméticamente datos codificados.

8. El decodificador aritmético definido en la Reivindicación 6, en que no se realiza ninguna renormalización después de decodificar un último evento en la secuencia de eventos.

9. Un método de decodificación que comprende:

producir una pluralidad de eventos de una secuencia de eventos a partir de un flujo de bits que comprende datos codificados aritméticamente, en que la pluralidad de eventos son eventos binarios convertidos a partir de valores enteros que representan los bloques de datos de video de muestra transformados y cuantificados, en que el flujo de bits incluye datos codificados de un segmento que incluyen una indicación de extremo de segmento que indica la finalización de la codificación aritmética; y

reconocer uno o más bytes de relleno precedidos por cero o más bits de alineación añadidos después de los datos codificados que incluye el extremo codificado de la indicación de segmento en el flujo de bits identificando un patrón asociado con los bytes de relleno y no realizando ninguna decodificación aritmética en el uno o más bytes de relleno, en que el uno o más bytes de relleno se utilizan para limitar la cantidad de eventos codificados aritméticamente como una combinación lineal del número de bits en el flujo de bits y el número de bloques de muestra codificados aritméticamente, en que la limitación adquiere la forma de la siguiente combinación lineal:

$$e \leq \alpha B + \beta S,$$

donde

e es el número de eventos codificados aritméticamente en el flujo de bits,

B es un número de bits en el flujo de bits,

S es el número de bloques de muestra codificados aritméticamente en el flujo de bits, y

α y β son valores predeterminados.

10. El método definido en la Reivindicación 9, en que se decodifica una cabecera antes de decodificar aritméticamente datos codificados.

11. El método definido en la Reivindicación 9, en que no se realiza ninguna renormalización después de decodificar un último evento en la secuencia de eventos.

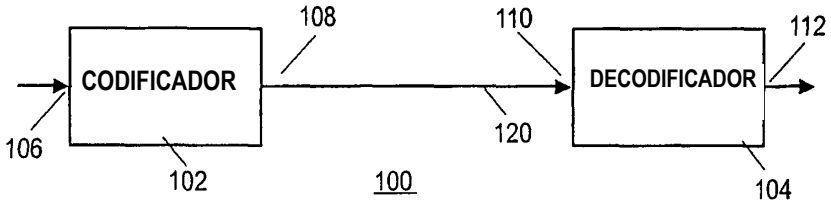
CODIFICADOR
DECODIFICADOR
106
102
108
110
120
112
104
100

FIG. 1

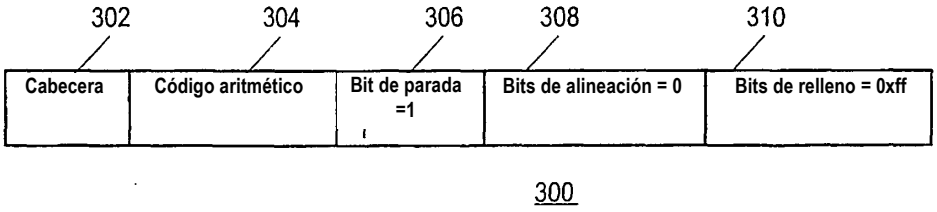
302
304
306
308
310
Cabecera
Código aritmético
Bit de parada =1
Bits de alineación = 0
Bits de relleno = 0xff
300

FIG. 3

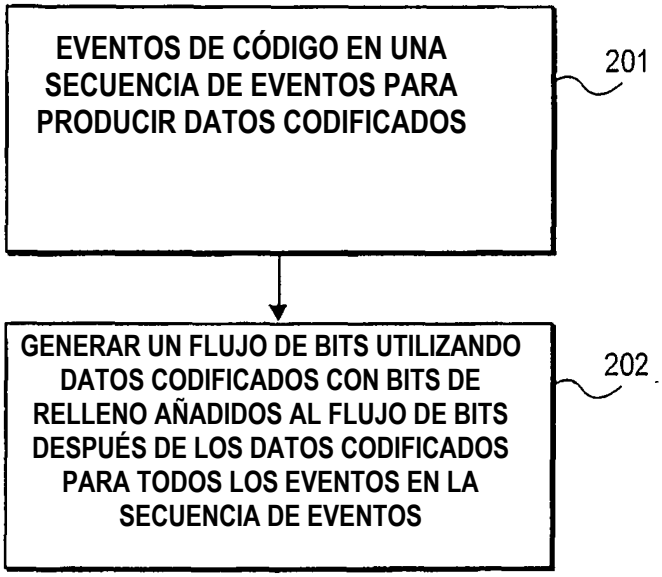
EVENTOS DE CÓDIGO EN UNA SECUENCIA DE EVENTOS PARA PRODUCIR DATOS CODIFICADOS
201
GENERAR UN FLUJO DE BITS UTILIZANDO DATOS CODIFICADOS CON BITS DE RELLENO AÑADIDOS AL FLUJO DE BITS DESPUÉS DE LOS DATOS CODIFICADOS PARA TODOS LOS EVENTOS EN LA SECUENCIA DE EVENTOS
202

FIG. 2

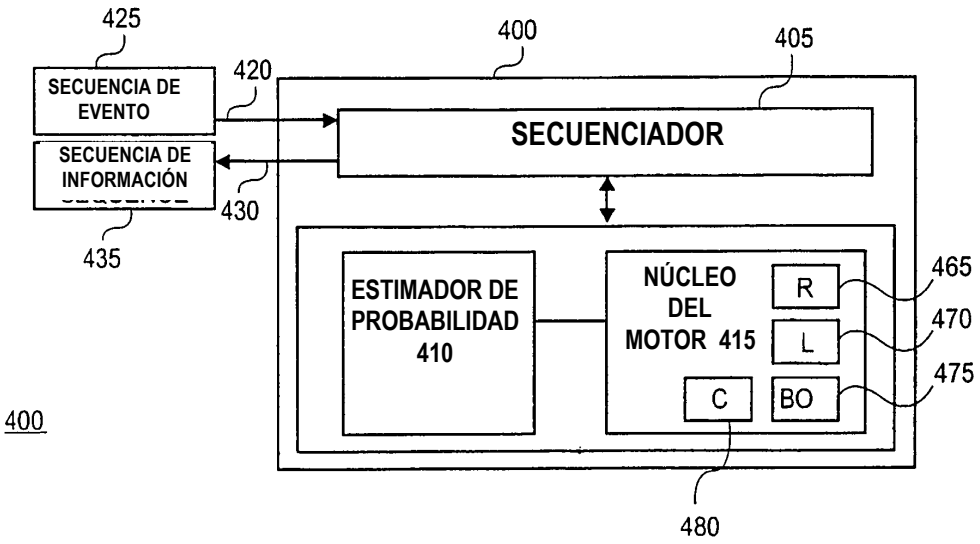
425
SECUENCIA DE EVENTO
420
SECUENCIA DE INFORMACIÓN
430
435
400
405
SECUENCIADOR
ESTIMADOR DE PROBABILIDAD 410
NÚCLEO DEL MOTOR 415
R
L
C
BO
465
470
475
480
400

FIG. 4

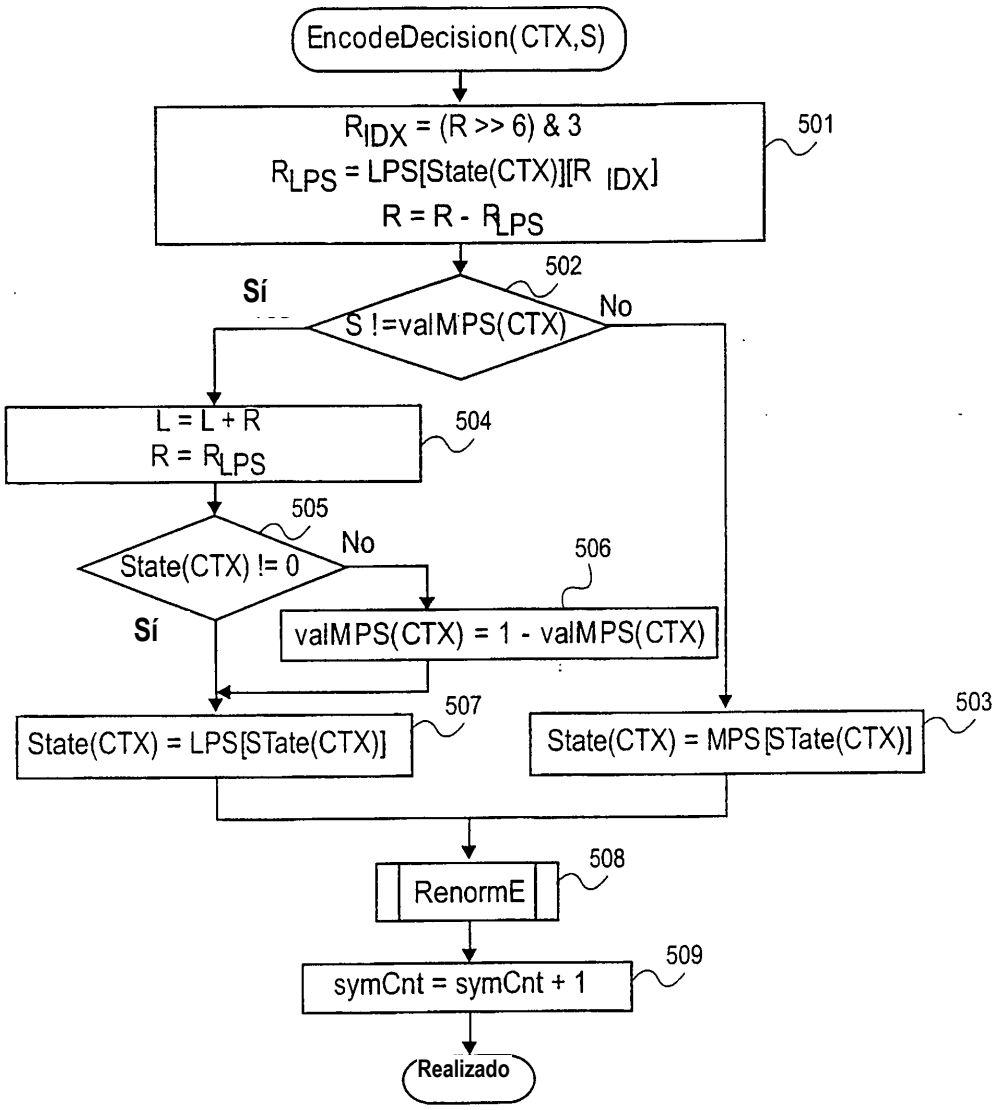
EncodeDecision(CTX,S)
R_IDX = (R >> 6) & 3
R_LPS = LPS[State(CTX)][R_IDX]
R = R - R_LPS
501
502
Sí
S !=valMPS(CTX)
No
L = L + R
R = R_LPS
504
505
State(CTX) != 0
No
506
Sí
valMPS(CTX) = 1 - valMPS(CTX)
507
State(CTX) = LPS[STate(CTX)]
503
State(CTX) = MPS[STate(CTX)]
508
RenormE
509
symCnt = symCnt + 1
Realizado

FIG. 5

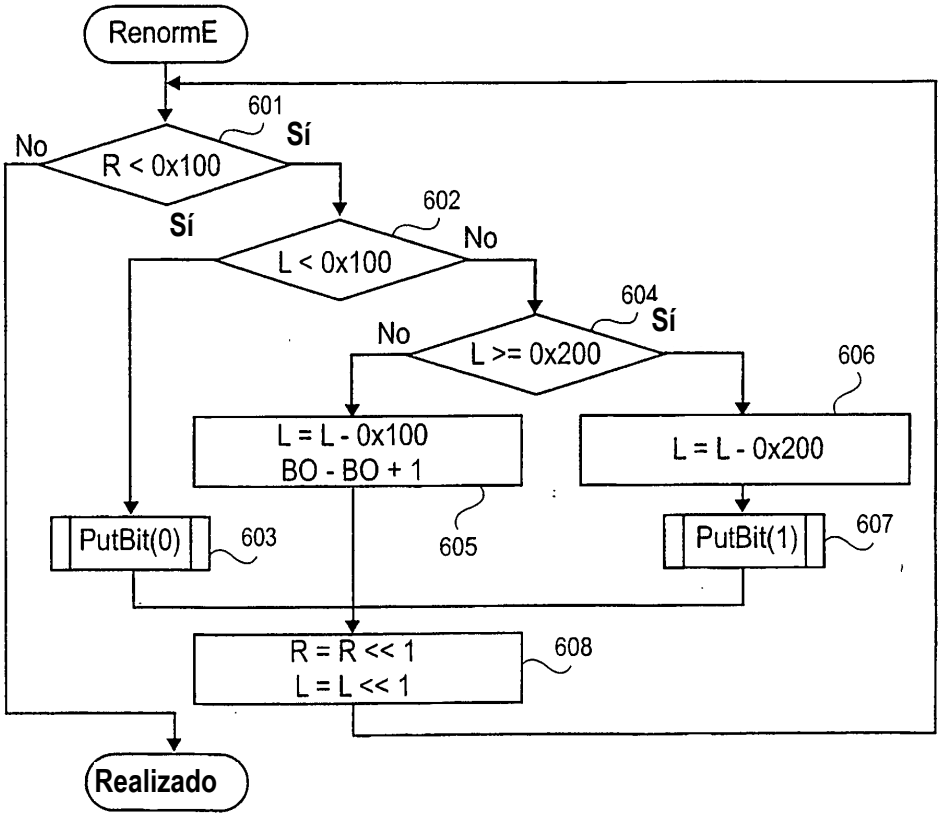
RenormE
601
Sí
No
R < 0x100
Sí
602
No
L < 0x100
604
Sí
No
L >= 0x200
606
L = L - 0x100
BO - BO + 1
L = L - 0x200
PutBit(0)
603
605
PutBit(1)
607
R = R << 1
L = L << 1
608
Realizado

FIG. 6

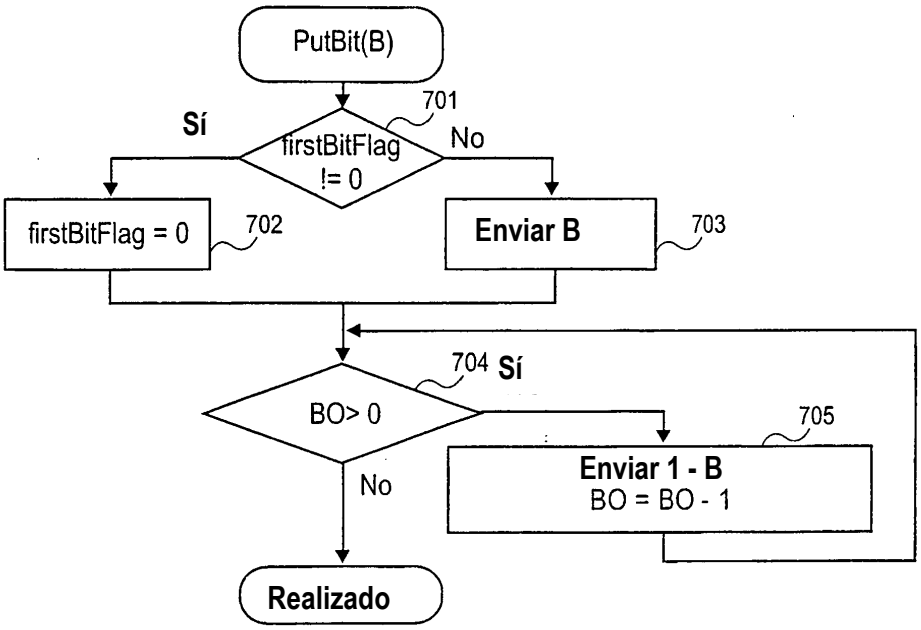
PutBit(B)
701
Sí
firstBitFlag
!= 0
No
firstBitFlag = 0
702
Enviar B
703
704 Sí
BO> 0
705
Enviar 1 - B
BO = BO - 1
No
Realizado

FIG. 7

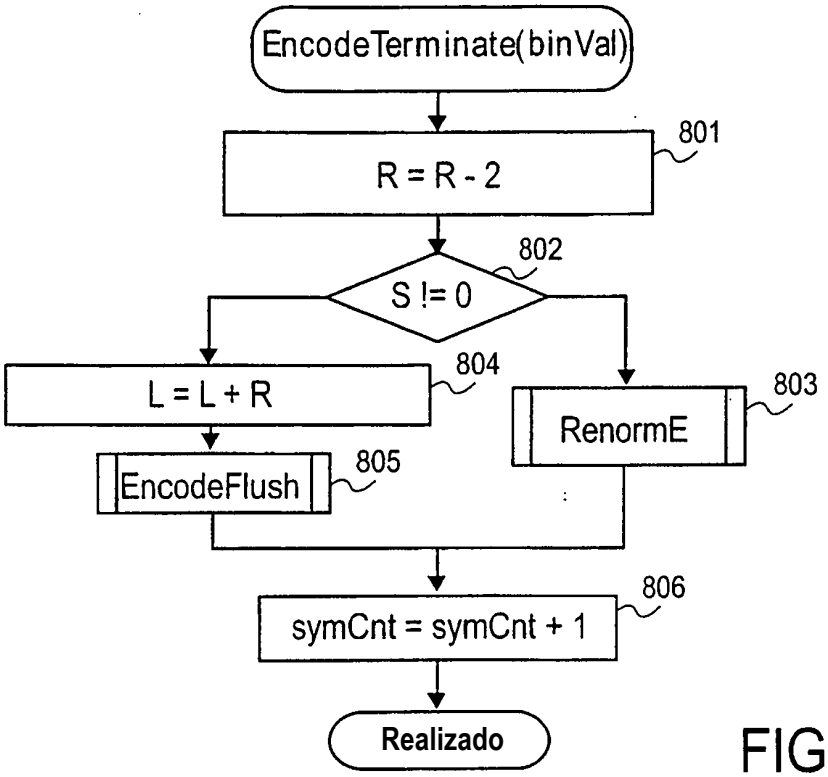
EncodeTerminate(binVal)
801
R = R - 2
802
S != 0
804
L = L + R
803
RenormE
805
EncodeFlush
806
symCnt = symCnt + 1
Realizado

FIG. 8

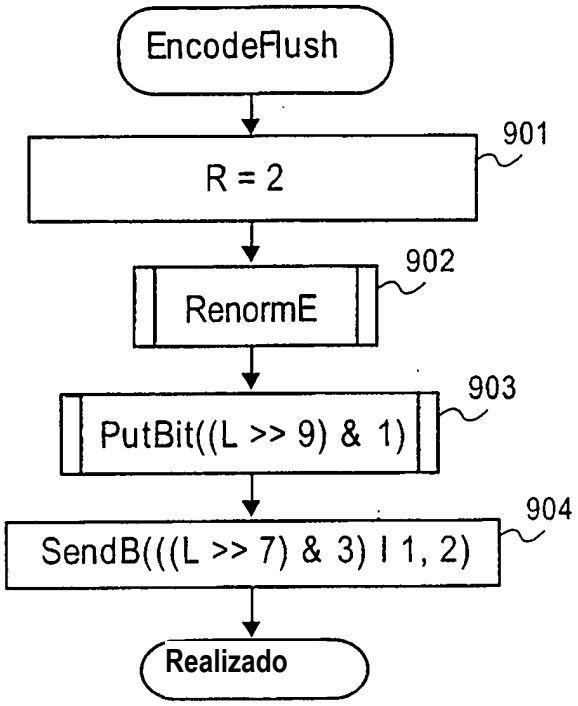
EncodeFlush
901
R = 2
902
RenormE
903
PutBit((L >> 9) & 1)
904
SendB(((L >> 7) & 3) I 1, 2)
Realizado

FIG. 9

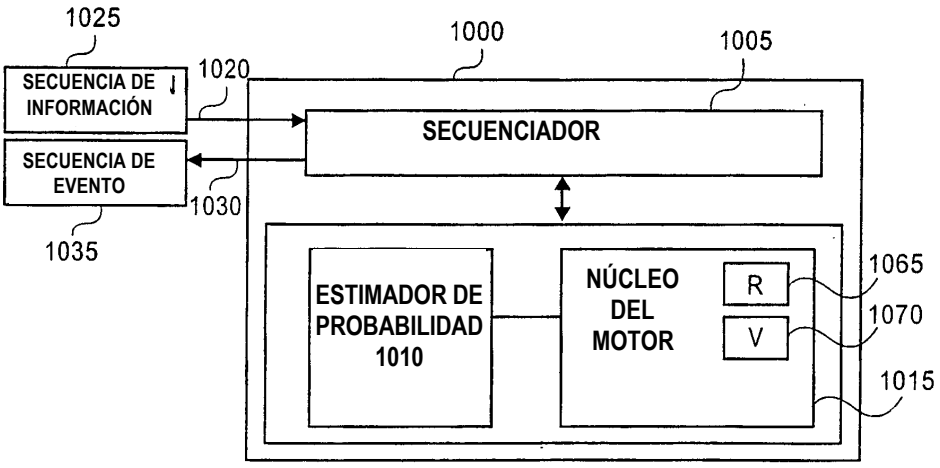
1025
SECUENCIA DE INFORMACIÓN
1020
SECUENCIA DE EVENTO
1030
1035
1000
1005
SECUENCIADOR
ESTIMADOR DE PROBABILIDAD 1010
NÚCLEO DEL MOTOR
R
V
1065
1070
1015

FIG. 10

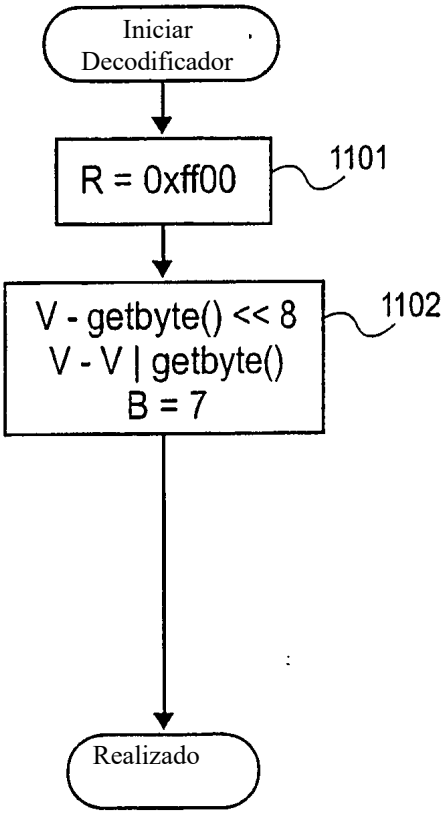
Iniciar Decodificador
R = 0xff00
1101
V - getbyte() << 8
V - V | getbyte()
B = 7
1102
Realizado

FIG. 11

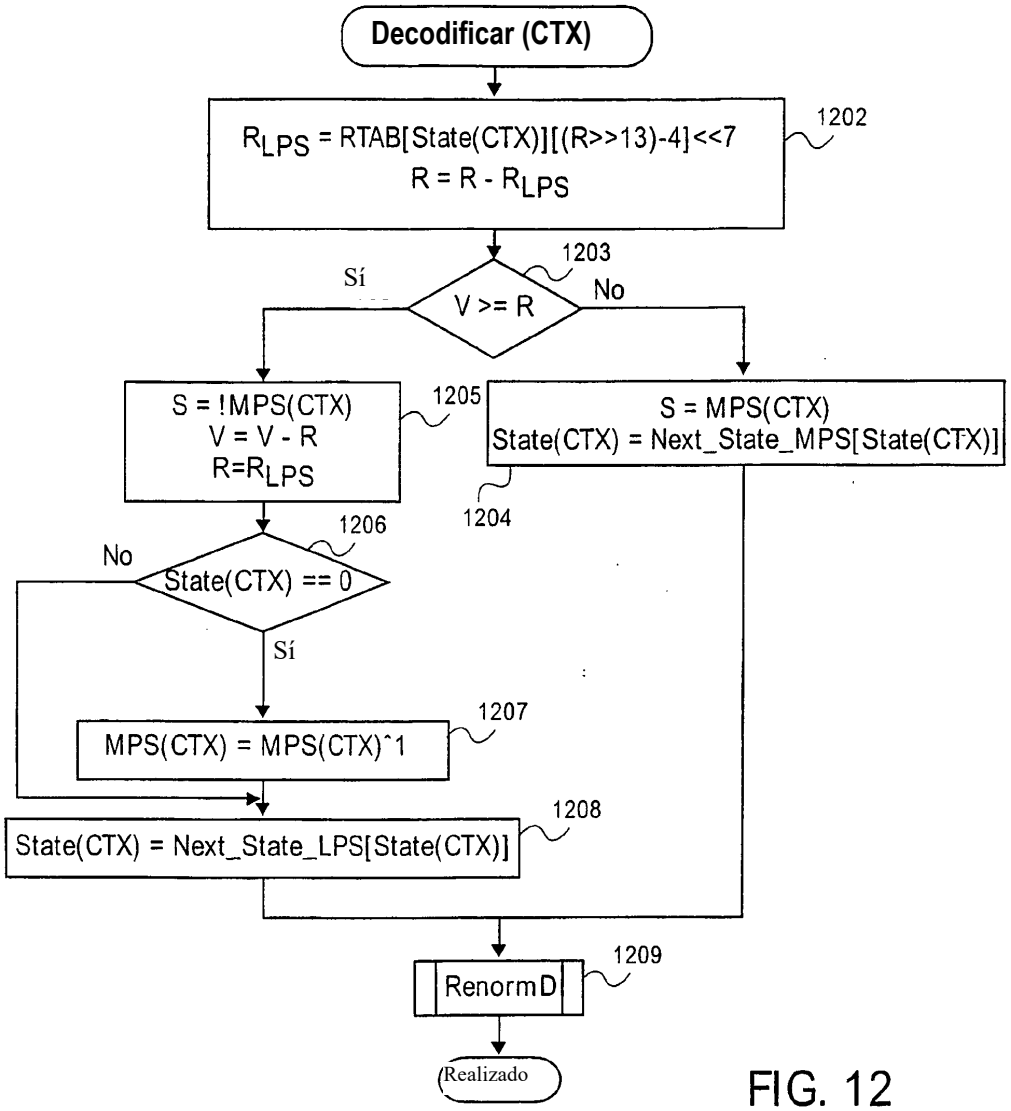
Decodificar (CTX)
R_{LPS} = RTAB[State(CTX)][(R>>13)-4]<<7
R = R - R_{LPS}
1202
1203
Sí
V >= R
No
S = !MPS(CTX)
V = V - R
R=R_{LPS}
1205
S = MPS(CTX)
State(CTX) = Next_State_MPS[State(CTX)]
1204
1206
No
State(CTX) == 0
Sí
1207
MPS(CTX) = MPS(CTX)^1
1208
State(CTX) = Next_State_LPS[State(CTX)]
1209
RenormD
Realizado

FIG. 12

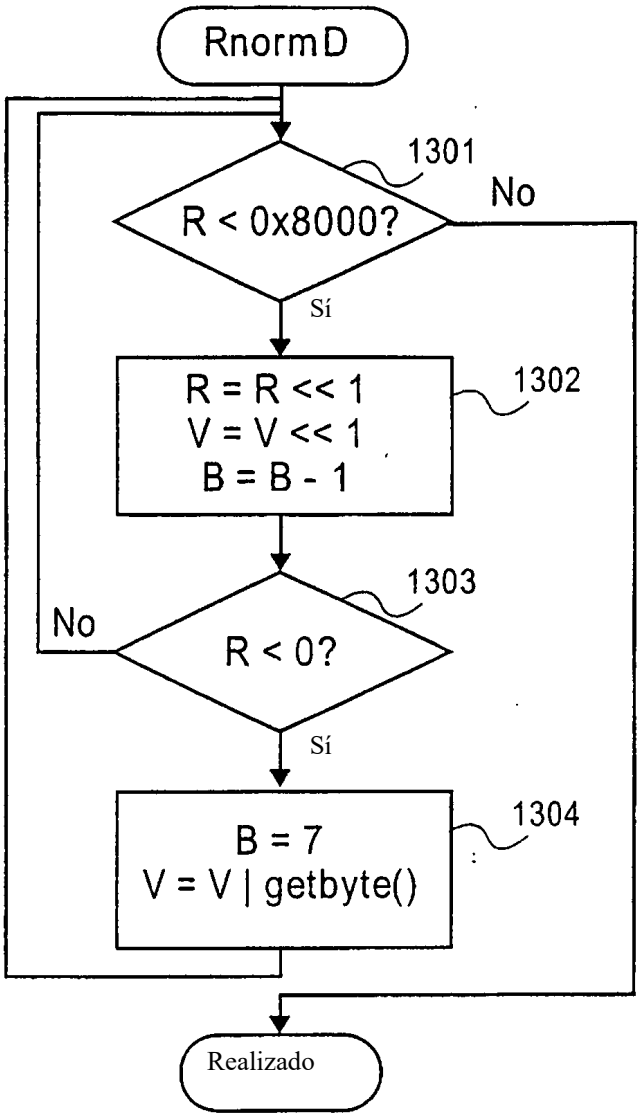
RnormD
1301
R < 0x8000?
No
Sí
R = R << 1
V = V << 1
B = B - 1
1302
1303
No
R < 0?
Sí
B = 7
V = V | getbyte()
1304
Realizado

FIG. 13

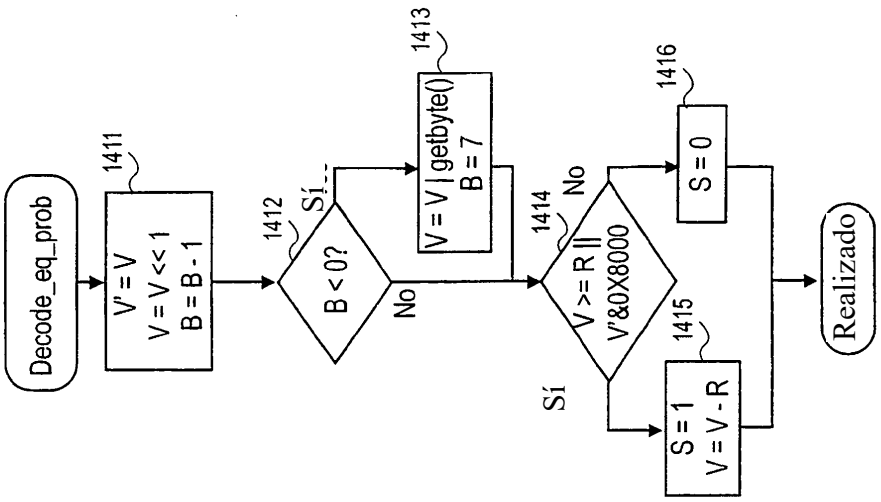

FIG. 14B

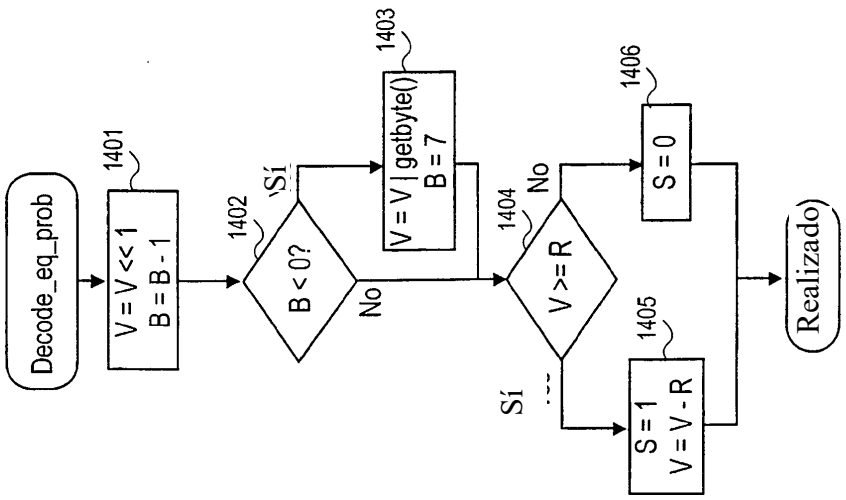

FIG. 14A

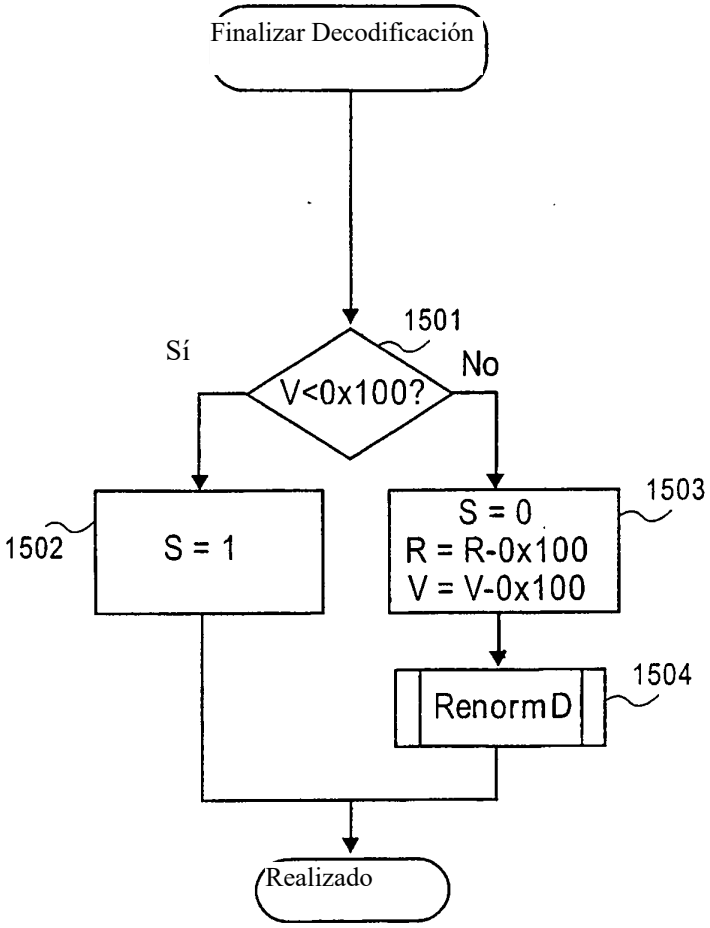
Finalizar Decodificación
1501
Sí
No
V<0x100?
1502
S = 1
1503
S = 0
R = R-0x100
V = V-0x100
1504
RenormD
Realizado

FIG. 15A

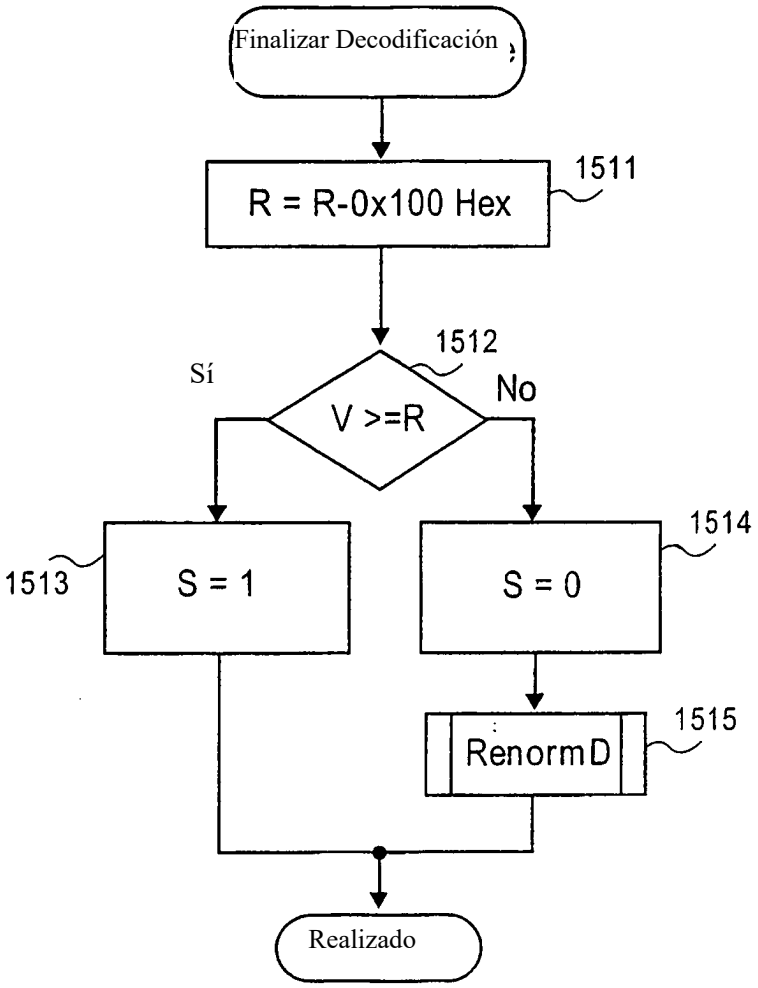
Finalizar Decodificación
R = R-0x100 Hex
1511
1512
Sí
V >=R
No
1513
S = 1
S = 0
1514
RenormD
1515
Realizado

FIG. 15B

Estado (CTX)	R_{IDX}				Estado (CTX)	R_{IDX}			
	0	1	2	3		0	1	2	3
0	128	176	208	240	32	27	33	39	45
1	128	167	197	227	33	26	31	37	43
2	128	158	187	216	34	24	30	35	41
3	123	150	178	205	35	23	28	33	39
4	116	142	169	195	36	22	27	32	37
5	111	135	160	185	37	21	26	30	35
6	105	128	152	175	38	20	24	29	33
7	100	122	144	166	39	19	23	27	31
8	95	116	137	158	40	18	22	26	30
9	90	110	130	150	41	17	21	25	28
10	85	104	123	142	42	16	20	23	27
11	81	99	117	135	43	15	19	22	25
12	77	94	111	128	44	14	18	21	24
13	73	89	105	122	45	14	17	20	23
14	69	85	100	116	46	13	16	19	22
15	66	80	95	110	47	12	15	18	21
16	62	76	90	104	48	12	14	17	20
17	59	72	86	99	49	11	14	16	19
18	56	69	81	94	50	11	13	15	18
19	53	65	77	89	51	10	12	15	17
20	51	62	73	85	52	10	12	14	16
21	48	59	69	80	53	9	11	13	15
22	46	56	66	76	54	9	11	12	14
23	43	53	63	72	55	8	10	12	14
24	41	50	59	69	56	8	9	11	13
25	39	48	56	65	57	7	9	11	12
26	37	45	54	62	58	7	9	10	12
27	35	43	51	59	59	7	8	10	11
28	33	41	48	56	60	6	8	9	11
29	32	39	46	53	61	6	7	9	10
30	30	37	43	50	62	6	7	8	9
31	29	35	41	48	63	2	2	2	2

FIG. 16A

Estado (CTX)	**0**	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
transIdxLPS	0	0	1	2	2	4	4	5	6	7	8	9	9	11	11	12
transIdxMPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Estado (CTX)	**16**	17	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
transIdxLPS	13	13	15	15	16	16	18	18	19	19	21	21	22	22	23	24
transIdxMPS	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Estado (CTX)	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
transIdxLPS	24	25	26	26	27	27	28	29	29	30	30	30	31	32	32	33
transIdxMPS	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Estado (CTX)	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	62	**63**
transIdxLPS	33	33	34	34	35	35	35	36	36	36	37	37	37	38	38	63
transIdxMPS	49	50	51	52	53	54	55	56	57	58	59	60	61	62	62	63

FIG. 16B

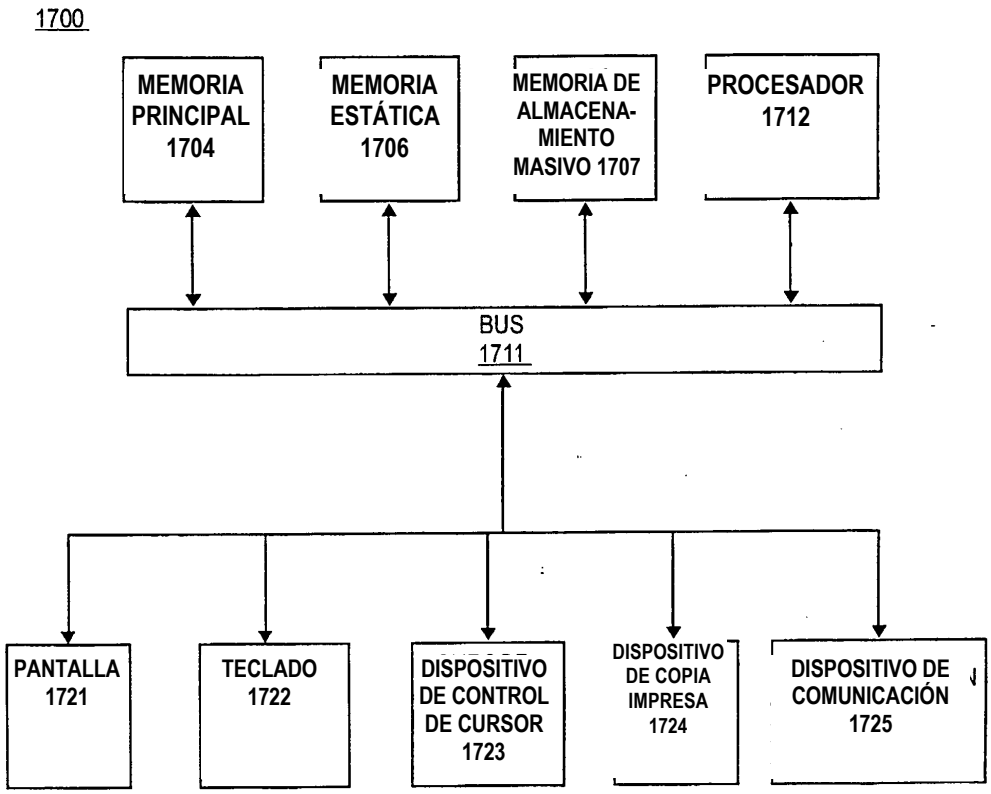
1700
MEMORIA PRINCIPAL 1704
MEMORIA ESTÁTICA 1706
MEMORIA DE ALMACENA-MIENTO MASIVO 1707
PROCESADOR 1712
BUS 1711
PANTALLA 1721
TECLADO 1722
DISPOSITIVO DE CONTROL DE CURSOR 1723
DISPOSITIVO DE COPIA IMPRESA 1724
DISPOSITIVO DE COMUNICACIÓN 1725

FIG. 17