(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 721 055**

(51) Int. CI.:

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

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **19.09.2003 E 16156283 (0)**

(97) Fecha y número de publicación de la concesión europea: **20.03.2019 EP 3079261**

(54) Título: **Método y aparato para 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:
26.07.2019

(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:

FÚSTER OLAGUIBEL, Gustavo Nicolás

DESCRIPCIÓN

Método y aparato para decodificación aritmética

Campo de la invención

La presente invención se refiere en general a la teoría de la información, la compresión de vídeo y la codificación aritmética. Más particularmente, la presente invención se refiere a un método y a un aparato para la terminación de la codificación aritmética y el relleno de bytes, así como a la creación y el uso de una máquina de estados 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 una transmisión en red de un documento, se reduce drásticamente cuando se usa compresión para disminuir el número de bits requeridos para recrear la imagen.

Existen muchas técnicas diferentes de compresión de datos en la técnica anterior. Las técnicas de compresión pueden dividirse en dos categorías amplias, codificación con pérdida y codificación sin pérdida. La codificación con pérdida implica una codificación que da como resultado la pérdida de información, de modo que no hay 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 cuestionables ni detectables. En la compresión sin pérdida, toda la información se conserva y los datos se comprimen de una manera que permite una reconstrucción perfecta. El borrador de la Recomendación H.263 de UIT-T “Video Coding for low Bitrate Communication” divulga un método de codificación aritmética para codificación de vídeo de baja tasa de bits.

La codificación aritmética es una técnica de compresión bien conocida que se usa en algunos sistemas de codificación y compresión de datos para reducir el número de bits o símbolos requeridos para la transmisión. Un codificador aritmético recibe una entrada, que incluye una secuencia de eventos (por ejemplo, eventos binarios) o símbolos. El codificador codifica la secuencia de entrada para dar una secuencia correspondiente de bits o bytes. En algunos casos, se producen menos bits de datos en la salida del codificador que los recibidos en la entrada del codificador, lo que da como resultado la compresión de datos. Un decodificador aritmético puede recibir o acceder 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 logra generando menos bits en las secuencias de información para los eventos que están codificándose, donde las relaciones de eventos con respecto a bits de información que están codificándose 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 de funcionamiento simétrico, 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 de antemano un grupo de bits. Sin embargo, dado que el decodificador lee de antemano un grupo de bits, puede producirse 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 convencional, no se generan bits codificados adicionales, pero se permite que el decodificador lea de antemano en el flujo de bits de datos codificados, y luego retroceda.

Ambas soluciones convencionales introducen ineficacias. Se desea una solución más eficaz para reducir la complejidad de los algoritmos de codificación y decodificación, para reducir los datos para la codificación, transmisión y decodificación y para reducir los requisitos de almacenamiento.

Sumario de la invención

La presente invención se define mediante las reivindicaciones adjuntas.

Breve descripción de los dibujos

La presente invención se entenderá más completamente a partir de la descripción detallada facilitada a continuación y a partir de los dibujos adjuntos de diversas realizaciones de la invención que, sin embargo, no debe considerarse que limitan la invención a las realizaciones específicas, sino que son solo para explicación y comprensión.

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 procedimiento de codificación para generar un flujo de bits.

La figura 3 ilustra un formato de datos a modo de ejemplo mediante el cual pueden transmitirse datos codificados 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 codificar 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 procedimiento para realizar una realización del procedimiento de colocación de bit.

La figura 8 es un diagrama de flujo de una realización de un procedimiento para decodificar un evento antes de la terminación.

La figura 9 ilustra un diagrama de flujo de una realización de un procedimiento para vaciado en la terminación.

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 procedimiento de inicialización de decodificador aritmético.

La figura 12 es un diagrama de flujo de una realización de un procedimiento 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 decodificar un evento binario con equiprobabilidad.

Las figuras 15A y 15B son diagramas de flujo de realizaciones para decodificar un indicador de fin de sector u otros eventos binarios antes de la terminación.

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 sistema informático a modo de ejemplo.

Descripción detallada de la presente invención

Se divulgan un método y un aparato para codificar y decodificar información, particularmente datos de vídeo. Durante la codificación y decodificación, se usa un indicador (por ejemplo, fin de sector) para indicar el final de los eventos que están codificándose aritméticamente. En una realización, también durante la codificación de la información, se añaden bits o bytes de información de relleno al flujo de bits de los datos codificados generados por un codificador. En lugar de rellenar estos bits adicionales en el medio del flujo de bits de datos codificados, los bytes (o bits) de relleno se adjuntan al final de los datos codificados. Tal relleno puede usarse para mantener una relación entre varios eventos que están codificándose, varios bloques de datos de vídeo (por ejemplo, macrobloques) y el tamaño de la secuencia de información que está generándose.

En la siguiente descripción, se exponen numerosos detalles para proporcionar una explicación más completa de la presente invención. Sin embargo, será evidente para un experto en la técnica 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 complicar la presente invención.

Algunas partes de las descripciones detalladas que siguen se presentan en términos de algoritmos y representaciones simbólicas de operaciones en bits de datos dentro de una memoria de ordenador. Estas descripciones y representaciones algorítmicas son los medios usados por los expertos en las técnicas de procesamiento de datos para transmitir lo más eficazmente el contenido de su trabajo a otros expertos en la técnica. En el presente documento, y en general, un algoritmo se concibe como una secuencia coherente de etapas que conducen a un resultado deseado. Las etapas son las que requieren manipulaciones físicas de cantidades físicas. Habitualmente, aunque no necesariamente, estas cantidades adoptan la forma de señales eléctricas o magnéticas capaces de almacenarse, transferirse, combinarse, compararse y manipularse de otro modo. En ocasiones se ha demostrado conveniente, principalmente por razones de uso común, referirse a estas señales como bits, valores, elementos, símbolos, caracteres, términos, números o similares.

Sin embargo, debe tenerse en cuenta que todos estos términos y términos similares van a asociarse con las cantidades físicas apropiadas y son simplemente calificaciones convenientes aplicadas a estas cantidades. A menos que se indique específicamente otra cosa tal como se desprende de la siguiente exposición, se aprecia que a lo largo de toda la descripción, las exposiciones que utilizan términos como “procesar” o “computar” o “calcular” o “determinar” o “visualizar” o similares, se refieren a la acción y los procedimientos de un sistema informático, o dispositivo informático electrónico similar, que manipula y transforma datos representados como cantidades físicas (electrónicas) dentro de los registros y memorias del sistema informático para dar otros datos representados de manera similar como cantidades físicas dentro de las memorias o registros del sistema informático u otros de tales dispositivos de almacenamiento, transmisión o visualización de información.

La presente invención también se refiere a un aparato para realizar las operaciones en el presente documento. Este aparato puede construirse especialmente para los fines requeridos, o puede comprender un ordenador de uso general activado o reconfigurado de manera selectiva por un programa informático almacenado en el ordenador. Tal programa informático puede almacenarse en un medio de almacenamiento legible por ordenador, tal como, pero sin limitarse a, cualquier tipo de disco, incluyendo discos flexibles, discos ópticos, CD-ROM y discos magnético-ópticos, memorias de solo lectura (ROM), memorias de acceso aleatorio (RAM), EPROM, EEPROM, tarjetas magnéticas u ópticas, o cualquier tipo de medio adecuado para almacenar instrucciones electrónicas, y cada uno de ellos acoplado a un bus de sistema informático.

Los algoritmos y pantallas presentados en el presente documento no están relacionados de forma inherente con ningún ordenador u otro aparato en particular. Pueden usarse diversos sistemas de uso general con programas según las enseñanzas en el presente documento, o puede resultar conveniente construir aparatos más especializados para realizar las etapas de método requeridas. La estructura requerida para una variedad de estos sistemas aparecerá a partir de la descripción siguiente. Además, la presente invención no se describe con referencia a ningún lenguaje de programación particular. Se apreciará que pueden usarse 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 máquina incluye cualquier mecanismo para almacenar o transmitir información de una forma legible por una máquina (por ejemplo, un ordenador). Por ejemplo, un medio legible por máquina incluye memoria de solo 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 otras formas de señales propagadas (por ejemplo, ondas portadoras, señales infrarrojas, señales digitales, etc.); etc.

Visión general del sistema de codificación y decodificación

La figura 1 es un diagrama de bloques de una realización de un sistema 100 de codificación y decodificación. En referencia a la figura 1, el sistema 100 incluye un codificador 102 y un decodificador 104 en comunicación a través de un canal 120. Alternativamente, el sistema 100 puede incluir sólo el codificador 102 o el decodificador 104.

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

El codificador 102 tiene una entrada 106 para recibir información de entrada, tal como datos de entrada (por ejemplo, información de vídeo). En una realización, el codificador 102 codifica los datos usando codificación aritmética. Por consiguiente, el codificador 102 puede incluir almacenamiento de datos, registros de manipulación y un motor de codificación aritmética. En una realización, el codificador 102 incluye un registro de rangos, o registro R, y un registro bajo, o registro L. Además, en una realización, el codificador 102 incluye una máquina de estados de estimación de probabilidad. El algoritmo de codificación realizado por el codificador 102 puede ser una codificación aritmética binaria adaptativa al contexto, denominada en el presente documento CABAC, que es bien conocida en la técnica. Además, las técnicas y estructuras descritas en el presente documento también pueden ampliarse 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, una decisión) que indica la terminación de los datos codificados aritméticos. En una realización, el evento que indica la terminación de datos codificados aritméticos comprende un indicador de fin de sector. El flujo de bits también puede incluir bytes (o bits) de relleno tal como se describe con más detalle a continuación.

El decodificador 104 tiene una entrada 110 para recibir los datos codificados a partir 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 observarse que el sistema 100 puede incluir más de un codificador y/o más de un decodificador.

El codificador 102 y el decodificador 104 pueden utilizarse en el procesamiento de datos de vídeo, tales como por ejemplo, datos de vídeo generados por un procesador de vídeo (por ejemplo, códec de vídeo). En una realización, se graba una imagen de vídeo y se divide en bloques de muestra de datos que pueden representar muestras de 16x16, 8x8 o 4x4 de la imagen grabada. A continuación, los bloques se transforman mediante el procesador de vídeo (por ejemplo, utilizando una transformada de coseno discreta) y se cuantifican para obtener valores enteros que representan el bloque de muestra. El procesador de vídeo convierte los valores enteros en una secuencia de eventos (por ejemplo, eventos binarios) y los envía al codificador para su codificación. Alternativamente, el procesador de vídeo puede operar directamente en muestras individuales, incluyendo la transformación y 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 procedimiento de codificación para generar un flujo de bits. El procedimiento se realiza mediante lógica de procesamiento que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

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

La figura 3 ilustra un formato 300 de datos a modo de ejemplo mediante el cual pueden transmitirse datos codificados en un sistema tal como el sistema de la figura 1. El formato 300 incluye un encabezado 302, un código 304 aritmético, uno o más bits 306 de parada, cero, uno o más bits 308 de alineación y cero, uno o más bytes 310 de relleno. En una realización alternativa, pueden usarse cero, uno o más bits de relleno en lugar de bytes.

Tal como se indicó anteriormente, el sistema de la figura 1 y el formato de datos de la figura 3 pueden usarse para codificar y transmitir información de vídeo, incluyendo datos relacionados con una secuencia de imágenes. En una realización, una imagen se divide en uno o más sectores, donde un sector contiene uno o más macrobloques que son matrices de 16x16 píxeles. Cada sector puede codificarse independientemente de otros sectores dentro de la imagen. Los datos de la imagen se codifican en el formato ilustrado en la figura 3.

En una realización, el encabezado 302 comienza en un límite de bytes y contiene datos codificados usando códigos o bien de longitud fija o bien de longitud variable (por ejemplo, codificación de Huffman). El encabezado 302 puede ser un encabezado de sector. Como encabezado de sector, el encabezado 302 puede ir precedido por un código de inicio (SC) y un indicador que identifica el tipo de datos de sector que sigue.

El código 304 aritmético es una secuencia de bits generada por un motor de codificación aritmética de un codificador tal como el codificador 102 (figura 1). En una realización, la secuencia de bits comienza en un límite de byte. Uno o más bits 306 de parada siguen el código 304 aritmético. En una realización alternativa, el bit 306 de parada puede incluirse en el código 304 aritmético. Varios bits 308 (de 0 a 7) de alineación finales siguen a los bits 306 de parada y, en una realización, garantizan la alineación de bytes de los bytes 310 de relleno. El número de bytes 310 de relleno adjuntos a los datos puede ser cero bytes, un byte o más de un byte, dependiendo del número de bytes requeridos para mantener la relación entre el número de eventos que se codifican, el número de bloques de datos de vídeo (por ejemplo, macrobloques) y el tamaño de la secuencia de información que está generándose.

Terminación de flujo codificado

En una realización, el codificador codifica un evento (por ejemplo, una decisión) que indica la terminación de datos codificados aritméticos a un decodificador. Esta terminación de los datos codificados aritméticos puede indicarse cuando se ha alcanzado un fin de sector. La terminación de datos codificados aritméticos también puede producirse cuando los datos codificados aritméticos en un flujo de bits se paran y les siguen datos codificados no aritméticos.

Con referencia de nuevo a la figura 3, en una realización, para cada macrobloque en un sector, el código 204 aritmético normalmente contiene los siguientes datos: un modo de macrobloque, opcionalmente vectores de movimiento y coeficientes de transformada, y también un end_of_slice_flag. El end_of_slice_flag permite que el decodificador 104 (figura 1) determine cuándo se ha decodificado el último macrobloque en un sector. Se usa este indicador, ya que el último bit del código aritmético puede contener datos que describen más de un macrobloque.

Los beneficios de codificar la terminación de datos codificados aritméticos pueden explicarse examinando implementaciones convencionales. En implementaciones convencionales, la terminación de un codificador aritmético se realiza habitualmente según una de las dos alternativas. En un primer enfoque, se transmite todo el registro L. En

un segundo enfoque, se añade un desplazamiento al contenido del registro L y solo se transmiten los bits más significativos del registro L. La ventaja del primer enfoque es que el decodificador lee exactamente el mismo número de bits que genera el codificador. Sin embargo, esto se produce a expensas de enviar bits adicionales. En el segundo enfoque, se ahorran bits, pero el decodificador lee más bits que los generados por el codificador. Esto puede superarse rellenando el flujo de bits en el decodificador.

Un enfoque divulgado en el presente documento ofrece lo mejor de ambos planteamientos: el decodificador lee el mismo número de bits que genera el codificador sin que el codificador envíe más bits de los necesarios. Esto se permite por el hecho de que un evento, el end_of_slice_flag, está codificado para señalar el final de un sector. Dada una probabilidad bien definida asignada a este evento, un decodificador puede decodificarlo, pero puede renunciar a la renormalización si el resultado del evento señaliza terminación. Es decir, normalmente, durante la codificación, para cada símbolo que se codifica, el valor R se multiplica por la probabilidad de obtener un subintervalo. Después, se realiza la renormalización para devolver el valor de R a un rango de valores. Los expertos en la técnica de codificación aritmética conocen bien la renormalización. La renormalización anterior garantiza que el número de bits leídos coincida con el número de bits generados por el codificador.

En una realización, la probabilidad asignada a un evento de end_of_slice (u otros eventos que indican la terminación de la codificación aritmética) se define por un número asignado al registro R durante la terminación de la codificación, antes de que se realice cualquier renormalización. En una realización, para garantizar que el codificador y el decodificador estén sincronizados, para el indicador de fin de sector, el cálculo del subintervalo no se realiza multiplicando el valor almacenado en R por la probabilidad. En cambio, al subintervalo se le asigna un valor fijo o constante. En una realización, se utiliza un valor fijo de 2. Más generalmente, el valor debe ser independiente del valor del contenido del registro R antes de codificar el end_of_slice_flag. Esto se hace para el último símbolo (bit) que se coloca en el flujo de bits. Al ajustar el subintervalo a un valor de 2, puede añadirse el valor de 1 al valor del registro L sin afectar al funcionamiento del decodificador. Esto permite que se envíe el contenido de todo el registro bajo (L) al flujo de bits. Puesto que se envía el contenido de todo el registro L, no es necesaria ninguna renormalización en este caso.

En una realización, el bit menos significativo del registro L se ajusta a 1 antes de enviar el contenido de L. Ajustar el bit menos significativo del registro L en 1 es equivalente a añadir 1 a L si su bit menos significativo es cero. Por 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.

Añadir 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 sector, tras un bit de parada y cero, uno o más bits de alineación. Los bits de alineación se añaden para garantizar que cualquier byte de relleno añadido se inserta en límites de byte. Uno de los beneficios de colocar los bytes de relleno después del bit de parada es que un decodificador no tendrá que decodificar los bytes de relleno. Por 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 mantener una relación entre el número de eventos que están introduciéndose en el codificador, el número de bloques de datos y el número de bits que están saliendo del codificador. La relación se describe en más detalle a continuación.

En una realización, los bytes de relleno tienen un patrón específico. El patrón puede ser único de manera que un decodificador puede determinar que los bytes de relleno están presentes identificando bits con este patrón particular después de un bit de parada y uno o más bits de alineación. Una vez que se realiza una determinación de este tipo, el decodificador no tiene que decodificar los bytes de relleno. En una realización, el decodificador incluye la funcionalidad de demultiplexación que evita que los bytes de relleno se envíen a un motor de decodificación aritmética en el decodificador, de manera similar a los bits de encabezado (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 decodificador reconoce el tercer byte (03) después de un fin de sector para identificar los bytes como bytes de relleno.

En una realización, el número de bytes 310 de relleno rellenado al final de un sector 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 el codificador 102 de la figura 1, puede usar un registro C para contar o, en cualquier caso, realizar un seguimiento de la relación de eventos (operaciones de decodificación) con respecto a 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 decrementa en 4 (o en 32 para cada byte producido). En una realización, el recuento tiene en cuenta todos los bits en el sector (u otro conjunto de eventos), incluyendo los bits de encabezado y alineación y parada finales.

Debe observarse que en una realización, las operaciones de decodificación para end_of_slice_flag no se cuentan con el contador C (aunque en una implementación alternativa pueden contarse). Sin embargo, se sabe que hay un evento de este tipo por macrobloque y el número de tales eventos está bien delimitado por el tamaño de la imagen. En este caso, no contar eventos de end_of_slice_flag es equivalente a contarlos (incrementando por tanto C en 1 una vez por macrobloque), pero decrementando al mismo tiempo C en 1 cada 256 píxeles (una vez por macrobloque). Alternativamente, C podría decrementarse en cualquier valor para cada macrobloque.

En una realización, adjuntar bytes de relleno de la manera descrita en el presente documento garantiza una longitud mínima para el sector codificado. En relación con la técnica convencional de insertar bits de relleno en el medio de un sector codificado, este avance simplifica las normas mediante las cuales el codificador codifica los datos, en particular, definiendo cuántos datos codificar.

El codificador puede restringir el número de eventos de la secuencia de eventos en 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 adoptar 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, macrobloques) representados en la secuencia de eventos, y

α y β representan un valor de decremento de un contador para mantener sustancialmente una restricción 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 para α y β normalmente se proporcionan a un controlador para un codificador aritmético, y la derivación de α y β se comentará a continuación. El valor α puede representar un valor de decremento para, por ejemplo, un contador tras la generación de un bit de información en el codificador, donde el valor β puede representar un valor de decremento para, por ejemplo, un contador tras la finalización del procesamiento de un bloque de datos. Como alternativa, el valor β puede decrementarse con respecto a un valor de contador al comienzo del procesamiento de un segmento, o en cualquier otro momento durante el procesamiento de un bloque de datos tal como será evidente para un experto en la técnica.

Dado que se conocen el número total de bloques, S, y el valor β, el producto de β x S puede restarse del número de eventos, e, para la secuencia de eventos después del procesamiento de los bloques (por ejemplo, macrobloques) de los datos de entrada. Por ejemplo, cuando se utiliza un contador para restringir el número de eventos que responden al número de bits que se han generado, el contador puede decrementarse inicialmente en un valor de β x S, y puede decrementarse en un valor α para cada bit de información generado, 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, normalmente en el rango de 1 a 100, y puede determinarse, por ejemplo, tal como se describe más adelante adicionalmente. El valor de α puede ser cualquier valor, normalmente en el rango de 1 a 10, y puede determinarse, por ejemplo, tal como se describe más adelante.

En algunas circunstancias, un número de bloques de los datos de entrada que van a procesarse no se conoce de antemano, por ejemplo, cuando el medio de comunicación limita el número de bits de información que pueden proporcionarse en la secuencia de información. Esto puede producirse, por ejemplo, cuando la secuencia de información va a transmitirse a través de Internet, como un paquete de Protocolo de Internet (IP), donde el paquete de IP tiene una limitación de tamaño máxima. En estas circunstancias, dependiendo de la complejidad de una imagen en particular, puede requerirse 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 puede no conocerse de antemano, puesto que puede no conocerse después de cuántos segmentos procesados se alcanzará el tamaño máximo de una secuencia de bits de información. Cuando no se conoce de antemano el número de segmentos de los datos de entrada que van a procesarse, el controlador puede tener en cuenta las secuencias de eventos a medida que se codifican uno o más bloques que representan una secuencia particular de eventos. Por ejemplo, cuando se utiliza un contador para restringir el número de eventos que responden al número de bits que se han generado, el contador puede decrementarse en un valor β para cada bloque procesado y puede decrementarse en un valor α para cada bit de información generado, mientras que el

contador puede incrementarse en “1” para cada evento de la secuencia de eventos procesados por el codificador de entropía.

Los valores de α y β pueden ser determinados de antemano, por un diseñador de sistemas del codificador que tiene en cuenta una o más de las limitaciones comentadas anteriormente, y que se proporciona al controlador. Alternativamente, o además, los valores de α y β pueden determinarse mediante el controlador, o mediante cualquier otro componente del codificador, según una o más de las limitaciones comentadas anteriormente, o como valores por defecto del codificador. Cuando el controlador determina los valores para α y β usando una o las dos limitaciones impuestas por la norma o por un dispositivo de decodificación, la información sobre una o más de las limitaciones puede almacenarse en una memoria (no mostrada) del controlador, y usarse por el controlador en la determinación de los valores α y β. Adicionalmente, o como alternativa, la información relacionada con las limitaciones puede proporcionarse al controlador, por ejemplo, mediante algún dispositivo externo, tal como una memoria externa (es decir, un disco de vídeo digital (DVD)), un dispositivo reproductor de DVD o mediante un ingeniero de sistemas, por ejemplo, manejando algunas de las funciones relacionadas con la codificación de los datos de entrada particulares. En este último caso, el ingeniero de sistemas puede introducir a una consola u otro dispositivo de entrada (no mostrado), o especificar de otro modo, información con respecto a las limitaciones impuestas como resultado de una norma de codificación y/o un dispositivo de decodificación, tal como apreciará un experto en la técnica.

Además, cuando se determinan los valores para α y β, pueden realizarse consideraciones en cuanto a si la restricción de complejidad es demasiado ajustada, por ejemplo, si los valores para 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 el 1% o el 2% de los bits de información de la secuencia de información) puede indicar que la restricción es demasiado ajustada. Un experto se daría cuenta de que otras proporciones pueden indicar una alta proporción de bits de información de relleno, por ejemplo, teniendo en cuenta la norma y/o el decodificador particulares que pueden usarse.

Cuando se determina, por ejemplo, que los valores para α y β son demasiado ajustados, los valores para α y β pueden aumentarse para reducir la probabilidad de que se añadan bytes de relleno (es decir, reducir la probabilidad de una penalización de calidad en la secuencia de información codificada). Cuando se aumentan los valores para α y β, pueden realizarse consideraciones en cuanto a los efectos sobre los límites de complejidad resultantes con respecto a un decodificador que se utilizará para decodificar la secuencia de información codificada. Tales consideraciones pueden incluir el coste de implementar el decodificador. Si el límite de complejidad es mayor, es posible que se requiera más capacidad de procesamiento en el decodificador. Un aumento en la capacidad de procesamiento requerida podría dar como resultado un mayor coste de implementación. Debe observarse que en una realización, los cambios para α y β pueden realizarse después de codificar los datos de cada macrobloque.

Los valores α y β pueden determinarse experimentalmente, utilizando técnicas de regresión lineal. Pueden codificarse varias secuencias de eventos, representando cada una S segmentos, sin imponer ninguna restricción de complejidad. Para cada secuencia z de eventos, se conoce para el número de eventos e(z), el número de bits de información generados resultantes B(z). Puede determinarse, usando regresión lineal, una línea e + c*B + d que se aproxima a los pares de datos (e(z), B(z)). A continuación, puede aumentarse un valor inicial de α y/o β tal como para reducir, y potencialmente minimizar, 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 para α y β según lo determinado por una o más de las diversas técnicas comentadas anteriormente, el codificador puede tener cuenta un valor de α (es decir, decrementar un contador en el valor de α) para cada bit de información generado, y puede tener en cuenta un valor de β (es decir, decrementar un contador en el valor de β) tras completar un segmento de los datos de entrada. Por ejemplo, cuando α y β son valores enteros, dicha consideración (es decir, decrementos en uno o más contadores) puede llevarse a cabo directamente.

Cuando, por ejemplo, uno o ambos de α y β son valores fraccionarios, puede determinarse un denominador común para proporcionar valores no fraccionarios para α y β. En esta circunstancia, pueden tenerse en cuenta los nuevos valores no fraccionarios para α y β tal como se describió anteriormente, por ejemplo, al decrementar un contador en los valores de α y β tras la generación de un bit de información y la finalización del procesamiento del segmento, respectivamente. Puede tenerse en cuenta el denominador común determinado, por ejemplo, añadiendo el valor del denominador común al valor de contador tras procesar cada evento de la secuencia de eventos. Por ejemplo, cuando se determina que los valores para α y β son 4/3 y 25 respectivamente, puede determinarse un denominador común como 3. Los valores no fraccionarios para α y β pueden determinarse por tanto como 4 y 75 respectivamente, utilizando común denominador. Por tanto, cuando se utiliza un contador para tener en cuenta los valores de α y β, el contador puede decrementarse en 4 para cada bit de información generado, puede decrementarse en 75 tras la finalización del procesamiento de cada segmento y puede incrementarse en 3 para cada evento procesado.

Funcionamiento del codificador a modo de ejemplo

La figura 4 ilustra un diagrama de bloques de una realización de un codificador aritmético. En referencia a la figura 4, el codificador 400 aritmético incluye un secuenciador 405, un estimador 410 de probabilidad y un motor 415 de codificación, que están acoplados entre sí. Una o más líneas 420 de datos de entrada proporcionan un puerto de entrada para recibir una secuencia 425 de eventos (por ejemplo, una secuencia ordenada de eventos binarios) al 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 comprende 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 enviar la secuencia 435 de información 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”.

Tras recibir la secuencia 425 de eventos, el secuenciador 405 transmite secuencialmente los eventos 425 tanto al estimador 410 de probabilidad como al motor 415 de codificación. Para cada evento binario de la secuencia 425 de eventos, el secuenciador 405 también transmite información de contexto al estimador 410 de probabilidad para el evento binario. El estimador 410 de probabilidad, usando la información de contexto recibida, genera una estimación de probabilidad P(A) que se transmite al motor 415 de codificación. En una realización, el estimador 410 de probabilidad envía múltiples estimaciones de probabilidad al motor 415 de codificación y el motor 415 de codificación selecciona una de las estimaciones de probabilidad basándose en el valor R. Alternativamente, el valor R puede enviarse al estimador 410 de probabilidad, que lo utiliza para seleccionar una estimación de probabilidad que va a enviarse. El estimador 410 de probabilidad actualiza entonces su estado interno basándose en el valor del evento binario recibido. El motor 415 de codificación produce 0 o más bits de información usando el evento binario recibido y la estimación de probabilidad correspondiente P(A).

En una realización, el motor 415 de codificación codifica un evento que indica una terminación de datos codificados aritméticos. El evento puede ser un indicador de fin de sector u otro indicador de que van a seguir datos codificados no aritméticos, si los hay, en el flujo de bits.

Al producir los cero o más bits de información, el motor 415 de codificación utiliza varios registros incluyendo un registro 465 de rangos, un registro 470 bajo, un registro 475 de bits pendientes y un registro 480 de contador. Se conoce bien en la técnica el funcionamiento del codificador 400 para realizar codificación aritmética.

En una realización, el codificador 400 limita una relación de eventos a bits de información, que se describe en otra parte en el presente documento. El codificador 400 realiza esta operación, en parte, insertando 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 codificar un evento. El procedimiento se realiza mediante lógica de procesamiento, que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos. Las entradas al procedimiento de codificación aritmética son los eventos binarios que están decodificándose, con el ID de contexto que identifica el contexto, y el valor R, L y symCnt, y en las salidas se escriben 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 comentó anteriormente, se representa mediante el valor del valor de L que señala al extremo inferior del subintervalo y el valor de R que especifica el rango correspondiente del subintervalo.

En una realización, el procedimiento de codificación se invoca solo después de haberse inicializado el motor de codificación. En una realización, la inicialización se realiza enviando el valor de L igual a cero y el valor de R igual a 0x01FE, ajustando un primer indicador de bit en uno, el valor bits pendientes (BO) y los contadores de symCnt (C) igual a cero. El primer indicador de bit se usa durante la codificación para indicar cuándo está pasando el codificador por primera vez por el procedimiento de colocación de bit. El contador symCnt almacena un valor que indica el número de eventos que están codificados.

En referencia a la figura 5, el procedimiento comienza codificando un solo evento (por ejemplo, un bit) derivando el valor R_{LPS} tal como sigue (bloque 501 de procesamiento). En una realización, la lógica de procesamiento deriva la variable R_{LPS} ajustando el índice de R (o R_{idx}) igual al valor de R desplazado seis posiciones a la derecha y se somete a Y con el número 3 Hex. A continuación, la lógica de procesamiento ajusta el valor de R_{LPS} igual a un valor determinado al acceder a una tabla de máquinas de la estación de estimación de probabilidad, tal como una tabla mostrada en la figura 16A usando el valor de R_{idx} y el valor del estado para el contexto actual asociado con el contexto. El valor de R se ajusta entonces al valor de R actual menos R_{LPS}.

Después de calcular el intervalo de subrango para el recuento de MPS, la lógica de procesamiento somete a prueba si el valor del evento binario que está codificándose no es igual al valor del MPS (bloque de procesamiento 502). Si el valor del evento binario es igual al MPS, entonces la lógica de procesamiento toma la ruta de MPS y pasa al bloque 503 de procesamiento donde la lógica de procesamiento actualiza la máquina de estados al siguiente estado

indicado en la máquina de estados para el contexto usando la tabla de la figura 16B y el procesamiento pasa al bloque 508 de procesamiento. Si la lógica de procesamiento determina que el evento binario que está codificándose no es igual al valor del MPS, entonces la lógica de procesamiento toma la ruta de LPS y pasa al bloque 504 de procesamiento, donde la lógica de procesamiento ajusta el valor de L igual al valor de L más el valor de R y ajusta el valor de R igual al valor de R_{LPS}.

Posteriormente, la lógica de procesamiento determina si el estado para el contexto particular no es igual a cero (bloque 505 de procesamiento). En una realización, el estado cero es un estado correspondiente a una probabilidad de 50/50. Alternativamente, el estado cero es un estado correspondiente a otra probabilidad tal como, por ejemplo, algo próximo a una probabilidad de 50/50. Si el estado para el contexto no es igual a cero, la lógica de procesamiento pasa al bloque 507 de procesamiento. Si el estado para el contexto es igual a cero, la lógica de procesamiento cambia el significado del MPS (bloque 506 de procesamiento) y el procesamiento pasa al bloque 507, y la lógica de procesamiento actualiza el número de estado del contexto al siguiente estado usando la tabla en la figura 16B (bloque 507 de procesamiento).

Tras efectuar los bloques 507 y 503 de procesamiento, el procesamiento pasa al bloque 508 de procesamiento donde la lógica de procesamiento realiza el procedimiento de renormalización, tal como la renormalización en la figura 6. A continuación, la lógica de procesamiento incrementa el valor del contador de eventos en 1 (bloque 509 de procesamiento) y el procesamiento finaliza.

La figura 6 es un diagrama de flujo de una realización de un procedimiento de renormalización del codificador. El procedimiento se realiza mediante lógica de procesamiento, que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la figura 6, la lógica de procesamiento somete a prueba si el valor de R es menor de 100 Hex (bloque 601 de procesamiento). En caso negativo, se realiza el procedimiento. En caso afirmativo, el procedimiento pasa al bloque 602 de procesamiento donde la lógica de procesamiento somete a prueba si el valor de L es menor de 100 Hex. En caso afirmativo, el bloque de procesamiento pasa al bloque 603 de procesamiento, donde se realiza un procedimiento de colocación de bit con el parámetro 0 y luego el procesamiento pasa al bloque 608 de procesamiento. Si la lógica de procesamiento determina que el valor de L es mayor o igual a 100 Hex, la lógica de procesamiento somete a prueba si el valor de L es mayor de 200 Hex. En caso negativo, la lógica de procesamiento ajusta el valor de L al resultado de restar 100 Hex del valor de L e incrementa el valor de los bits pendientes (BO) en uno con el parámetro 1 (bloque 605 de procesamiento) y el procesamiento pasa al bloque 608 de procesamiento. Si el valor de L es mayor de o igual a 200 Hex, el procesamiento pasa al bloque 606 de procesamiento, donde la lógica de procesamiento ajusta el valor de L al resultado de restar 200 Hex del valor L, realiza el procedimiento de colocación de bit (bloque 607 de procesamiento) y pasa al bloque 608 de procesamiento.

Al procesar el bloque 608, la lógica de procesamiento desplaza el valor de R hacia la izquierda en una posición y desplaza el valor de L en una posición. A continuación, el procesamiento pasa al bloque 601 de procesamiento y el procedimiento se repite.

La figura 7 ilustra una realización del procedimiento para llevar a cabo una realización del procedimiento de colocación de bit. El procedimiento de colocación de bit escribe cero o más bits en el flujo de bits. El procedimiento se realiza mediante lógica de procesamiento, que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la figura 7, la lógica de procesamiento comprueba inicialmente si el primer indicador de bit no es igual a cero (bloque 701 de procesamiento). Si el primer indicador de bit se ajusta a 1, entonces la lógica de procesamiento ajusta el primer indicador de bit igual a cero (bloque 702 de procesamiento) y el procesamiento pasa al bloque 704 de procesamiento. En caso negativo, la lógica de procesamiento envía un bit con valor B (bloque 703 de procesamiento) y la lógica de procesamiento pasa al bloque 704 de procesamiento).

En el bloque 704 de procesamiento, la lógica de procesamiento somete a prueba si el valor de los bits pendientes (BO) es mayor de cero. En caso negativo, el procedimiento finaliza. En caso afirmativo, la lógica de procesamiento envía un bit con valor 1-B y decrementa el valor de BO en uno (bloque 705 de procesamiento). Posteriormente, la lógica de procesamiento pasa al bloque 704 de procesamiento.

La figura 8 es un diagrama de flujo de una realización de un procedimiento para codificar un evento antes de la terminación. Este procedimiento puede utilizarse para codificar el fin de sector, así como cualquier otro evento binario que señalice la terminación de la codificación aritmética. El procedimiento se realiza mediante lógica de procesamiento, que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la figura 8, la lógica de procesamiento inicialmente decrementa el valor de R en 2 (bloque 801 de procesamiento). A continuación, la lógica de procesamiento somete a prueba si el valor del evento binario que se está codificando no es igual a cero (bloque 802 de procesamiento). Si el evento es igual a cero, la lógica de procesamiento realiza un procedimiento de renormalización como el mostrado en la figura 6 (bloque 803 de procesamiento), y el procesamiento pasa al bloque 806 de procesamiento. Si el valor del evento binario que va a codificarse no es igual a cero, entonces, la lógica de procesamiento ajusta el valor de L al resultado de añadir el valor de L más el valor de R (bloque 804 de procesamiento), realiza un procedimiento de vaciado del codificador (bloque 805 de procesamiento) y pasa al bloque 806 de procesamiento. En el bloque 806 de procesamiento, la lógica de procesamiento incrementa el valor del contador de eventos en 1 y el procedimiento de codificación finaliza.

Tal como se observa en el procedimiento anterior, en una realización, cuando el valor del evento binario es igual a 1, la codificación aritmética se termina y el procedimiento de vaciado se aplica después de codificar el evento. Cuando se codifica un evento de este tipo, 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 procedimiento para el vaciado en la terminación. El procedimiento se realiza mediante lógica de procesamiento, que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la figura 9, la lógica de procesamiento ajusta inicialmente el valor de R a 2 (bloque 901 de procesamiento). La lógica de procesamiento realiza entonces un procedimiento de renormalización, tal como el procedimiento de renormalización mostrado en la figura 6 (bloque 902 de procesamiento). La lógica de procesamiento realiza entonces el procedimiento de colocación de bit mostrado en la figura 7 en un valor igual al valor de L desplazado nueve lugares a la derecha y se somete a Y con el valor de 1 Hex (bloque 903 de procesamiento). Los resultados de realizar la operación de Y en el contenido desplazado del valor del registro L hacen que se genere el bit en la posición de bit 10.ª (contada a partir del bit significativo reciente) y posteriormente se emita usando el procedimiento de colocación de bit.

Finalmente, la lógica de procesamiento envía dos bits iguales al valor del registro L desplazado siete lugares a la derecha, se somete a Y con un valor de 3 Hex, y luego a O con 1 Hex (bloque 904 de procesamiento). La operación de O con 1 Hex se realiza para añadir el bit de parada.

Funcionamiento del decodificador a modo de ejemplo

La figura 10 es un diagrama de bloques de una realización de un decodificador 1000 aritmético. En referencia a la figura 10, el decodificador 1000 incluye un secuenciador 1005, un estimador 1010 de probabilidad y un motor 1015 de decodificación acoplados entre sí. Una entrada 1020 proporciona un puerto para una secuencia 1025 de información (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 realización, el decodificador 1000 procesa la secuencia de información para generar una secuencia 1035 de eventos. 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 los valores de un solo bit. La secuencia de eventos se proporciona a la salida 1030, que incluye al menos un puerto de salida del decodificador 1000.

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

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

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

Los siguientes diagramas de flujo representan operaciones de decodificación realizadas en un sector por una realización de un decodificador, tal como el decodificador 1000. En una realización, el decodificador realiza la decodificación según los diagramas de flujo representados en las figuras 12, 14A, 14B, 15A o 15B basándose en el valor de un contexto. Los procedimientos ilustrados pueden incorporarse en otros procedimientos, modificarse o adaptarse de otro modo para obtener los beneficios de las mejoras incorporadas en ellos. 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 procedimiento de inicialización de decodificador aritmético. El procedimiento se realiza mediante lógica de procesamiento que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la figura 11, el procedimiento comienza con la lógica de procesamiento que ajusta el rango R a un número predeterminado (bloque 1101 de procesamiento). En una realización, el número predeterminado es 0xff00. Después de inicializar el rango R, la lógica de procesamiento lee dos bytes de datos comprimidos en el registro V (bloque 1102 de procesamiento). En una realización, el registro V almacena los bits comprimidos, un byte cada vez. El registro V puede implementarse para almacenar los datos comprimidos un bit cada vez, pero tendrían que cambiarse en consecuencia las constantes usadas en el procedimiento descrito en el presente documento.

Más específicamente, como se muestra, la lógica de procesamiento lee en un byte y lo desplaza 8 lugares hacia la izquierda y luego obtiene otro byte y lo añade al registro V con una operación aritmética de O. Una vez que los datos comprimidos se han leído en el registro V, la lógica de procesamiento ajusta el valor del registro B a un valor predeterminado. El registro B indica el número de bits adicionales en el registro V que están disponibles para su procesamiento. Cuando el valor en el registro B se hace menor de 0, entonces es necesario recuperar 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 procedimiento para decodificar un evento binario. El procedimiento se realiza mediante lógica de procesamiento que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la figura 12, el procedimiento comienza calculando el tamaño del intervalo para el LPS (bloque 1202 de procesamiento). En una realización, este cálculo se realiza mediante una multiplicación. La multiplicación puede aproximarse utilizando una búsqueda en tabla que se basa en el estado asociado con el contexto (CTX). En una realización, se utiliza una máquina de estados finitos para indicar cuál es la probabilidad dependiendo del estado de la máquina. A continuación, para la búsqueda, el valor del estado y los siguientes dos bits más significativos de R después del bit más significativo de R. En la figura 16A se muestra una tabla a modo de ejemplo para realizar la búsqueda. A continuación también se facilita un método a modo de ejemplo para generar la tabla.

El resultado de la búsqueda en tabla se desplaza en 7 porque esta implementación lee bytes cada vez en lugar de bits. El resultado desplazado de la búsqueda en tabla es el intervalo de subrango del LPS denominado R_{LPS}.

También como parte del bloque 1202 de procesamiento, la lógica de procesamiento calcula el rango de subintervalo para el MPS restando el intervalo de subrango del LPS R_{LPS} del valor del registro R. La lógica de procesamiento ajusta el valor de R igual al resultado de la resta.

Tras el cálculo del intervalo de subrango para el MPS, la lógica de procesamiento somete a prueba si el valor del registro V es mayor que o igual al subintervalo del MPS almacenado en el registro R, lo que indica que el bit actual que está procesándose está en el subrango de LPS (bloque 1203 de procesamiento). En caso negativo, la lógica de procesamiento toma la ruta de MPS y pasa al bloque 1204 de procesamiento, donde la lógica de procesamiento ajusta el valor que está decodificándose (es decir, el resultado que se devuelve) S igual al valor que se define que es el MPS para ese contexto particular y actualiza la máquina de estados para el contexto al siguiente estado indicado en la máquina de estados para el contexto usando la tabla en la figura 16B. En una realización, para un MPS, la actualización de la máquina de estados comprende incrementar el estado en la tabla de estado en uno.

Si la lógica de procesamiento determina que el valor V es mayor o igual que el valor en el registro R, entonces la lógica de procesamiento toma la ruta de LPS y pasa al bloque 1205 de procesamiento, donde el resultado S se ajusta igual al LPS (no al MPS) para el contexto CTX particular, el valor V se ajusta igual al resultado de restar el valor del rango R del valor actual de V, y el rango R se ajusta igual al rango para el LPS, concretamente, R_{LPS} (bloque 1205 de procesamiento).

La lógica de procesamiento también comprueba si el estado para el contexto del evento binario es cero (bloque 1206 de procesamiento). En una realización, el estado 0 es el estado correspondiente a una probabilidad de 50/50. Alternativamente, el estado cero es un estado correspondiente a otra probabilidad tal como, por ejemplo, algo próximo a una probabilidad de 50/50. Si no lo es, entonces el procesamiento pasa al bloque 1208 de procesamiento. Si lo es, la lógica de procesamiento cambia el significado del MPS (bloque 1207 de procesamiento).

Después, el número de estado del contexto se actualiza al siguiente estado usando la tabla en la figura 16B (bloque 1208 de procesamiento) y la lógica de procesamiento realiza un procedimiento de renormalización (bloque 1209 de procesamiento), que se comenta en más detalle a continuación.

La figura 13 es un diagrama de flujo de un procedimiento de renormalización. El procedimiento se realiza mediante lógica de procesamiento que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la figura 13, el procedimiento comienza mediante la lógica de procesamiento que somete a prueba si R es menor de 8000 Hex (bloque 1301 de procesamiento). Si R es mayor de o igual a 8000 Hex, el procedimiento de renormalización finaliza. En caso negativo, la lógica de procesamiento duplica los valores de R y V (bloque 1302 de procesamiento). En una realización, la lógica de procesamiento duplica los valores de R y V al desplazar los bits de R y V una posición hacia la izquierda. El valor de B también se decrementa en 1, ya que el desplazamiento ha provocado que esté disponible un bit menos para su procesamiento. La lógica de procesamiento comprueba entonces si el valor de B es menor de 0 (bloque 1303 de procesamiento). En caso negativo, entonces el procesamiento pasa al bloque 1301 de procesamiento y el procedimiento se repite. Si el valor de B es menor de 0, entonces el procesamiento pasa al bloque 1304 de procesamiento, donde el valor de B se ajusta a 7 y el otro byte que va a procesarse se recupera y se somete a la lógica de O con el contenido actual del registro V. Después, el procesamiento pasa al bloque 1301 de procesamiento y se repite el procedimiento.

Las figuras 14A y 14B ilustran diagramas de flujo para decodificar un evento con equiprobabilidad. La figura 14A puede usarse cuando el tamaño del registro V es mayor de 16 bits, mientras que la figura 14B puede usarse cuando el tamaño del registro V es de 16 bits. Estas implementaciones pueden usarse cuando se recupera un byte cada vez.

Los procedimientos se realizan mediante lógica de procesamiento que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

Estos procedimientos pueden usarse cuando las distribuciones se centran alrededor de cero y la probabilidad de obtener un valor positivo o negativo es aproximadamente la misma. Por ejemplo, pueden usarse cuando se procesa un valor de signo de coeficientes. En lugar de estimar la probabilidad de que sea positiva o negativa, se usan estimaciones fijas que reconocen que la probabilidad es de 50/50. Por tanto, no hay necesidad de realizar una búsqueda en tabla para la multiplicación de R con una probabilidad. Debe observarse que esto no afecta a la terminación.

En referencia a la figura 14A, el procedimiento comienza mediante la lógica de procesamiento que duplica el valor de V y que decrementa el valor de B en 1 (lógica 1401 de procesamiento). La duplicación del valor de V puede realizarse desplazando los bits de V una posición hacia la izquierda.

A continuación, la lógica de procesamiento comprueba si el valor de B es menor de 0 (bloque 1402 de procesamiento). En caso negativo, entonces el procesamiento pasa al bloque 1404 de procesamiento. Si el valor de B es menor de 0, entonces el procesamiento pasa al bloque 1403 de procesamiento, donde el valor de B se ajusta a 7 y el otro byte que va a procesarse se recupera y se somete a la lógica de O con el contenido actual del registro V.

En el bloque 1404 de procesamiento, la lógica de procesamiento somete a prueba si el valor de V es mayor que o igual al valor de R. En caso afirmativo, la lógica de procesamiento ajusta el resultado S a 1 y ajusta el valor de V al resultado de restar el valor R del valor V (bloque 1405 de procesamiento) y el procedimiento finaliza. En caso negativo, la lógica de procesamiento ajusta el resultado S a 0 (bloque 1406 de procesamiento) y el procedimiento finaliza.

En referencia a la figura 14B, el procedimiento comienza mediante la lógica de procesamiento que ajusta el valor V' igual a V, que duplica el valor de V y que decrementa el valor de B en 1 (lógica 1411 de procesamiento). La duplicación del valor de V puede realizarse desplazando los bits de V una posición hacia la izquierda.

A continuación, la lógica de procesamiento comprueba si el valor de B es menor de 0 (bloque 1412 de procesamiento). En caso negativo, entonces el procesamiento pasa al bloque 1414 de procesamiento. Si el valor de B es menor de 0, entonces el procesamiento pasa al bloque 1413 de procesamiento donde el valor de B se ajusta a 7 y el otro byte que va a procesarse se recupera y se somete a la lógica de O con el contenido actual del registro V.

En el bloque 1414 de procesamiento, la lógica de procesamiento somete a prueba si el valor de V es mayor de o igual al valor de R o si V' es mayor de o igual a 8000 Hex. En caso afirmativo, la lógica de procesamiento ajusta el resultado S a 1 y ajusta el valor de V al resultado de restar el valor R del valor V (bloque 915 de procesamiento) y el procedimiento finaliza. En caso negativo, la lógica de procesamiento ajusta el resultado S a 0 (bloque 916 de

procesamiento) y el procedimiento finaliza.

La figura 15A es un diagrama de flujo de una realización para decodificar eventos codificados que indican la terminación de la codificación aritmética. Un evento de este tipo puede comprender un indicador de fin de sector. Con respecto al indicador de fin de sector, puede usarse sintaxis para indicar a un decodificador la presencia de un indicador de fin de sector. En una realización, este procedimiento se realiza para cada macrobloque; sin embargo, solo para el último macrobloque en el sector el resultado indicará un fin de sector (por ejemplo, emite un resultado que es 1).

Puede usarse un evento para señalizar la terminación de la codificación aritmética (para un decodificador) cuando los datos van a seguir la codificación aritmética en el flujo de bits que no está comprimido o está comprimido dentro de otra técnica de codificación diferente de la codificación aritmética. Debe observarse que datos codificados aritméticos adicionales pueden seguir a estos datos no comprimidos o a datos comprimidos con una técnica de codificación no aritmética. Por tanto, el evento para señalizar la terminación puede usarse en casos donde los datos codificados no aritméticos se entrelazan en flujo de bits con datos codificados aritméticos.

El procedimiento se realiza mediante lógica de procesamiento que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la figura 15A, la lógica de procesamiento somete a prueba si el valor de V es menor de 100 Hex (bloque 1501 de procesamiento), indicando de ese modo que se ha alcanzado el último macrobloque en el sector. En caso afirmativo, la lógica de procesamiento ajusta el resultado S, que representa el símbolo decodificado, a 1 (bloque 1502 de procesamiento) y el procedimiento de decodificación para el sector finaliza. En caso negativo, la lógica de procesamiento ajusta el resultado de salida S a 0, ajusta el valor de R al resultado de restar 100 Hex del valor de R, y ajusta el valor de V al resultado de restar 100 Hex del valor de V (lógica 1503 de procesamiento). A continuación, la lógica de procesamiento realiza el procedimiento de renormalización de la figura 3 (bloque 1504 de procesamiento) y el procedimiento finaliza.

Debe observarse que en una realización, puede cambiarse la convención entre MPS y LPS. La figura 15B es un diagrama de flujo de una realización de un procedimiento para codificar un evento antes de la terminación cuando se cambia una convención entre MPS y LPS. El procedimiento puede realizarse mediante lógica de procesamiento, que puede comprender hardware (por ejemplo, conjunto de circuitos, lógica dedicada, etc.), software (tal como el que se ejecuta en un sistema informático de uso general o una máquina dedicada), o una combinación de ambos.

En referencia a la figura 15B, la lógica de procesamiento comienza restando 100 Hex del valor de R (bloque 1511 de procesamiento). La lógica de procesamiento somete entonces a prueba si el valor de V es mayor de o igual al valor de R (bloque 1512 de procesamiento). En caso afirmativo, la lógica de procesamiento ajusta el resultado de salida S, que representa el símbolo decodificado a uno (bloque 1513 de procesamiento) y el procedimiento de decodificación para decodificar el evento antes de que finalice la terminación. Por tanto, no se realiza ninguna renormalización. En caso negativo, la lógica de procesamiento ajusta el resultado de salida S a cero (bloque 1514 de procesamiento) y realiza el procedimiento de renormalización de la figura 13 (bloque 1515 de procesamiento) y el procedimiento finaliza.

Construcción de máquina de estados para la estimación de probabilidad

En el código C a continuación se facilita un procedimiento a modo de ejemplo para construir la máquina de estados en las figuras 16A y 16B.

Código C:

```
# 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);
    si (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 estados 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, inclusive) y un indicador MPS (determina si 0 ó 1 es el MPS).

En una realización, los estados se organizan de manera que los números de estado más altos corresponden a las probabilidades más bajas para el LPS. La máquina de estados se define para aproximar el 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 una tasa de adaptación. Alfa normalmente está en el rango de 0,9 a 1, pero puede extenderse o estar en otros rangos según la adaptación deseada.

En el código anterior, alfa se ajusta 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 observarse que pow(a, b) es el número a para la capacidad b.

La matriz denominada prob64 contiene valores de punto flotante que representan las probabilidades de un LPS asociado con cada estado. Prob64[i] se ajusta a 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 tras la observación de un MPS. Si i es diferente de N-1, el estado aumenta en 1. De otro modo, el estado permanece sin cambios. Dada la combinación de Prob64[i] y Next_state_MPS_64[i], la parte (a) del procedimiento de actualización definido anteriormente está bien aproximado.

Para aproximar la parte (b) del procedimiento de actualización, Next_state_LPS_64[i] debe ajustarse a i-(-log((prob64[i]*alpha+(1-alpha))/prob64[i])/log(alpha)). Este valor, sin embargo, no es un número entero y debe buscarse una aproximación de número entero. En una realización, el valor se redondea al entero más cercano. Sin embargo, en una realización alternativa, para un mejor equilibrio entre redondeo hacia arriba y hacia abajo, se usa una suma variable de tal manera que, en promedio, la diferencia introducida por el redondeo está próxima a cero.

El valor de RTAB[i][j] se calcula de manera que se aproxime a R*prob64[i]. La variable j se determina por el intervalo en el que se encuentra R. La variable j se ajusta igual a 0 para R en [256, 319], a 1 para [320, 383], a 2 para [384, 447] y a 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 j.

Para permitir implementaciones más rápidas, es deseable garantizar que al codificar un MPS, se produzca como máximo una iteración de renormalización. Para este fin, RTAB[i][0] se abrevia a ONE/4. Por tanto, R no puede ser más pequeño que ONE/4 antes de la renormalización. Más generalmente, en una realización, RTAB[i][j] se abrevia a (ONE/4) + (ONE/8)*j pero este caso no ocurre para j diferente de 0 en el presente ejemplo.

Por tanto, usando la técnica descrita anteriormente, puede generarse la tabla de estados de las figuras 16A y 16B, con la excepción de un estado en una realización. En la figura 16A, el estado 63 incluye valores de R de 2, 2, 2, 2. En la figura 16B, una vez en el estado 63, el siguiente estado es el estado 63. Por tanto, independientemente de que

se produzca un LPS o que se produzca un MPS, el estado no cambia. También en la figura 16B, una vez en el estado 62, el estado permanece en el estado 62 tras la aparición de un MPS.

Realizaciones a modo de ejemplo en código fuente

A continuación se facilitan codificadores de muestra y un decodificador de muestra en código C. Estos métodos pueden implementarse usando cualquier dispositivo de procesamiento adecuado para codificar y decodificar los datos (por ejemplo, datos de vídeo). En algunas realizaciones, el procedimiento puede realizarse mediante una combinación de elementos de hardware y software. Pueden realizarse otras adaptaciones. Las funciones para la codificación y decodificación se describen a continuación en forma C.

Codificador:

```
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();
    }
  }
```

Decodificador (basado en byte)

```
  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;
  }
```

Decodificación de fin de sector basado en byte alternativo para su uso cuando se cambia la convención de MPS/LPS

```
int decode_end_of_slice_flag(){
  R -= 0x100;
  if (V >= R)
    bit = 1;
  else {
    bit = 0
    decode_renorm();
  }
  return bit;
}
```

Decodificador (basado en bit)

```
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;
}
```

Decodificación del indicador end_of_slice basada en bit alternativa para su uso cuando se cambia la convención MPS/LPT

```
int decode_end_of_slice_flag(){
    R -= 2;
    if (V >= R)
      bit = 1
    else {
      bit = 0;
      decode_renorm();
    }
    return bit;
  }
```

Debe observarse que en el codificador aritmético descrito anteriormente, hay 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 MPS y LPS a intervalos comprende asignar un 1 a un intervalo y un 0 al otro. En el código fuente anterior, cuando el intervalo se divide para codificar end_of_slice_flag, al MPS (valor 0) se le asigna el subintervalo superior. También es posible asignar el MPS al subintervalo inferior.

El siguiente código ilustra otro codificador a modo de ejemplo. Debe observarse que en el código, S es el número mínimo de bytes en el sector para satisfacer la relación de delimitación descrita anteriormente.

```
void start_encode () {
  send_NAL_first_byte ();
  encode_slice_header ();
  while (! byte_aligned ())
    send_bit (0);
  R = 0x1fe; // rango
  L = 0; // bajo
  BO = 0; // bits pendientes
  C = 0; // contador de eventos
  FB = 1; // primer indicador de bit
}

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); // escribir bytes directamente en la unidad NAL
}

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 anterior, el envío del primer byte, que forma parte del encabezado, a una unidad NAL se realiza para indicar el tipo de datos que deben seguirse. La unidad NAL y su uso se conocen bien en la técnica.

La llamada de función RBSP_to_EBSP() hace que los datos se inserten en el flujo de bits. Más preferiblemente, en una realización, se inserta un byte de Hex 03 después de los bytes de 0000 Hex en 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 usarse como marcadores de resincronización. Cuando un decodificador encuentra el patrón de 000003 Hex, un procedimiento inverso elimina el “03” del flujo de bits.

Aunque un uso de este tipo para los codificadores y decodificadores comentados en el presente documento se produce en la codificación y decodificación de datos de vídeo, un experto en la técnica se dará cuenta de que el codificador y el decodificador descritos en el presente documento pueden utilizarse en cualquier situación en la que se comprime una secuencia de eventos para dar 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 exposición previa del codificador está en el contexto de procesar una secuencia de eventos que comprende múltiples eventos binarios para dar una secuencia de información que comprende al menos un bit, y para el decodificador está en el contexto de procesar una secuencia de información que comprende al menos un bit para dar una secuencia de eventos que comprende múltiples eventos binarios, que el codificador y el decodificador podrán operar en secuencias de eventos y secuencias de información compuestas por eventos, que son de naturaleza M (es decir, cada evento M representa más de un bit de datos) usando las enseñanzas descritas en el presente documento, tal como apreciará un experto en la técnica.

Un sistema informático a modo de ejemplo

La figura 17 es un diagrama de bloques de un sistema informático a modo 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 integrarse en un dispositivo tal como, por ejemplo, un teléfono móvil, para realizar las técnicas descritas en el presente documento.

En referencia a la figura 17, el sistema 1700 informático 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™, Alpha™, etc.

El sistema 1700 comprende además una memoria de acceso aleatorio (RAM) u otro dispositivo 1704 de almacenamiento dinámico (denominado memoria principal) acoplado al bus 1711 para almacenar información e instrucciones que van a ejecutarse por el procesador 1712. La memoria 1704 principal también puede usarse para almacenar variables temporales u otra información intermedia durante la ejecución de instrucciones mediante el procesador 1712.

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

El sistema 1700 informático también puede acoplarse a un dispositivo 1721 de visualización, como un tubo de rayos catódicos (CRT) o una pantalla de cristal líquido (LCD), acoplado al bus 1711 para presentar visualmente la información al usuario de un ordenador. Un dispositivo 1722 de entrada alfanumérico, que incluye teclas alfanuméricas y otras, también puede estar acoplado al bus 1711 para comunicar información y selecciones de comando al procesador 1712. Un dispositivo de entrada de usuario adicional es el control 1723 de cursor, como un ratón, bola de seguimiento, panel táctil, lápiz, o teclas de dirección del cursor, acoplado al bus 1711 para comunicar información de dirección y selecciones de comando al procesador 1712, y para controlar el movimiento del cursor en la pantalla 1721.

Otro dispositivo que puede acoplarse al bus 1711 es el dispositivo 1724 de copia impresa, que puede usarse para

imprimir instrucciones, datos u otra información en un medio, como papel, película o tipos de medios similares. Además, un dispositivo de grabación y reproducción de sonido, tal como un altavoz y/o micrófono, puede acoplarse opcionalmente al bus 1711 para la interfaz de audio con el sistema 1700 informático. Otro dispositivo que puede acoplarse al bus 1711 es una capacidad 1725 de comunicación alámbrica/inalámbrica para comunicación con un teléfono, dispositivo portátil u otro dispositivo.

Debe observarse que en la presente invención puede usarse cualquiera o todos los componentes del sistema 1700 y el hardware asociado. Sin embargo, puede apreciarse que otras configuraciones del sistema informático pueden incluir algunos o todos los dispositivos.

Aunque muchas alteraciones y modificaciones de la presente invención sin duda serán evidentes para un experto habitual en la técnica después de haber leído la descripción anterior, debe entenderse que no se pretende que cualquier realización particular mostrada y descrita a modo de ilustración se considere limitativa. Por tanto, no se pretende que las referencias a detalles de varias realizaciones limiten el alcance de las disposiciones, que en sí mismas enumeran solo aquellas características consideradas esenciales para la invención.

REIVINDICACIONES

1. Un método de decodificación aritmética que usa codificación aritmética binaria adaptativa al contexto que comprende:

introducir una secuencia (1025) de información de uno o más bits, incluyendo la secuencia (1025) de información datos codificados aritméticamente;

almacenar un valor de bits de la secuencia (1025) de información que incluye datos codificados aritméticamente en un registro (1070) de valores;

transmitir el uno o más bits a un motor (1015) de decodificación y transmitir un identificador de contexto para un evento de una secuencia de eventos a un estimador (1010) de probabilidad;

generar una estimación de probabilidad basándose en valores del contexto recibido, estimación de probabilidad que se envía al motor (1015) de decodificación y es usada por el motor (1015) de decodificación para generar el evento;

asignar un valor a un rango para un MPS, basándose en un valor de estado como la estimación de probabilidad asociada con el identificador de contexto y un valor almacenado en un registro (1065) de rangos y enviar cada evento binario generado al estimador (1010) de probabilidad y a un secuenciador (1005);

actualizar un estado interno del estimador (1010) de probabilidad basándose en el valor recibido a partir del motor (1015) de decodificación;

determinar un valor de un intervalo de subrango para un LPS basándose en un resultado de una búsqueda en tabla que tiene como entradas un valor de un estado asociado con el identificador de contexto y un valor almacenado en el registro (1065) de rangos, y actualizar el valor del registro (1065) de rangos restando el intervalo de subrango del LPS del valor del registro;

determinar un valor del evento binario como un resultado de decodificación basándose en el resultado de comparación entre el valor del registro (1070) de valores y el valor del rango para el MPS;

y si el valor del registro (1070) de valores es mayor o igual que el valor del rango para el MPS,

restar el valor del rango para el MPS del valor del registro (1070) de valores, ajustar el valor del registro (1070) de valores igual al resultado, ajustar el resultado de decodificación al LPS para el contexto, ajustar el rango igual al intervalo de subrango para el LPS, comprobar si el estado para el contexto del evento binario es igual a cero para determinar si cambiar el significado del MPS, y actualizar el número de estado del contexto al siguiente estado para el LPS indicado en la máquina de estados para el contexto;

y si el valor del registro (1070) de valores es menor que el valor del rango para el MPS,

mantener el valor del registro (1070) de valores sin cambios, ajustar el resultado decodificado al valor que se define que es el MPS para ese contexto particular, y actualizar la máquina de estados para el contexto al siguiente estado para el MPS indicado en la máquina de estados para el contexto,

en el que la pluralidad de eventos son los eventos binarios convertidos a partir de valores enteros que representan los bloques de muestra transformados y cuantificados de datos de vídeo, en el que el flujo de bits incluye datos codificados de un sector que incluye una indicación de fin de sector que indica la terminación 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 incluyen la indicación de fin de sector codificada en el flujo de bits identificando un patrón asociado con los bytes de relleno y no realizando decodificación aritmética en el uno o más bytes de relleno,

usándose el uno o más bytes de relleno para restringir 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 el que la restricción adopta la forma de la siguiente combinación lineal:

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

donde

e es la cantidad 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. Un decodificador (1000) aritmético que usa codificación aritmética binaria adaptativa al contexto que comprende:

una sección (1020) de entrada para introducir una secuencia de información de uno o más bits, incluyendo la secuencia (1025) de información datos codificados aritméticamente;

un secuenciador (1005) para transmitir el uno o más bits a un motor de decodificación y para transmitir un identificador de contexto para un evento de una secuencia de eventos a un estimador (1010) de probabilidad;

el estimador (1010) de probabilidad para generar una estimación de probabilidad basándose en valores del contexto recibido, estimación de probabilidad que se envía al motor (1015) de decodificación y es utilizada por el motor de decodificación para generar el evento;

y el motor (1015) de decodificación que incluye un registro (1070) de valores que almacena un valor de bits a partir de la secuencia de información y un registro (1065) de rangos para asignar un valor a un rango para un MPS, y configurado para enviar cada evento binario generado al estimador (1010) de probabilidad y al secuenciador (1005),

en el que el secuenciador, el estimador (1010) de probabilidad y el motor de decodificación están acoplados entre sí, y el estimador (1010) de probabilidad actualiza su estado interno basándose en el valor del evento binario recibido a partir del motor (1015) de decodificación,

en el que el motor (1015) de decodificación está configurado para determinar el valor de un intervalo de subrango para un LPS basándose en el resultado de una búsqueda en tabla que tiene como entradas un valor de un estado asociado con el identificador de contexto y un valor almacenado en el registro (1065) de rangos, y

está configurado para actualizar el valor del registro (1065) de rangos restando el intervalo de subrango del LPS del valor del registro,

y el motor de decodificación está configurado además para determinar un valor del evento binario como resultado de decodificación, basándose en un resultado de comparación entre el valor del registro de valores y el valor del rango para el MPS;

y si el registro de valores es mayor o igual que el valor del rango para el MPS, está configurado para

restar el valor del rango para el MPS del valor del registro de valores y el valor del registro de valores se ajusta igual al resultado,

y el motor de decodificación está configurado para fijar el resultado de decodificación al LPS para el contexto, y el rango se fija igual al intervalo de subrango para el LPS

y está configurado para comprobar si el estado para el contexto del evento binario es igual a cero para determinar si se cambia el significado del MPS,

y está configurado para actualizar el número de estado del contexto al siguiente estado para el LPS indicado en la máquina de estados para el contexto,

y si el valor del registro de valores es menor que el valor del rango para el MPS, está configurado para mantener el valor del registro de valores sin cambios,

y está configurado para ajustar el resultado de decodificación al valor que se define que es el MPS para ese contexto particular y

para actualizar la máquina de estados para el contexto al siguiente estado para el MPS indicado en la máquina de estados para el contexto,

en el que la pluralidad de eventos son los eventos binarios convertidos a partir de valores enteros que representan los bloques de muestra transformados y cuantificados de datos de vídeo, en el que el flujo de bits incluye datos codificados de un sector que incluye una indicación de fin de sector que indica la terminación de la codificación aritmética; y

reconoce uno o más bytes de relleno precedidos por cero o más bits de alineación añadidos después de los datos codificados que incluyen la indicación de fin de sector codificada en el flujo de bits identificando un patrón asociado con los bytes de relleno y no realizando decodificación aritmética en el uno o más bytes de relleno,

usándose el uno o más bytes de relleno para restringir 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 el que la restricción adopta la forma de la siguiente combinación lineal:

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

e es la cantidad 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.

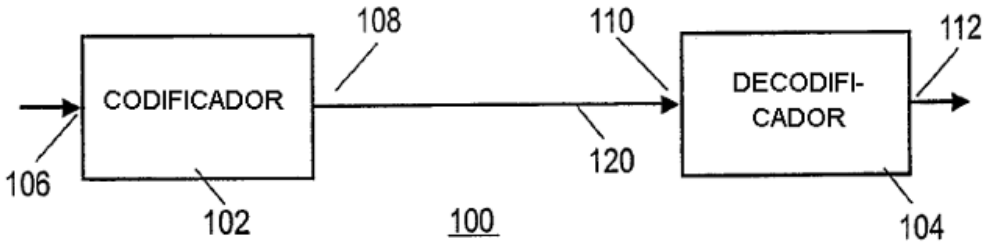
CODIFICADOR
DECODIFI-
CADOR
106
102
108
110
112
120
104
100

FIG. 1

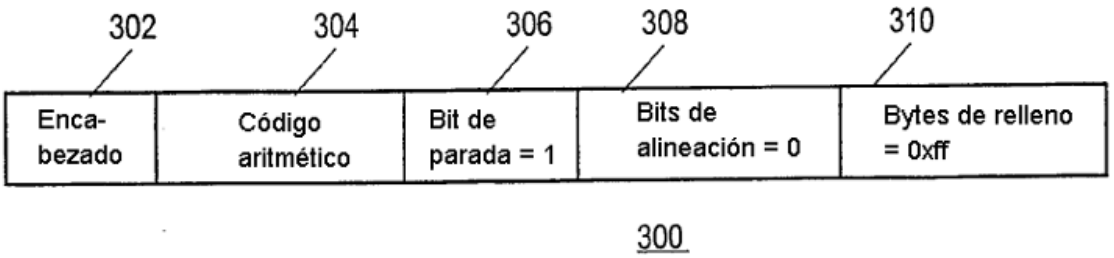
302
304
306
308
310
Enca-
bezado
Código
aritmético
Bit de
parada = 1
Bits de
alineación = 0
Bytes de relleno
= 0xff
300

FIG. 3

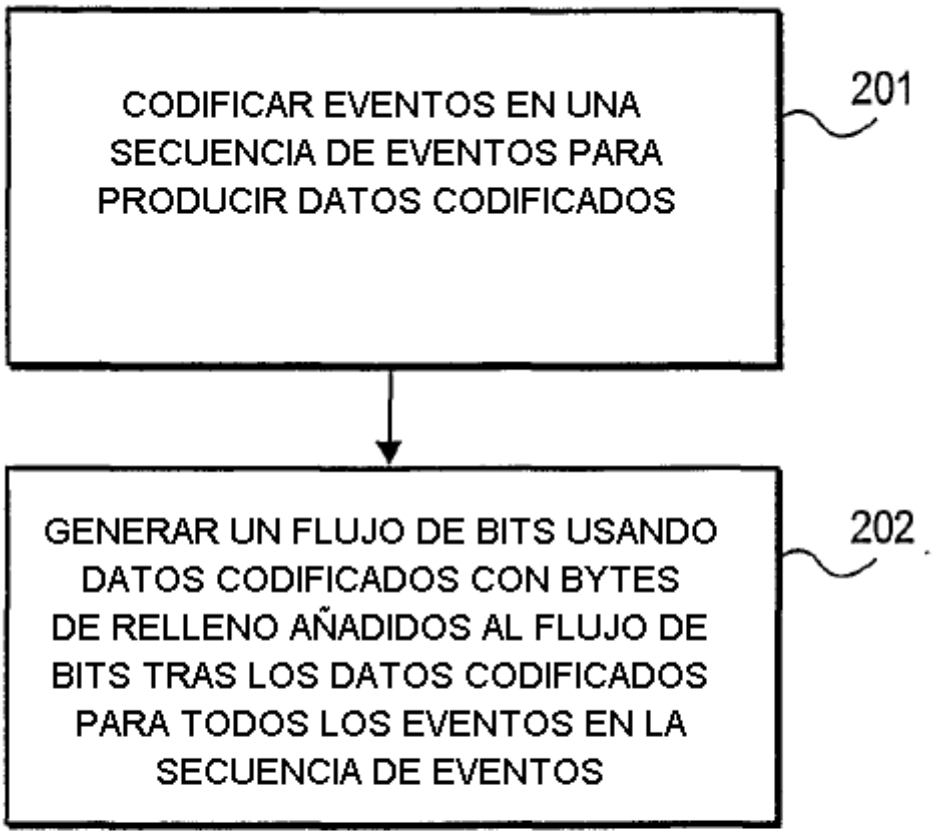
CODIFICAR EVENTOS EN UNA SECUENCIA DE EVENTOS PARA PRODUCIR DATOS CODIFICADOS
201
GENERAR UN FLUJO DE BITS USANDO DATOS CODIFICADOS CON BYTES DE RELLENO AÑADIDOS AL FLUJO DE BITS TRAS LOS DATOS CODIFICADOS PARA TODOS LOS EVENTOS EN LA SECUENCIA DE EVENTOS
202

FIG. 2

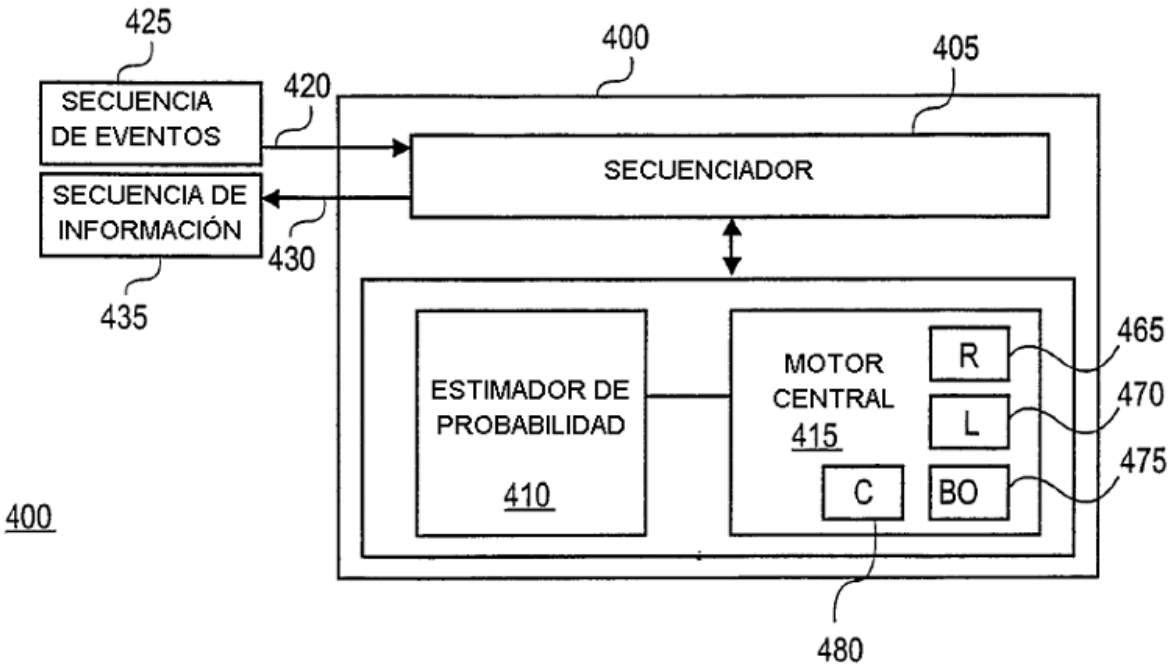
425
SECUENCIA DE EVENTOS
420
SECUENCIA DE INFORMACIÓN
435
430
400
405
SECUENCIADOR
ESTIMADOR DE PROBABILIDAD
410
MOTOR CENTRAL
415
R
L
C
BO
465
470
475
480
400

FIG. 4

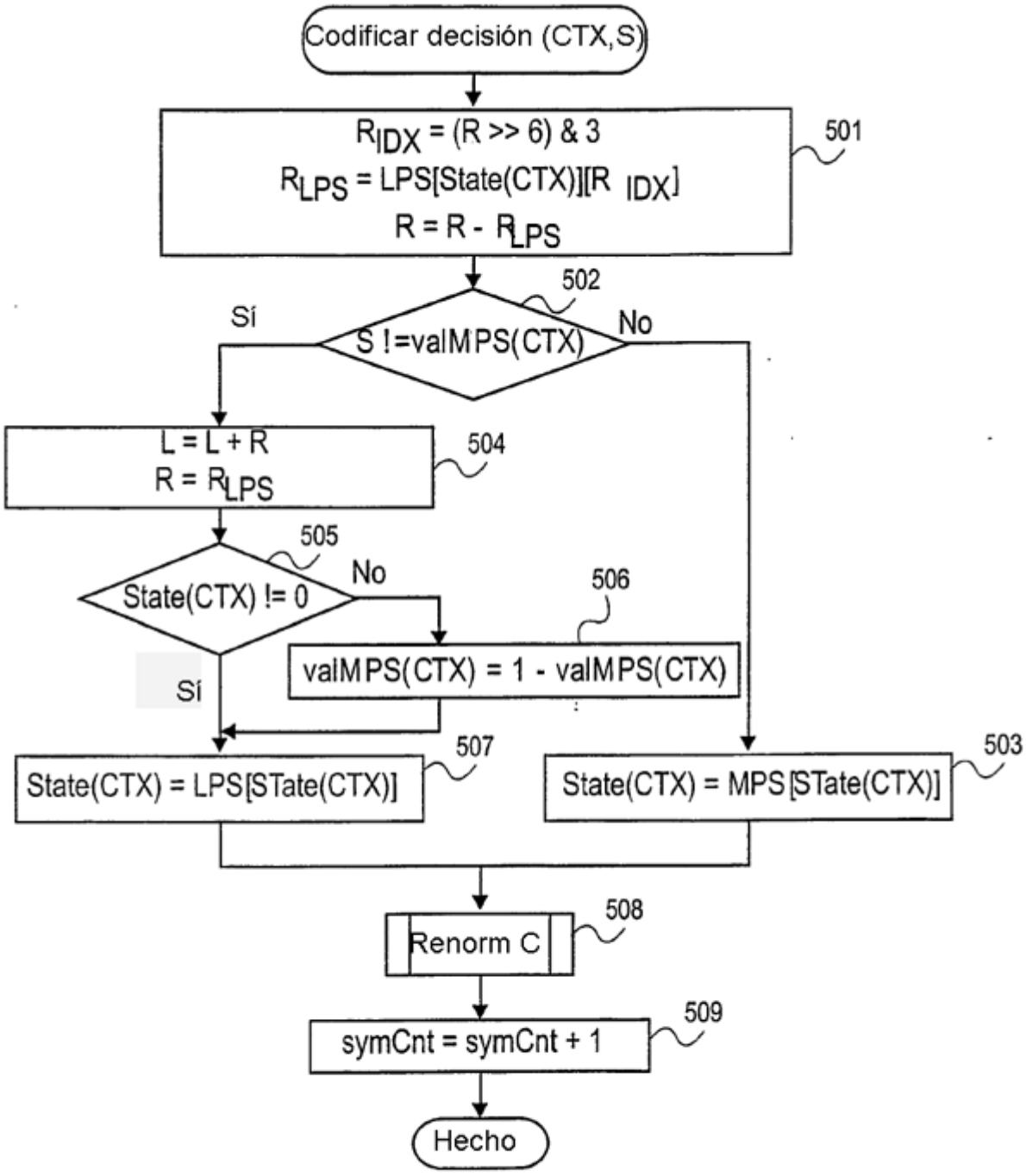
Codificar decisión (CTX,S)
$R_{IDX} = (R >> 6) \& 3$
$R_{LPS} = LPS[State(CTX)][R_{IDX}]$
$R = R - R_{LPS}$
501
502
Sí
S !=valMPS(CTX)
No
L = L + R
$R = R_{LPS}$
504
505
State(CTX) != 0
No
506
valMPS(CTX) = 1 - valMPS(CTX)
Sí
507
State(CTX) = LPS[STate(CTX)]
503
State(CTX) = MPS[STate(CTX)]
508
Renorm C
509
symCnt = symCnt + 1
Hecho

FIG. 5

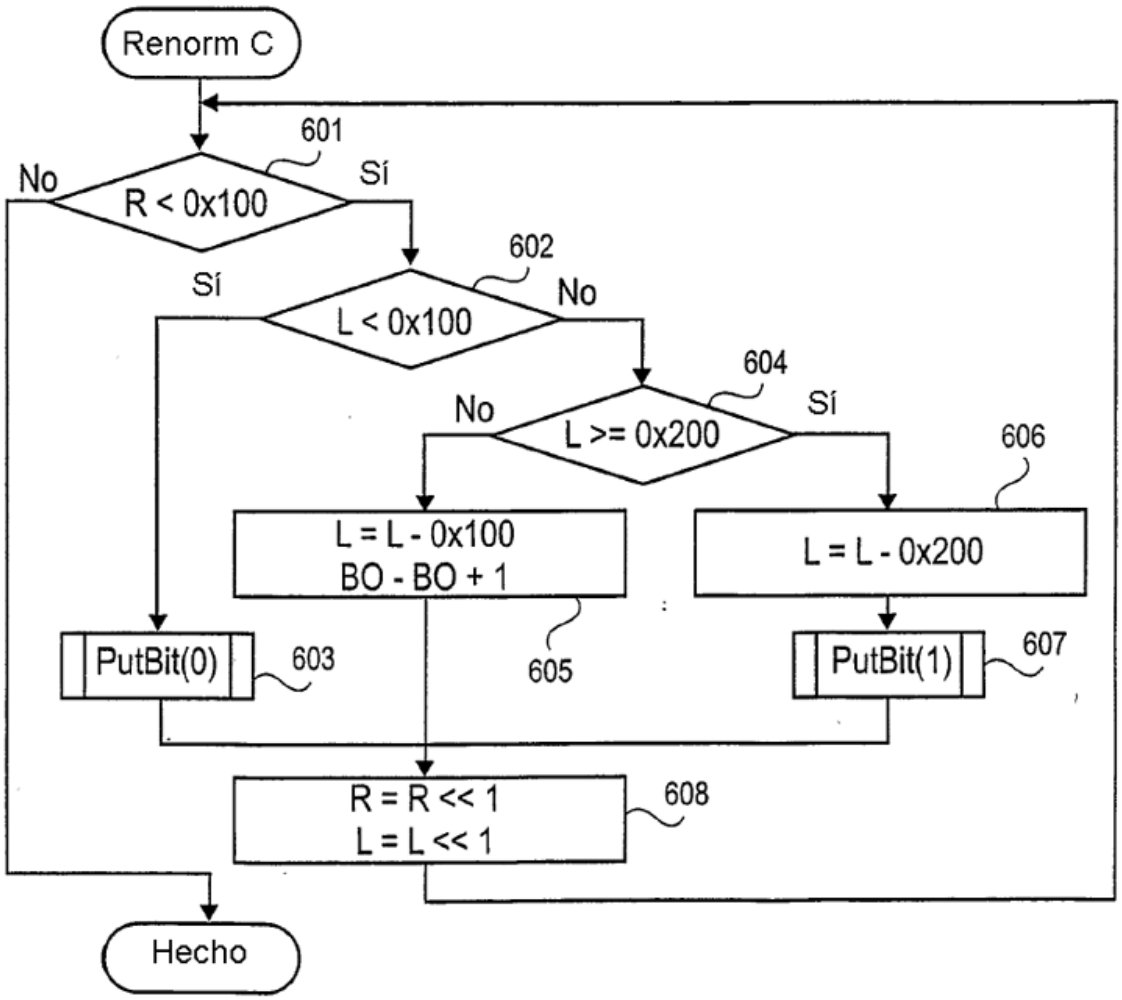
Renorm C
601
No
R < 0x100
Sí
602
Sí
L < 0x100
No
604
No
L >= 0x200
Sí
606
L = L - 0x100
BO - BO + 1
L = L - 0x200
PutBit(0)
603
605
PutBit(1)
607
R = R << 1
L = L << 1
608
Hecho

FIG. 6

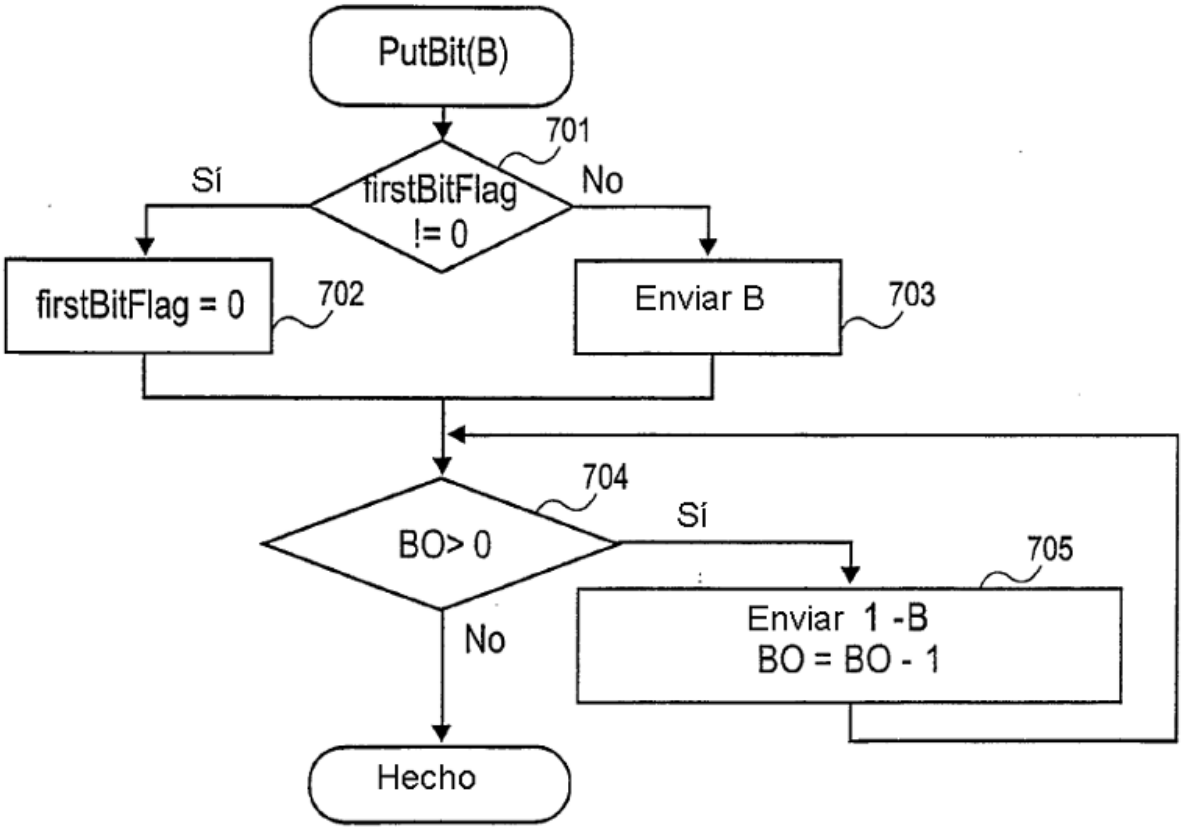
PutBit(B)
701
firstBitFlag
!= 0
Sí
No
firstBitFlag = 0
702
Enviar B
703
704
BO> 0
Sí
705
Enviar 1 -B
BO = BO - 1
No
Hecho

FIG. 7

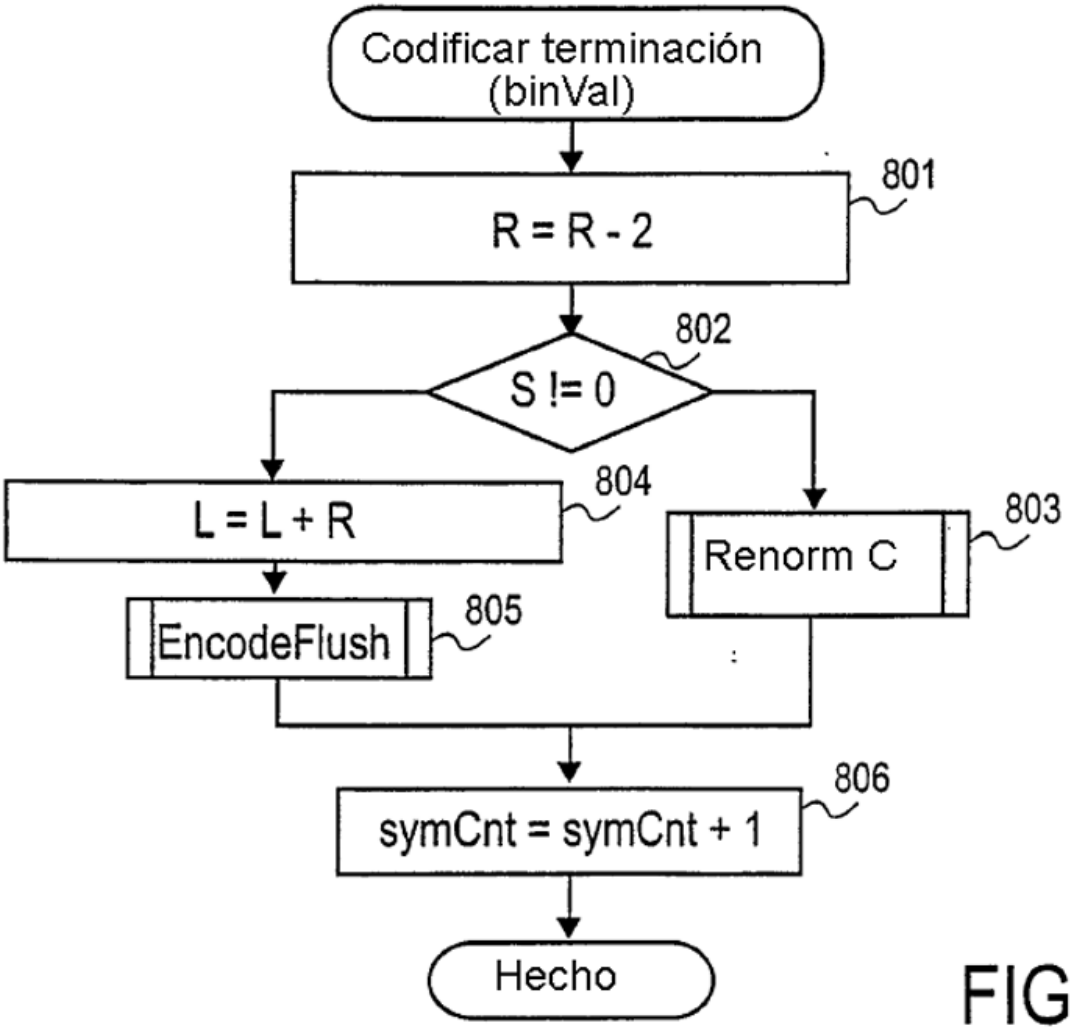
Codificar terminación
(binVal)
801
R = R - 2
802
S != 0
804
L = L + R
803
Renorm C
805
EncodeFlush
806
symCnt = symCnt + 1
Hecho

FIG. 8

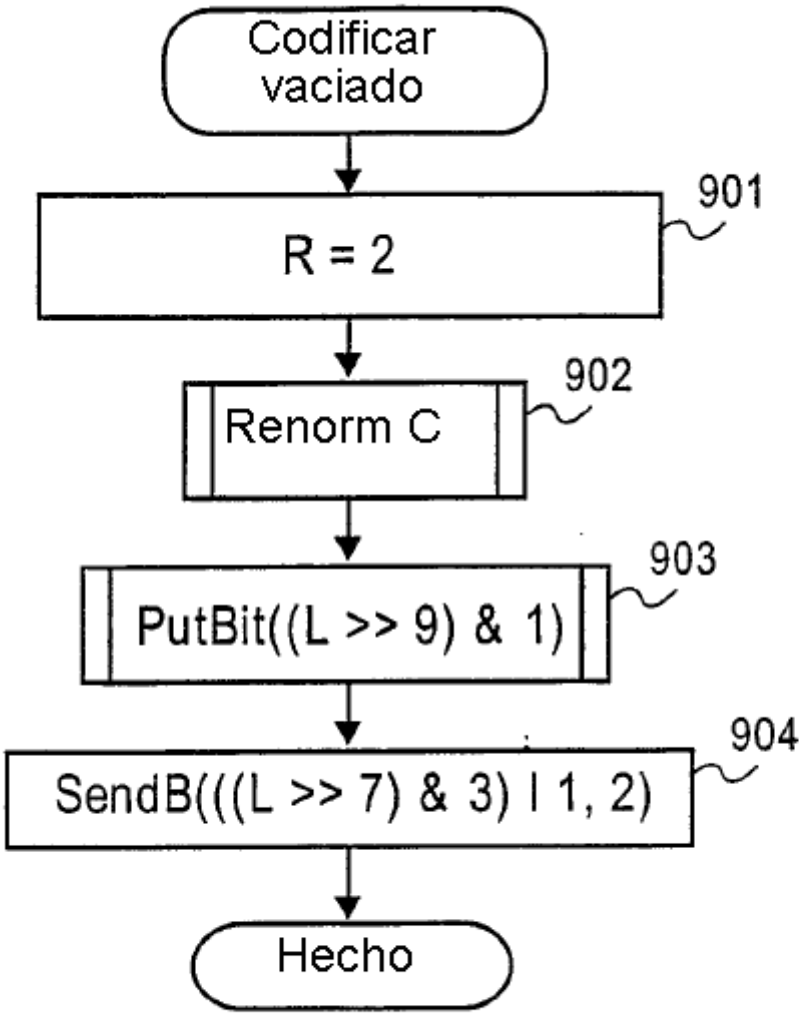
Codificar vaciado
R = 2
901
Renorm C
902
PutBit((L >> 9) & 1)
903
SendB(((L >> 7) & 3) | 1, 2)
904
Hecho

FIG. 9

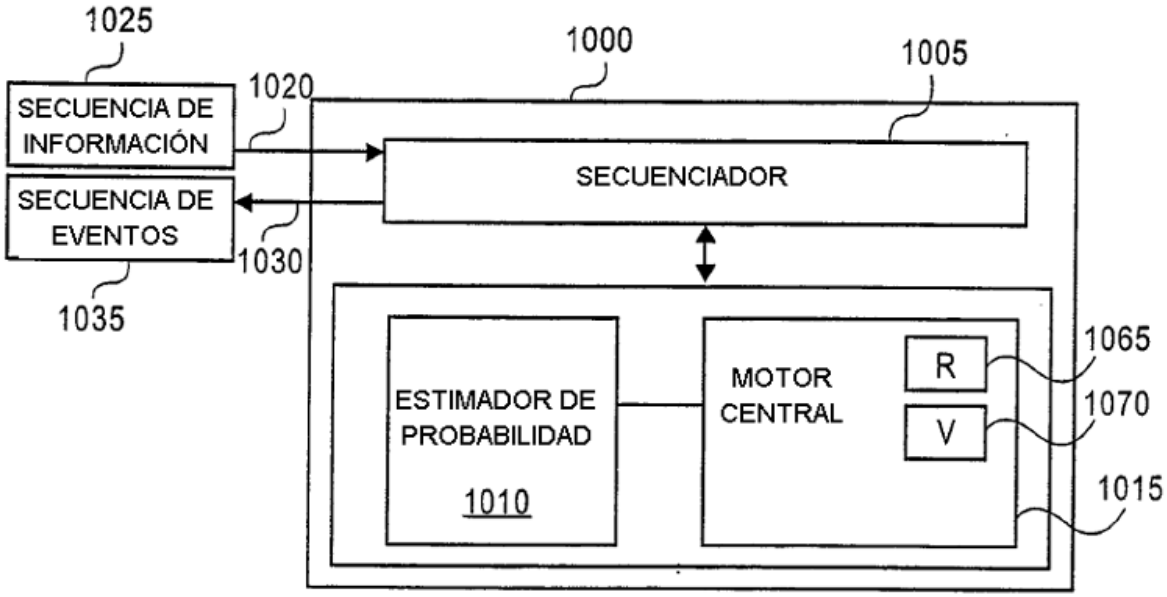
1025
SECUENCIA DE INFORMACIÓN
1020
SECUENCIA DE EVENTOS
1030
1035
1000
1005
SECUENCIADOR
ESTIMADOR DE PROBABILIDAD
1010
MOTOR CENTRAL
R
V
1065
1070
1015

FIG. 10

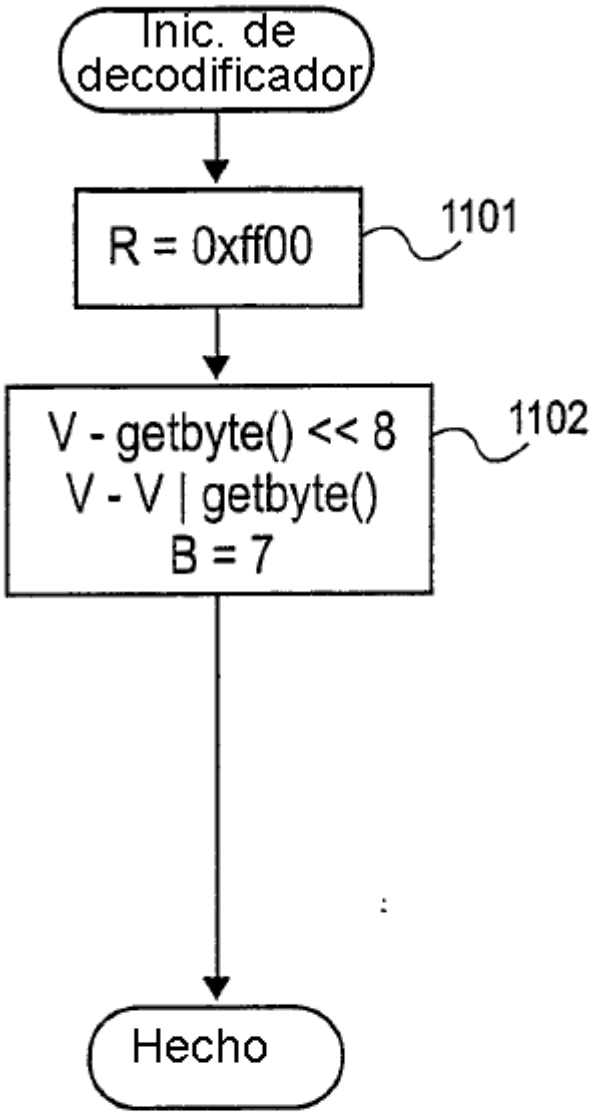
Inic. de decodificador
R = 0xff00
1101
V - getbyte() << 8
V - V | getbyte()
B = 7
1102
Hecho

FIG. 11

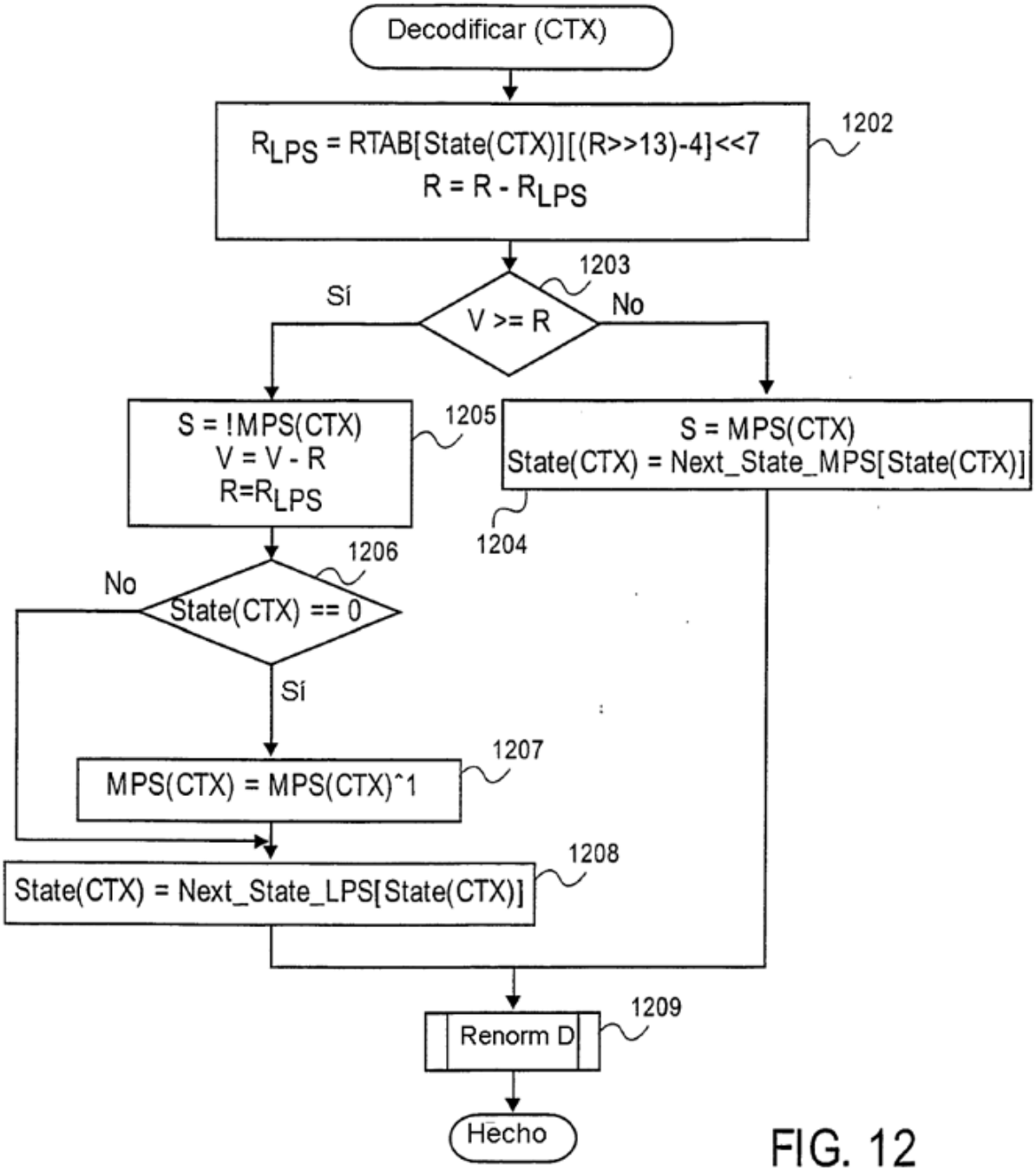
Decodificar (CTX)
R_{LPS} = RTAB[State(CTX)][(R>>13)-4]<<7
R = R - R_{LPS}
1202
1203
Sí
V >= R
No
S = !MPS(CTX)
V = V - R
R=R_{LPS}
1205
S = MPS(CTX)
State(CTX) = Next_State_MPS[State(CTX)]
1204
1206
No
State(CTX) == 0
Sí
1207
MPS(CTX) = MPS(CTX)^1
1208
State(CTX) = Next_State_LPS[State(CTX)]
1209
Renorm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

FIG. 12

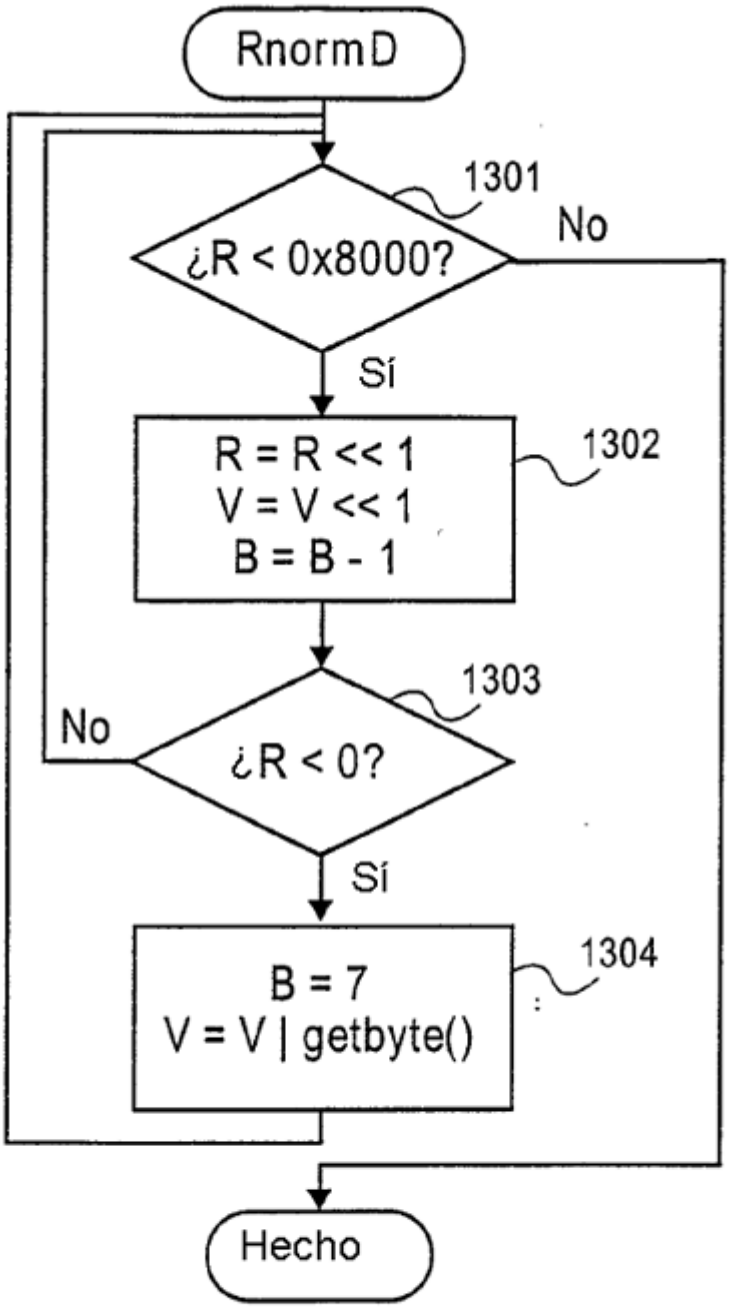
RnormD
1301
¿R < 0x8000?
No
Sí
R = R << 1
V = V << 1
B = B - 1
1302
1303
No
¿R < 0?
Sí
B = 7
V = V | getbyte()
1304
Hecho

FIG. 13

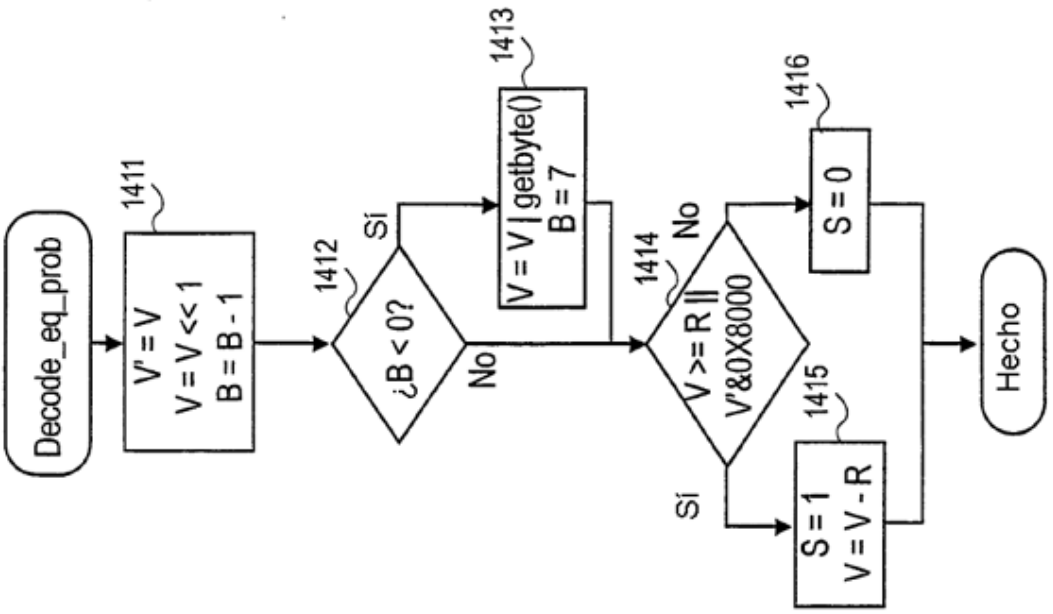

FIG. 14B

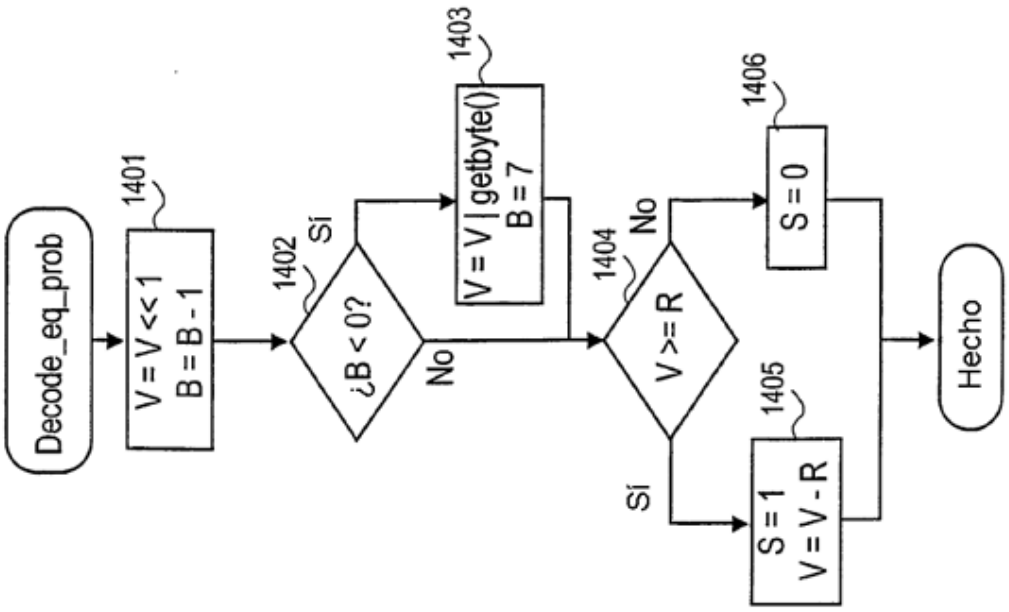

FIG. 14A

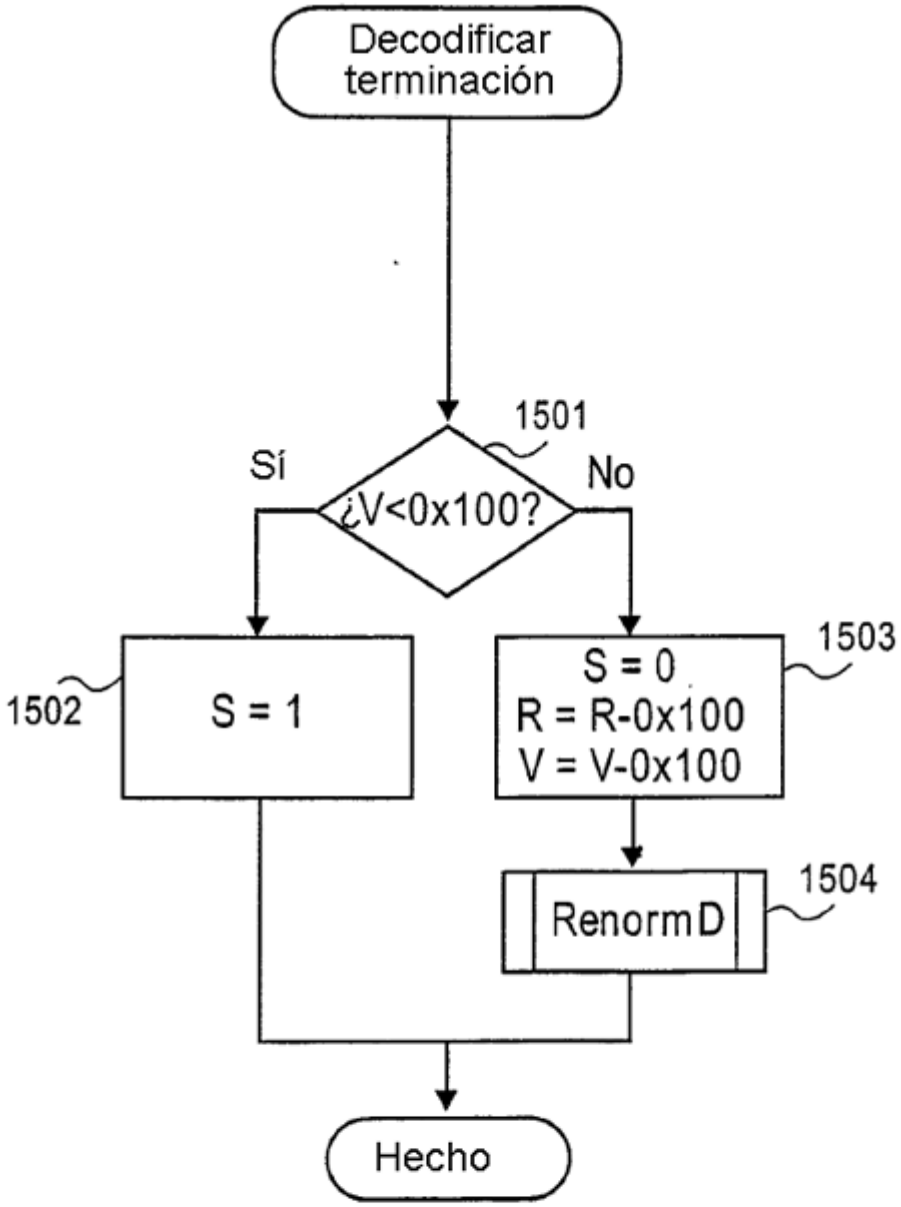
Decodificar terminación
1501
Sí
¿V<0x100?
No
1502
S = 1
1503
S = 0
R = R-0x100
V = V-0x100
1504
RenormD
Hecho

FIG. 15A

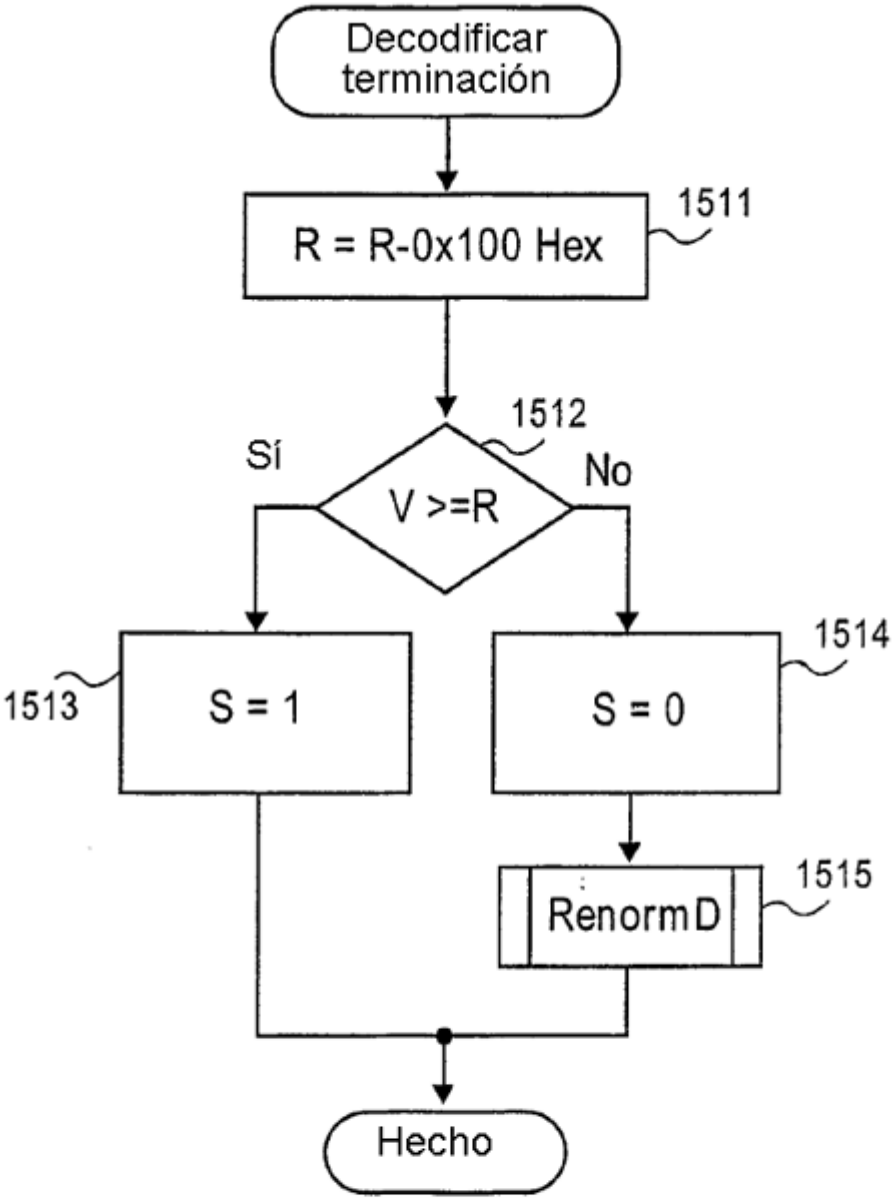
Decodificar terminación
R = R-0x100 Hex
1511
1512
Sí
No
V >=R
1513
S = 1
S = 0
1514
RenormD
1515
Hecho

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

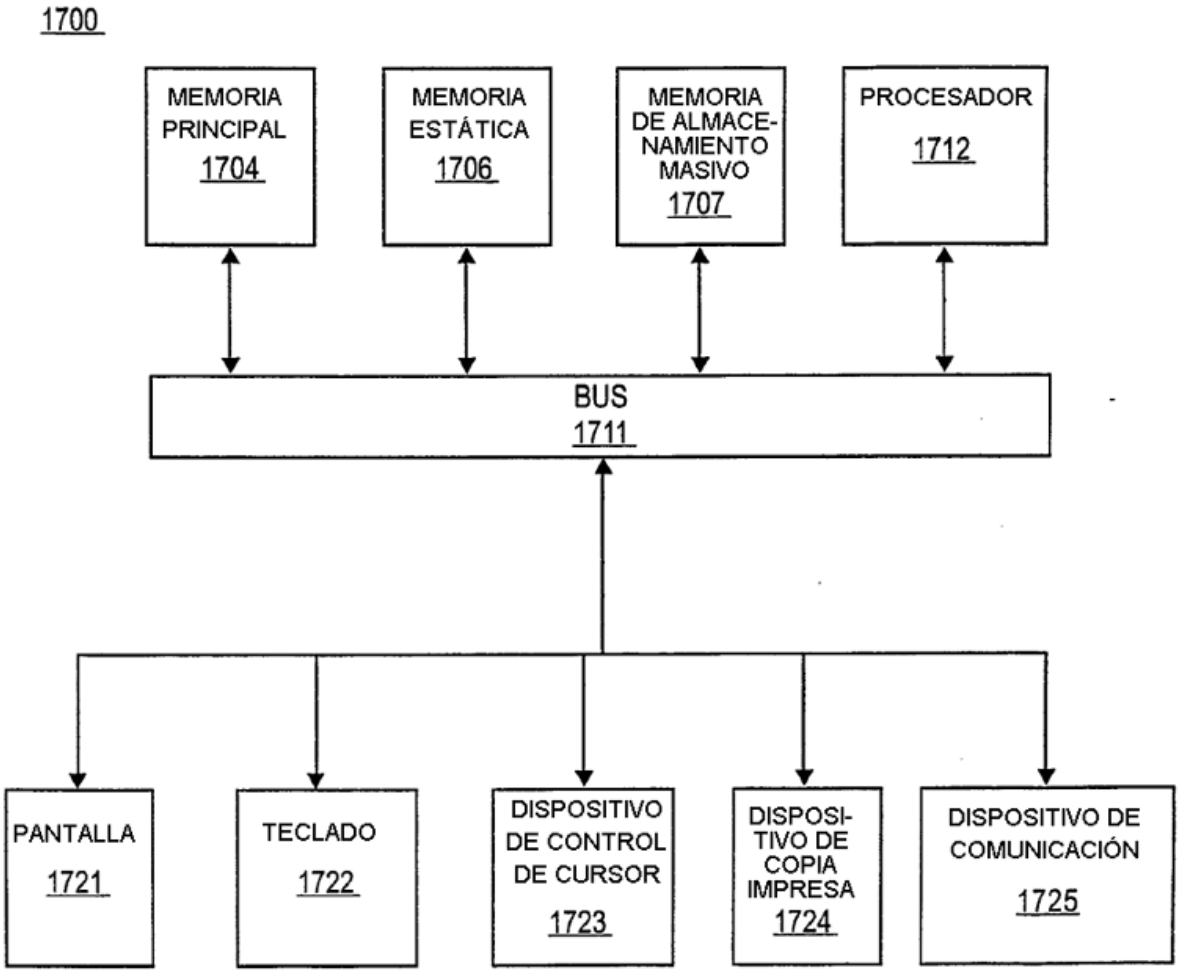
1700
MEMORIA PRINCIPAL 1704
MEMORIA ESTÁTICA 1706
MEMORIA DE ALMACE-NAMIENTO MASIVO 1707
PROCESADOR 1712
BUS 1711
PANTALLA 1721
TECLADO 1722
DISPOSITIVO DE CONTROL DE CURSOR 1723
DISPOSI-TIVO DE COPIA IMPRESA 1724
DISPOSITIVO DE COMUNICACIÓN 1725

FIG. 17