UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T G.728 – Anexo G

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

(11/94)

ASPECTOS GENERALES DE LOS SISTEMAS DE TRANSMISIÓN DIGITAL

CODIFICACIÓN DE SEÑALES VOCALES A 16 kbit/s UTILIZANDO PREDICCIÓN LINEAL CON EXCITACIÓN POR CÓDIGO DE BAJO RETARDO

ANEXO G: ESPECIFICACIÓN DE COMA FIJA A 16 kbit/s

Recomendación UIT-T G.728 – Anexo G

(Anteriormente «Recomendación del CCITT»)

PREFACIO

El UIT-T (Sector de Normalización de las Telecomunicaciones) es un órgano permanente de la Unión Internacional de Telecomunicaciones (UIT). Este órgano estudia los aspectos técnicos, de explotación y tarifarios y publica Recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

La Conferencia Mundial de Normalización de las Telecomunicaciones (CMNT), que se celebra cada cuatro años, establece los temas que han de estudiar las Comisiones de Estudio del UIT-T, que a su vez producen Recomendaciones sobre dichos temas.

La aprobación de Recomendaciones por los Miembros del UIT-T es el objeto del procedimiento establecido en la Resolución N.° 1 de la CMNT (Helsinki, 1 al 12 de marzo de 1993).

La Recomendación UIT-T G.728 – Anexo G ha sido preparada por la Comisión de Estudio 15 (1993-1996) del UIT-T y fue aprobada por el procedimiento de la Resolución N.° 1 de la CMNT el 1 de noviembre de 1994.

NOTA

En esta Recomendación, la expresión «Administración» se utiliza para designar, en forma abreviada, tanto una administración de telecomunicaciones como una empresa de explotación reconocida de telecomunicaciones.

ÍNDICE

Página

CODIFICACIÓN DE SEÑALES VOCALES A 16 kbit/s UTILIZANDO PREDICCIÓN LINEAL CON EXCITACIÓN POR CÓDIGO DE BAJO RETARDO

(Ginebra, 1992)

Anexo G

Especificación de coma fija a 16 kbit/s

(Ginebra, 1994)

(Este anexo es parte integrante de la presente Recomendación)

G.1 Introducción

El objetivo de este anexo es describir con suficiente detalle cómo puede aplicarse en un dispositivo aritmético de coma fija la Recomendación UIT-T G.728 para LD-CELP a 16 kbit/s. Una implementación de coma fija basada en esta descripción debería ser capaz de interfuncionar plenamente con una versión de coma flotante de la Recomendación G.728 y de producir una señal de salida de calidad equivalente, tanto si dicha señal es una señal vocal o una señal de datos dentro de banda. Por aritmética de coma fija se entiende aquella que utiliza un tamaño de palabras de 16 bits. La mayoría de los dispositivos de 16 bits tienen además otros tamaños de palabras. Por ejemplo, el producto de dos palabras de 16 bits es una palabra de 32 bits, por lo que el registro de productos de un dispositivo como ése tiene una anchura típica de 32 bits. El acumulador almacena la suma de productos, por lo que también debe tener una anchura de por lo menos 32 bits. Así pues, aunque se hable de una «implementación de 16 bits», algunas de las variables de estado internas tienen una precisión distinta de la de 16 bits.

El presente anexo pretende dar una descripción completa y exacta a nivel de bit, de todas las operaciones necesarias para la implementación de la Recomendación G.728 en un procesador de señales digitales de coma fija de 16 bits, que tenga un registro de productos de 32 bits y por lo menos dos acumuladores de 32 bits (o superiores). En muchas ocasiones, a lo largo del anexo, se indican posibles métodos alternativos de efectuar las operaciones con los que se obtiene el mismo resultado exacto. En tales casos puede elegirse un método alternativo, pero si no se llega al mismo resultado exacto con todas las entradas posibles, no deberá efectuarse esa sustitución. El número de métodos alternativos posibles es muy grande, por lo que la mayoría de ellos no se han indicado.

Este anexo consta de siete subcláusulas. La primera subcláusula es una introducción y contiene información adicional sobre el procesamiento de las señales de coma fija y los convenios utilizados a lo largo del anexo. La segunda subcláusula contiene información sobre cambios de algoritmo que se introdujeron especialmente para la implementación en coma fija de la Recomendación G.728. En la tercera subcláusula se da un seudocódigo de coma fija para los restantes módulos del codificador. En la cuarta subcláusula se hace un resumen global de las representaciones de las variables de estado del codificador de coma fija. Las últimas subcláusulas contienen tablas relativas al adaptador de ganancia vectorial hacia atrás.

G.1.1 Principios generales

Este anexo es un anexo a la Recomendación UIT-T G.728, por lo que resulta innecesario repetir todos los detalles y análisis de la misma. Cuando así convenga, se examinarán algunos de los detalles. En la Recomendación se da la totalidad de los detalles de cálculo para una implementación de coma flotante. Si los detalles de cálculo permanecen inalterados, salvo por lo que se refiere a la sustitución de las operaciones aritméticas de coma flotante por las de coma fija, no se indican dichos detalles de cálculo en el presente anexo.

Los cambios más importantes al pasar de la versión del codificador en coma flotante a la presente son:

1) la introducción de tipos diferentes de operaciones aritméticas y precisiones para las variables de estado;

2) un método modificado, pero matemáticamente equivalente, de adaptación de ganancia vectorial hacia atrás; y

3) la introducción de precisión variable en el cálculo de los coeficientes del predictor de la recursión de Levinson-Durbin.

En el resto de esta primera subcláusula del anexo se dan detalles sobre las diferentes representaciones numéricas y la aritmética de la coma fija. La segunda subcláusula contiene detalles sobre los dos principales cambios algorítmicos mencionados más arriba: la adaptación de ganancia vectorial hacia atrás y la recursión de Levinson-Durbin. En la tercera subcláusula se da un seudocódigo para el módulo de ventanización híbrida, bloque 49 de la Recomendación G.728. El algoritmo para este módulo no se cambia, pero la implementación se complica por la utilización de la aritmética de coma fija. El referido seudocódigo es un buen ejemplo de los tipos de cambio que es preciso introducir en los otros módulos del codificador. La cuarta subcláusula del anexo contiene un cuadro, que se corresponde con el Cuadro 2/G.728, en el que se da la representación numérica de todas las variables de estado utilizadas en el codificador y el decodificador.

Por coherencia, en todas las representaciones de este anexo se supone que a lo largo del mismo se utiliza la aritmética de complemento a 2. Pueden emplearse representaciones alternativas, que produzcan resultados matemáticos equivalentes, para implementar el codificador.

G.1.2 Representación numérica

La unidad básica de una implementación de coma fija de 16 bits es la palabra de 16 bits. Cuando se representan enteros puros, la palabra tiene una gama de –32768 a +32767. La representación del 1 viene dada por 0000000000000001 y la representación de –32 768 es 1000000000000000. El bit situado más a la derecha representa aquí el bit menos significativo (LSB, *least significant bit*) y el situado más a la izquierda representa el bit más significativo (MSB, *most significant bit*). En la aritmética de complemento a 2, si el MSB es 0, el número es positivo mientras que si el MSB es 1, el número es negativo. Los bits se pueden numerar de 0 a 15, siendo el bit 0 el LSB y el bit 15 el MSB.

Para representar números con partes fraccionarias es preciso asignar una coma decimal entre dos de los bits. Por ejemplo, para representar números entre –1,0 y +1,0 se asignaría la coma decimal entre los bits 14 y 15. Este formato particular se llama Q15, porque hay 15 bits a la derecha de la coma decimal. Se define como formato Qn aquel que tiene n bits a la derecha de la coma decimal. Los datos que sean enteros puros se representarán en formato Q0.

Algunos datos requieren una precisión mayor que la representación mediante una palabra de 16 bits. Para dar acomodo a tales datos se define el formato de precisión doble, lo que significa que hay 32 bits de información. Mientras que las palabras de 16 bits son capaces de representar datos con una precisión de 1 en 2^{15}, los registros de 32 bits, tales como el registro de productos o el acumulador de la mayoría de las microplaquetas DSP disponibles comercialmente, pueden representar datos con una precisión de 1 en 2^{31}. Estas palabras se dice que son de precisión doble. Una vez más, debe haber una coma decimal que indique también la gama dinámica de la palabra.

Algunos datos tienen una gama mayor que la que puede representarse mediante cualquier formato fijo de 16 bits. Quizá 16 bits de precisión sea lo adecuado pero el escalamiento del valor ha de ser dinámico. Esos datos pueden representarse mediante coma flotante de precisión simple. Esto significa que los datos se representan con dos palabras. La primera palabra de 16 bits contiene un número cuya magnitud está comprendida entre 16384 y 32767. Esta es la mantisa del valor y se dice que su valor está representado en formato normalizado debido a la gama de su magnitud. Si el valor es positivo, el bit 14 de la mantisa es un 1. La segunda palabra contiene el número de desplazamientos a la izquierda (NLS, *number of left shifts*) utilizados para poner el valor en formato normalizado. Así pues, la segunda palabra especifica el formato Q de la mantisa. Si este formato se emplea para un solo valor, se dice que es de coma flotante escalar.

También es posible representar un ordenamiento de n valores con n + 1 palabras utilizando coma flotante de bloque. Mediante este formato, el valor de mayor magnitud del ordenamiento se representaría de la misma manera que acaba de describirse para el caso de coma flotante escalar. Todos los demás valores del ordenamiento compartirían el mismo NLS. Su mantisa no tendría necesariamente formato normalizado. Una ampliación de esta representación es la de coma flotante de bloque segmentado. En este caso, un ordenamiento de mn valores se representa mediante m(n + 1) palabras. El ordenamiento se subdivide en m subordenamientos de tamaño n y cada subordenamiento se representa en coma flotante de bloque con n palabras que representan la magnitud y 1 palabra que representa el NLS.

El otro tipo de representación utilizado es el de coma flotante de precisión doble. En este caso, se utilizan enteros de precisión doble para la mantisa y una palabra de precisión simple para representar el NLS. En definitiva, los diferentes tipos de representación utilizados son los formatos de coma fija de precisión simple, coma fija de precisión doble para los acumuladores y el registro de productos, coma flotante de precisión simple escalar y coma flotante de bloque de precisión simple y doble.

G.1.3 Operaciones aritméticas

El resultado de la multiplicación de dos palabras de 16 bits es un número de 32 bits. Esta es la razón por la que los registros de productos son normalmente de precisión doble. Puesto que los registros de productos pueden sumarse a los acumuladores, los acumuladores han de tener también una anchura de por lo menos 32 bits. En un cálculo del tipo suma de productos, como en el caso de convolución o filtrado FIR, podría desbordarse el acumulador. El problema del desbordamiento se trata de diferentes maneras en las microplaquetas DSP disponibles comercialmente.

En el filtrado IIR, la suma de sus productos, o el producto de las operaciones de multiplicación-acumulación, pasa a ser parte de la memoria del filtro y se utiliza de nuevo cuando se efectúe la operación de filtrado siguiente. De manera específica, los 16 bits de la palabra alta (de mayor peso) de la salida se utilizarán como una entrada al multiplicador. Un desbordamiento que convierte un valor positivo grande en un valor negativo grande o viceversa se conoce como reciclaje (de mayor peso) y provoca una gran diferencia en la salida del filtro. Como protección contra esto se utiliza la aritmética del modo saturación en todos los filtros IIR y dondequiera que una suma de productos vaya a utilizarse más tarde como entrada a un multiplicador. Modo saturación significa que, si la palabra alta pasa a ser mayor que 32767 o menor que –32768, se recortará a esos valores para evitar el reciclaje.

G.1.3.1 Desplazamiento y redondeo

Al analizar las operaciones aritméticas se empieza con el desplazamiento y el redondeo. Si se multiplica un valor en formato Qn por un número cuyo valor está en formato Qm, el resultado del registrador de productos tendrá el formato de precisión doble Q(n + m). Si es preciso almacenar o adicionar el resultado con una precisión diferente, habrá que desplazarlo y/o redondearlo a la precisión correcta.

Dos tipos de desplazamientos son posibles: desplazamientos a la izquierda y desplazamientos a la derecha. En las microplaquetas DSP disponibles comercialmente, los desplazamientos pueden efectuarse normalmente en el acumulador. También es posible, normalmente, desplazar el resultado que se halla en un registro de productos antes de adicionarlo o almacenarlo en el acumulador. Tal como indican sus nombres, en un desplazamiento a la izquierda los bits se desplazan hacia la izquierda y en un desplazamiento a la derecha, hacia la derecha. Si un valor se desplaza k bits hacia la derecha, se pierden los k bits menos significativos del valor anterior. Si un valor se desplaza hacia la izquierda es preciso verificar la posibilidad de desbordamientos. La expresión con la que se indica un desplazamiento de k bits hacia la derecha de una variable TMP es

$$TMP = TMP >> k$$

y la expresión para indicar un desplazamiento de k bits hacia la izquierda es

$$TMP = TMP << k$$

En algunos casos, k es variable y puede incluso ser negativo. Cuando k es negativo, un desplazamiento de k bits a la izquierda se define como un desplazamiento de –k bits a la derecha. De manera similar, un desplazamiento de k bits a la derecha, cuando k es negativo, es equivalente a un desplazamiento de –k bits a la izquierda. Cuando existe la posibilidad de que k sea negativo, el seudocódigo incluye una prueba de dicha posibilidad seguida de un desplazamiento inverso de –k bits, si k es negativo. Aunque los desplazamientos negativos puedan definirse matemáticamente, como acaba de hacerse, no es posible su aplicación en la mayoría de los dispositivos o en algunos lenguajes de computador.

Conviene señalar una anomalía particular de los desplazamientos a la derecha en el caso de aritmética de complemento a 2. Supóngase que el valor que ha de desplazarse a la derecha es 3 y el desplazamiento es de un bit. La representación de 3 con 16 bits viene dada por 0000000000000011. Si esto se desplaza un bit a la derecha se obtiene 0000000000000001 = 1. Si el valor que ha de desplazarse a la derecha es –3, la representación es 1111111111111101. Después de un desplazamiento a la derecha el resultado es 1111111111111110 = –2. Lo primero que se observa es que, en un desplazamiento a la derecha, se produce una extensión del bit de signo. La anomalía consiste en que las magnitudes de las respuestas de estos dos ejemplos no concuerdan. Si se utilizara una representación a base de signo y magnitud, sí habría concordancia. Los realizadores han de ser conscientes de esta diferencia.

Al simular el codificador se ha encontrado otra diferencia, más sutil, que depende del compilador. Es posible que se genere en el algoritmo la instrucción de efectuar el desplazamiento de una palabra a la derecha superior al tamaño de la misma. Podría ser, por ejemplo, desplazar 18 bits una palabra de 16 bits. Si la operación se efectuara realizando 18 desplazamientos individuales de 1 bit a la derecha, el resultado de la misma sería 0 ó –1, dependiendo del signo del dato original. Sin embargo, se ha encontrado que algunos compiladores consideran que un desplazamiento de 18 bits es una instrucción ilícita que produce resultados espúreos. Los realizadores deben verificar cómo trataría un caso así su soporte físico y su compilador de lenguaje ideales.

El redondeo es el proceso de conversión del acumulador de precisión doble a precisión simple. Normalmente, se efectúa inmediatamente antes de almacenar en memoria el valor de una palabra de 16 bits. Un acumulador consta de una palabra alta y una palabra baja (y, posiblemente, de los bits adicionales a la izquierda de la palabra alta). Por lo general, es posible almacenar en memoria la palabra alta o la palabra baja (de menor peso), o ambas, con dos instrucciones sucesivas. Si se considera que el acumulador tiene una coma decimal situada entre la palabra alta y la palabra baja, el redondeo consiste entonces en la operación de hacer pasar el acumulador al valor entero más próximo al valor no entero almacenado en las dos palabras originales. El convenio usual para números en complemento a 2 consiste en verificar el MSB de la palabra baja. Si es 1, se añade 1 al valor de la palabra alta y a continuación se anula la palabra baja. Por ejemplo, si el valor del acumulador es 1,5, la palabra alta viene dada por 0000000000000001 y la palabra baja por 1000000000000000. Puesto que el MSB de la palabra baja es 1, se añade 1 a la palabra alta y a continuación se da el valor cero a la palabra baja. El resultado es 0000000000000010 para la palabra alta o 2. Si el valor del acumulador es –1,5 la palabra alta viene dada por 1111111111111110 y la palabra baja viene dada por 1000000000000000. Puesto que el MSB de la palabra baja es 1, se añade 1 a la palabra alta y a continuación se da el valor cero a la palabra baja. El resultado es 1111111111111111 = –1, lo cual es similar a la anomalía de los desplazamientos hacia la derecha.

Al efectuar la función de redondeo es preciso tener presente la posibilidad de desbordamiento. Por ejemplo, si el valor de la palabra alta es 0111111111111111 (=32767) y la palabra baja tiene un 1 en el MSB, el seguimiento del convenio usual da lugar a un desbordamiento. Dependiendo del procesador, la palabra de resultado podría pasar a ser 1000000000000000, lo que representa –32768. En este caso no se sigue el convenio usual sino que se satura el valor, para evitar un valor no representable.

En los ejemplos de seudocódigo, la función de redondeo descrita más arriba se representa por RND (.).

Seudocódigo para VSCALE

Un módulo nuevo de seudocódigo, que es preciso introducir en este punto, efectúa el escalamiento de vectores para la representación de coma flotante de bloque. El nombre dado a dicho módulo es el de VSCALE. Su objetivo es escalar un vector de modo que la mayor magnitud de sus elementos esté justificada a la izquierda según se desea, es decir, representada en formato normalizado. Este módulo se puede utilizar para vectores en los que se sabe que el primer elemento tiene el elemento mayor o para vectores en los que se desconoce la ubicación del elemento mayor. Las entradas a VSCALE son IN, el vector de entrada que ha de escalarse, LEN, la longitud del vector de entrada, SLEN, la duración de la búsqueda para encontrar el valor máximo, y MLS, el número máximo de desplazamientos a la izquierda permitidos. Las salidas de VSCALE son OUT, el vector de salida, y NLS, el número de desplazamientos a la izquierda utilizados para escalar el vector de entrada. Se supone que los vectores de entrada y salida son del mismo tipo y pueden ser de coma flotante de bloque de precisión simple (enteros de 16 bits) o de coma flotante de bloque de precisión doble (enteros de 32 bits). En el caso de vectores de precisión simple, MLS = 14, mientras que en los vectores de precisión doble, MLS = 30. A veces se desea utilizar menos de 16 bits o de 32 bits para representar una variable. Por ejemplo, hay algunas variables que tienen especificado 14 ó 15 bits de precisión, en cuyo caso se hace MLS =12 ó 13, respectivamente. Debido a esta posibilidad, también cabe el que, para normalizar la variable, se requieran desplazamientos a la derecha en vez de desplazamientos a la izquierda. En tal caso, el valor de NLS devuelto será negativo. Si, por ejemplo, se devuelve NLS = –1, ello indica que ha sido necesario un desplazamiento de 1 bit a la derecha. El módulo supone que hay un acumulador (AA0) disponible para el desplazamiento y que tiene por lo menos 32 bits de precisión. Si se sabe que el elemento máximo es el primero, se hace SLEN = 1. De no ser así, SLEN = LEN y se busca en todo el vector para encontrar el valor máximo.

El código siguiente se atiene al convenio de que los datos se representan en complemento 2. Trata por separado los casos en que los valores de mayor magnitud son positivos o negativos.

```
      SUBROUTINE VSCALE(IN, LEN, SLEN, MLS, OUT, NLS)
      AA0 = IN(1)                                          | Encontrar el máximo valor positivo de la entrada
      AA1 = IN(1)                                          | Encontrar el máximo valor negativo de la entrada
      If SLEN = 1, skip the next 3 lines
          For I = 2, 3, ..., SLEN, do the next two lines
              If IN(I) > AA0, set AA0 = IN(I)
              If IN(I) < AA1, set AA1 = IN(I)

                                                           | Caso 1: vector de entrada cero
      If AA0 = 0 and AA1 = 0, do the next 3 lines
          For I = 1, 2, ..., LEN, set OUT(I) = 0
          NLS = MLS + 1                                    | Hacer que 0 tenga un bit más de desplazamiento
          Exit this subroutine                             | a la izquierda que 1

      NLS = 0                                              | Inicializar NLS

                                                           | Determinar caso 2 o caso 3
      If AA0 < 0 or AA1 < –AA0, then do the following indented lines
                                                           | Caso 2: el número negativo es mayor
          MAXI = –2^MLS                                    | Límite inferior de la mantisa después del desplazamiento
          MINI = 2 * MAXI
          If AA1 < MINI, then do the following doubly indented lines to find the number of right shifts needed and then scale the
          elements

LOOP1R:       AA1 = AA1 >> 1
              NLS = NLS – 1                                | NLS negativo = => desplazamientos a la derecha
              If AA1 < MINI, go to LOOP1R
              For I = 1, 2, 3, ..., LEN, do the next line
                  OUT(I) = IN(I) >> –NLS
              Exit this subroutine

LOOP1L:   If AA1 < MAXI, go to SCALE1                      | Encontrar el número de desplazamientos a
                                                           | la izquierda
              AA1 = AA1 << 1
              NLS = NLS + 1
              Go to LOOP1L

SCALE1:   For I = 1, 2, 3, ..., LEN, do the next line
              OUT(I) = IN(I) << NLS
          Exit this subroutine

      Else, do the following indented lines
                                                           | Caso 3: el número positivo es mayor

          MINI = 2^MLS                                     | Límite inferior de la mantisa después del desplazamiento
          MAXI = MINI – 1                                  | 2 * MIN desbordará si MLS = 30
          MAXI = MAXI + MINI                               | Límite superior de la mantisa
          If AA0 > MAXI, then do the following doubly indented lines to find the number of right shifts needed and then scale the
          elements

LOOP2R:       AA0 = AA0 >> 1
              NLS = NLS – 1
              If AA0 > MAXI, go to LOOP2R
              For I = 1, 2, 3, ..., LEN, do the next line
                  OUT(I) = IN(I) >> –NLS
              Exit this subroutine

LOOP2L:   If AA0 ≥ MINI, go to SCALE2
              AA0 = AA0 << 1
              NLS = NLS + 1
              Go to LOOP2L

SCALE2:   For I = 1, 2, 3, ..., LEN, do the next line
              OUT(I) = IN(I) << NLS
          Exit this subroutine
```

En algunos casos se encuentra que, en realidad, no se desea reescalar los datos sino, simplemente, encontrar el número de desplazamientos a la izquierda que serían necesarios si se quisiera ese reescalamiento. La rutina que sigue utiliza las mismas entradas que VSCALE, si bien sólo proporciona NLS como salida. Omite el escalamiento del vector de entrada pero es, por lo demás, igual que VSCALE.

```
SUBROUTINE FINDNLS(IN, SLEN, MLS, NLS)
AA0 = IN(1)                                  | Encontrar el máximo valor positivo de la entrada
AA1 = IN(1)                                  | Encontrar el máximo valor negativo de la entrada
If SLEN = 1, skip the next 3 lines
    For I = 2, 3, ..., SLEN, do the next two lines
        If IN(I) > AA0, set AA0 = IN(I)
        If IN(I) < AA1, set AA1 = IN(I)

                                             | Caso 1: vector de entrada cero
If AA0 = 0 and AA1 = 0, do the next 2 lines
    NLS = MLS + 1                            | Hacer que 0 tenga 1 bit más de desplazamiento
    Exit this subroutine                     | a la izquierda que 1

NLS = 0                                      | Inicializar NLS

                                             | Determinar caso 2 o caso 3

If AA0 < 0 or AA1 < –AA0, then do the following indented lines
                                             | Caso 2: el número negativo es mayor
    MAXI = –2^MLS                            | Límite inferior de la mantisa después del desplazamiento
    MINI = 2 * MAXI
    If AA1 < MINI, then do the following doubly indented lines to find the number of right shifts needed

LOOP1R:     AA1 = AA1 >> 1
            NLS = NLS – 1                    | NLS negativo = => desplazamientos a la derecha
            If AA1 < MINI, go to LOOP1R
            Exit this subroutine

LOOP1L: If AA1 < MAXI, exit this subroutine  | Encontrar NLS
        AA1 = AA1 << 1
        NLS = NLS + 1
        Go to LOOP1L

    Else, do the following indented lines
                                             | Caso 3: el número positivo es mayor

    MINI = 2^MLS                             | Límite inferior de la mantisa después del desplazamiento
    MAXI = MINI – 1                          | 2 * MIN desbordará si MLS = 30
    MAXI = MAXI + MINI                       | Límite superior de la mantisa
    If AA0 > MAXI, then do the following doubly indented lines to find the number of right shifts needed

LOOP2R:     AA0 = AA0 >> 1
            NLS = NLS – 1
            If AA0 > MAXI, go to LOOP2R
            Exit this subroutine

LOOP2L: If AA0 ≥ MINI, exit this subroutine  | Encontrar NLS
        AA0 = AA0 << 1
        NLS = NLS + 1
        Go to LOOP2L
```

G.1.3.2 Multiplicación

La multiplicación de dos números de coma fija da lugar a un número de 32 bits, almacenado normalmente en un registro de productos de un DSP. Si los dos números de coma fija estuvieran en formatos Qn y Qm, el resultado del registro de productos estaría en formato Q(n + m). Antes de adicionar el resultado a un acumulador, puede ser necesario desplazarlo tal como se explica en la subcláusula precedente.

La multiplicación de dos palabras de coma flotante se efectúa mediante la multiplicación en coma fija de las dos mantisas y la adición de los dos NLS. Tal como se ha descrito más arriba, el producto es una palabra de 32 bits con formato Q(n + m). Si el producto ha de convertirse de nuevo a coma flotante, quizá haga falta renormalizarlo. Por ejemplo, cuando se multiplican dos palabras de coma flotante positivas, el producto ha de tener un 1 en el bit 30 o en el bit 29. La renormalización es necesaria si el bit 30 es 0. Esto significa que es preciso un desplazamiento adicional a la izquierda, después del cual el producto está representado en formato Q(n + m + 1). Si el producto ha de almacenarse en coma flotante escalar, debe ser redondeado antes del almacenamiento. Si el producto ha de estar en coma flotante de bloque, es preciso renormalizar la totalidad del ordenamiento, teniendo en cuenta qué valor es ahora el de mayor magnitud.

Aunque en algunas partes de este codificador se utilizan variables de precisión doble, no hay multiplicaciones de precisión doble. En algunos casos, se multiplican variables de precisión doble, pero entonces sólo se utilizan los 16 bits más significativos. En el seudocódigo se señalan tales casos.

G.1.3.3 Adición

La adición de números de coma fija exige que ambos estén almacenados en el mismo formato Q. Por lo general, el valor almacenado con la gama dinámica mayor determina qué valor ha de cambiarse al formato Q apropiado. Por ejemplo, si se adicionan valores almacenados en formatos Q9 y Q11, el valor en formato Q11 debe ser desplazado dos bits a la derecha antes de añadirlo al valor almacenado en formato Q9.

La adición de números de coma flotante escalar es similar. Los dos valores han de tener el mismo NLS. De nuevo aquí es preciso desplazar a la derecha el valor con NLS más elevado para que haya concordancia con el NLS del otro valor. Si la suma requiere 17 bits para la representación, la suma del acumulador puede ser desplazada un bit a la derecha y redondeada a continuación a 16 bits y el nuevo NLS será inferior en una unidad al del formato previo. Como ejemplo, considérese el caso de la adición de dos valores cuyos NLS son 5 y 7. El valor cuyo NLS es 7 ha de desplazarse 2 bits a la derecha antes de que pueda añadirse al otro valor. Si ambos valores tienen el mismo signo, la suma de las dos mantisas puede ser de una magnitud superior a 32767. En este caso, el valor del acumulador ha de ser desplazado un bit y redondeado a continuación. El NLS de la suma será 4. Si los dos valores son de signo opuesto, el resultado del acumulador puede tener una mantisa cuya magnitud sea inferior a 16384. En este caso, el resultado ha de ser renormalizado por desplazamiento a la izquierda hasta que la magnitud sea superior o igual a 16384 y el NLS incrementado en el número de desplazamientos a la izquierda. En el ejemplo considerado, siendo el NLS de 5 y 7 inicialmente, el NLS final no puede ser superior a 6 ni inferior a 4.

La adición de números de coma flotante de bloque se complica por el hecho de que las constricciones se basan en el valor de mayor magnitud. En este caso, si dos vectores tienen NLS de 5 y 7, aquel cuyo NLS es 7 debe ser desplazado 2 bits a la derecha. Se suma cada uno de los pares y la suma resultante mayor determina si es necesaria la renormalización.

G.1.3.4 División

La división se utiliza con mucha menos frecuencia que la adición o la multiplicación. Las únicas divisiones utilizadas son divisiones con coma flotante escalar. El numerador y el denominador se representan en formato normalizado, al igual que el cociente. El NLS del cociente se calcula sustrayendo el NLS del denominador del NLS del numerador y añadiendo 14. Para comprender esta adición de 14 considérese el caso en que el numerador fuese ligeramente mayor que el denominador y ambos tuvieran un NLS igual a 0. El cociente tendría entonces un NLS igual a 14 y sería normalizado convenientemente. Si la mantisa del numerador fuese inferior a la del denominador, el numerador debería ser desplazado un bit a la derecha y su NLS incrementado en 1 para calcular el NLS del cociente. Así se garantiza que la mantisa del cociente estará en formato normalizado.

La división se produce en la recursión de Durbin; una rutina que exige una precisión de 16 bits total en el resultado, por lo que las rutinas de división aproximada no son suficientes. La mantisa del resultado ha de tener una precisión de 16 bits total, incluido el redondeo del resultado de 17 bits. El seudocódigo para una división como ésa se da un poco más adelante.

Si el numerador o el denominador no están almacenados inicialmente con coma flotante escalar, ha de procederse en primer lugar a su conversión a ese formato. En el seudocódigo la función FLOAT(.) se utiliza para representar esa conversión. El argumento puede ser de coma fija de precisión simple o de precisión doble.

Seudocódigo para división en coma flotante

Esta rutina se utiliza para calcular la división en coma flotante en un dispositivo de coma fija de 16 bits. Se supone que hay por lo menos un acumulador de 32 bits disponible. Todas las entradas y salidas son palabras de 16 bits.

Entrada: NUM, NUMNLS, DEN, DENNLS

Salida: QUO, QUONLS

Función: Calcular el cociente. NUM y NUMNLS son la mantisa y el formato Q del numerador. DEN y DENNLS son la mantisa y el formato Q del denominador. QUO y QUONLS son la mantisa y el formato Q del cociente. Se supone que todos ellos están en formato normalizado. No se comprueba si DEN es cero; se supone que no lo es.

```
SUBROUTINE DIVIDE(NUM, MUMNLS, DEN, DENNLS, QUO, QUONLS)
SIGN = 1                                  | Determinar primero
P = NUM * DEN                             | el bit de signo
If P < 0, set SIGN = –1                   | del cociente

QUONLS = NUMNLS – DENNLS + 14             | Calcular a continuación QUONLS
A0 = | NUM |                              | A0 es un acumulador de 32 bits
                                          | | NUM | se halla en los 16 bits más bajos
A1 = | DEN |                              | A1 puede ser un registro de 16 ó 32 bits,
                                          | si es de 32 bits, | DEN | se halla en
                                          | los 16 bits más bajos.
If A0 < A1, do the next 2 lines
     QUONLS = QUONLS + 1
     A0 = A0 << 1

QUO = 0                                   | Inicialización del cociente
I = 0                                     | Inicialización del contador del bucle

LOOP: QUO = QUO << 1                      | Bucle de división larga
If A0 ≥ A1, do the next 2 lines
     QUO = QUO + 1
     A0 = A0 – A1
A0 = A0 << 1
I = I + 1
If I < 15, GO TO LOOP

If A0 ≥ A1, set QUO = QUO + 1             | Atención al redondeo

If SIGN < 0, set QUO = –QUO               | Atención al signo
```

G.2 Cambios de los algoritmos

G.2.1 Cambios en el adaptador de ganancia vectorial hacia atrás (bloque 20)

NOTA – Esta subcláusula se refiere a 3.8/G.728. Los lectores deben familiarizarse con 3.8 antes de intentar comprender esta subcláusula. Los cambios que aquí se indican corresponden a los cálculos efectuados una vez por vector para el adaptador de ganancia vectorial hacia atrás. Cuando ha sido posible se ha utilizado la misma notación que en la Recomendación G.728.

En esta subcláusula se describen, de manera sucinta, las operaciones del adaptador de ganancia vectorial hacia atrás una vez por vector de la Recomendación G.728 aplicado en coma flotante. A continuación se describe un método matemáticamente equivalente que puede aplicarse más fácilmente y con mayor exactitud en los procesadores de coma fija. En el addéndum al presente anexo figuran cuadros con los valores requeridos por este método alternativo.

Las operaciones en coma flotante pueden describirse brevemente como a continuación se indica. El ordenamiento de las variables de estado internas GSTATE, representado por el símbolo δ, contiene las 10 ganancias logarítmicas anteriores con el desplazamiento eliminado. El símbolo $\delta(n)$ indica la ganancia logarítmica con el desplazamiento eliminado del vector n. La salida del predictor de la ganancia logarítmica [la versión predicha de $\delta(n)$] para el vector n viene dada por:

$$\hat{\delta}(n) = -\sum_{i=1}^{10} \alpha_i \, \delta(n-i) \qquad \text{(G-1)}$$

Como se muestra en la Figura 6/G.728, antes de convertir $\hat{\delta}(n)$ al dominio lineal ha de añadirse un desplazamiento de ganancia de 32 dB y verificar el resultado para tener la seguridad de que:

$$0 \leq \hat{\delta}(n) + 32 \leq 60 \tag{G-2}$$

De manera similar, puede decirse que la gama permitida para $\hat{\delta}(n)$ es:

$$-32 \leq \hat{\delta}(n) \leq 28 \tag{G-3}$$

La ganancia estimada en el dominio lineal viene dada por:

$$\sigma(n) = 10^{(\hat{\delta}(n) + 32)/20} \tag{G-4}$$

El valor de $\sigma(n)$ se utiliza primero para normalizar el vector objetivo VQ de excitación. Una vez completada la búsqueda del código cifrado, se utiliza $\sigma(n)$ para escalar el mejor vector de código seleccionado. Suponiendo que para el vector n se hayan elegido el índice i del código cifrado de ganancia y el índice j del código cifrado de forma, el vector de excitación $e(n)$ viene dado por:

$$e(n) = \sigma(n)\, g_i\, y_j \tag{G-5}$$

donde y_j es el j-ésimo vector de código de forma y g_i es el i-ésimo nivel de ganancia del código cifrado de ganancia. A continuación se utiliza el vector de excitación $e(n)$ para calcular $\delta(n)$. Primero se calcula el valor al cuadrado de la media cuadrática de $e(n)$ [la «potencia» de $e(n)$] que viene dado por:

$$P[e(n)] = \frac{1}{5} \sum_{k=1}^{5} e_k^2(n) \tag{G-6}$$

Para cualquier vector dado x, se utiliza el símbolo $P[x]$ en representación de la potencia de x, que se define como la energía de x dividida por la dimensión del vector de x. Antes de convertir $P[e(n)]$ al valor en dB del dominio logarítmico, se recorta $P[e(n)]$ a 1, si es menor que 1. La gama permitida para $P[e(n)]$ es, entonces, como sigue:

$$P[e(n)] \geq 1 \tag{G-7}$$

Con esto se trata de evitar el desbordamiento en la conversión logarítmica, o un valor en dB sumamente pequeño. Obsérvese que, si bien esta acción limitadora de la gama no se muestra explícitamente en la Figura 6/G.728, sí está implementada en el «seudocódigo» del 5.7/G.728. La ganancia logarítmica en dB, tras deducirle el desplazamiento, del vector n se obtiene entonces de la siguiente manera:

$$\delta(n) = 10 \log_{10} P[e(n)] - 32 \tag{G-8}$$

Obsérvese que la ecuación (G-7) implica que:

$$\delta(n) \geq -32 \tag{G-9}$$

A continuación se utiliza la ganancia $\delta(n)$, calculada mediante la ecuación (G-8), para predecir las ganancias de excitación siguientes y actualizar los coeficientes del predictor de ganancia logarítmica. Así se termina el breve análisis de la operación en coma flotante del adaptador de ganancia vectorial hacia atrás.

Seguidamente se describe el método equivalente matemáticamente para la implementación de coma fija. Sea y_{jk} el elemento k-ésimo del vector de código j-ésimo del código cifrado de forma. Combinando las ecuaciones (G-5) y (G-6) se tiene:

$$P[e(n) = \sum_{k=1}^{5} \left(\sigma(n)\, g_i\, y_{jk} \right)^2 \tag{G-10}$$

$$= \sigma^2(n)\, g_i^2 \left(\frac{1}{5} \sum_{k=1}^{5} y_{jk}^2 \right) \tag{G-11}$$

$$= \sigma^2(n)\, g_i^2\ P[y_j] \tag{G-12}$$

Entrando con la ecuación (G-12) en la ecuación (G-8) se obtiene:

$$\delta(n) = 20 \log_{10} \sigma(n) - 32 + 20 \log_{10} |\, g_i\, | + 10 \log_{10} P[y_j] \tag{G-13}$$

A continuación, utilizando la ecuación (G-4), se puede expresar $\delta(n)$ de la siguiente manera:

$$\delta(n) = \hat{\delta}(n) + 20 \log_{10} |\, g_i\, | + \log_{10} P[y_j] \tag{G-14}$$

En otras palabras, $\delta(n)$ es simplemente la ganancia logarítmica predicha $\hat{\delta}(n)$ más dos «términos de corrección»:

1) $20 \log_{10} |\, g_i\, |$, el valor en dB del mejor nivel de ganancia seleccionado a partir del código cifrado de ganancia; y
2) $10 \log_{10} P[y_j]$, el valor en dB de la potencia del mejor vector de código de forma seleccionado a partir del código cifrado de forma. (En cierto sentido, esto equivale a un codificador predictivo convencional para la ganancia, pero utilizado en el dominio logarítmico.)

La Figura G.1 muestra el diagrama de bloques de este método matemáticamente equivalente. Puesto que sólo hay cuatro valores posibles de $|\, g_i\, |$ y 128 valores posibles de $P[y_j]$; se puede efectuar un cálculo previo de sus valores en dB y almacenarlos en dos tablas de ganancia logarítmica (bloques 93 y 94 de la Figura G.1).

Las unidades de retardo 91 y 92 ponen a disposición los mejores índices de código cifrado de ganancia y de forma elegidos en la búsqueda del código cifrado de excitación del vector anterior. Estos dos índices se utilizan para encontrar los valores de $20 \log_{10} |\, g_i\, |$ y $10 \log_{10} P[y_j]$ consultando las tablas de ganancia logarítmica de los bloques 93 y 94. La unidad 95 de retardo de una muestra retiene la ganancia logarítmica predicha (y posiblemente de gama limitada) anterior $\hat{\delta}(n-1)$. El sumador 96 suma la salida de los bloques, 93, 94 y 95 produciendo un valor $\hat{\delta}(n-1)$ no recortado de acuerdo con la ecuación (G-14). El limitador 97 aplica entonces la desigualdad (G-9) recortando la salida del sumador 96 a –32 dB, si es inferior a –32 dB.

La salida del limitador 97 equivale matemáticamente a la salida del sumador 42 de la Figura 6/G.728, por lo que los bloques 43 a 46 de la Figura G.1 son idénticos a sus correspondientes de la Figura 6/G.728. La operación del limitador de ganancia logarítmica 98 es similar a la del limitador 47 de la Figura 6/G.728, salvo que la gama permitida ha sido desplazada hacia abajo en 32 dB. El sumador 99 suma el valor de desplazamiento de ganancia logarítmica de 32 dB, almacenado en el bloque 41, a la salida del limitador de ganancia logarítmica 98. El valor de ganancia logarítmica resultante es convertido al dominio logarítmico, a continuación, por el calculador logarítmico inverso 48, que es idéntico a su correspondiente de la Figura 6/G.728. De esta manera se completa la descripción del método matemáticamente equivalente para implementación de coma fija.

El método equivalente mostrado en la Figura G.1 tiene dos ventajas importantes con respecto al método original de la Figura 6/G.728.

a) Elimina la necesidad de calcular la función logarítmica (bloque 40 de la Figura 6/G.728). En las implementaciones de DSP, la función logarítmica se calcula normalmente utilizando una expansión de serie potencial y, por lo general, exige el cálculo de un gran número de ciclos de instrucción. Por eso, la sustitución del cálculo logarítmico por la consulta de una tabla podría suponer un considerable ahorro de ciclos de DSP. Además, las entradas en la tabla pueden precalcularse con el grado de precisión máxima deseada.

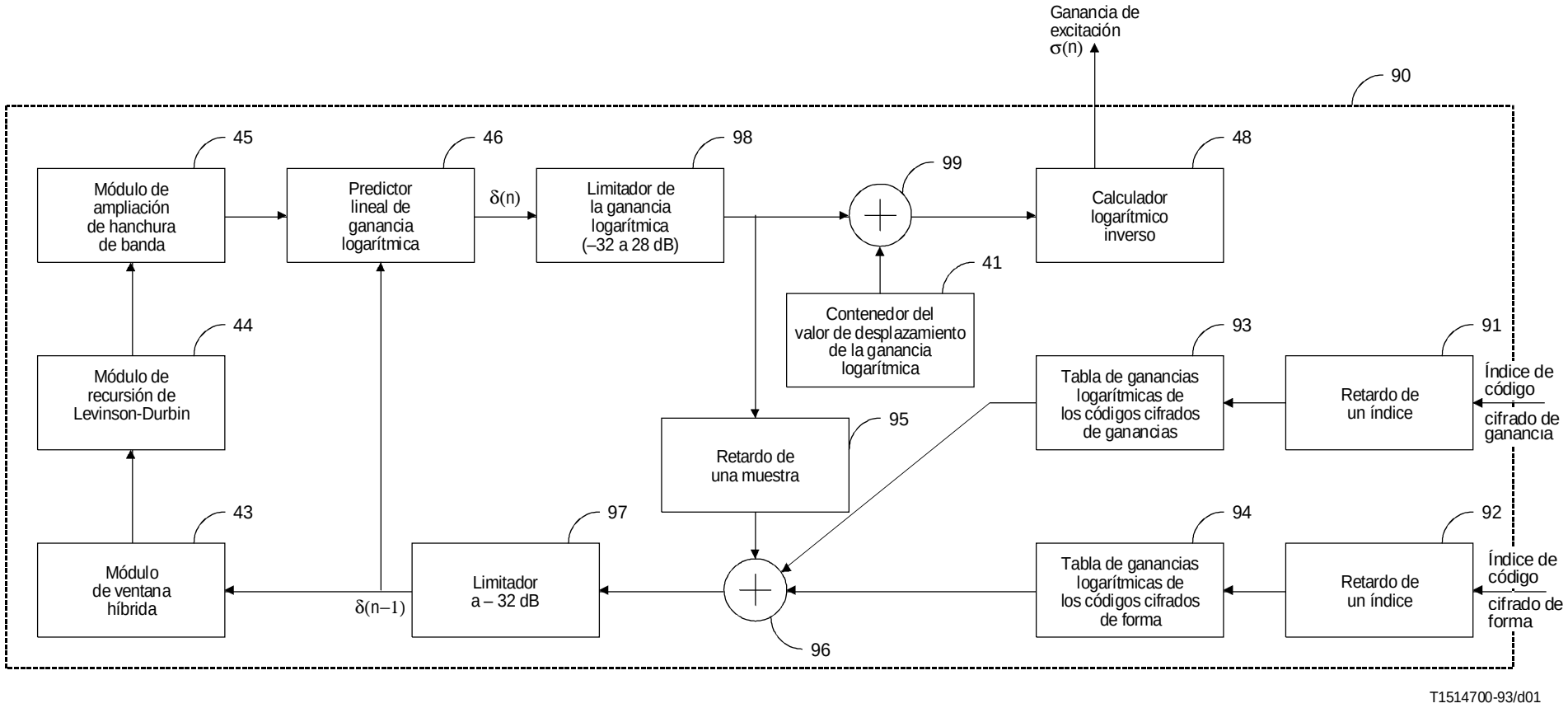

FIGURA G.1/G.728

Adaptador de ganancia vectorial hacia atrás para implementación de coma fija

b) Probablemente dé resultados numéricos más exactos que el método original cuando se utilice un procesador de coma fija. Debido a la adaptación hacia atrás, hay un bucle de realimentación en el proceso de adaptación de ganancia. En la Figura 6/G.728, dicho bucle de realimentación es muy largo. Va del calculador logarítmico inverso 48 a la unidad de escalamiento de ganancia 21 (de la Figura 2/G.728) y a continuación vuelve a los bloques 67, 39, 40 y 42 a 48. Cuantos más cálculos se hagan en este bucle más probable es que se acumulen errores numéricos debidos a la precisión finita en el bucle de realimentación. Esto es especialmente válido si el procesador de coma fija no alcanza siempre la máxima precisión posible para la función logarítmica. Por el contrario, el bucle de realimentación de la Figura G.1 es lo más compacto posible. Obsérvese que la unidad de escalamiento de ganancia, el cálculo de energía y de potencia de $e(n)$, el calculador logarítmico e incluso el sumador para el restablecimiento del desplazamiento de la ganancia logarítmica están ahora, todos ellos, fuera del bucle de realimentación. Salvo por los bloques 43 a 46, que son comunes a ambos métodos, el bucle de realimentación sólo implica dos limitadores y dos adiciones, que pueden ser implementados con un grado de precisión muy alto por los procesadores de coma fija.

Como resultado de este cambio se mejora el interfuncionamiento entre implementaciones de coma fija y de coma flotante de la Recomendación G.728. La principal desventaja de este nuevo método es que necesita más palabras de memoria muerta (ROM, *read only memory*). Hay 128 vectores de forma y 4 vectores de ganancia posibles. La memoria adicional requerida es de 128 + 4 = 132 palabras, lo que representa una fracción muy pequeña del espacio de memoria ROM ya necesario según la Recomendación G.728.

Este método nuevo ha cambiado la entrada al adaptador de ganancia vectorial hacia atrás (bloques 20 y 30) de $e(n)$ a los índices i y j del código cifrado de forma. Para reflejar este hecho, deberían haberse rehecho aquí las Figuras 1/G.728 a 3/G.728, de modo que el adaptador de ganancia vectorial hacia atrás obtuviera su entrada a partir del bloque de códigos cifrados VQ de excitación. Sin embargo, se han omitido esas figuras modificadas ya que el cambio necesario resulta trivial y debe haber quedado muy claro con la descripción dada más arriba.

G.2.2 Cambios en los módulos de recursión de Levinson-Durbin

En esta subcláusula se tratan los cambios a introducir en los módulos de recursión de Levinson-Durbin utilizados en la Recomendación G.728. Hay tres módulos de ese tipo, designados como bloques 37, 44 y 50, y utilizados por el filtro de ponderación perceptual, el predictor lineal de ganancia logarítmica y el filtro de síntesis, respectivamente. Los lectores deberán consultar, para más detalles, 5.5/G.728 y 5.6/G.728. En la subcláusula presente se utilizará el seudocódigo para el bloque 50 de 5.6/G.728 como ejemplo y se mostrará cómo debe modificarse en el caso de implementación de coma fija. Para los bloques 37 (filtro de ponderación perceptual) y 44 (predictor de ganancia logarítmica) es preciso introducir cambios similares. Se empieza con un listado del seudocódigo de coma flotante.

```
If RTMP (LPC + 1) = 0, go to LABEL                | Saltar si cero
                                                  |
If RTMP(1) ≤ 0, go to LABEL                       | Saltar si señal cero
                                                  |

RC1 = –RTMP(2)/RTMP(1)
ATMP(1) = 1                                       |
ATMP(2) = RC1                                     | Predictor de primer orden
ALPHATMP = RTMP(1) + RTMP(2) * RC1                |
If ALPHATMP ≤ 0, go to LABEL                      | Abortar si hay desajuste
For MINC = 2, 3, 4, ..., LPC, do the following
    SUM = 0
    For IP = 1, 2, 3, ..., MINC, do the next 2 lines
        N1 = MINC – IP + 2
        SUM = SUM + RTMP(N1) * ATMP(IP)

                                                  |
    RC = –SUM/ALPHATMP                            | Coeficiente de reflexión
    MH = MINC/2 + 1                               |
    For IP = 2, 3, 4, ..., MH, do the next 4 lines
        IB = MINC – IP + 2
        AT = ATMP(IP) + RC * ATMP(IB)             |
        ATMP(IB) = ATMP(IB) + RC * ATMP(IP)       | Actualizar los coeficientes del predictor
        ATMP(IP) = AT                             |
```

```
        ATMP(MINC + 1) = RC                          |
        ALPHATMP = ALPHATMP + RC * SUM               | Energía residual de predicción
        If ALPHATMP ≤ 0, go to LABEL                 | Abortar si hay desajuste
    Repeat the above for the next MINC

                                                     | La recursión ha terminado normalmente
    Exit this program                                | si la ejecución llega hasta aquí
```

LABEL: Si el programa llega hasta aquí, se había producido desajuste; saltar entonces el bloque 51 y no actualizar los coeficientes del filtro de síntesis (es decir, utilizar los coeficientes del filtro de síntesis del ciclo de adaptación anterior).

La mejor manera de empezar es analizar las variables de coma flotante a las que se hace referencia en este seudocódigo. Son RC, RC1, RTMP, SUM, ALPHATMP y ATMP. (Las otras variables del código, MINC, IP, IB y N1 son índices que son, todos ellos, enteros.)

RC se refiere a los coeficientes de reflexión, que en este módulo se calculan como una variable intermedia. Los coeficientes de reflexión tienen la propiedad de que su magnitud es siempre inferior a la unidad en el caso de un filtro LPC estable. Por eso RC puede representarse en formato Q15, lo que significa que se utiliza un bit como bit de signo y los otros 15 bits para representar la parte fraccionaria del valor.

Se señala que RC se calcula en cada iteración, que es utilizada para esa iteración y después ya no se vuelve a utilizar. La única excepción consiste en que, para el análisis del filtro LPC de síntesis en el decodificador, se guarda RC1 a fin de utilizarla más tarde en el posfiltro. Sólo el valor RC1 se guarda más arriba, para ahorrar memoria. Todos los demás valores de RC se escriben en una sola ubicación, sobre la que se reescribe en la iteración siguiente. Esto representa un cambio con respecto al seudocódigo de coma flotante original, pero no influye en los resultados de salida y puede utilizarse también en las implementaciones de coma flotante.

RTMP se refiere a los valores de la función de autocorrelación. Dichos valores tienen una gama dinámica enorme. Por necesidad, RTMP debe mantenerse en coma flotante de bloque, lo que significa que todos los valores son normalizados por la misma potencia de 2. En teoría, RTMP(1) debería tener el valor mayor. Se sabe además que ha de ser positivo. La representación utilizada será tal que la magnitud mayor de RTMP esté comprendida entre 0,5 y 1. Si éste es el caso, todo lo de RTMP puede representarse en formato Q15. Todo lo de RTMP se representa en coma flotante de bloque en ventana híbrida, pero en la recursión de Durbin sólo se necesitan las mantisas.

Otra observación a propósito de RTMP(LPC + 1), y tal como se indica en la primera línea, es que si esta variable tiene un valor de 0, deberá terminarse el módulo. Si RTMP(LPC + 1) está representada por un entero de 16 bits, es mucho más probable que se dé esa condición que en el caso en que la misma RTPM(LPC + 1) se representa mediante un número de coma flotante de 32 bits en una implementación de coma flotante. De aquí se derivan problemas de interfuncionamiento. Al calcular RTMP(LPC + 1) en el módulo anterior (ventana híbrida, bloque 49), el valor se acumula en el acumulador, que es por lo menos de 32 bits en todos los DSP de coma fija. Se propone que se efectúe la verificación de cero en este valor de 32 bits del acumulador cuando se ha completado el cálculo. Con este cambio puede evitarse la terminación prematura (y, por tanto, los problemas de interfuncionamiento). En el código nuevo se prueba una variable lógica llamada ILLCOND para ver si es verdadera o falsa. Su valor depende de los resultados de la prueba de RTMP(LPC + 1) en el módulo de ventana híbrida. ILLCOND se utiliza más tarde como variable de salida de este bloque, para indicar si los valores a la salida deben ser utilizados o ignorados.

En cuanto al posfiltro, existe la posibilidad de que los desajustes se produzcan después de la décima iteración. Para este caso se ha determinado un nuevo conjunto de coeficientes de predicción posfiltro adaptativo a corto plazo, que son válidos, si bien los coeficientes de filtro de síntesis de quincuagésimo orden no lo son. Una segunda variable lógica, ILLCONDP, indica la situación de los coeficientes de predicción posfiltro.

Las dos variables siguientes son SUM y ALPHATMP. Ambas son valores que se acumulan, pero que nunca se multiplican, y el valor mantenido en el acumulador es de 32 bits. No obstante, estas dos variables se dividen para calcular RC. Tanto SUM como ALPHATMP se convierten a coma fija de 16 bits para la división. El resultado de la división se representa en formato de coma fija Q15 y se asigna a RC. La variable SUM no aparece explícitamente en el seudocódigo de coma fija. En el formato de 32 bits, es el acumulador AA0 y en el formato de 16 bits, la variable SIGN.

Obsérvese que ALPHATMP es, naturalmente, un número de 32 bits que se acumula en un acumulador. Sin embargo, en las implementaciones de DSP reales es necesario guardar ALPHATMP en memoria para la próxima recursión de orden superior, porque el acumulador se necesitará para otros cálculos antes de que ALPHATMP se actualice de nuevo. Por ello se pueden ahorrar algunos ciclos de DSP si sólo se guarda y se carga la palabra alta de 16 bits redondeada de ALPHATMP en vez de la palabra completa de 32 bits. En la práctica se ha visto que aunque sólo se guarda la palabra alta de ALPHATMP después de cada actualización, no se degrada la calidad de funcionamiento del codificador. Sólo la palabra alta de ALPHATMP se guarda, por consiguiente, después de cada actualización de ALPHATMP. Para que quede claro cuándo está representada ALPHATMP por 16 bits y cuándo por 32 bits, el nombre ALPHATMP se utiliza en el seudocódigo solamente si es un número de 16 bits. Cuando se trata de un número de 32 bits se hace referencia a un acumulador.

El vector restante de variables es ATMP, que representa los coeficientes del predictor. En todas las simulaciones, el valor máximo observado para ATMP ha sido inferior a 4. Esto induciría a utilizar el formato Q13 en todas partes. Sin embargo, las mismas simulaciones han mostrado también que el empleo del formato Q13 en la recursión de Durbin no proporciona un grado suficiente de interfuncionamiento con las implementaciones de coma flotante. Se ha visto que, para lograr un mayor interfuncionamiento entre implementaciones de coma fija y de coma flotante, es mejor utilizar el formato Q15, excepto cuando provoca desbordamientos.

En una microplaqueta DSP, los resultados del cálculo de ATMP(IB) o ATMP(IP) se hallan inicialmente en el acumulador. Los acumuladores de la mayoría de las microplaquetas DSP coma fija de 16 bits tienen una anchura de por lo menos 32 bits. Los valores de ATMP(IB) y ATMP(IP) deben redondearse a una precisión de 16 bits. En el código de coma fija, es importante que el acumulador contenga bits de guarda o proporcione una bandera de desbordamiento, de tal manera que, cuando se calcule ATMP(IB) o ATMP(IP), se detecte el desbordamiento, si es que se produce.

Para la elección del formato de ATMP se ha adoptado la estrategia que se indica a continuación. Las iteraciones pueden numerarse de acuerdo con el valor de MINC. Se empieza con MINC = 2 y formato Q15 para ATMP. Si nunca se produce un desbordamiento, es decir que, para todo IP, $|\text{ATMP(IP)}| < 1$, la representación final de ATMP es en Q15. La otra posibilidad es que se produzca un desbordamiento durante una de las iteraciones. Supóngase que ocurre en la iteración K. En este caso, todos los valores de ATMP calculados durante las iteraciones K y K – 1 deben convertirse al formato Q14 por desplazamiento a la derecha. A continuación se reinicia la iteración K utilizando el formato Q14. Las iteraciones subsiguientes, de K + 1 en adelante, se calculan también utilizando el formato Q14. Los desbordamientos se pueden producir también cuando se utiliza el formato Q14. En ese caso, se sigue el mismo procedimiento y el cálculo continúa en formato Q13. Se ha observado empíricamente que tales desbordamientos nunca ocurren en formato Q13. El motivo por el que se utilizan los otros formatos es porque, si se impiden los desbordamientos, el resultado es más exacto. La representación final de ATMP antes de salir de este bloque es en Q13, Q14 o Q15.

También se ha observado que, tras las operaciones de ampliación de anchura de banda, los coeficientes del filtro a la salida de los módulos de esa ampliación (bloques 38, 45, 51 y 85) son siempre representables en formato Q14. Se ha observado además que la representación en formato Q15, cuando es posible, no mejora de todos modos la SNR de la transdecodificación más allá de lo observado utilizando el formato Q14. Por ello esos tres módulos de ampliación de anchura de banda convierten siempre sus ordenamientos de coeficientes de salida a Q14, con independencia de si sus ordenamientos de coeficientes de entrada (es decir, la salida de los módulos de recursión de Levinson-Durbin) están en Q13, Q14 o Q15. Esto significa que, cuando un módulo de recursión de Levinson-Durbin produce un ordenamiento de coeficientes de salida en Q13 o Q15, debe señalar el correspondiente módulo de ampliación de anchura de banda, de modo que pueda efectuarse un desplazamiento adicional para convertir el ordenamiento a Q14. Por este motivo, en el módulo de recursión de Levinson-Durbin que se da a continuación, se ha añadido una bandera NLSATMP adicional como una de las salidas de dicho módulo.

En el decodificador se interrumpe la recursión de Levinson-Durbin después de haber derivado los coeficientes de predicción de décimo orden. Estos valores se guardan para el posfiltro adaptativo. Hay, en consecuencia, dos posibles condiciones de comienzo. En el caso normal, la recursión se empieza con MINCO = 1. La otra posibilidad es MINCO = 10 en el decodificador. En este último caso deben guardarse los valores de NRS y ALPHATMP. Se señala además que, cuando se ejecuta el módulo de ampliación de anchura de banda, debe guardarse el valor de NLSATMP hasta que ICOUNT = 3.

Se señala por último que esta rutina emplea tres acumuladores. El tercer acumulador, AA2, se utiliza para retener el valor con precisión de 17 bits de RC, a fin de actualizar el coeficiente de predicción más reciente.

El seudocódigo siguiente describe la versión de coma fija de los módulos de recursión de Levinson-Durbin.

```
If MINC0 > 1, go to RECURSION
MINC0 = 1                                   | Inicializaciones para
ILLCONDP = .FALSE.                          | decodificador solamente

If ILLCOND = .TRUE., go to FAILED           | Saltar si RTMP(LPC + 1) es cero
If RTMP(1) ≤ 0, go to FAILED                | Saltar si señal cero
NRS = 0                                     | Formato Q15 inicialmente

DEN = RTMP(1)                               | Calcular predictor de primer orden
NUM = RTMP(2)
If NUM < 0, set NUM = –NUM
Call SIMPDIV(NUM, DEN, AA0)                 | | RTMP(2) | /RTMP(1)

AA0 = AA0 << 15
RC1 = RND(AA0)
If RTMP(2) > 0, set RC1 = –RC1              | Añadir información de signo
RC = RC1                                    | Coeficiente del predictor de primer orden
ATMP(2) = RC1

AA0 = RTMP(1) << 16                         |
P = RTMP(2) * RC                            |
AA0 = AA0 + (P << 1)                        |
ALPHATMP = RND(AA0)                         | Guardar en memoria palabra alta del
                                            | acumulador de DSP
```

RECURSIÓN:

```
For MINC = MINC0 + 1, MINC0 + 2, ..., LPC, do the following indented lines
    AA0 = 0
    For IP = 2, 3, ..., MINC, do the next 3 lines
        N1 = MINC – IP + 2
        P = RTMP(N1) * ATMP(IP)
        AA0 = AA0 + P                       | 32 bits para SUM
    AA0 = AA0 << 1

    AA0 = AA0 << NRS
    AA1 = RTMP(MINC + 1) << 16
    AA0 = AA0 + AA1                         |
    SIGN = RND(AA0)                         | Guardar el signo de la palabra alta
    NUM = SIGN
    If NUM < 0, set NUM = –NUM
    If NUM ≥ ALPHATMP, go to FAILED         |
    Call SIMPDIV(NUM, ALPHATMP, AA0)        | Dividir para obtener RC
    AA2 = AA0 << 15                         | AA2 almacena RC de 17 bits
    RC = RND(AA2)
    If SIGN > 0, set RC = –RC

                                            | Actualizar ahora ALPHATMP

    AA1 = ALPHATMP << 16
    P = RC * SIGN
    AA1 = AA1 + (P << 1)
    If AA1 ≤ 0, go to FAILED
    ALPHATMP = RND(AA1)

    MH = MINC/2 + 1                         | Parte fraccionaria de MINC/2 truncada;
                                            | MH = entero
                                            | Empezar a actualizar los coeficientes
                                            | del predictor
```

```
For IP = 2, 3, 4, ..., MH, do the following doubly indented lines
    IB = MINC – IP + 2
    AA0 = ATMP(IP) << 16                          | Cargar palabra alta de AA0
    P = RC * ATMP(IB)                             | RC en Q15 por lo que << 1
    AA0 = AA0 + (P << 1)
    If AA0 overflowed, then do the following triply indented lines
        NRS = NRS + 1
        For LP = 2, 3, ..., MINC, set ATMP(LP) = ATMP(LP) >> 1
        AA0 = ATMP(IP) << 16                      | Primero reescalar ATMP
        P = RC * ATMP(IB)                         | A continuación recalcular
        AA0 = AA0 + (P << 1)                      | AA0 desbordado
    AA1 = ATMP(IB) << 16


        P = RC * ATMP(IP)
        AA1 = AA1 + (P << 1)
        If AA1 overflowed, then do the following triply indented lines


            NRS = NRS + 1
            For LP = 2, 3, ..., MINC, set ATMP(LP) = ATMP(LP) >> 1
            AA0 = ATMP(IP) << 16                  | Primero reescalar ATMP(IP)
            P = RC * ATMP(IB)                     | A continuación recalcular AA0
            AA0 = AA0 + (P << 1)                  |
            AA1 = ATMP(IB) << 16                  | A continuación reescalar ATMP(IB)
            P = RC * ATMP(IP)                     | A continuación recalcular
            AA1 = AA1 + (P << 1)                  | AA1 desbordado
        ATMP(IP) = RND(AA0)
        ATMP(IB) = RND(AA1)



                                                  | Actualizar ATMP(MINC + 1)
    AA0 = AA2 >> NRS                              | AA2 contiene RC de 17 bits
    AA0 = RND(AA0)                                | Salida en la palabra baja de AA0
    If SIGN > 0, set AA0 = –AA0
    ATMP(MINC + 1) = AA0                          | Palabra baja almacenada en ATMP


Repeat the above indented lines for the next MINC


NLSATMP = 15 – NRS
If NLSATMP < 13, go to FAILED
Exit this program                                 | La recursión ha terminado normalmente
                                                  | si la ejecución llega hasta aquí


FAILED:  Set ILLCOND = .TRUE.
         If MINC ≤ 10, set ILLCONDP = .TRUE.


    Si el programa llega aquí, se ha producido desajuste; saltar, entonces, el bloque 51 y no actualizar los coeficientes del filtro de síntesis (es decir, utilizar los coeficientes del filtro de síntesis del ciclo de adaptación anterior).
```

La tabla siguiente contiene la relación de todas las variables del seudocódigo anterior, con su formato de representación para facilitar su referencia.

Variable	Formato	Tamaño	Temp./perm.	Antigua/nueva
AA0, AA1, AA2	DP-entero	1	temp.	nueva
ALPHATMP	SFL	1	temp.	antigua
ATMP	Q13/Q14/Q15	51	perm.	antigua
IB	entero	1	temp.	antigua
ILLCOND	lógico	1	perm.	nueva
ILLCONDP	lógico	1	perm.	nueva
IP	entero	1	temp.	antigua
LP	entero	1	temp.	nueva
MH	entero	1	temp.	antigua
MINC	entero	1	temp.	antigua
NLSATMP	entero	1	temp.	nueva
NRS	entero	1	temp.	nueva
NUM	entero	1	temp.	nueva
RC	Q15	1	temp.	nueva
RC1	Q15	1	temp.	nueva
RTMP	Q15	51	perm.	antigua
SIGN	entero	1	temp.	nueva

SFL	Coma flotante escalar de 16 bits
DP-entero	Registro de 32 bits, tal como un acumulador o los registradores de productos (AA1, AA2 & P)
entero	Entero de 16 bits
Q13/Q14/Q15	Entero de 16 bits con una de estas representaciones

El código anterior se escribió para el bloque 50 y en él se utilizan nombres de variables asociados con dicho bloque. No obstante, puede utilizarse para los bloques 37 y 44. En la tabla que sigue se indica la correspondencia entre los nombres de variables que son específicos del bloque 50 y los nombres que son específicos de cada uno de los otros bloques.

Bloque 50	Bloque 37	Bloque 44
ATMP	AWZTMP	GPTMP
ILLCOND	ILLCONDW	ILLCONDG
NLSATMP	NLSAWZTMP	NLSGPTMP
RTMP	R	R

En el código indicado más arriba se utiliza un algoritmo de división distinto y más sencillo que el utilizado a lo largo del resto del algoritmo, al que se hace referencia como SIMPDIV. El seudocódigo de SIMPDIV se da a continuación. Las entradas son NUM y DEN; ambas son enteros de 16 bits. La salida es AA0 con los resultados en los 17 bits más bajos.

```
        Subroutine SIMPDIV(NUM, DEN, AA0)
        AA0 = 0
        AA1 = NUM
        K = 0
LOOP:   AA0 = AA0 << 1
        AA1 = AA1 << 1
        If AA1 ≥ DEN, then set AA1 = AA1 – DEN and AA0 = AA0 + 1
        K = K + 1
        If K < 16, go to LOOP
```

G.3 Seudocódigo para otros módulos de la Recomendación G.728

En esta subcláusula se presenta el seudocódigo para otros módulos de la Recomendación G.728. El seudocódigo para la recursión de Levinson-Durbin se indicó en la subcláusula anterior, junto con los cambios algorítmicos para el adaptador de ganancia vectorial hacia atrás. Para cada módulo se presenta primero el seudocódigo de coma flotante, seguido de unos comentarios y a continuación el seudocódigo de coma fija. La tabla siguiente puede utilizarse como referencia para encontrar el seudocódigo para un determinado módulo del codificador.

Los seudocódigos para todos los bloques proporcionan una especificación exacta de bits y son la definición definitiva del codificador G.728 de coma fija. Toda desviación con respecto a estos seudocódigos puede dar lugar a una simulación o implementación incorrecta.

Número de bloque de la Recomendación G.728 de coma fija, descripción y nombre de seudocódigo

Bloque	Descripción	Seudocódigo
1	Conversión del formato MIC de entrada	No se necesita
2	Memoria intermedia de vector	No se necesita
3	Adaptador para el filtro de ponderación	Utilizar bloque 45
4	Filtro de ponderación	Bloque 4
5-7	Conmutador para ZIR/actualización de la memoria	No se necesita
8	Decodificador simulado	Véanse más adelante los bloques detallados
9	Filtro de síntesis para ZIR	Blockzir
10	Filtro de ponderación para ZIR	Blockzir
9, 10	Bloques 9 & 10 para actualización de la memoria	Bloque 9
11	Cálculo del vector objetivo VQ	Bloque 11
12	Calculador del vector de respuesta a impulsos	Bloque 12
13	Convolución con inversión del tiempo	Bloque 13
14	Convolución del vector de código de forma	Bloque 14
15	Calculador de la tabla de energía de código cifrado	Bloque 14
16	Normalización del vector objetivo VQ	Bloque 16
17	Calculador de error de búsqueda VQ	Bloque 17
18	Selector del mejor índice de código cifrado	Bloque 17
19	Códigos cifrados VQ de excitación	Bloque 19
20	Adaptador de ganancia vectorial hacia atrás	Véanse más adelante los bloques detallados
21	Unidad de escalamiento de ganancia	Bloque 19
22	Filtro de síntesis	Utilizar bloque 9
23	Adaptador del filtro de síntesis	Véanse los bloques 49-51

Número de bloque de la Recomendación G.728 de coma fija, descripción y nombre de seudocódigo *(fin)*

Bloque	Descripción	Seudocódigo
24	Módulo de búsqueda de código cifrado	Véanse los bloques 12-18
28	Conversión al formato MIC de salida	No se necesita
29	Códigos cifrados de excitación del decodificador	Utilizar bloque 19
30	Adaptador de ganancia hacia atrás del decodificador	Igual que el bloque 20
31	Unidad de escalamiento de ganancia del decodificador	Utilizar bloque 19
32	Filtro de síntesis del decodificador	Bloque 32
33	Adaptador de filtro de síntesis del decodificador	Igual que el bloque 23
34	Posfiltro	Véanse los bloques 71-77
35	Adaptador del posfiltro	Véanse los bloques 81-85
36	Ventana híbrida para W(z)	Bloque 36
37	Recursión de Durbin para W(z)	Véase G.2
38	Calculador de los coeficientes de W(z)	Bloque 38
43	Ventana híbrida para GP(z)	Bloque 43
44	Recursión de Durbin para GP(z)	Véase G.2
45	Ampliación de anchura de banda de GP(z)	Bloque 45
46	Predictor lineal de ganancia logarítmica	Bloque 46
48	Calculador logarítmico inverso	Bloque 46
49	Ventana híbrida para A(z)	Bloque 49
50	Recursión de Durbin para A(z)	Véase G.2
51	Calculador de los coeficientes de A(z)	Bloque 51
71-77	Bloques dentro del posfiltro	Bloques correspondientes
81-85	Bloques en el adaptador del posfiltro	Bloques correspondientes
91	Unidad de retardo de un índice de código cifrado de ganancia	No se necesita
92	Unidad de retardo de un índice de código cifrado de forma	No se necesita
93	Tabla de ganancia logarítmica de los códigos cifrados de ganancia	Tabla en G.5
94	Tabla de ganancia logarítmica de los códigos cifrados de forma	Tabla en G.5
95	Retardo de una muestra para ganancia logarítmica	No se necesita
96	Sumador para actualizar la ganancia logarítmica	Bloque 46
97	Limitador de la ganancia logarítmica a –32 dB	Bloque 46
98	Limitador de la ganancia logarítmica: –32 a 28 dB	Bloque 46
99	Sumador para restablecer el desplazamiento de la ganancia	Bloque 46

G.3.1 Bloque 4 – Seudocódigo para filtro de ponderación

Lo que sigue es el seudocódigo de coma flotante para el bloque 4, el filtrado de la señal vocal de entrada por el filtro de ponderación perceptual.

```
For   K = 1, 2, ..., IDIM, do the following
      SW(K) = S(K)
      For  J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines
           SW(K) = SW(K) + WFIR(J) * AWZ(J + 1)          | Parte «todos ceros»
           WFIR(J) = WFIR(J – 1)                        | del filtro

      SW(K) = SW(K) + WFIR(1) * AWZ(2)                  | Tratar la última
      WFIR(1) = S(K)                                    | de manera diferente

      For  J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines
           SW(K) = SW(K) – WIIR(J) * AWP(J + 1)         | Parte «todos polos»
           WIIR(J) = WIIR(J – 1)                        | del filtro

      SW(K) = SW(K) – WIIR(1) * AWP(2)                  | Tratar la última
      WIIR(1) = SW(K)                                   | de manera diferente
```

Repeat the above for the next K

Para la versión en coma fija de este seudocódigo hay los valores de NLS asociados con WFIR y WIIR. El cálculo debe hacerse de manera que la señal vocal de entrada tenga el mismo NLS que WFIR y el resultado de este cálculo tenga el mismo NLS que WIIR. En este caso, los valores de NLS para WIIR y WFIR están fijados en el mismo valor que la señal vocal de entrada. AWZ y AWP están en Q14. El valor para la señal vocal de entrada, NLSS, es 2. Se supone que diferentes formatos de entrada (lineal de 16 bits, ley μ, ley A, etc.) se convierten a la gama de [–4096 a +4095,75] representada en un formato Q2.

Para entrada lineal de K bits se supone que los datos ocupan los K bits menos significativos de una palabra de 16 bits, K_BIT_SAMPLE. La representación apropiada viene dada por:

$$NLS = 15 - K$$

$$S = K\_BIT\_SAMPLE << NLS$$

Para señales de entrada lineales de 16 bits (16_BIT_SAMPLE), se requiere un desplazamiento de 1 bit a la derecha:

$$S = 16\_BIT\_SAMPLE >> 1$$

Para MIC de ley μ (MU) (MULAW_SAMPLE), el valor de muestra de mayor magnitud es 4015,5 y se supone que esto se representaría en formato Q1 como 8031. Para convertir al formato Q2 es necesario un desplazamiento de 1 bit a la izquierda:

$$S = MULAW\_SAMPLE << 1$$

Para MIC de ley A (ALAW_SAMPLE), la muestra de mayor magnitud es 2016, pero algunos valores de muestra tienen una parte fraccionaria de 0,5. En consecuencia, 2016 se representaría como 4032 en una palabra de 16 bits. Para poner este valor en la gama apropiada es preciso un desplazamiento de 2 bits a la izquierda:

$$S = ALAW\_SAMPLE << 2$$

Lo que sigue es el seudocódigo de coma fija para el bloque 4, el filtrado de la señal vocal de entrada por el filtro de ponderación perceptual.

```
For   K = 1, 2, ..., IDIM, do the following
      AA0 = S(K)
      AA0 = AA0 << 14
      For  J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines
           AA0 = AA0 + WFIR(J) * AWZ(J + 1)              | Parte «todos ceros»
           WFIR(J) = WFIR(J – 1)                        | del filtro

      AA0 = AA0 + WFIR(1) * AWZ(2)                      | Tratar la última
      WFIR(1) = S(K)                                    | de manera diferente
```

```
For  J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines
     AA0 = AA0 – WIIR(J) * AWP(J + 1)          | Parte «todos polos»
     WIIR(J) = WIIR(J – 1)                     | del filtro

AA0 = AA0 – WIIR(1) * AWP(2)                   | Tratar la última
AA0 = AA0 >> 14                                | de manera diferente

If AA0 > 32767, set AA0 = 32767                | Modo saturación para entrada
If AA0 < –32768, set AA0 = –32768              | de multiplicador más adelante

WIIR(1) = AA0                                  | Guardada palabra baja de 16 bits
SW(K) = AA0                                    | SW está en Q2
```

Repeat the above for the next K

G.3.2 Blockzir – Seudocódigo para filtros de ponderación perceptual y de síntesis durante el cálculo de la respuesta a entrada cero

Lo que sigue es el seudocódigo de coma flotante para el bloque 9 (el filtro de síntesis) durante el cálculo de la respuesta a entrada cero.

```
For  K = 1, 2, ..., IDIM, do the following
     TEMP(K) = 0
     For  J = LPC, LPC – 1, ..., 3, 2, do the next 2 lines
          TEMP(K) = TEMP(K) – STATELPC(J) * A(J + 1)     | Multiplicar – adicionar
          STATELPC(J) = STATELPC(J – 1)                 | Cambio de memoria

     TEMP(K) = TEMP(K) – STATELPC(1) * A(2)             | Tratar la última
     STATELPC(1) = TEMP(K)                              | de manera diferente
```

Repeat the above for the next K

Lo que sigue es el seudocódigo de coma flotante para el bloque 10 (el filtro de ponderación perceptual) durante el cálculo de la respuesta a entrada cero.

```
For  K = 1, 2, ..., IDIM, do the following
     TMP = TEMP(K)
     For  J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines
          TEMP(K) = TEMP(K) + ZIRWFIR(J) * AWZ(J + 1)    | Parte «todos ceros»
          ZIRWFIR(J) = ZIRWFIR(J – 1)                   | del filtro

     TEMP(K) = TEMP(K) + ZIRWFIR(1) * AWZ(2)            | Tratar la última
     ZIRWFIR(1) = TMP

     For  J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines
          TEMP(K) = TEMP(K) – ZIRWIIR(J) * AWP(J + 1)    | Parte «todos polos»
          ZIRWIIR(J) = ZIRWIIR(J – 1)                   | del filtro

     ZIR(K) = TEMP(K) – ZIRWIIR(1) * AWP(2)             | Tratar la última
     ZIRWIIR(1) = ZIR(K)                                | de manera diferente
```

Repeat the above for the next K

En el código de coma fija se observa que STATELPC es de coma flotante de bloque segmentado y tiene asociado NLSSTATE. Puesto que hay entrada cero no es necesario hacer concordar NLSSTATE con el NLS de la entrada. Los valores de A( ), AWZ( ) y AWP( ) se representan siempre en formato Q14.

Lo que sigue es el seudocódigo de coma fija para el bloque 9 (el filtro de síntesis) durante el cálculo de la respuesta a entrada cero.

```
NLSSTATE(11) = NLSSTATE(1)
For  K = 2, 3, 4, ..., 10, do the next line     | Encontrar NLSSTATE mínimo
     If NLSSTATE(K) < NLSSTATE(11), set NLSSTATE(11) = NLSSTATE(K)

For  K = 1, 2, ..., IDIM, do the following
     I = 1
```

```
    L = 6 – K
    J = LPC
    AA0 = 0
    For  LL = 1, ..., L, do the next 3 lines
        AA0 = AA0 – STATELPC(J) * A(J + 1)          | Multiplicar – adicionar
        STATELPC(J) = STATELPC(J – 1)               | Cambio de memoria
        J = J – 1
    NLS = NLSSTATE(I) – NLSSTATE(11)
    AA1 = AA0 >> NLS

    For  I = 2, ..., 10, do the next 8 lines
        AA0 = 0
        For   LL = 1, 2, ..., IDIM, do the next 3 lines
            AA0 = 0 – STATELPC(J) * A(J + 1)
            STATELPC(J) = STATELPC(J – 1)           | STATELPC(0) = intrascendente si J = 1; es correcto
            J = J – 1
        NLS = NLSSTATE(I) – NLSSTATE(11)
        AA0 = AA0 >> NLS                            | Desplazar para alinear
        AA1 = AA1 + AA0

    If K = 1, go to SHIFT2
    L = K – 1
    AA0 = 0
    For  LL = 1, 2, ..., L, do the next 3 lines
        AA0 = AA0 – STATELPC(J) * A(J + 1)
        STATELPC(J) = STATELPC(J – 1)               | STATELPC(0) = intrascendente si J = 1; es correcto
        J = J – 1
    AA1 = AA1 + AA0                                 | No es necesario un desplazamiento por ahora

SHIFT2: AA1 = AA1 >> 14                             | A( ) estaba en Q14, NLS de AA1
                                                    | es ahora NLSSTATE(11)
    If AA1 > 32 767, set AA1 = 32 767               | Recortar a 16 bits si es necesario puesto que
    If AA1 < –32 768, set AA1 = –32 768             | STATELPC(1) será una entrada de multiplicador

    STATELPC(1) = AA1                               | Guardar la palabra baja 16 bits para
                                                    | STATELPC
    IR = NLSSTATE(11) – 2                           | Poner TEMP en formato Q2
    If IR > 0, set AA1 = AA1 >> IR                  |
    If IR < 0, set AA1 = AA1 << –IR                 |
    TEMP(K) = AA1

Repeat the above for the next K

Call VSCALE(STATELPC, IDIM, IDIM, 13, STATELPC, NLS)
NLSSTATE(11) = NLSSTATE(11) + NLS                   | Renormalizar el nuevo STATELPC a 15 bits

For     L = 1, 2, ..., 10, do the next line         | Actualizar NLSSTATE
        NLSSTATE(L) = NLSSTATE(L + 1)
```

En el seudocódigo de coma fija para el bloque 10, TEMP, ZIRWFIR y ZIRWIIR están en Q2. En el bloque anterior se creó TEMP explícitamente con este valor. Por eso no se necesita normalizar para sumarlos. Lo que sigue es el seudocódigo de coma fija para el bloque 10 (filtro de ponderación perceptual) durante el cálculo de la respuesta a entrada cero.

```
For K = 1, 2, ..., IDIM, do the following
    AA0 = TEMP(K) << 14                             | Porque AWZ está en Q14
    For  J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines
        AA0 = AA0 + ZIRWFIR(J) * AWZ(J + 1)         | Parte «todos ceros»
        ZIRWFIR(J) = ZIRWFIR(J – 1)                 | del filtro

    AA0 = AA0 + ZIRWFIR(1) * AWZ(2)                 | Tratar la última
    ZIRWFIR(1) = TEMP(K)                            | de manera diferente
```

```
For  J = LPCW, LPCW – 1, ..., 3, 2, do the next 2 lines
     AA0 = AA0 – ZIRWIIR(J) * AWP(J + 1)        | Parte «todos polos»
     ZIRWIIR(J) = ZIRWIIR(J – 1)                | del filtro

AA0 = AA0 – ZIRWIIR(1) * AWP(2)                 | Tratar la última
AA0 = AA0 >> 14                                 | de manera diferente

If AA0 > 32767, set AA0 = 32767                 | Recortar puesto que ZIR y ZIRWIIR
If AA0 < –32768, set AA0 = –32768               | serán entradas de multiplicador

ZIR(K) = AA0                                    | Guardar la palabra baja de 16 bits
ZIRWIIR(1) = AA0                                | para ZIR y ZIRWIIR
```

Repeat the above for the next K

G.3.3 Bloques 9 y 10 – Seudocódigo para actualizaciones de la memoria del filtro de ponderación perceptual y de síntesis

Lo que sigue es el seudocódigo de coma flotante para los bloques 9 y 10, la actualización de la memoria del filtro.

```
ZIRWFIR(1) = ET(1)                              | ZIRWFIR es ahora un ordenamiento transitorio
TEMP(1) = ET(1)
For  K = 2, 3, ..., IDIM, do the following
     A0 = ET(K)
     A1 = 0
     A2 = 0
     For  I = K, K – 1, ..., 2, do the next 5 lines
          ZIRWFIR(I) = ZIRWFIR(I – 1)
          TEMP(I) = TEMP(I – 1)
          A0 = A0 – A(I) * ZIRWFIR(I)           | Calcular las respuestas de estado cero
          A1 = A1 + AWZ(I) * ZIRWFIR(I)         | en varias etapas
          A2 = A2 – AWP(I) * TEMP(I)            | del filtro en cascada

     ZIRWFIR(1) = A0
     TEMP(1) = A0 + A1 + A2
```

Repeat the above indented section for the next K

```
                                                | Actualizar ahora la memoria del filtro
                                                | añadiendo las respuestas de estado cero
                                                | a las respuestas de entrada cero

For  K = 1, 2, ..., IDIM, do the next 4 lines
     STATELPC(K) = STATELPC(K) + ZIRWFIR(K)
     If STATELPC(K) > MAX, set STATELPC(K) = MAX  | Limitar la gama
     If STATELPC(K) < MIN, set STATELPC(K) = MIN
     ZIRWIIR(K) = ZIRWIIR(K) + TEMP(K)

For  I = 1, 2, ..., LPCW, do the next line      | Asignar ahora ZIRWFIR
     ZIRWFIR(I) = STATELPC(I)                   | al valor correcto

I = IDIM + 1
For  K = 1, 2, ..., IDIM, do the next line      | Obtener la señal vocal cuantificada
     ST(K) = STATELPC(I – K)                    | invirtiendo el orden de la memoria
                                                | del filtro de síntesis
```

Lo que sigue es el seudocódigo de coma fija para el mismo bloque. STATELPC tiene 10 exponentes almacenados en NLSSTATE(1), ..., NLSSTATE(10). Asociado con el ordenamiento ET está NLSET. ZIRWIIR y ZIRWFIR están en Q2 después de la actualización. ZIRWFIR se utiliza inicialmente como un ordenamiento transitorio. Al entrar en este código, tanto ET como los 5 elementos superiores de STATELPC [STATELPC(1) a STATELPC(5)] son ordenamientos de coma flotante de bloque de 15 bits. Cuando ET es filtrado por el filtro de síntesis LPC sin memoria, la salida (es decir, la respuesta de estado cero del filtro LPC) puede rebasar la gama de 15 bits. Si tal cosa ocurre, se desplaza ET 1 bit a la derecha y se repite el cálculo hasta que la salida quepa en 15 bits. Empíricamente, el proceso se repite a lo

sumo 3 veces (ó 4, si se cuenta la primera vez). Obsérvese que sólo hay 10 «multiplicar – adicionar» por cada repetición del cálculo, porque el cálculo de la respuesta de estado cero del filtro de ponderación se ha trasladado a un bucle separado. La respuesta de estado 0 del filtro LPC calculada de esta manera se puede representar siempre con 15 bits o menos. Cuando esta respuesta se añade a continuación a STATELPC de 15 bits para actualizar STATELPC, el resultado de la adición tiene garantizada su representabilidad mediante 16 bits. Antes de salir de este código, se escala STATELPC a 14 bits, para evitar desbordamientos en el cálculo posterior de la respuesta de entrada cero.

```
                                                    | Calcular primero la respuesta de estado cero
LABEL1: ZIRWFIR(1) = ET(1)                          | del filtro de síntesis LPC
For   K = 2, 3, ..., IDIM, do the following indented lines
      AA0 = ET(K) << 14                             | Porque A(1) = 1 en Q14 = 16384
      For  I = K, K – 1, ..., 2, do the next 3 lines
           ZIRWFIR(I) = ZIRWFIR(I – 1)
           P = A(I) * ZIRWFIR(I)                    | Multiplicación de Q14
           AA0 = AA0 – P                            | Calcular las respuestas de estado cero

      AA1 = AA0 << 3
      If AA1 overflowed above, do the next 4 lines  | Asegurarse de que después de AA0 >> 14,
           For   I = 1, 2, ..., IDIM, do the next line | el resultado no excede de 15 bits.
                 ET(I) = ET(I) >> 1                 | Si excediera, entonces ET >> 1
           NLSET = NLSET – 1                        | y se repite el cálculo
           GO TO LABEL1                             | hasta que quepa

      AA0 = AA0 >> 14                               | Compensar porque A( ) está en Q14
      ZIRWFIR(1) = AA0                              | Mantener los 16 bits más bajos

Repeat the above indented section for the next K

N = IDIM + 1                                        | Calcular ahora la respuesta de estado cero
TEMP(1) = ZIRWFIR(IDIM)                             | del filtro de ponderación
For   K = 2, 3, ..., IDIM, do the following indented lines
      AA1 = ZIRWFIR(N – K) << 14                    | Porque AWZ(1) = 1 (en Q14 = 16384)
      M = IDIM – K
      For  I = K, K – 1, ..., 2, do the next 5 lines
           TEMP(I) = TEMP(I – 1)                    | Cambiar la parte «todos polos» de la memoria del
           P = AWZ(I) * ZIRWFIR(I + M)              | filtro. Parte «todos ceros» del filtro de ponderación
           AA1 = AA1 + P
           P = AWP(I) * TEMP(I)                     | Parte «todos polos» del filtro de ponderación

           AA1 = AA1 – P

      AA1 = AA1 >> 14
      If AA1 > 32767, set AA1 = 32767               | Recortar si es necesario, puesto que TEMP(1)
      If AA1 < –32768, set AA1 = –32768             | será una entrada de 16 bits de multiplicador
      TEMP(1) = AA1                                 | Mantener los 16 bits más bajos

Repeat the above indented section for the next K

IR = NLSET – 2
For   K = 1, ..., IDIM, do the next 2 lines         | Cambiar ahora TEMP a Q2 como
      If IR > 0, set TEMP(K) = TEMP(K) >> IR        | ZIRWIIR
      If IR < 0, set TEMP(K) = TEMP(K) << –IR

                                                    | Actualizar ahora la memoria del filtro añadiendo
                                                    | las respuestas de estado cero a las respuestas
                                                    | a entrada cero. Primero hay que hacer concordar
                                                    | el NLS de ZIRWFIR y de STATELPC

If NLSET = NLSSTATE(10), go to LABEL2               | No se necesitan cambios
```

```
If NLSET < NLSSTATE(10), do the next 5 lines          | Pérdida de precisión en STATELPC
      NLSD = NLSSTATE(10) – NLSET                     | por los bits de NLSD
      For  K = 1, 2, ..., IDIM, do the next line
           STATELPC(K) = STATELPC(K) >> NLSD
      NLSSTATE(10) = NLSET
      go to LABEL2                                    | El único caso dejado es
                                                      | NLSET > NLSSTATE
NLSD = NLSET – NLSSTATE(10)                           | Pérdida de precisión en ZIRWFIR
For   K = 1, 2, ..., IDIM, do the next line           | por los bits de NLSD
      ZIRWFIR(K) = ZIRWFIR(K) >> NLSD

LABEL2:                                               | Ahora está preparado
AA1 = 4095                                            | 4095 = nivel de recorte de STATELPC
If NLSSTATE(10) ≥ 0, set AA1 = AA1 << NLSSTATE(10)    | Cambiar el nivel de recorte para
If NLSSTATE(10) < 0, set AA1 = AA1 >> –NLSSTATE(10)   | alinear con STATELPC

For   K = 1, 2, ..., IDIM, do the following indented lines
      AA0 = STATELPC(K) + ZIRWFIR(K)                  | Actualizar la memoria del filtro LPC.
      If AA0 > AA1, set AA0 = AA1                     | Si es necesario, efectuar el recorte tal como
      If AA0 < –AA1, set AA0 = –AA1                   | se especifica en la Rec. G.728 para coma flotante.
                                                      | Obsérvese que estos valores fueron escalados,
                                                      | por lo que, si 32767 < | AA0 | < AA1, es necesario
      If AA0 > 32767, set AA0 = 32767                 | recortar AA0 a 16 bits puesto que STATELPC(K)
      If AA0 < –32768, set AA0 = –32768               | será más tarde una entrada de 16 bits
      STATELPC(K) = AA0                               |  de multiplicador

      AA0 = ZIRWIIR(K) + TEMP(K)                      | Actualizar la parte «todos polos» de la memoria
      If AAO > 32767, set AA0 = 32767                 | de W(z). Recortar de nuevo a 16 bits, si es necesario,
      If AA0 < –32768, set AA0 = –32768               | puesto que ZIRWIIR(K) será más tarde
      ZIRWIIR(K) = AA0                                | una entrada de 16 bits de multiplicador

Repeat the above indented section for the next K

Call VSCALE(STATELPC, IDIM, IDIM, 12, STATELPC, NLS)  | Escalar STATELPC a 14 bits
NLSSTATE(10) = NLSSTATE(10) + NLS                     | para evitar el desbordamiento en el cálculo
                                                      | posterior de la respuesta a entrada cero

IR = NLSSTATE(10) – 2

For   I = 1, 2, ..., 5, do the next 4 lines           | Asignar ahora ZIRWFIR,
      AA0 = STATELPC(I)                               | la parte «todos ceros» de la memoria W(z)
      If IR > 0, set AA0 = AA0 >> IR                  | a los valores correctos en formato Q2
      If IR < 0, set AA0 = AA0 << –IR                 |
      ZIRWFIR(I) = AA0                                |
IR = NLSSTATE(9) – 2                                  |
For   I = 6, 7, ..., 10, do the next 4 lines          |
      AA0 = STATELPC(I)                               |
      If IR > 0, set AA0 = AA0 >> IR                  |
      If IR < 0, set AA0 = AA0 << –IR                 |
      ZIRWFIR(I) = AA0                                |

I = IDIM + 1
For   K = 1, 2, ..., IDIM, do the next line           | Obtener la señal vocal cuantificada
      ST(K) = STATELPC(I – K)                         | invirtiendo el orden de las cinco ubicaciones
NLSST = NLSSTATE(10)                                  | superiores de la memoria del filtro de síntesis
                                                      | NLSST sólo se utiliza en el decodificador
```

G.3.4 Bloque 11 – Cálculo del vector objetivo VQ

Lo que sigue es el seudocódigo de coma flotante para el bloque 11, cálculo de vector objetivo VQ.

```
For    K = 1, 2, ..., IDIM, do the next line
       TARGET(K) = SW(K) – ZIR(K)
```

Para el código de coma fija, SW y ZIR están en formato Q2, al igual que la señal vocal de entrada, por lo que NLSTARGET = 2. El seudocódigo de coma fija es como sigue.

```
set NLSTARGET = 2

For    K = 1, 2, ..., IDIM, do the next 6 lines
       AA0 = SW(K)
       AA1 = ZIR(K)
       AA0 = AA0 – AA1
       If AA0 > 32767, set AA0 = 32767                  | Recortar si es necesario
       If AA0 < –32768, set AA0 = –32768
       TARGET(K) = AA0
```

G.3.5 Bloque 12 – Cálculo del vector de respuesta a impulsos

Lo que sigue es el seudocódigo de coma flotante para el bloque 12.

```
TEMP(1) = 1                                             | TEMP = memoria del filtro de síntesis
WS(1) = 1                                               | WS = memoria de la parte «todos polos» de W(z)
For    K = 2, 3, ..., IDIM, do the following
       A0 = 0
       A1 = 0
       A2 = 0
       For  I = K, K – 1, ..., 3, 2, do the next 5 lines
            TEMP(I) = TEMP(I – 1)
            WS(I) = WS(I – 1)                           |
            A0 = A0 – A(I) * TEMP(I)                    | Filtrado
            A1 = A1 + AWZ(I) * TEMP(I)                  |
            A2 = A2 – AWP(I) * WS(I)

       TEMP(1) = A0
       WS(1) = A0 + A1 + A2

Repeat the above indented section for the next K

ITMP = IDIM + 1                                         | Obtener h(n) invirtiendo el orden de la
For    K = 1, 2, ..., IDIM, do the next line            | memoria de la sección «todos polos» de W(z)
       H(K) = WS(ITMP – K)                              |
```

Los valores de los coeficientes del predictor, A( ), AWZ( ) y AWP( ) están almacenados, todos ellos, en formato Q14. En el seudocódigo de coma fija que viene a continuación, sólo están indicados dos acumuladores de 32 bits, AA0 y AA1. Los acumuladores A1 y A2 del seudocódigo de coma flotante se han combinado. No se necesitan bits de guarda. El ordenamiento de salida, H( ), se almacena en formato Q13. El seudocódigo de coma fija es como sigue.

```
TEMP(1) = 8192                                          | TEMP = memoria del filtro de síntesis
WS(1) = 8192                                            | WS = memoria de la sección «todos polos» de W(z)
                                                        | WS & TEMP son palabras de 16 bits en Q13

For    K = 2, 3, ..., IDIM, do the following
       AA0 = 0
       AA1 = 0
       For  I = K, K – 1, ..., 3, 2, do the next 5 lines
            TEMP(I) = TEMP(I – 1)
            WS(I) = WS(I – 1)                           |
            AA0 = AA0 – A(I) * TEMP(I)                  | Filtrado
            AA1 = AA1 + AWZ(I) * TEMP(I)                |
            AA1 = AA1 – AWP(I) * WS(I)
```

```
    AA1 = AA0 + AA1
    AA0 = AA0 >> 14                          | >> 14 porque A( ), AWZ( ) y
    AA1 = AA1 >> 14                          | AWP( ) estaban en formato Q14
    TEMP(1) = AA0
    WS(1) = AA1
```

Repeat the above indented section for the next K

```
ITMP = IDIM + 1                              | Obtener h(n) invirtiendo el orden
For  K = 1, 2, ..., IDIM, do the next line   | de la memoria de la sección «todos polos» de W(z)
     H(K) = WS(ITMP – K)                     |
```

G.3.6 Bloque 13 – Convolución con inversión del tiempo

Este módulo efectúa la convolución con inversión del tiempo en preparación de la búsqueda de códigos cifrados. El seudocódigo de coma flotante original era:

```
For  K = 1, 2, ..., IDIM, do the following
     K1 = K – 1
     PN(K) = 0
     For  J = K, K + 1, ..., IDIM, do the next line
          PN(K) = PN(K) + TARGET(J) * H(J – K1)
```

Repeat the above for the next K

En la versión en coma fija, H( ) se representa en formato Q13 y TARGET se representa en coma flotante de bloque. NLSTARGET se determina en el bloque 16. NLSPN está fijado en 7.

```
For  K = 1, 2, ..., IDIM, do the following
     K1 = K – 1
     AA0 = 0                                 | Acumulador puesto a cero
     For  J = K, K + 1, ..., IDIM, do the next 2 lines
          P = TARGET(J) * H(J – K1)
          AA0 = AA0 + P
     AA0 = AA0 >> 13 + (NLSTARGET – 7)       | Desplazamiento a la derecha para poner en Q7
     If AA0 > 32767, set AA0 = 32767         | Recortar AA0 a 16 bits puesto que PN
     If AA0 < –32768, set AA0 = –32768       | será una entrada de multiplicador
     PN(K) = AA0                             | AA0 en modo saturación
```

Repeat the above for the next K

G.3.7 Bloque 14 – Convolución de los vectores de código de forma y cálculo de la energía

Lo que sigue es el seudocódigo para el cálculo de la energía de los vectores de código, bloques 14 y 15.

```
For  J = 1, 2, ..., NCWD, do the following   | Un vector de código por bucle
     J1 = (J – 1) * IDIM
     For  K = 1, 2, ..., IDIM, do the next 4 lines
          K1 = J1 + K + 1
          TEMP(K) = 0
          For  I = 1, 2, ..., K, do the next line
               TEMP(K) = TEMP(K) + H(I) * Y(K1 – I)   | Convolución

     Repeat the above 4 lines for the next K

     Y2(J) = 0
     For  K = 1, 2, ..., IDIM, do the next line
          Y2(J) = Y2(J) + TEMP(K) * TEMP(K)   | Calcular la energía
```

Repeat the above for the next J

En el seudocódigo de coma fija, H( ) se representa en formato Q13 e Y( ) en formato Q11. Se ha encontrado empíricamente que, tras la convolución de H( ) e Y( ), no se producen desbordamientos en el acumulador, es decir, no hay resultados de magnitud superior a 2**31. Por eso, en el seudocódigo que viene a continuación, no se efectúa la prueba del desbordamiento en el acumulador. Esto hace que el código DSP correspondiente vaya más deprisa. El subsiguiente desplazamiento en 14 bits es necesario para posibilitar la representación de TEMP( ) en formato Q10. Se ha visto que las representaciones utilizadas funcionan con una diversidad de ficheros, NLSY2 = (NLSH + NLSY – 14) * 2 – 15. Puesto que NLSY = 11 y NLSH = 13, NLSY2 es igual a 5.

```
For   J = 1, 2, ..., NCWD, do the following                | Un vector de código por bucle
      J1 = (J – 1) * IDIM
      For  K = 1, 2, ..., IDIM, do the next 7 lines
           K1 = J1 + K + 1
           AA0 = 0
           For I = 1, 2, ..., K, do the next 2 lines       |
                P = H(I) * Y(K1 – I)                       | Convolución
                AA0 = AA0 + P                              |

                AA0 = AA0 >> 14
                TEMP(K) = AA0                              | Los 16 bits más bajos solamente

      Repeat the above 7 lines for the next K

      AA0 = 0
      For  K = 1, 2, ..., IDIM, do the next 2 lines
           P = TEMP(K) * TEMP(K)                           | Calcular la energía
           AA0 = AA0 + P
      AA0 = AA0 >> 15
      Y2(J) = AA0                                          | Los 16 bits más bajos solamente

Repeat the above for the next J
```

G.3.8 Bloque 16 – Normalización del vector objetivo VQ

Primero se da el seudocódigo de coma flotante para este módulo.

```
TMP = 1./GAIN
For       K = 1, 2, ..., IDIM, do the next line
          TARGET(K) = TARGET(K) * TMP
```

Para el seudocódigo de coma fija es preciso considerar el NLS de la ganancia y el NLS de TARGET. De entrada, NLSTARGET=2. En el proceso se creará el NLS para TMP.

```
                                                           | Numerador de la división = 16384
                                                           | NLS = 14 para el numerador
                                                           | NLS del denominador es NLSGAIN
                                                           | NLSGAIN se determina en el bloque 46

Call DIVIDE(16384, 14, GAIN, NLSGAIN, TMP, NLSTMP)

For   K = 1, 2, ..., IDIM, do the next 2 lines
      AA0 = TMP * TARGET(K)                                | AA0 es de 32 bits
      TARGET(K) = AA0 >> 15                                | Guardar los 16 bits más bajos solamente
                                                           | TARGET está en Q2 en este punto

NLSTARGET = 2 + NLSTMP – 15                                | Poner TARGET en coma flotante
                                                           | de bloque
Call VSCALE(TARGET, IDIM, IDIM, 14, TARGET, NLS)
NLSTARGET = NLSTARGET + NLS                                | NLS se cambió en VSCALE
```

G.3.9 Bloque 17 – Calculador de error de búsqueda VQ y selector del mejor índice de código cifrado

Lo que sigue es el seudocódigo de coma flotante para el calculador de error y selector del mejor índice de código cifrado (bloques 17 y 18).

```
Initialize DISTM to the largest number representable in the hardware
N1 = NG/2
For   J = 1, 2, ..., NCWD, do the following
      J1 = (J – 1) * IDIM
      COR = 0
      For  K = 1, 2, ..., IDIM, do the next line
           COR = COR + PN(K) * Y(J1 + K)                    | Calcular el producto interno Pj

      If COR > 0, then do the next 5 lines
           IDXG = N1
           For   K = 1, 2, ..., N1 – 1, do the next "if" statement
                 If    COR < GB(K) * Y2(J), do the next 2 lines
                       IDXG = K                             | Mejor ganancia positiva hallada
                       GO TO LABEL

      If COR ≤ 0, then do the next 5 lines
           IDXG = NG
           For   K = N1 + 1, N1 + 2, ..., NG – 1, do the next "if" statement
                 If    COR > GB(K) * Y2(J), do the next 2 lines
                       IDXG = K                             | Mejor ganancia negativa hallada
                       GO TO LABEL

LABEL: D = –G2(IDXG) * COR + GSQ(IDXG) * Y2(J)              | Calcular la distorsión D^
      If D < DISTM, do the next 3 lines
           DISTM = D                                        | Guardar la distorsión más baja
           IG = IDXG                                        | y los mejores índices de código cifrado
           IS = J                                           | hallados hasta ahora

Repeat the above inserted section for the next J

ICHAN = (IS – 1) * NG + (IG – 1)
```

Lo que sigue es el seudocódigo de coma fija para el calculador de error y selector del mejor índice de código cifrado (bloques 17 y 18). El código se ha escrito de manera que se utiliza el valor absoluto de la correlación con el valor objetivo, AA0 más abajo, para la búsqueda de la ganancia y a continuación se determina de nuevo el signo de la correlación. Aunque parezca más laborioso, este procedimiento evita tener una rama en medio del bucle de búsqueda. En la mayoría de las implementaciones de DSP, evitar una rama significa ahorrar instrucciones. De manera alternativa, se puede guardar el signo y evitarse el recálculo pero esto exige normalmente, también una instrucción adicional en el bucle de búsqueda.

```
DISTM = 2147483647
For   J = 1, 2, ..., NCWD, do the following
      J1 = (J – 1) * IDIM
      AA0 = 0
      For  K = 1, 2, ..., IDIM, do the next 2 lines
           P = PN(K) * Y(J1 + K)                            | Calcular el producto interno Pj
           AA0 = AA0 + P                                    | NLS para AA0 es 7 + 11 = 18
      If AA0 < 0, set AA0 = – AA0                           | Tomar el valor absoluto

      IDXG = 1
      P = GB(1) * Y2(J)                                     | NLS para P es 13 + 5 = 18
      If AA0 ≥ P, set IDXG = IDXG + 1
      P = GB(2) * Y2(J)
      If AA0 ≥ P, set IDXG = IDXG + 1
      P = GB(3) * Y2(J)
      If AA0 ≥ P, set IDXG = IDXG + 1
```

```
    AA0 = AA0 >> 14                                  | NLS para AA0 = 4
    If AA0 > 32767, set AA0 = 32767                  | Recortar AA0; AA0 en modo saturación
    AA1 = GSQ(IDXG) * Y2(J)                          | NLSGSQ = 11 y NLSY2 = 5 por lo que NLSAA1 = 16
    P = G2(IDXG) * AA0                               | NLSG2 = 12 y NLSAA0 = 4 por lo que NLSP = 16

    AA1 = AA1 – P
    If AA1 < DISTM, do the next 3 lines
        DISTM = AA1                                  | DISTM de precisión doble
        IG = IDXG
        IS = J
```

Repeat the above inserted section for the next J

```
AA0 = 0                                              | Encontrar ahora el bit de signo
J1 = (IS – 1) * IDIM
For  K = 1, 2, ..., IDIM, do the next 2 lines
    P = PN(K) * Y(J1 + K)                            | Calcular el producto interno
    AA0 = AA0 + P
If AA0 ≤ 0, set IG = IG + 4

ICHAN = (IS – 1) * NG + (IG – 1)
```

En el código anterior se han utilizado las cuatro líneas siguientes.

```
    AA0 = AA0 >> 14                                  | NLS para AA0 = 4
    If AA0 > 32 767, set AA0 = 32 767                | Recortar AA0
    AA1 = GSQ(IDXG) * Y2(J)                          | NLSGSQ = 11 y NLSY2 = 5 por lo que NLSAA1 = 16
    P = G2(IDXG) * AA0                               | NLSG2 = 12 y NLSAA0 = 4 por lo que NLSP = 16
```

En las microplaquetas DSP que tienen una función de «recorte» estas líneas pueden sustituirse por el código indicado a continuación que da los mismos resultados exactos.

```
    AA0 = AA0 << 2                                   | NLS para AA0 = 20
    AA0 = CLIP(AA0)                                  | AA0 está en modo saturación
    AA0 = AA0 >> 16                                  | Tomar la palabra alta; NLS para AA0 = 4
    AA1 = GSQ(IDXG) * Y2(J)                          | NLSGSQ = 11 y NLSY2 = 5 por lo que NLSAA1 = 16
    P = G2(IDXG) * AA0                               | NLSG2 = 12 y NLSAA0 = 4 por lo que NLSP = 16
```

La función CLIP y el modo saturación se refieren al concepto de no permitir que AA0 desborde cuando se efectúa la operación << 2. Para que no desborde, se fija AA0 al número máximo positivo o negativo, dependiendo de su signo original. En este caso, AA0 es siempre positivo. Esta alternativa depende del DSP y quizás necesite más de un acumulador de 32 bits. La alternativa del seudocódigo principal siempre puede ser implementada.

G.3.10 Bloque 19 – Códigos cifrados VQ de excitación y bloque 21 – Unidad de escalamiento de ganancia

Lo que sigue es la versión en coma flotante del seudocódigo para el bloque 19, los códigos cifrados VQ de excitación.

```
NN = (IS – 1) * IDIM
For  K = 1, 2, ..., IDIM, do the next line
    YN(K) = GQ(IG) * Y(NN + K)
```

A continuación se da la versión en coma flotante del seudocódigo para el bloque 21, unidad de escalamiento de ganancia.

```
For  K = 1, 2, ..., IDIM, do the next line
    ET(K) = GAIN * YN(K)
```

Para el seudocódigo de coma fija se combinan los bloques 19 y 21 en un solo módulo. Tanto Y como GQ tienen formatos Q fijos, Q11 y Q13 respectivamente. El valor de GAIN tiene asociado NLSGAIN. Para conseguir la máxima precisión se normaliza el producto GQ(IG) * GAIN a 32 bits antes de efectuar el redondeo de los 16 bits más altos. A NNGQ(I) se le asigna el valor [1 + el número de desplazamientos a la izquierda necesarios para normalizar GQ(I) en Q13], con lo que NNGQ(I) = 3 para I = 1, 2, 5, 6, NNGQ(I) = 2 para I = 3, 7 y NNGQ(I) = 1 para I = 4, 8. El seudocódigo puede entonces escribirse como a continuación se indica.

```
AA0 = GQ(IG) * GAIN                                  | AA0 tiene NNGQ(IG) ceros delanteros
AA0 = AA0 << NNGQ(IG)                                | Desplazar NNGQ(IG) bits a la izquierda
                                                     | para normalizar AA0
TMP = RND(AA0)                                       | Redondear a los 16 bits más altos y asignar a TMP

NLSAA0 = 13 + NLSGAIN                                | Formato Q del producto GQ(IG) * GAIN
NLSTMP = NLSAA0 + NNGQ(IG) – 16                      | Formato Q de TMP por el desplazamiento de AA0
                                                     | a la izquierda en NNGQ(IG) bits, redondear
                                                     | a continuación y tomar los 16 bits más altos

NN = (IS – 1) * IDIM                                 | Normalizar el vector de código
Call VSCALE(Y(NN + 1), IDIM, IDIM, 14, TEMP, NLS)    | de forma seleccionado a 16 bits; poner en TEMP

For   K = 1, 2, ..., IDIM, do the next 2 lines       | TMP y TEMP normalizados ambos a 16 bits
      AA0 = TMP * TEMP(K)                            | por lo que el producto tiene un cero delantero.
      ET(K) = RND(AA0)                               | El redondeo directo a la palabra alta
                                                     | da un ordenamiento ET de 15 bits

NLSET = NLSTMP + 11 + NLS – 16                       | Calcular el NLS para ET
```

G.3.11 Bloque 32 – Filtro de síntesis del decodificador

Lo que sigue es el seudocódigo de coma flotante para el bloque 32, el filtro de síntesis del decodificador.

```
For   K = 1, 2, ..., IDIM, do the next 6 lines
      TEMP(K) = 0
      For  J = LPC, LPC – 1, ..., 3, 2, do the next 2 lines
           TEMP(K) = TEMP(K) – STATELPC(J) * A(J + 1)   | Respuesta de entrada cero
           STATELPC(J) = STATELPC(J – 1)

      TEMP(K) = TEMP(K) – STATELPC(1) * A(2)           | Tratar la última de manera diferente
      STATELPC(1) = TEMP(K)                            |

Repeat the above for the next K

TEMP(1) = ET(1)
For   K = 2, 3, ..., IDIM, do the next 5 lines
      A0 = ET(K)
      For  I = K, K – 1, ..., 2, do the next 2 lines
           TEMP(I) = TEMP(I – 1)
           A0 = A0 – A(I) * TEMP(I)                    | Calcular la respuesta de estado cero

      TEMP(1) = A0

Repeat the above 5 lines for the next K

                                                       | Actualizar ahora la memoria del filtro
                                                       | sumando las respuestas de estado
                                                       | cero a las respuestas de entrada cero
For   K = 1, 2, ..., IDIM, do the next 3 lines
      STATELPC(K) = STATELPC(K) + TEMP(K)               | ZIR + ZSR
      If STATELPC(K) > MAX, set STATELPC(K) = MAX       | Limitar la gama
      If STATELPC(K) < MIN, set STATELPC(K) = MIN       |

I = IDIM + 1
For   K = 1, 2, ..., IDIM, do the next line            | Obtener la señal vocal cuantificada
      ST(K) = STATELPC(I – K)                          | invirtiendo el orden de la memoria
                                                       | del filtro de síntesis
```

El seudocódigo de coma fija para el bloque 32 sigue la misma metodología utilizada en el bloque 9, excepto que no hay actualización de la memoria del filtro de ponderación perceptual.

```
NLSSTATE(11) = NLSSTATE(1)
For   K = 2, 3, 4, ..., 10, do the next line                    | Encontrar NLSSTATE mínimo
      If NLSSTATE(K) < NLSSTATE(11), set NLSSTATE(11) = NLSSTATE(K)

For   K = 1, 2, ..., IDIM, do the following
      I = 1
      L = 6 – K
      J = LPC
      AA0 = 0
      For  LL = 1, ..., L, do the next 3 lines
           AA0 = AA0 – STATELPC(J) * A(J + 1)                   | Multiplicar – adicionar
           STATELPC(J) = STATELPC(J – 1)                        | Cambio de la memoria
           J = J – 1
      NLS = NLSSTATE(I) – NLSSTATE(11)
      AA1 = AA0 >> NLS

      For  I = 2, ..., 10, do the next 8 lines
           AA0 = 0
           For LL = 1, 2, ..., IDIM, do the next 3 lines
                AA0 = AA0 – STATELPC(J) * A(J + 1)
                STATELPC(J) = STATELPC(J – 1)                   | STATELPC(0) = intrascendente si J = 1; es correcto
                J = J – 1
           NLS = NLSSTATE(I) – NLSSTATE(11)
           AA0 = AA0 >> NLS                                     | Desplazar para alinear
           AA1 = AA1 + AA0

      If K = 1, go to SHIFT2
      L = K – 1
      AA0 = 0
      For  LL = 1, 2, ..., L, do the next 3 lines
           AA0 = AA0 – STATELPC(J) * A(J + 1)
           STATELPC(J) = STATELPC(J – 1)                        | STATELPC(0) = intrascendente si J = 1; es correcto
           J = J – 1
      AA1 = AA1 + AA0                                           | No es necesario un desplazamiento por ahora

SHIFT2: AA1 = AA1 >> 14                                         | A( ) estaba en Q14, NLS del AA1
                                                                | es ahora NLSSTATE(11)
      If AA1 > 32767, set AA1 = 32767                           | Recortar 16 bits si es necesario puesto que
      If AA1 < –32768, set AA1 = –32768                         | STATELPC(1) será entrada de multiplicador

      STATELPC(1) = AA1                                         | Guarda la palabra baja de 16 bits
      IR = NLSSTATE(11) – 2                                     | para STATELPC
      If IR > 0, set AA1 = AA1 >> IR                            | Poner TEMP Q2 en format Q2
      If IR < 0, set AA1 = AA1 << – IR                          |
      TEMP(K) = AA1

Repeat the above for the next K

Call VSCALE(STATELPC, IDIM, IDIM, 13, STATELPC, NLS)
NLSSTATE(11) = NLSSTATE(11) + NLS                               | Renormalizar el nuevo STATELPC a 15 bits

For   L = 1, 2, ..., 10, do the next line                       | Actualizar NLSSTATE
      NLSSTATE(L) = NLSSTATE(L + 1)

                                                                | Calcular primero la respuesta de estado cero
LABEL1: TEMP(1) = ET(1)                                         | del filtro de síntesis LPC
For   K = 2, 3, ..., IDIM, do the following indented lines
      AA0 = ET(K) << 14                                         | Porque A(1) = 1 en Q14 = 16384
      For  I = K, K – 1, ..., 2, do the next 3 lines
           TEMP(I) = TEMP(I – 1)
           P = A(I) * TEMP(I)                                   | Multiplicación en Q14
           AA0 = AA0 – P                                        | Calcular las respuestas de estado cero
```

```
    AA1 = AA0 << 3
    If AA1 overflowed above, do the next 4 lines          | Asegurarse de que después de AA0 >> 14
        For  I = 1, 2, ..., IDIM, do the next line        | el resultado no excede de 15 bits.
            ET(I) = ET(I) >> 1                            | Si excediera, entonces ET >> 1
        NLSET = NLSET – 1                                 |  y se repite el cálculo hasta
        GO TO LABEL1                                      | que quepa

    AA0 = AA0 >> 14                                       | Compensar porque A( ) está en Q14
    TEMP(1) = AA0                                         | Mantener los 16 bits más bajos

Repeat the above indented section for the next K

If NLSET = NLSSTATE(10), go to LABEL2                     | No se necesitan cambios

If NLSET < NLSSTATE(10), do the next 5 lines              | Pérdida de precisión en STATELPC
    NLSD = NLSSTATE(10) – NLSET                           | por los bits de NLSD
    For  K = 1, 2, ..., IDIM, do the next line
        STATELPC(K) = STATELPC(K) >> NLSD
    NLSSTATE(10) = NLSET
    go to LABEL2                                          | El único caso debajo es NLSET > NLSSTATE
                                                          | Pérdida de precisión en TEMP
NLSD = NLSET – NLSSTATE(10)                               | por los bits de NLSD
For  K = 1, 2, ..., IDIM, do the next line                |
    TEMP(K) = TEMP(K) >> NLSD

LABEL2:                                                   | Ahora está preparado
AA1 = 4095                                                | 4095 = nivel de recorte de STATELPC
If NLSSTATE(10) ≥ 0, set AA1 = AA1 << NLSSTATE(10)        | Cambiar el nivel de recorte
If NLSSTATE(10) < 0, set AA1 = AA1 >> –NLSSTATE(10)       | para alinear con STATELPC

For  K = 1, 2, ..., IDIM, do the following indented lines
    AA0 = STATELPC(K) + TEMP(K)                           | Actualizar la memoria del filtro LPC
    If AA0 > AA1, set AA0 = AA1                           | Si es necesario, efectuar el recorte tal como
    If AA0 < –AA1, set AA0 = –AA1                         | se especifica en la Rec. G.728 para coma flotante.
                                                          | Obsérvese que estos valores fueron escalados,
                                                          | por lo que si 32767 > | AA0 | < AA1,
    If AA0 > 32767, set AA0 = 32767                       | es necesario recortar AA0 a 16 bits,
    If AA0 < –32768, set AA0 = –32768                     | ya que STATELPC(K) será más tarde una entrada
    STATELPC(K) = AA0                                     | de 16 bits de multiplicador

Repeat the above indented section for the next K

Call VSCALE(STATELPC, IDIM, IDIM, 12, STATELPC, NLS)      | Escalar STATELPC a 14 bits
NLSSTATE(10) = NLSSTATE(10) + NLS                         | para evitar el desbordamiento en
                                                          | el cálculo posterior de la respuesta de estado cero

I = IDIM + 1
For  K = 1, 2, ..., IDIM, do the next line                | Obtener la señal vocal cuantificada invirtiendo
    ST(K) = STATELPC(I – K)                               | el orden de las cinco ubicaciones superiores
NLSST = NLSSTATE(10)                                      | de la memoria del filtro de síntesis
                                                          | NLSST se utiliza más tarde en el decodificador
```

G.3.12 Bloque 36 – Seudocódigo para el módulo de ventanización híbrida

En esta subcláusula se dan tanto el seudocódigo de coma flotante como el de coma fija para el bloque 36. Primero se presenta el seudocódigo de coma flotante.

```
    N1 = LPCW + NFRSZ                                     | Calcular algunas constantes (pueden calcularse
    N2 = LPCW + NONRW                                     | y almacenarse previamente en memoria)
    N3 = LPCW + NFRSZ + NONRW

    For  N = 1, 2, ..., N2, do the next line
        SBW(N) = SBW(N + NFRSZ)                           | Cambiar la antigua memoria intermedia de señal
    For  N = 1, 2, ..., NFRSZ, do the next line
        SBW(N2 + N) = STMP(N)                             | Introducir la nueva señal
                                                          | SBW(N3) es la muestra más reciente
```

```
K = 1
For  N = N3, N3 – 1, ..., 3, 2, 1, do the next 2 lines
     WS(N) = SBW(N) * WNRW(K)                    | Multiplicar la función de ventana
     K = K + 1

For  I = 1, 2, ..., LPCW + 1, do the next 4 lines
     TMP = 0
     For   N = LPCW + 1, LPCW + 2, ..., N1, do the next line
           TMP = TMP + WS(N) * WS(N + 1 – I)
     REXPW(I) = (1/2) * REXPW(I) + TMP           | Actualizar la componente recursiva

For  I = 1, 2, ..., LPCW + 1, do the next 3 lines
     R(I) = REXPW(I)
     For   N = N1 + 1, N1 + 2, ..., N3, do the next line
           R(I) = R(I) + WS(N) * WS(N + 1 – I)   | Añadir la componente no recursiva

R(1) = R(1) * WNCF                               | Corrección por ruido blanco
```

A continuación se da la versión en coma fija del mismo módulo. En este código se han añadido varias variables nuevas. NLSREXPW es una variable global que contiene el número de desplazamientos a la izquierda para normalizar REXPW. Esta variable se inicializa con un valor de 31.

```
N1 = LPCW + NFRSZ (= 10 + 20)                    | Calcular algunas constantes (pueden calcularse
N2 = LPCW + NONRW (= 10 + 30)                    | y almacenarse previamente en memoria)
N3 = LPCW + NFRSZ + NONRW (= 10 + 20 + 30)

For  N = 1, 2, ..., N2, do the next line
     SBW(N) = SBW(N + NFRSZ)                     | Cambiar la antigua memoria intermedia de señal
For  N = 1, 2, ..., NFRSZ, do the next line
     SBW(N2 + N) = STMP(N)                       | SBW(N3) es la muestra más reciente
                                                 | Todas las SBW están en Q2 y se representan
                                                 | con una precisión de 15 bits

Call FINDNLS(SBW, N3, N3, 14, NLS)               | Encontrar el número de desplazamientos
                                                 | a la izquierda necesarios en el próximo
                                                 | bucle para obtener un margen de 2 bits.
                                                 | En realidad no es necesario efectuar el escalamiento.
                                                 | Se utiliza simplemente NLS

NLSTMP = NLS – 1
K = 1
For  N = 60, 59, ..., 1, do the next 4 lines
     P = SBW(N) * WNRW(K)                        | WNRW está en Q15; desplazada NLSWS bits
     AA0 = P << NLSWS                            | a la izquierda hará que el mayor
     WS(N) = RND(AA0)                            | elemento WS(N) sea un número
     K = K + 1                                   | de 14 bits (2 bits de margen para
                                                 | acumulación posterior)

NLSATTW = 15
Call
HWMCORE(LPCW, N1, N3, NLSATTW, WS, NLSTMP, REXPW, NLSREXPW, R, ILLCONDW)

If NLSREXPW > 41, set NLSREXPW = 41              | Para evitar que se reduzca la precisión en
                                                 | REXPW( ) y R( ) durante los largos periodos
                                                 | de señal de entrada cero
```

La subrutina HWMCORE figura en G.3.18.

En el código anterior, una invocación de FINDNLS busca en la totalidad de la memoria intermedia SBW de 60 muestras. Sin embargo, se puede utilizar un sustituto exacto a nivel de bit, que emplea dos palabras más de memoria, para reducir esa computación. SBW contendrá siempre 40 muestras antiguas y 20 nuevas, que se pueden dividir en tres vectores de 20 muestras cada uno. Se lleva la cuenta del NLS de cada uno de los tres vectores y se elige a continuación el valor mínimo, para utilizarlo al aplicar la ventana híbrida. Puesto que dos de los vectores se componen de muestras antiguas, su NLS respectivo será ya conocido. Sólo se necesita comprobar el vector más reciente para encontrar su NLS. A continuación hay que almacenar el NLS del vector más reciente y el del menos antiguo de los otros dos para la siguiente computación. Con este método se acaba seleccionando exactamente el mismo NLS que con el procedimiento mostrado en el seudocódigo anterior.

La tabla siguiente contiene la relación de todas las variables de este seudocódigo, con su formato de representación y tamaño para facilitar su referencia. En la tabla se indica si cada una de las variables es temporal (temp.), lo que significa que no es necesario almacenarla una vez que se haya completado el módulo, o permanente (perm.), en cuyo caso el valor se necesitará también después del cálculo en curso. En la tabla se indica además qué variables no estaban incluidas en el seudocódigo anterior de coma flotante (antigua/nueva).

Variable	Formato	Tamaño	Temp./perm.	Antigua/nueva
NLS	entero	1	temp.	nueva
NLSREXPW	entero	1	perm.	nueva
NLSTMP	entero	1	temp.	nueva
REXPW	BFL	11	perm.	antigua
R	BFL	1	perm.	antigua
SBW	Q2	60	perm.	antigua
STMP	Q2	20	perm.	antigua
WS	BFL	60	temp.	antigua
BFL Coma flotante de bloque Entero entero de 16 bits				

G.3.13 Bloque 38 – Calculador de los coeficientes del filtro de ponderación

Se empieza con el seudocódigo de coma flotante para este bloque.

```
If ICOUNT ≠ 3, skip the execution of this block
Otherwise, do the following

For  I = 2, 3, ..., 11, do the next line
     AWP(I) = WPCFV(I) * AWZTMP(I)          | Escalar los coeficientes del denominador

For  I = 2, 3, ..., 11, do the next line
     AWZ(I) = WZCFV(I) * AWZTMP(I)          | Escalar los coeficientes del numerador
```

En el seudocódigo de coma fija se debe considerar la posibilidad de que haya un desajuste en la recursión de Durbin, o que AWZTMP no pueda expresarse ni siquiera en Q13. (Nunca se ha observado que Q13 fuese insuficiente, pero de todos modos hay que tener en cuenta esa posibilidad.) La variable ILLCONDW es una bandera del bloque 37 que indica si los resultados de dicho bloque son válidos o no. En la Recomendación G.728 se supone implícitamente que no se utilizarán los resultados de Durbin si ILLCONDW es verdad. Esto es, AWZ y AWP no se actualizarán nunca a partir de AWZTMP. La misma suposición se establece aquí. Si ILLCONDW es verdad, no se actualizan AWP o AWZ. No es necesario hacerlo porque se van a seguir utilizando los valores anteriores.

A continuación hay que considerar la posibilidad de que los coeficientes AWZTMP( ) de la recursión de Durbin estén en Q13, Q14 o Q15. NLSAWZTMP es el número de desplazamientos a la izquierda de AWZTMP. Se quiere que los coeficientes del numerador y del denominador, AWZ y AWP, estén en Q14 para la salida. Puede ocurrir que AWZ no pueda representarse en Q14. Si tal es el caso, no se actualizan AWZ o AWP. El seudocódigo de coma fija viene dado por lo siguiente.

```
If ICOUNT ≠ 3, skip the execution of this block
Otherwise, do the following

                                        | Comprobar primero si ILLCONDW es verdad
```

```
If ILLCONDW = .TRUE., skip the execution of this block
Otherwise, do the following
                                                  | Hacer a continuación los coeficientes del
                                                  | numerador. Si desbordan para Q14,
                                                  | no actualizar AWZ o AWP
                                                  | En caso de desbordamiento se utiliza el ordenamiento
                                                  | temporal WS, con lo que se preserva AWZ

    For  I = 2, 3, ..., 7, do the next 6 lines
         AA0 = WZCFV(I) * AWZTMP(I)               | WZCFV está en Q14,
         If NLSAWZTMP = 13, AA0 = AA0 << 3        | AA0 es 14 + NLSAWZTMP. Poner AA0
         If NLSAWZTMP = 14, AA0 = AA0 << 2        | en Q30 en los 3 casos mediante
         If NLSAWZTMP = 15, AA0 = AA0 << 1        | el número apropiado de desplazamiento
         If AA0 overflowed above, go to LABEL     | Si es verdad, Q14 desbordará;
         WS(I) = RND(AA0)                         | redondear a la palabra alta para WS
                                                  | En los demás casos no se puede
                                                  | producir desbordamiento
                                                  | Si se ha llegado hasta aquí,
                                                  | continuar sin las comprobaciones
                                                  | Copiar a continuación WS en AWZ

    For  I = 8, 9, 10, 11, do the next 5 lines
         AA0 = WZCFV(I) * AWZTMP(I)
         If NLSAWZTMP = 13, AA0 = AA0 << 3
         If NLSAWZTMP = 14, AA0 = AA0 << 2
         If NLSAWZTMP = 15, AA0 = AA0 << 1
         WS(I) = RND(AA0)
    For  I = 2, 3, ..., 11, do the next line
         AWZ(I) = WS(I)                           | No hay desbordamiento, por lo que se
                                                  | copia WS en AWZ

                                                  | Hacer ahora los coeficientes
                                                  | del denominador
                                                  | Si el numerador no desbordó, tampoco
                                                  | puede desbordar el denominador

    For  I = 2, 3, ..., 11, do the next 5 lines
         AA0 = WPCFV(I) * AWZTMP(I)               | WPCFV está en Q14; AA0 es 14 + NLSAWZTMP
         If NLSAWZTMP = 13, AA0 = AA0 << 3        | Poner AA0 en Q30 en los 3 casos
         If NLSAWZTMP = 14, AA0 = AA0 << 2        | mediante el número apropiado de desplazamientos
         If NLSAWZTMP = 15, AA0 = AA0 << 1        |
         AWP(I) = RND(AA0)                        | Redondear a la palabra alta para AWP

    Exit this subroutine

LABEL:                                            | Si el programa llega hasta aquí, se producirá un
                                                  | desbordamiento si se intenta representar AWZ
                                                  | en Q14. En este caso no se actualizan los
                                                  | coeficientes del filtro de ponderación (es decir,
                                                  | se siguen utilizando los coeficientes del filtro del ciclo
                                                  | de adaptación anterior).
```

G.3.14 Bloque 43 – Módulo de ventanización híbrida

En esta subcláusula se dan el seudocódigo de coma flotante y el de coma fija para el bloque 43. Primero se presenta el seudocódigo de coma flotante.

```
    N1 = LPCLG + NUPDATE                          | Calcular algunas constantes (pueden
    N2 = LPCLG + NONRLG                           | calcularse y almacenarse previamente en memoria)
    N3 = LPCLG + NUPDATE + NONRLG

    For  N = 1, 2, ..., N2, do the next line
         SBLG(N) = SBLG(N + NUPDATE)              | Cambiar la antigua memoria intermedia de señal
    For  N = 1, 2, ..., NUPDATE, do the next line
         SBLG(N2 + N) = GTMP(N)                   | Introducir la nueva señal
                                                  | SBW(N3) es la muestra más reciente
```

```
K = 1
For  N = N3, N3 – 1, ..., 3, 2, 1, do the next 2 lines
     WS(N) = SBLG(N) * WNRLG(K)                    | Multiplicar la función de ventana
     K = K + 1

For  I = 1, 2, ..., LPCLG + 1, do the next 4 lines
     TMP = 0
     For   N = LPCLG + 1, LPCLG + 2, ..., N1, do the next line
           TMP = TMP + WS(N) * WS(N + 1 – I)
     REXPLG(I) = (3/4) * REXPLG(I) + TMP           | Actualizar la componente recursiva

For  I = 1, 2, ..., LPCLG + 1, do the next 3 lines
     R(I) = REXPLG(I)
     For   N = N1 + 1, N1 + 2, ..., N3, do the next line
           R(I) = R(I) + WS(N) * WS(N + 1 – I)     | Añadir la componente no recursiva

R(1) = R(1) * WNCF                                 | Corrección por ruido blanco
```

Obsérvese que antes de invocar esta rutina se asigna GTMP( ) como:

```
GTMP(1) = GSTATE(4)
GTMP(2) = GSTATE(3)
GTMP(3) = GSTATE(2)
GTMP(4) = GSTATE(1)
```

y los valores iniciales de GSTATE( ) son –32 en coma flotante, lo que equivale a –16384 en coma fija en Q9. A continuación se da la versión en coma fija del mismo módulo. En este código se han añadido varias variables nuevas. NLSREXPLG es una variable global que contiene el número de desplazamientos a la izquierda para normalizar REXPLG. Esta variable se inicializa con un valor de 31.

```
N1 = LPCLG + NUPDATE (= 10 + 4)                    | Calcular algunas constantes (pueden
N2 = LPCLG + NONRLG (= 10 + 20)                    | calcularse y almacenarse previamente en memoria)
N3 = LPCLG + NUPDATE + NONRLG (= 10 + 4 + 20)

For  N = 1, 2, ..., N2, do the next line
     SBLG(N) = SBLG(N + NUPDATE)                   | Cambiar la antigua memoria intermedia de señal
For  N = 1, 2, ..., NUPDATE, do the next line
     SBLG(N2 + N) = GTMP(N)                        | SBLG(N3) es la muestra más reciente
                                                   | Todas las SBLG están en Q9 y se representan
                                                   | con una precisión de 16 bits
Call FINDNLS(SBLG, N3, N3, 14, NLS)                | Encontrar el número de desplazamientos a
NLSTMP = NLS – 1                                   | la izquierda necesarios en el próximo bucle para
                                                   | un margen futuro de 2 bits

K = 1
For  N = 34, 33, ..., 1, do the next 5 lines
     P = SBLG(N) * WNRLG(K)                        | WNRLG está en Q15
     If NLSTMP = –1, set AA0 = P >> 1
     If NLSTMP > –1, set AA0 = P << NLSTMP
     WS(N) = RND(AA0)                              | WS(N) es de 14 bits o menos
     K = K + 1

NLSATTLG = 14
Call HWMCORE(LPCLG, N1, N3, NLSATTLG, WS, NLSTMP, REXPLG, NLSREXPLG, R, ILLCONDG)
```

La subrutina HWMCORE figura en G.3.18.

La tabla siguiente contiene la relación de todas las variables de este seudocódigo, con su formato de representación y tamaño para facilitar su referencia. En la tabla se indica si cada una de las variables es temporal (temp.), lo que significa que no es necesario almacenarla una vez que se haya completado el módulo, o permanente (perm.), en cuyo caso el valor se necesitará también después del cálculo actual. En la tabla se indica además qué variables no estaban incluidas en el seudocódigo previo de coma flotante (antigua/nueva).

Variable	Formato	Tamaño	Temp./perm.	Antigua/nueva
GTMP	Q9	4	perm.	antigua
NLS	entero	1	temp.	nueva
NLSREXPLG	entero	1	perm.	nueva
NLSTMP	entero	1	temp.	nueva
REXPLG	BFL	11	perm.	antigua
R	BFL	11	perm.	antigua
SBLG	Q9	34	perm.	antigua
WS	BFL	34	temp.	antigua
BFL Coma flotante de bloque entero Entero de 16 bits				

G.3.15 Bloque 45 – Módulo de ampliación de anchura de banda

Lo que sigue es el seudocódigo de coma flotante para el bloque 45, el módulo de ampliación de anchura de banda.

```
If ICOUNT ≠ 2, skip the execution of this block
For  I = 2, 3, ..., LPCLG + 1, do the next line
     GP(I) = FACGPV(I) * GPTMP(I)                    | Escalar coeficiente
```

Las tablas para FACGPV se dan en formato Q14, al igual que las tablas para los otros coeficientes de ampliación de anchura de banda. Los valores del ordenamiento GPTMP de entrada están en formato Q13, Q14 o Q15. Tal como se vio en la descripción del módulo de recursión de Durbin-Levinson de coma fija, NLSGPTMP viene dado por dicho módulo para indicar qué formato se utiliza para GPTMP. Después de la multiplicación FACGPV(I) * GPTMP(I) se necesita el número correspondiente de desplazamientos a la izquierda.

Los valores finales de GP se representan siempre en formato Q14. Empíricamente, los ordenamientos de coeficientes de salida del bloque 45 nunca han sido lo suficientemente grandes como para ser representados en Q14 (es decir, que requieren formato Q13 o inferior). Sin embargo, como medida de seguridad, hay que estar preparado para hacer frente al evento poco probable de un desbordamiento de Q14 a la salida de los bloques de ampliación de anchura de banda. En el seudocódigo que se indica más adelante se comprueba la posibilidad de un desbordamiento de Q14. Si se detecta esa posibilidad, se hace algo similar a los módulos de recursión de Levinson-Durbin: no se actualizan los coeficientes del predictor y se siguen utilizando los coeficientes antiguos, del ciclo de adaptación anterior. Podría emplearse un formato Q14/Q13 conmutable, con una bandera para señalar a los módulos de filtrado cuál de los dos posibles formatos Q se utiliza, pero esto aumentaría innecesariamente la complejidad del código DSP y el tiempo de ejecución. Puesto que nunca se ha observado un desbordamiento de Q14 a la salida de los módulos de ampliación de anchura de banda, basta con una simple comprobación de seguridad como la implementada a continuación.

Lo que sigue es el seudocódigo de coma fija para el bloque 45.

```
If ICOUNT ≠ 2, skip the execution of this block
Otherwise, do the following
                                                     | Comprobar primero si ILLCONDG es verdad

If ILLCONDG = .TRUE., skip the execution of this block
Otherwise, do the following
     GPTMP(1) = 16384
     For  I = 2, 3, 4, ..., LPCLG + 1, do the next 6 lines
          AA0 = FACGPV(I) * GPTMP(I)                 | AA0 está en Q27, Q28 o Q29
          If NLSGPTMP = 13, AA0 = AA0 << 3           | Poner AA0 en Q30 en los tres casos
          If NLSGPTMP = 14, AA0 = AA0 << 2           | mediante el número apropiado
          If NLSGPTMP = 15, AA0 = AA0 << 1           | de desplazamientos
          If AA0 overflowed above, go to LABEL       | Si no es verdad,
          GPTMP(I) = RND(AA0)                        | redondear a la palabra alta para GP
```

```
    For  I = 2, 3, 4, ..., LPCLG + 1, do the next line          | Todo es normal, copiar GPTMP
         GP(I) = GPTMP(I)                                       | a GP y salir a continuación
    Exit this program

LABEL:                                                          | Si el programa llega hasta aquí, se producirá un
                                                                | desbordamiento si se intenta representar GP en Q14.
                                                                | En este caso no se actualizan los coeficientes
                                                                | del predictor de ganancia logarítmica (es decir,
                                                                | utilizando los coeficientes del predictor de ganancia
                                                                | logarítmica del ciclo de adaptación anterior).
```

G.3.16 Bloque 46 – Predictor lineal de ganancia logarítmica

Lo que sigue es el seudocódigo de coma flotante del predictor lineal de ganancia logarítmica, bloque 46.

```
LOGGAIN = 0
For   I = LPCLG, LPCLG – 1, ..., 3, 2, do the next 2 lines
      LOGGAIN = LOGGAIN – GP(I + 1) * GSTATE(I)
      GSTATE(I) = GSTATE(I – 1)

LOGGAIN = LOGGAIN – GP(2) * GSTATE(1)
```

LOGGAIN y GSTATE se representan en formato Q9 a lo largo del codificador. GP se representa en formato Q14. A continuación se indica el seudocódigo de coma fija.

```
AA0 = 0
For   I = LPCLG, LPCLG – 1, ..., 3, 2, do the next 3 lines
      P = GP(I + 1) * GSTATE(I)
      AA0 = AA0 – P
      GSTATE(I) = GSTATE(I – 1)

P = GP(2) * GSTATE(1)
AA0 = AA0 – P
AA0 = AA0 >> 14
LOGGAIN = AA0
```

Lo que sigue es el seudocódigo de coma flotante para el bloque 98, el limitador de la ganancia logarítmica. Puesto que este código se basa en modificaciones introducidas para el de coma fija, no aparece en la Recomendación G.728. Se incluye aquí a efectos de comparación con el seudocódigo de coma fija que figura más adelante.

```
If LOGGAIN > 28., set LOGGAIN = 28
If LOGGAIN < –32., set LOGGAIN = –32
```

LOGGAIN se representa en formato Q9, por lo que los umbrales máximo y mínimo se multiplican por 512. Estos valores se utilizan en el seudocódigo de coma fija que se da a continuación.

```
If LOGGAIN > 14336, set LOGGAIN = 14336
If LOGGAIN < –16384, set LOGGAIN = –16384
```

Lo que sigue es el seudocódigo de coma flotante del sumador del desplazamiento de la ganancia logarítmica, que es el bloque 99.

```
Z = LOGGAIN + GOFF
```

El valor en coma flotante de GOFF es 32 y su valor en coma fija es 16384, que corresponde a 512*32. Puesto que la gama de LOGGAIN va de –32 a +28, Z tiene una gama de 0 a 60. El código de coma fija es idéntico al de coma flotante.

Lo que sigue es el código de coma flotante del bloque 48, el calculador logarítmico inverso.

$GAIN = 10^{(Z/20)}$

De esta manera, el valor completo que se desea obtener puede expresarse en términos del antilogaritmo de 2 de la siguiente manera:

$$10^{0,05\,Z} = 2^{0,05\log_2(10)\,Z} = 2^{0,1660964\,Z}$$

Se hace X = 0,1660964 Z, que tendrá una gama de 0 a 9,97 y, por último, X = [X] + x, en donde [X] es el mayor entero inferior o igual a X y x, la parte fraccionaria. El valor de $2^{[X]}$ es exacto y puede representarse por su exponente solamente. Lo que queda es el problema de calcular el valor de la parte fraccionaria.

En el cálculo de X se representa 0,1660964 en formato Q21, lo cual corresponde a un número que puede representarse como 10 en los 16 bits superiores y como 20649 en los 15 bits inferiores. Se multiplica Z por ambas partes separadamente para obtener una buena precisión de X y a continuación se separan [X] y x. Al calcular el exponente de la parte fraccionaria se tiene en cuenta que $0 < x < 1$, por lo que $1 < 2^x < 2$. Se puede utilizar, por tanto, la siguiente representación fija: x en Q15 y 2^x en Q14. Para el cálculo de 2^x se utiliza una ampliación de la serie de Taylor:

$$2^x = \Big(((c4\ x\ +\ c3)\ x\ +\ c2)\ \ x\ +\ c1\Big)x\ +\ c0$$
$$= c4\ x^4\ +\ c3\ x^3\ +\ c2\ x^2\ +\ c1\ x\ +\ c0$$

Los valores de c están almacenados en Q14 o Q15 y vienen dados por:

$c4 = 323 = 0{,}0098571$ en Q15
$c3 = 1874 = 0{,}0571899$ en Q15
$c2 = 7866 = 0{,}2400512$ en Q15
$c1 = 22702 = 0{,}6928100$ en Q15
$c0 = 16384 = 1{,}0$ en Q14

A continuación se indica el seudocódigo para el cálculo de $10^{0{,}05\ GAIN}$ en un DSP de 16 bits con dos acumuladores de 32 bits. Se ha supuesto que GAIN está en formato Q9 y que ya se le ha agregado el desplazamiento de 32 dB.

```
AA0 = 10 * Z                    | Z está en Q9 y 10 en Q6, por lo que AA0 está en Q15
AA1 = 20649 * Z                 | 20649 está en Q21, por lo que AA1 está en Q30
AA1 = AA1 << 1                  | Poner AA1 en Q31
AA1 = RND(AA1)                  | Redondear AA1 para ponerlo en Q15 en
                                | la palabra baja
AA0 = AA0 + AA1                 | AA0 = [X] + x en Q15
AA1 = AA0 >> 15                 | Se desea [X] en Q0 y x en Q15
NLS = AA1                       | NLS = [X]
AA1 = AA1 << 15
x = AA0 – AA1                   | x es la parte fraccionaria en Q15

                                | Calcular 2 ** x
AA0 = c4 * x                    | Q15 * Q15 = => AA0 está en Q30
AA0 = AA0 << 1                  | AA0 está ahora en Q31
AA1 = c << 16                   | AA1 está en Q31
AA0 = AA0 + AA1
TMP = RND(AA0)                  | TMP está en Q15, utilizar el redondeo
AA0 = TMP * x                   | Q15 * Q15 = => AA0 está en Q30
AA0 = AA0 << 1                  | AA0 está ahora en Q31
AA1 = c2 << 16                  | AA1 está en Q31
AA0 = AA0 + AA1
TMP = RND(AA0)                  | TMP está en Q15, utilizar el redondeo
AA0 = TMP * x                   | Q15 * Q15 = => AA0 está en Q30
AA0 = AA0 << 1                  | AA0 está ahora en Q31
AA1 = c1 << 16                  | AA1 está en Q31
AA0 = AA0 + AA1
TMP = RND(AA0)                  | TMP está en Q15, utilizar el redondeo
AA0 = TMP * x                   | Q15 * Q15 = => AA0 está en Q30
                                | ¡Sin desplazamiento a la izquierda esta vez!
AA1 = c0 = << 16                | AA1 está en Q30
AA0 = AA0 + AA1
GAIN = RND(AA0)                 | GAIN está en Q14 y contiene 2 ** x
                                | NLSGAIN es 14 – NLS para 2 ** X
NLSGAIN = 14 – NLS              | Factor Q como resultado
```

Lo que sigue es el seudocódigo de coma flotante de los bloques 96 y 97, el cálculo del nuevo valor de GSTATE(1). Este bloque no aparece en la Recomendación G.728 y se da aquí a efectos de comparación con el seudocódigo de coma fija que se indica inmediatamente después. Las variables de entrada son el valor de GAIN a la salida del bloque 98, GCBLG, el valor en dB de la entrada del código cifrado de ganancia seleccionado para el vector de excitación anterior, y SHAPELG, el valor en dB del vector de código de forma seleccionado para el vector de excitación anterior. Estos valores se indican en los Cuadros G.3 y G.4, respectivamente. Dichos cuadros dan las representaciones de los valores en coma flotante y coma fija. Las representaciones en coma fija están en formato Q11. El seudocódigo de coma flotante es como sigue:

```
GSTATE(1) = LOGGAIN + GCBLG(IG) + SHAPELG(IS)
If GSTATE(1) < –32., set GSTATE(1) = –32
```

El seudocódigo de coma fija es como sigue:

```
AA0 = LOGGAIN << 7                          | Alinear las comas decimales en el límite
AA0 = AA0 + (GCBLG(IG) << 5)                | entre las palabras alta y baja
AA0 = AA0 + (SHAPELG(IS) << 5)              | del acumulador

AA0 = AA0 >> 7                              | Desplazamiento a la derecha para volver a formato Q9

IF AA0 < –16384, set AA0 = –16384           | Comprobar límite inferior

GSTATE(1) = AA0                             | Guardada palabra de 16 bits más bajos
```

G.3.17 Bloque 49 – Módulo de ventana híbrida para filtro de síntesis

Se empieza con el seudocódigo de coma flotante para el módulo de ventanización híbrida.

```
N1 = LPC + NFRSZ                            | Calcular algunas constantes (pueden
N2 = LPC + NONR                             | calcularse y almacenarse previamente en memoria)
N3 = LPC + NFRSZ + NONR

For   N = 1, 2, ..., N2, do the next line
      SB(N) = SB(N + NFRSZ)                 | Cambiar la antigua memoria intermedia de señal
For   N = 1, 2, ..., NFRSZ, do the next line
      SB(N2 + N) = STTMP(N)                 | Introducir la nueva señal
                                            | SB(N3) es la muestra más reciente
K = 1
For   N = N3, N3 – 1, ..., 3, 2, 1, do the next 2 lines
      WS(N) = SB(N) * WNR(K)                | Multiplicar la función de ventana
      K = K + 1

For   I = 1, 2, ..., LPC + 1, do the next 4 lines
      TMP = 0
      For   N = LPC + 1, LPC + 2, ..., N1, do the next line
            TMP = TMP + WS(N) * WS(N + 1 – I)
      REXP(I) = (3/4) * REXP(I) + TMP       | Actualizar la componente recursiva

For   I = 1, 2, ..., LPC + 1, do the next 3 lines
      RTMP(I) = REXP(I)
      For   N = N1 + 1, N1 + 2, ..., N3, do the next line
            RTMP(I) = RTMP(I) + WS(N) * WS(N + 1 – I)
                                            | Añadir la componente no recursiva

RTMP(1) = RTMP(1) * WNCF                    | Corrección por ruido blanco
```

El seudocódigo de coma fija para el módulo de ventanización híbrida (bloque 49) es mucho más complicado que la versión en coma flotante. Ello se debe al tratamiento especial del formato de coma flotante de bloque segmentado (SBFL, *segmental block floating point*) que se necesita para mantener una precisión numérica suficiente.

El ordenamiento STTMP contiene 4 vectores de señales vocales cuantificadas del ciclo de adaptación anterior. Cuando se calculó cada uno de estos 4 vectores de señales vocales cuantificadas (el ordenamiento ST), se representó en formato BFL de 14 bits de precisión. El número de desplazamientos a la izquierda (NLS) de los 4 vectores cuantificados será, por lo general, diferente. Es por esto por lo que se dice que el ordenamiento STTMP se almacena en formato SBFL, ya que es la concatenación de 4 vectores de ST en BFL. El ordenamiento SB es la concatenación de 21 vectores de ST en BFL y por eso se almacena en el mismo formato SBFL de 14 bits de precisión. Por cada uno de los 4 vectores que componen el STTMP hay un valor de NLS asociado. Estos valores se almacenan en el ordenamiento NLSSTTMP( ). Los valores de NLS correspondientes a los 21 vectores que componen el SB se almacenan en el ordenamiento NLSSB( ).

A continuación se indica el seudocódigo de coma fija para el módulo de ventanización híbrida.

```
N1 = LPC + NFRSZ (= 70)                          | Calcular algunas constantes (pueden
N2 = LPC + NONR (= 85)                           | calcularse y almacenarse previamente en memoria)
N3 = LPC + NFRSZ + NONR (= 105)
N4 = N3/IDIM (= 21)
N5 = NFRSZ/IDIM (= 4)
N6 = N4 – N5 (= 17)

For   N = 1, 2, ..., N2, do the next line
      SB(N) = SB(N + NFRSZ)                      | Cambiar la parte antigua del SB de
For   N = 1, 2, ..., N6, do the next line        | la memoria intermedia
      NLSSB(N) = NLSSB(N + N5)                   | Cambiar el NLSSB antiguo
For   N = 1, 2, ..., NFRSZ, do the next line
      SB(N2 + N) = STTMP(N)                      | Introducir la parte nueva del SB
For   N = 1, 2, ..., N5, do the next line
      NLSSB(N6 + N) = NLSSTTMP(N)                | Introducir el NLSSB nuevo

                                                 | Encontrar ahora el NLSSB mínimo,
                                                 | con lo que se determina el NLSWS
NLSTMP = Min{NLSSB(1), NLSSB(2), ..., NLSSB(N4)}

K = 1                                            | Multiplicar ahora el SB por la
N = N3                                           | función de ventana híbrida
For   J = 1, 2, ..., N4, do the next 8 lines
      NRSH = NLSSB(J) – NLSTMP – 1               | –1 para compensar la multiplicación en Q15
      For  M = 1, 2, ..., IDIM, do the next 6 lines
           P = SB(K) * WNR(N)                    | WNR es multiplicación en Q15
           If NRSH = –1, set AA0 = P << 1
           If NRSH > –1, set AA0 = P >> NRSH
           WS(K) = RND(AA0)                      | Redondear la palabra superior y almacenar en WS
           N = N – 1
           K = K + 1

NLSATT50 = 14
Call HWMCORE(LPC, N1, N3, NLSATT50, WS, NLSTMP, REXP, NLSREXP, RTMP, ILLCOND)
```

NOTA – La tabla siguiente contiene la relación de todas las variables de este seudocódigo, con su formato de representación y tamaño para facilitar su referencia. En la tabla se indica si cada una de la variables es temporal (temp.), lo que significa que no es necesario almacenarla una vez que se haya completado el módulo, o permanente (perm.), en cuyo caso el valor se necesitará también después del cálculo en curso. En la tabla se indica además qué variables no estaban incluidas en el seudocódigo anterior de coma flotante (antigua/nueva).

Variable	Formato	Tamaño	Temp./perm.	Antigua/nueva
NLSSB	entero	21	perm.	nueva
NLSREXP	entero	1	perm.	nueva
NLSSTTMP	entero	4	perm.	nueva
NLSTMP	entero	1	temp.	nueva
NRSH	entero	1	temp.	nueva
REXP	BFL	51	perm.	antigua
RTMP	BFL	51	perm.	antigua
SB	SBFL	105	perm.	antigua
STTMP	SBFL	20	perm.	antigua
WS	BFL	105	temp.	antigua

BFL Coma flotante de bloque
entero Entero de 16 bits
SBFL Coma flotante de bloque segmentado con precisión de 14 bits
WS BFL con precisión de 14 bits
REXP y RTMP Precisión de 16 bits

G.3.18 HWMCORE – Núcleo del módulo de ventana híbrida

Este módulo se utiliza para completar el cálculo de la ventana híbrida de los bloques 36, 43 y 49. Cada uno de estos bloques tiene su propia porción inicial. Las variables se pasan de esos bloques a este módulo. Para evitar confusiones, se han dado nombres nuevos a determinadas variables para no utilizar en este seudocódigo nombres asociados con ninguno de esos tres bloques. En la tabla que sigue se indica la correspondencia entre los nombres utilizados en este módulo y los nombres utilizados en los bloques 36, 43 y 49.

Variable	Bloque 36	Bloque 43	Bloque 49
LPO	LPCW (=10)	LPCLG (=10)	LPC (=50)
NLSATT	NLSATTW	NLSATTLG	NLSATT50
NLSRREC	NLSREXPW	NLSREXPLG	NLSREXP
N1	30	14	70
N3	60	34	105
R	R	R	RTMP
RREC	REXPW	REXPLG	REXP

Además de estas variables, también se pasan de esos bloques a este módulo el ordenamiento transitorio WS y el número correspondiente de desplazamientos, NLSTMP.

Lo que sigue es el seudocódigo de coma fija para este módulo.

```
SUBROUTINE HWMCORE(LPO, N1, N3, NLSATT, WS, NLSTMP, RREC, NLSRREC, R, ILLCOND)

NLSAA0 = 2 * NLSTMP
AA0 = 0                                          | Calcular la parte recursiva de RREC(1)
For N = LPO + 1, ..., N1, do the next 2 lines
    P = WS(N) * WS(N)                            | WS tiene 2 bits de margen
    AA0 = AA0 + P                                | AA0 tendrá 5 bits de margen
                                                 | para el cálculo de la energía


                                                 | Caso 1: NLSRREC > NLSAA0
If NLSRREC > NLSAA0, do the next 22 lines
    AA0 = AA0 >> 1
    IR = NLSRREC – NLSAA0 + 1
    AA1 = RREC(1) << NLSATT                      | Esto puede hacerse por multiplicación
    AA1 = –AA1 + (RREC(1) << 16)                 | Escalar RREC mediante un factor de atenuación
    AA1 = AA1 >> IR                              | Alinear AA0 y AA1
    AA0 = AA0 + AA1
    Call VSCALE(AA0, 1, 1, 30, AA0, NLSRE)       | Encontrar NLS para RREC
    RREC(1) = RND(AA0)                           | Guardados los 16 bits más altos de AA1
    NLSRREC = NLSAA0 – 1 + NLSRE
    For I = 1, 2, ..., LPO, do the next 11 lines
        AA0 = 0                                  | Calcular la parte recursiva de RREC(I + 1)
        For N = LPO + 1, ..., N1, do the next 2 lines
            P = WS(N) * WS(N – I)
            AA0 = AA0 + P
        AA0 = AA0 >> 1
        AA1 = RREC(I + 1) << NLSATT              | Escalar RREC por 3/4 ó 1/2
        AA1 = AA1 + (RREC(I + 1) << 16)          |
        AA1 = AA1 >> IR
        AA0 = AA0 + AA1
        AA0 = AA0 << NLSRE
        RREC(I + 1) = RND(AA0)                   | Guardados los 16 bits más altos de AA0
    Go to FIN_RECUR


                                                 | Caso 2: NLSRREC = NLSAA0
If NLSRREC = NLSAA0, do the next 21 lines
    AA1 = RREC(1) << NLSATT                      | Escalar RREC por 3/4 ó 1/2
    AA1 = –AA1 + (RREC(1) << 16)                 |
    AA0 = AA0 >> 1
    AA1 = AA1 >> 1
    AA0 = AA0 + AA1
    Call VSCALE(AA0, 1, 1, 30, AA0, NLSRE)       | Encontrar NLS para RREC
    RREC(1) = RND(AA0)                           | Guardados los 16 bits más altos de AA1
    NLSRREC = NLSRREC – 1 + NLSRE
    For I = 1, 2, ..., LPO, do the next 11 lines
        AA0 = 0                                  | Calcular la parte recursiva de RREC(I + 1)
        For N = LPO + 1, ..., N1, do the next 2 lines
            P = WS(N) * WS(N – I)
            AA0 = AA0 + P
        AA0 = AA0 >> 1
        AA1 = RREC(I + 1) << NLSATT              | Escalar RREC por 3/4 ó 1/2
        AA1 = –AA1 + (RREC(I + 1) << 16)         |
        AA1 = AA1 >> 1
        AA0 = AA0 + AA1
        AA0 = AA0 << NLSRE
        RREC(I + 1) = RND(AA0)                   | Guardados los 16 bits más altos de AA0
    Go to FIN_RECUR
```

```
                                                    | Caso 3: NLSRREC < NLSAA0
    If NLSRREC < NLSAA0, do the next 21 lines
        IR = NLSAA0 – NLSRREC + 1
        AA0 = AA0 >> IR
        AA1 = RREC(1) << NLSATT                     | Escalar RREC por 3/4 ó 1/2
        AA1 = –AA1 + (RREC(1) << 16)                |
        AA1 = AA1 >> 1
        AA0 = AA0 + AA1
        Call VSCALE(AA0, 1, 1, 30, AA0, NLSRE)
        RREC(1) = RND(AA0)                          | Guardados los 16 bits más altos de AA1
        NLSRREC = NLSRREC – 1 + NLSRE
        For I = 1, 2, ..., LPO, do the next 11 lines
            AA0 = 0                                 | Calcular la parte recursiva de RREC(I + 1)
            For N = LPO + 1, ..., N1, do the next 2 lines
                    P = WS(N) * WS(N – I)
                    AA0 = AA0 + P
            AA0 = AA0 >> IR
            AA1 = RREC(I + 1) << NLSATT             | Escalar RREC por 3/4 ó 1/2
            AA1 = –AA1 + (RREC(I + 1) << 16)        |
            AA1 = AA1 >> 1
            AA0 = AA0 + AA1
            AA0 = AA0 << NLSRE
            RREC(I + 1) = RND(AA0)                  | Guardados los 16 bits más altos de AA0

FIN_RECUR:                                          | Cuando se alcanza este punto se ha
                                                    | calculado la componente recursiva

    AA0 = 0                                         | Calcular la parte no recursiva de R(1)
    For N = N1 + 1, ..., N3, do the next 2 lines
        P = WS(N) * WS(N)
        AA0 = AA0 + P

                                                    | Caso 1: NLSRREC > NLSAA0
    If NLSRREC > NLSAA0, do the next 21 lines
        IR = NLSRREC – NLSAA0 + 1
        AA1 = RREC(1) << 16
        AA1 = AA1 >> IR
        AA0 = AA0 >> 1
        AA1 = AA0 + AA1
        AA0 = AA1 >> 8                              | Aplicar un factor de corrección por
        AA1 = AA1 + AA0                             | ruido blanco

        Call VSCALE(AA1, 1, 1, 30, AA1, NLSRR)      | Guardados los 16 bits más altos de AA1
        R(1) = RND(AA1)

    For I = 1 , 2, ..., LPO, do the next 10 lines
        AA0 = 0                                     | Calcular la parte no recursiva de R(I + 1)
        For N = N1 + 1, ..., N3, do the next 2 lines
            P = WS(N) * WS(N – I)
            AA0 = AA0 + P
        AA0 = AA0 >> 1
        AA1 = RREC(I + 1) << 16
        AA1 = AA1 >> IR
        AA1 = AA0 + AA1
        AA1 = AA1 << NLSRR
        R(I + 1) = RND(AA1)                         | Guardar los 16 bits más altos
    Go to END
```

```
                                                    | Caso 2: NLSRREC = NLSAA0
    If NLSRREC = NLSAA0, do the next 18 lines
        AA0 = AA0 >> 1
        AA1 = RREC(1) << 15                         | Esto puede hacerse por multiplicación
        AA1 = AA0 + AA1
        AA0 = AA1 >> 8                              | Aplicar un factor de corrección por
                                                    | ruido blanco
        AA1 = AA1 + AA0
        Call VSCALE(AA1, 1, 1, 30, AA1, NLSRR)
        R(1) = RND(AA1)                             | Guardados los 16 bits más altos de AA1
    For I = 1, 2, ..., LPO, do the next 9 lines
        AA0 = 0                                     | Calcular la parte no recursiva de R(I + 1)
        For N = N1 + 1, ..., N3, do the next 2 lines
            P = WS(N) * WS(N – I)
            AA0 = AA0 + P
        AA0 = AA0 >> 1
        AA1 = RREC(I + 1) << 15
        AA1 = AA0 + AA1
        AA1 = AA1 << NLSRR
        R(I + 1) = RND(AA1)                         | Guardar los 16 bits superiores
    Go to END


                                                    | Caso 3: NLSRREC < NLSAA0
    If NLSRREC < NLSAA0, do the next 18 lines
        IR = NLSAA0 – NLSRREC + 1
        AA0 = AA0 >> IR
        AA1 = RREC(1) << 15                         | Esto puede hacerse por multiplicación
        AA1 = AA0 + AA1
        AA0 = AA1 >> 8                              | Aplicar un factor de corrección
                                                    | por ruido blanco
        AA1 = AA1 + AA0
        Call VSCALE(AA1, 1, 1, 30, AA1, NLSRR)
        R(1) = RND(AA1)                             | Guardados los 16 bits superiores de AA1
        For I = 1, 2, ..., LPO, do the next 9 lines
        AA0 = 0                                     | Calcular la parte no recursiva de R(I + 1)
        For N = N1 + 1, ..., N3, do the next 2 lines
            P = WS(N) * WS(N – I)
            AA0 = AA0 + P
        AA0 = AA0 >> IR
        AA1 = RREC(I + 1) << 15
        AA1 = AA0 + AA1
        AA1 = AA1 << NLSRR
        R(I + 1) = RND(AA1)                         | Guardar los 16 bits superiores


END:                                                | Como última tarea, comprobar
                                                    | si hay desajustes
    ILLCOND = .FALSE.
    If AA1 = 0, set ILLCOND = .TRUE                 | AA1 contiene todavía R(LPO + 1) de 32 bits
```

NOTA – La tabla siguiente contiene la relación de todas las variables de este seudocódigo, con su formato de representación y tamaño para facilitar la referencia. En la tabla se indica si cada una de las variables es temporal (temp.), lo que significa que no es necesario almacenarla una vez que se haya completado el módulo, o permanente (perm.), en cuyo caso el valor se necesitará también después del cálculo en curso. En la tabla se indica además qué variables no estaban incluidas en el seudocódigo anterior de coma flotante (antigua/nueva).

Variable	Formato	Tamaño	Temp./perm.	Antigua/nueva
NLSRE	entero	1	temp.	nueva
NLSRR	entero	1	temp.	nueva
NLSRREC	entero	1	perm.	nueva
NLSTMP	entero	1	temp.	nueva
RREC	BFL	51	perm.	antigua
R	BFL	51	perm.	antigua
WS	BFL	105	temp.	antigua

BFL	Coma flotante de bloque
entero	Entero de 16 bits
SBFL	Coma flotante de bloque segmentado con precisión de 14 bits
WS	BFL con precisión de 14 bits
RREC y R	Precisión de 16 bits
RREC	Representa REXP, REXPW o REXPLG, dependiendo de si este módulo se convoca desde el bloque 49, 36 ó 43

G.3.19 Bloque 51 – Módulo de ampliación de anchura de banda

Lo que sigue es el seudocódigo de coma flotante para el bloque 51, el módulo de ampliación de anchura de banda. Existe un código similar para el bloque 45, el módulo de ampliación de anchura de banda del predictor lineal de ganancia logarítmica. En ese caso, se utiliza una tabla diferente y el número de coeficientes del filtro es mayor.

```
For I = 2, 3, ..., LPC + 1, do the next line
    ATMP(I) = FACV(I) * ATMP(I)                    | Escalar coeficiente

Wait until ICOUNT = 3, then
for I = 2, 3, ..., LPC + 1, do the next line
    A(I) = ATMP(I)
```

Las tablas para FACV se dan en formato Q14, al igual que las tablas para los otros coeficientes de ampliación de anchura de banda. Los valores del ordenamiento ATMP de entrada están en formato Q13, Q14 o Q15. Tal como se vio en la descripción del módulo de recursión de Levinson-Durbin de coma fija, NLSATMP viene dado por dicho módulo para indicar qué formato se utiliza para ATMP. Después de la multiplicación FACV(I) * ATMP(I) se necesita el número correspondiente de desplazamientos a la izquierda.

Los valores finales de ATMP se representan siempre en formato Q14. Empíricamente, los valores de ATMP nunca han sido lo suficientemente grandes como para ser representados en Q14 (es decir, que requieren formato Q13 o menor). Sin embargo, como medida de seguridad, hay que estar preparado para hacer frente al evento poco probable de un desbordamiento en Q14 a la salida del módulo de ampliación de anchura de banda. En el seudocódigo que se indica más adelante se comprueba la posibilidad de un desbordamiento en Q14. Si se detecta esa posibilidad, se hace algo similar a los módulos de recursión de Levinson-Durbin: no se actualizan los coeficientes del predictor y se siguen utilizando los coeficientes antiguos, del ciclo de adaptación anterior. Podría emplearse un formato Q14/Q13 conmutable, con una bandera para señalar a los módulos de filtrado cuál de los dos posibles formatos Q se utiliza, pero esto aumentaría innecesariamente la complejidad del código DSP y el tiempo de ejecución. Puesto que nunca se ha observado un desbordamiento en Q14 a la salida de los módulos de ampliación de anchura de banda, basta con una simple comprobación de seguridad como la implementada a continuación.

Lo que sigue es el seudocódigo de coma fija para el bloque 51.

```
If ICOUNT ≠ 3, skip the following
Otherwise, do the following.
                                                    | Comprobar primero si ILLCOND es verdad
If ILLCOND = .TRUE., skip the execution of this block
Otherwise, do the following
        ATMP(1) = 16384
        For I = 2, 3, 4, ..., LPC + 1, do the next 6 lines
            AA0 = FACV(I) * ATMP(I)                 | AA0 está en Q27, Q28 o Q29
            If NLSATMP = 13, AA0 = AA0 << 3         | Poner AA0 en Q30 en los tres casos
            If NLSATMP = 14, AA0 = AA0 << 2         | mediante el número apropiado de desplazamientos
            If NLSATMP = 15, AA0 = AA0 << 1
            If AA0 overflowed above, go to LABEL    | Si no es verdad,
            ATMP(I) = RND(AA0)                      | redondear a la palabra alta para ATMP

        For I = 2, 3, ..., LPC + 1, do the next line
            A(I) = ATMP(I)

        Exit this module

LABEL:                                              | Si el programa llega hasta aquí, se produciría un
                                                    | desbordamiento si se intenta representar A en Q14.
                                                    | En este caso no se actualizan los coeficientes del filtro
                                                    | de síntesis (es decir, se siguen utilizando los
                                                    | coeficientes del filtro de síntesis del ciclo
                                                    | de adaptación anterior).
```

G.3.20 Bloques 71 y 72 – Posfiltros de largo plazo y de corto plazo

Los bloques 71 y 72 se combinan para preservar la precisión de la variable intermedia TEMP, que se pasó entre ellos en el seudocódigo de coma flotante. En primer lugar se indica el seudocódigo de coma flotante para ambos bloques.

```
        For K = 1, 2, ..., IDIM, do the next line
            TEMP(K) = GL * (ST(K) + B * ST(K – KP))     | Posfiltrado de largo plazo

        For K = –NPWSZ – KPMAX + 1, ..., –2, –1, 0, do the next line
            ST(K) = ST(K + IDIM)                        | Cambiar la memoria intermedia
                                                        | de señal vocal decodificada

        For K = 1, 2, ..., IDIM, do the following
            TEMP = TEMP(K)
            For J = 10, 9, ..., 3, 2, do the next 2 lines
                TEMP(K) = TEMP(K) + STPFFIR(J) * AZ(J + 1)   | Parte «todos ceros»
                STPFFIR(J) = STPFFIR(J – 1)                 | del filtro
            TEMP(K) = TEMP(K) + STPFFIR(1) * AZ(2)          | Ultimo multiplicador
            STPFFIR(1) =  TMP

            For J = 10, 9, ..., 3, 2, do the next 2 lines
                TEMP(K) =  TEMP(K) – STPFIIR(J) * AP(J + 1)  | Parte «todos polos»
                STPFIIR(J) = STPFIIR(J – 1)                 | del filtro
            TEMP(K) = TEMP(K) – STPFIIR(1) * AP(2)          | Ultimo multiplicador
            STPFIIR(1) = TEMP(K)
            TEMP(K) = TEMP(K) + STPFIIR(2) * TILTZ          | Filtro de compensación
                                                            | de la «inclinación» espectral
```

A continuación se da el seudocódigo de coma fija. Las variables STPFFIR y STPFIIR están en Q2 a lo largo del mismo.

```
For K = 1, 2, ..., IDIM, do the following indented lines
                                                    | Hacer primero el posfiltro de largo plazo
    AA0 = GL * SST(K)                               | GL está en Q14, SST (1:5) está en Q2
    AA0 = AA0 + GLB * SST(K – KP)                   | GLB está en Q16, SST (–239:0) está en Q0
```

```
AA1 = AA0                                          | Hacer a continuación el posfiltro de corto plazo
                                                   | Efectuar la parte FIR del filtro
For J = 10, 9, ..., 3, 2, do the next 2 lines
     AA1 = AA1 + STPFFIR(J) * AZ(J + 1)            | AZ está en Q14, STPFFIR(J) está en Q2
     STPFFIR(J) = STPFFIR(J – 1)
AA1 = AA1 + STPFFIR(1) * AZ(2)
AA0 = AA0 << 2
STPFFIR(1) = RND(AA0)                              | Q2 para STPFFIR

                                                   | Efectuar ahora la parte IIR del filtro
For J = 10, 9, ..., 3, 2, do the next 2 lines
     AA1 = AA1 – STPFIIR(J) * AP(J + 1)            | AP está en Q14, STPFIIR(J) está en Q2
     STPFIIR(J) = STPFIIR(J – 1)
AA1 = AA1 – STPFIIR(1) * AP(2)

AA0 = AA0 >> 14
                                                   | Comprobar ahora si hay saturación
If AA0 > 32767, set AA0 = 32767
If AA0 < –32768, set AA0 = –32768
STPFIIR(1) = AA0

                                                   | Hacer ahora el filtro de
                                                   | compensación de la «inclinación» espectral
AA1 = AA1 + STPFIIR(2) * TILTZ                     | TILTZ está en Q14
AA1 = AA1 >> 14
If AA1 > 32767, set AA1 = 32767
If AA1 < –32768, set AA1 = –32768
TEMP(K) = AA1

                                                   | Cambiar ahora la memoria
                                                   | intermedia del posfiltro de largo plazo
For K = –NPWSZ – KPMAX + 1, ..., –7, –6, –5, do the next line
     SST(K) = SST(K + IDIM)                        | Cambiar la memoria intermedia
                                                   | de señal vocal decodificada
For K = –4, –3, ..., 0, do the next line
     SST(K) = SST(K + IDIM) >> 2                   | Cambiar la memoria intermedia
                                                   | de señal vocal decodificada y pasar de Q2 a Q0
```

G.3.21 Bloques 73 y 74 – Calculador de la suma de los valores absolutos

Los bloques 73 y 74 son muy similares. Sus resultados se mantienen con precisión doble. Como aquí se indica, no es preciso almacenar esos resultados antes del bloque 75. A continuación se da el seudocódigo de coma flotante para el bloque 73. Obsérvese que a la variable ST se le ha dado el nombre SST para mantener coherencia entre este código, de coma flotante, y el código de coma fija presentado más abajo.

Recuérdese que SST(1:5) se representa en Q2.

```
SUMUNFIL = 0
For K = 1, 2, ..., IDIM, do the next line
     SUMUNFIL = SUMUNFIL + | SST(K) |
```

El seudocódigo para el bloque 74 viene dado por lo siguiente.

```
SUMFIL = 0
For K = 1, 2, ..., IDIM, do the next line
     SUMFIL = SUMFIL + absolute value of TEMP(K)
```

El seudocódigo de coma fija para estos bloques es como a continuación se indica.

```
AA1 = 0
AA0 = 0
For K = 1, 2, ..., IDIM, do the next 2 lines
        AA0 = AA0 + | SST(K) |                  | Añadir el valor absoluto de SST(K)
        AA1 = AA1 + | TEMP(K) |                 | Añadir el valor absoluto de TEMP(K)
                                                | AA0 = SUMUNFIL
                                                | AA1 = SUMFIL
                                                | SST y TEMP están en Q2, por lo que
                                                | AA0 y AA1 están también en Q2
                                                | AA0 y AA1 se utilizarán en el bloque 75
```

G.3.22 Bloque 75 – Calculador del factor de escala

El bloque 75 calcula la relación SUMUNFIL/SUMFIL y el resultado se almacena en SCALE con una precisión de NLSSCALE. SUMUNFIL(AA0) y SUMFIL(AA1) proceden de los bloques 73 y 74, respectivamente. El seudocódigo de coma flotante viene dado por lo siguiente.

```
If SUMFIL > 1, set SCALE = SUMUNFIL/SUMFIL
Otherwise, set SCALE = 1
```

El seudocódigo de coma fija es como a continuación se indica.

```
If AA1 > 4, do the following indented lines
    Call VSCALE(AA1, 1, 1, 30, AA1, NLSDEN)
    DEN = RND(AA1)
    Call VSCALE(AA0, 1, 1, 30, AA0, NLSNUM)
    NUM = RND(AA0)                              | NLSNUM y NLSDEN están los dos
                                                | desplazados de 16, entonces anulación
    Call DIVIDE(NUM, NLSNUM, DEN, NLSDEN, SCALE, NLSSCALE)
    Otherwise, set SCALE = 16384 and NLSSCALE = 14
```

G.3.23 Bloque 76 – Filtro de paso bajo de primer orden y bloque 77 – Unidad de escalamiento de ganancia de salida

El seudocódigo de coma flotante para estos dos bloques viene dado por lo siguiente.

```
For K = 1, 2, ..., IDIM, do the following
    SCALEFIL = AGCFAC * SCALEFIL + (1 – AGCFAC) * SCALE     | Filtrado de paso bajo
    SPF(K) = SCALEFIL * TEMP(K)                             | Escalar salida
```

En el seudocódigo de coma fija, el segundo término se calcula una vez y a continuación se añade en cada iteración, para ahorrarse la sustracción y la multiplicación dentro del bucle. El seudocódigo de coma fija es como a continuación se indica.

```
AA1 = AGCFAC1 * SCALE                   | AGCFAC1 = 20972 en Q21 = 0,010000228
NRS = NLSSCALE – 14 + (21 – 14)         | Calcular desplazamiento a la derecha
If NRS ≥ 0, AA1 = AA1 >> NRS            | Hacer que AA1 esté en Q28
If NRS < 0, AA1 = AA1 << –NRS           | Desplazar a la derecha si NRS es negativo

For K = 1, 2, ..., IDIM, do the following
                                        | Filtrado de paso bajo
    AA0 = AA1 + AGCFAC * SCALEFIL       | AGCFAC = 16 220 en Q14 y SCALEFIL
    AA0 = AA0 << 2                      | está en Q14
    SCALEFIL = RND(AA0)                 | Poner SCALEFIL en Q14

                                        | Escalar salida
    AA0 = SCALEFIL * TEMP(K)            | TEMP(K) está en Q2
    AA0 = AA0 << 2
    SPF(K) = RND(AA0)                   | SPF(K) está en Q2
```

G.3.24 Bloque 81 – Filtro inverso LPC de décimo orden

Lo que sigue es la versión en coma flotante del seudocódigo para el bloque 81, el filtro inverso LPC de décimo orden.

```
If IP = NPWSZ, then set IP = NPWSZ – NFRSZ                  | Comprobar y actualizar IP

For   K = 1, 2, ..., IDIM, do the next 7 lines
      ITMP = IP + K
      D(ITMP) = ST(K)
      For J = 10, 9, ..., 3, 2, do the next 2 lines
            D(ITMP) = D(ITMP) + STLPCI(J) * APF(J + 1)    | Filtrado FIR
            STLPCI(J) = STLPCI(J – 1)                     | Introducir la señal de entrada por desplazamiento
      D(ITMP) = D(ITMP) + STLPCI(1) * APF(2)              | Tratar el último
      STLPCI(1) =  ST(K)                                  | Introducir la señal de entrada por desplazamiento

IP = IP + IDIM                                            | Actualizar
```

En el código de coma fija es necesario, en primer lugar, convertir ST de coma flotante de bloque a formato Q2 de coma fija y, a continuación, escribir la versión en Q2 de ST en la memoria intermedia de posfiltro de largo plazo, SST, para su utilización posterior por el posfiltro de largo plazo. Obsérvese que esta memoria intermedia se identificó anteriormente mediante ST, en el seudocódigo de coma flotante. ST es de coma flotante de bloque y la memoria intermedia está en Q2. Para evitar confusiones fue necesario cambiar el nombre de la memoria intermedia a SST. A continuación se calcula el filtro inverso LPC. Se señala que los coeficientes del filtro LPC, APF, se representan en Q13.

```
NLS = 16 – NLSST + 2                                      | Calcular el número de desplazamientos a la izquierda
For K = 1, 2, ..., IDIM, do the next 2 lines              | para Q2
      AA0 = ST(K) << NLS
      SST(K) = RND(AA0)                                   | SST es la nueva memoria intermedia de
                                                          | posfiltro de largo plazo
If IP = NPWSZ, then set IP = NFRSZ                        | Comprobar y actualizar IP

                                                          | Iniciar el filtrado inverso LPC
For K = 1, 2, ..., IDIM, do the next 10 lines
      AA0 = SST(K)
      AA0 = AA0 << 13
      For J = 10, 9, ..., 3, 2, do the next 2 lines
            AA0 = AA0 + STLPCI(J) * APF(J + 1)
            STLPCI(J) = STLPCI(J – 1)
      AA0 = AA0 + STLPCI(1) * APF(2)
      STLPCI(1) = SST(K)
      ITMP = IP + K
      AA0 = AA0 << 2
      D(ITMP) = RND(AA0)                                  | D(ITMP) está en Q1

IP = IP + IDIM
```

G.3.25 Bloque 82 – Módulo de extracción del periodo de tono

Se empieza con la versión en coma flotante del seudocódigo para el bloque 82, el módulo de extracción del periodo de tono.

```
IF ICOUNT ≠ 3, skip the execution of this block
Otherwise, do the following
                                                          | Filtrado de paso bajo y reducción del número
                                                          | de muestras 4:1
For K = NPWSZ – NFRSZ + 1, ..., NPWSZ, do the next 7 lines
      TMP = D(K) – STLPF(1) * AL(1) – STLPF(2) * AL(2) – STLPF(3) * AL(3)    | Filtro IIR
      If K is divisible by 4, do the next 2 lines
            N = K/4                                       | Hacer el filtrado FIR sólo si se necesita
            DEC(N) = TEMP * BL(1) + STLPF(1) * BL(2) + STLPF(2) * BL(3) + STLPF(3) * BL(4)

      STLPF(3) = STLPF(2)
      STLPF(2) = STLPF(1)                                 | Cambiar la memoria del filtro de paso bajo
      STLPF(1) = TMP
```

```
        M1 = KPMIN/4                                          | Iniciar la toma de valores de correlación
        M2 = KPMAX/4                                          | en el dominio residual LPC diezmado
        CORMAX = most negative number of the machine
        For J = M1, M1 + 1, ..., M2, do the next 6 lines
            TMP = 0
            For N = 1, 2, ..., NPWSZ/4, do the next line
                TMP = TMP + DEC(N) * DEC(N – J)               | TMP = correlacion en el dominio diezmado
            If TMP > CORMAX, do the next 2 lines
                CORMAX = TMP                                  | Encontrar la correlación máxima
                KMAX = J                                      | y el retardo correspondiente
        For N = –M2 + 1, –M2 + 2, ..., (NPWSZ – NFRSZ)/4, do the next line
            DEC(N) = DEC(N + IDIM)                            | Cambiar la memoria intermedia
                                                              | del residuo LPC diezmado

        M1 = 4 * KMAX – 3                                     | Iniciar la toma de valores de correlación
                                                              | en el dominio no diezmado
        M2 = 4 * KMAX + 3
        If M1 < KPMIN, set M1 = KPMIN                         | Comprobar si M1 está fuera de gama
        If M2 > KPMAX, set M2 = KPMAX                         | Comprobar si M2 está fuera de gama
        CORMAX = most negative number of the machine

        For J = M1, M1 + 1, ..., M2, do the next 6 lines
            TMP = 0
            For K = 1, 2, ..., NPWSZ, do the next line
                TMP = TMP + D(K) * D(K – J)                   | Correlación en el dominio no diezmado
            If TMP > CORMAX, do the next 2 lines
                CORMAX = TMP                                  | Encontrar la correlación máxima y
                KP = J                                        | el retardo correspondiente

        M1 = KP1 – KPDELTA                                    | Determinar la gama de búsqueda alrededor
        M2 = KP1 + KPDELTA                                    | del periodo de tono de la trama anterior
        If KP < M2 + 1, go to LABEL                           | Si es verdad, KP no puede ser un tono múltiplo
                                                              | de la frecuencia fundamental de tono
        If M1 < KPMIN, set M1 = KPMIN                         | Comprobar si M1 está fuera de la gama
        CMAX = most negative number of the machine
        For J = M1, M1 + 1, ..., M2, do the next 6 lines
            TMP = 0
            For K = 1, 2, ..., NPWSZ, do the next line
                TMP = TMP + D(K) * D(K – J)                   | Correlación en el dominio no diezmado
            If TMP > CMAX, do the next 2 lines
                CMAX = TMP                                    | Encontrar la correlación máxima y
                KPTMP = J                                     | el retardo correspondiente
        SUM = 0
        TMP = 0                                               | Iniciar el cálculo de los pesos de
                                                              | las derivaciones
        For K = 1, 2, ..., NPWSZ, do the next 2 lines
            SUM = SUM + D(K – KP) * D(K – KP)
            TMP =  TMP + D(K – KPTMP) * D(K – KPTMP)
        If SUM = 0, set TAP = 0; otherwise, set TAP = CORMAX/SUM
        If TMP = 0., set TAP1 = 0.; otherwise, set TAP1 = CMAX/TMP
        If TAP > 1, set TAP = 1                               | Fijar TAP entre 0 y 1
        If TAP < 0, set TAP = 0
        If TAP1 > 1, set TAP1 = 1                             | Fijar TAP1 entre 0 y 1
        If TAP1 < 0, set TAP1 = 0
                                                              | Reemplazar KP por el tono fundamental
                                                              | si TAP1 es sufientemente grande
        If TAP1 > TAPTH  *  TAP, then set KP = KPTMP

LABEL:  KP1 = KP                                              | Actualizar el periodo de tono de la
                                                              | trama anterior
        For K = –KPMAX + 1, –KPMAX + 2, ..., NPWSZ – NFRSZ, do the next line
            D(K) = D(K + NFRSZ)                               | Cambiar la memoria intermedia del residuo LPC
```

En la versión en coma fija de este bloque, el ordenamiento D y las variables de estado del filtro de paso bajo se representan en Q1, para evitar el desbordamiento en los cálculos de la correlación y la energía. El seudocódigo de coma fija viene dado por lo siguiente.

```
If ICOUNT ≠ 3, skip the execution of this block
Otherwise, do the following

For K = NPWSZ – NFRSZ + 1, ..., NPWSZ, do the next 17 lines
    AA0 = D(K) * BL(0)                          | Hacer primero la parte FIR
    AA0 = AA0 + LPFFIR(1) * BL(1)               | D(K) está en Q1, BL( ) está en Q19
    AA0 = AA0 + LPFFIR(2) * BL(2)               | BL(0) = 18721, BL(1) = –3668
    AA0 = AA0 + LPFFIR(3) * BL(3)               | BL(2) = –3668, BL(3) = 18721
    LPFFIR(3) = LPFFIR(2)
    LPFFIR(2) = LPFFIR(1)
    LPFFIR(1) = D(K)                            | LPFFIR está en Q1
    AA0 = AA0 >> 6                              | Hacer ahora la parte IIR
    AA0 = AA0 – LPFIIR(1) * AL(1)               | AL( ) están en Q13, LPFIIR( ) están en Q1
    AA0 = AA0 – LPFIIR(2) * AL(2)               | AL(1) = –19172, AL(2) = 16481
    AA0 = AA0 – LPFIIR(3) * AL(3)               | AL(3) = –5031
    LPFIIR(3) = LPFIIR(2)
    LPFIIR(2) = LPFIIR(1)
    AA0 = AA0 << 3
    LPFIIR(1) = RND(AA0)                        | LPFIIR está en Q1
    N = (K >> 2)
    If K = (N << 2), set DEC(N) = LPFIIR(1)     | DEC(N) está en Q1

M1 = KPMIN/4                                    | Iniciar la toma de valores de correlación en
M2 = KPMAX/4                                    | el dominio residual LPC diezmado
AA1 = –2147483648                               | = –2^31

For J = M1, M1 + 1, ..., M2, do the next 6 lines
    AA0 = 0
    For N = 1, 2, ..., NPWSZ/4, do the next line
        AA0 = AA0 + DEC(N) * DEC(N – J)
    If AA0 > AA1, do the next 2 lines
        AA1 = AA0                               | Encontrar la correlación máxima y
        KMAX = J                                | el retardo correspondiente

For N = –M2 + 1, –M2 + 2, ..., (NPWSZ – NFRSZ)/4, do the next line
    DEC(N) = DEC(N + IDIM)

M1 = 4 * KMAX – 3                               | Iniciar la toma de valores de correlación en
                                                | el dominio no diezmado
M2 = 4 * KMAX + 3
If M1 < KPMIN, set M1 = KPMIN                   | Comprobar si M1 está fuera de la gama
If M2 > KPMAX, set M2 = KPMAX                   | Comprobar si M2 está fuera de la gama
AA1 = –2147483648                               | = –2^31

For J = M1, M1 + 1, ..., M2, do the next 6 lines
    AA0 = 0
    For K = 1, 2, ..., NPWSZ, do the next line
        AA0 = AA0 + D(K) * DEC(K – J)           | Correlación en el dominio no diezmado
    If AA0 > AA1, do the next 2 lines
        AA1 = AA0
        KP = J
CORMAX = AA1                                    | Guardar con precisión doble en CORMAX

M1 = KP1 – KPDELTA                              | Determinar la gama de búsqueda alrededor
M2 = KP1 + KPDELTA                              | del periodo de tono de la trama anterior
If KP < M2 + 1, go to LABEL                     | Si es verdad, KP no puede ser un tono
                                                | múltiplo de la frecuencia fundamental de tono
If M1 < KPMIN, set M1 = KPMIN                   | Comprobar si M1 está fuera de la gama
If M2 > KPMAX, set M2 = KPMAX                   | Comprobar si M2 está fuera de la gama
                                                | Este último enunciado no figura en
                                                | coma flotante
AA1 = –2147483648                               | = –2^31
```

```
        For J = M1, M1 + 1, ..., M2, do the next 6 lines
            AA0 = 0
            For K = 1, 2, ..., NPWSZ, do the next line
                AA0 = AA0 + D(K) * D(K – J)          | Correlación en el dominio no diezmado
            If AA0 > AA1, do the next 2 lines
                AA1 = AA0                            | Encontrar la correlación máxima y
                KPTMP = J                            | el retardo correspondiente
        CMAX = AA1                                   | Guardar con precisión doble en CMAX

    AA0 = 0
    AA1 = 0
    For K = 1, 2, ..., NPWSZ, do the next 2 lines
        AA0 = AA0 + D(K – KP) * D(K – KP)
        AA1 = AA1 + D(K – KPTMP) * D(K – KPTMP)

                                                     | Encontrar TAP
                                                     | Recortar los pesos de las derivaciones
                                                     | si es necesario
    If AA0 = 0, set CORMAX = 0
    If AA1 = 0, set CMAX = 0
    If CORMAX > AA0, set CORMAX = AA0
    If CORMAX < 0, set CORMAX = 0
    If CMAX > AA1, set CMAX = AA1
    If CMAX < 0, set CMAX = 0

    If AA0 > AA1, do the next 2 lines
        call VSCALE(AA0, 1, 1, 30, AA0, NLS)
        AA1 = AA1 << NLS
    otherwise do the next 2 lines
        call VSCALE(AA1, 1, 1, 30, AA1, NLS)
        AA0 = AA0 << NLS

    SUM = AA0 >> 16
    TMP = AA1 >> 16
    AA0 = CORMAX << NLS
    CORMAX = AA0 >> 16
    AA0 = CMAX << NLS
    CMAX = AA0 >> 16
    AA1 = CORMAX * TMP
    AA1 = AA1 >> 16
    AA1 = AA1 * ITAPTH                               | ITAPTH = 26214 en Q16
    AA0 = CMAX * SUM
    If AA0 > AA1, set KP – KPTMP

LABEL:  KP1 = KP                                     | Actualizar KP1 y cambiar el residuo LPC
    For  K = –KPMAX + 1, –KPMAX + 2, ..., NPWSZ – NFRSZ, do the next line
        D(K) = D(K + NFRSZ)                          | Cambiar la memoria intermedia del residuo LPC
```

G.3.26 Bloque 83 – Calculador de derivación de predictor de tono

Se empieza con la versión en coma flotante del seudocódigo. Aquí se utiliza SST en vez de ST para el nombre de la memoria intermedia del posfiltro de largo plazo.

```
    If ICOUNT ≠ 3, skip the execution of this block
    Otherwise, do the following
    SUM = 0
    TMP = 0
    For K = –NPWSZ + 1, –NPWSZ + 2, ..., 0, do the next 2 lines
        SUM = SUM + SST(K – KP) * SST(K – KP)
        TMP = TMP + SST(K) * SST(K – KP)
    If SUM = 0, set PTAP = 0; otherwise, set PTAP = TMP/SUM
```

En lo que sigue se da el seudocódigo de coma fija. Obsérvese que SST( ) está en Q0 de 13 bits. Al efectuar las correlaciones, la multiplicación de SST por sí misma o por una muestra retardada da un resultado que está en Q0 de 25 bits.

```
If ICOUNT ≠ 3, skip the execution of this block
Otherwise, do the following
    AA0 = 0
    AA1 = 0
    For K = –NPWSZ + 1, –NPWSZ + 2, ..., 0, do the next 4 lines
        P = SST(K – KP) * SST(K – KP)
        AA0 = AA0 + P
        P = SST(K) * SST(K – KP)
        AA1 = AA1 + P


    If AA0 = 0, set PTAP = 0 and return to calling program
    If AA1 ≤ 0, set PTAP = 0 and return to calling program
    If AA1 ≥ AA0, set PTAP = 16384                          | NLSPTAP = 14
    Otherwise, do the following
        Call VSCALE(AA0, 1, 1, 30, AA0, NLSDEN)
        Call VSCALE(AA1, 1, 1, 30, AA1, NLSNUM)
        NUM = RND(AA1)
        DEN = RND(AA0)
        Call DIVIDE(NUM, NLSNUM, DEN, NLSDEN, PTAP, NLSPTAP)
        NRS = NLSPTAP – 14
        PTAP = PTAP >> NRS                                  | NLSPTAP = 14
```

G.3.27 Bloque 84 – Calculador de los coeficientes del posfiltro de largo plazo

Se empieza con el seudocódigo de coma flotante para el bloque 84.

```
    If ICOUNT ≠ 3, skip the execution of this block
    Otherwise, do the following
    If PTAP > 1, set PTAP = 1                               | Fijar PTAP en 1
    If PTAP < PPFTH, set PTAP = 0                           | Desactivar el posfiltro de tono
                                                            | si PTAP es más pequeño que el umbral


    B = PPFZCF * PTAP
    GL = 1/(1 + B)
```

A continuación se da el seudocódigo de coma fija. Se define una variable adicional, GLB, que es el producto de GL y B. De esta manera se evitan ulteriores multiplicaciones. B y GLB son salidas en Q16 y GL es salida en Q14.

```
                                                            | Obsérvese que PTAP es < 16385
                                                            | según bloque 83
If ICOUNT ≠ 3, skip the execution of this block
Otherwise, do the following
    If PTAP < PPFTH, set PTAP = 0                           | PPFTH = 9830 en Q14
    AA0 = PPFZCF * PTAP                                     | PPFZCF = 9830 en Q16, PTAP está en Q14
    B = AA0 >> 14                                           | Guardar B en Q16
    AA0 = AA0 >> 16                                         | AA0 = B en Q14
    AA0 = AA0 + 16384
    DEN = AA0                                               | DEN está en Q14
    Call DIVIDE(16384, 14, DEN, 14, GL, NLS)
    AA0 = GL * B                                            | NLS = 14 ó 15, NLS de B = 16
    GLB = AA0 >> NLS                                        | GLB es GL * B y aquí se precalcula en Q16
                                                            | para el bloque 71
    NRS = NLS – 14
    If NRS > 0, SET GL = GL >> NRS                          | Poner GL en Q14
```

G.3.28 Bloque 85 – Calculador de los coeficientes del posfiltro de corto plazo

Se empieza con el seudocódigo de coma flotante para este bloque.

```
If ICOUNT ≠ 1, skip the execution of this block
Otherwise, do the following
For I = 2, 3, ..., 11, do the next 2 lines
    AP(I) = SPFPCFV(I) * APF(I)          | Escalar los coeficientes del denominador
    AZ(I) = SPFZCFV(I) * APF(I)          | Escalar los coeficientes del numerador
TILTZ = TILTF * RC1                      | «Inclinar» los coeficientes del filtro
                                         | de compensación
```

En el seudocódigo de coma fija se debe considerar la posibilidad de que haya un desajuste en la recursión de Durbin, o que los coeficientes de predicción no puedan expresarse ni siquiera en Q13. (Nunca se ha observado que Q13 fuese insuficiente, pero de todos modos hay que tener en cuenta esa posibilidad.) La variable ILLCONDP es una bandera del bloque 50 que indica si los resultados de dicho bloque son válidos o no. En la Recomendación G.728 se supone implícitamente que no se utilizarán los resultados de Durbin si ILLCONDP es verdad. Esto es, ATMP no será copiado en APF una vez que se haya completado la recursión de décimo orden. La misma suposición se establece aquí. Si ILLCONDP es verdad, no se actualizan AP, AZ o TILTZ.

A continuación hay que considerar la posibilidad de que los coeficientes APF( ) de la recursión de Durbin estén en Q13, Q14 o Q15. NLSAPF es el número de desplazamientos a la izquierda de APF. A la salida, se desea guardar también APF( ) en Q13 para su utilización posterior en la operación de filtrado inverso LPC. Se quiere que los coeficientes del numerador y del denominador, AP( ) y AZ( ), estén en Q14 para la salida. TILTZ es la salida en Q14. Puede ocurrir que AP no pueda representarse en Q14. Si tal es el caso, no se actualizan AP, AZ o TILTZ pero pueden utilizarse los nuevos valores para APF. Deben estar ya en formato Q13. El seudocódigo de coma fija viene dado por lo siguiente.

```
If ICOUNT ≠ 1, skip the execution of this block
Otherwise, do the following

                                         | Comprobar primero que ILLCONDP es verdad

If ILLCONDP = .TRUE., skip the execution of this block
Otherwise, do the following

                                         | Hacer a continuación los coeficientes del denominador
                                         | Si desbordan para Q14,
                                         | no se actualizan AP, AZ o TILTZ
                                         | El ordenamiento temporal WS se utiliza en caso
                                         | de desbordamiento, con lo que se preserva AP
For I = 2 and 3, do the next 6 lines
    AA0 = SPFPCFV(I) * APF(I)            | SPFPCFV está en Q14, AA0 es 14 + NLSAPF
    If NLSAPF = 13, AA0 = AA0 << 3       | Poner AA0 en Q30 en los 3 casos mediante
    If NLSAPF = 14, AA0 = AA0 << 2       | el número apropiado de desplazamientos
    If NLSAPF = 15, AA0 = AA0 << 1
    If AA0 overflowed above, go to LABEL
    WS(I) = RND(AA0)                     | Redondear a la palabra alta para WS
                                         | El desbordamiento sólo puede ocurrir
                                         | para 2 y 3, por lo que se copian éstos en AP
                                         | y se continúa
For I = 2 and 3, do the next line
    AP(I) = WS(I)

                                         | Hacer ahora el resto
For I = 4, 5, ..., 11, do the next 5 lines
    AA0 = SPFPCFV(I) * APF(I)            | SPFPCFV está en Q14, AA0 es 14 + NLSAPF
    If NLSAPF = 13, AA0 = AA0 << 3       | Poner AA0 en Q30 en los 3 casos mediante
    If NLSAPF = 14, AA0 = AA0 << 2       | el número apropiado de desplazamientos
    If NLSAPF = 15, AA0 = AA0 << 1
    AP(I) = RND(AA0)                     | Redondear a la palabra alta para AP
                                         | Hacer ahora los coeficientes del numerador
                                         | Si el denominador no desbordó,
                                         | tampoco puede desbordar el numerador
```

```
    For I = 2, 3, ..., 11, do the next 5 lines
        AA0 = SPFZCFV(I) * APF(I)          | SPFZCFV está en Q14, AA0 es 14 + NLSAPF
        If NLSAPF = 13, AA0 = AA0 << 3     | Poner AA0 en Q30 en los 3 casos mediante
        If NLSAPF = 14, AA0 = AA0 << 2     | el número apropiado de desplazamientos
        If NLSAPF = 15, AA0 = AA0 << 1
        AZ(I) = RND(AA0)                   | Redondear a la palabra alta para AZ

                                           | Actualizar ahora TILTZ
    AA0 = TILTF * RC1                      | TILTZ = 4915 en Q15
    TILTZ = RND(AA0)                       | RC1 está en Q15
                                           | TILTZ está en Q14

LABEL:                                     | Guardar APF( ) en Q3 para ulterior
                                           | filtrado inverso LPC
                                           | Caso 1: NLSAPF = 13, no hacer nada

    If NLSAPF = 14, do the next 3 lines    | Caso 2: NLSAPF = 14, desplazar 15,
        For I = 2, 3, 4, ..., 11, do the next 2 lines | redondear
            AA0 = APF(I) << 15
            APF(I) = RND(AA0)

    If NLSAPF = 15, do the next 3 lines    | Caso 3: NLSAPF = 15, desplazar 14,
        For I = 2, 3, 4, ..., 11, do the next 2 lines | redondear
            AA0 = APF(I) << 14
            APF(I) = RND(AA0)
```

Obsérvese que, en el código anterior, los bucles «For» que contienen tres instrucciones «If NLSAPF = ...» pueden ser eliminados si se reescribe el código en su totalidad para cada uno de los tres posibles valores de NLSAPF. Este código más largo producirá exactamente los mismos resultados, pero su actuación será más rápida en la mayoría de los dispositivos programables.

G.4 Representaciones de variables internas de LD-CELP

En esta subcláusula se presentan versiones actualizadas de los Cuadros 1/G.728 y 2/G.728. El Cuadro G.1 es una versión abreviada del Cuadro 1/G.728. Las constantes cuya relación figura en el mismo son sólo aquellas que no son enteros inherentemente y no se dan en ningún otro sitio de la presente Recomendación. Se han suprimido las columnas Símbolo equivalente y Valor del Cuadro 1/G.728 para dejar sitio para el formato de coma fija y la representación requerida de cada variable. El Cuadro G.2 es la versión entera del Cuadro 2/G.728. Se han suprimido las columnas Símbolo equivalente y Valor inicial del Cuadro 2/G.728 para presentar el formato de coma fija. En la relación de variables figuran algunas variables nuevas que están relacionadas solamente con la especificación de coma fija.

CUADRO G.1/G.728

Parámetros básicos del codificador que no son enteros inherentemente y no se dan en ningún otro sitio

Nombre	Valor en coma flotante	Valor en coma fija	Formato Q	Descripción
AGCFAC	0,99	16220	Q14	Factor de control de la velocidad de adaptación de AGC
AGCFAC1	0,01	20972	Q21	Valor de (1 – AGCFAC)
GOFF	32	16384	Q9	Valor de desplazamiento de la ganancia logarítmica
PPFTH	0,6	9830	Q14	Umbral de derivación para desactivar el posfiltro de tono
PPFZCF	0,15	9830	Q16	Factor de control de ceros del posfiltro de tono
TAPTH	0,4	26214	Q16	Umbral de derivación para el reemplazo del tono fundamental
TILTF	0,15	4915	Q15	Factor de control de la compensación de la inclinación espectral

CUADRO G.2/G.728

Variables de procesamiento interno del algoritmo LD-CELP

Nombre	Gama de los índices del ordenamiento	Formato de coma fija	Descripción
A	1 a LPC + 1	Q14	Coeficientes del filtro de síntesis
AL	1 a 3	Q13	Coeficientes del denominador del filtro de paso bajo de 1 kHz
AP	1 a 11	Q14	Coeficientes del denominador del posfiltro de corto plazo
APF	1 a 11	Q13	Coeficientes del filtro LPC de décimo orden
ATMP	1 a LPC + 1	Q13/Q14/Q15	Memoria intermedia temporal para los coeficientes del filtro de síntesis
AWP	1 a LPCW + 1	Q14	Coeficientes del denominador del filtro de ponderación perceptual
AWZ	1 a LPCW + 1	Q14	Coeficientes del numerador del filtro de ponderación perceptual
AWZTMP	1 a LPCW + 1	Q13/Q14/Q15	Memoria intermedia temporal para los coeficientes del filtro de ponderación
AZ	1 a 11	Q14	Coeficientes del numerador del posfiltro de corto plazo
B	1	Q16	Coeficientes del posfiltro de largo plazo
BL	1 a 4	Q19	Coeficientes del numerador del filtro de paso bajo de 1 kHz
D	–139 a 100	Q1	Residuo de la predicción LPC
DEC	–34 a 25	Q1	Residuo de la predicción LPC diezmado 4:1
ET	1 a IDIM	15b BFL	Vector de excitación con escalamiento de ganancia
FACV	1 a LPC + 1	Q14	Vector de ampliación de la anchura de banda del filtro de síntesis
FACGPV	1 a LPCLG + 1	Q14	Vector de ampliación de la anchura de banda del predictor de ganancia
G2	1 a NG	Q12	2 veces los niveles de ganancia en la tabla de códigos cifrados de ganancia
GAIN	1	SFL	Ganancia de excitación lineal
GB	1 a NG – 1	Q13	Punto medio entre niveles de ganancia adyacentes
GL	1	Q14	Factor de escala del posfiltro de largo plazo
GLB	1	Q16	Término del producto del posfiltro de largo plazo
GP	1 a LPCLG + 1	Q14	Coeficientes del predictor de ganancia logarítmica, valor inicial = 16384, –16384, 0, ..., 0
GPTMP	1 a LPCLG + 1	Q13/Q14/Q15	Ordenamiento temporal para los coeficientes del predictor lineal de ganancia logarítmica
GQ	1 a NG	Q13	Niveles de ganancia en la tabla de códigos cifrados de ganancia
GSQ	1 a NG	Q11	Cuadrados de los niveles de ganancia en la tabla de códigos cifrados de ganancia
GSTATE	1 a LPCLG	Q9	Memoria del predictor de ganancia logarítmica, valor inicial = –16384
GTMP	1 a 4	Q9	Memoria intermedia temporal de ganancia logarítmica, valor inicial = –16384
H	1 a IDIM	Q13	Vector de respuesta a los impulsos de F(z) W(z)
ICHAN	1	Q0	Mejor índice de código cifrado que ha de transmitirse
ICOUNT	1	Q0	Contador de vector de señal vocal (indizado de 1 a 4)
IG	1	Q0	Mejor índice de código cifrado de ganancia de 3 bits
ILLCOND	1	Q0	Bandera de desajuste para filtro de síntesis
ILLCONDG	1	Q0	Bandera de desajuste para predictor de ganancia logarítmica
ILLCONDP	1	Q0	Bandera de desajuste para posfiltro
ILLCONDW	1	Q0	Bandera de desajuste para filtro de ponderación
IP	1	Q0	Puntero de dirección del residuo de predicción LPC
IS	1	Q0	Mejor índice de código cifrado de forma de 7 bits
KP	1	Q0	Periodo de tono de la trama actual
KP1	1	Q0	Periodo de tono de la trama anterior

CUADRO G.2/G.728 *(continuación)*

Variables de procesamiento interno del algoritmo LD-CELP

Nombre	Gama de los índices del ordenamiento	Formato de coma fija	Descripción
LOGGAIN	1	Q9	Logaritmo de la ganancia de excitación
LPFFIR	3	Q1	Memoria FIR del filtro de paso bajo
LPFIIR	3	Q1	Memoria IIR del filtro de paso bajo
NLSATMP	1	Q0	Bandera de precisión de recursión de Durbin para ATMP
NLSAWZTMP	1	Q0	Bandera de precisión de recursión de Durbin para AWZTMP
NLSGPTMP	1	Q0	Bandera de precisión de recursión de Durbin para GPTMP
NLSET	1	Q0	NLS para ET
NLSGAIN	1	Q0	NLS para GAIN lineal
NLSREXP	1	Q0	NLS para REXP, valor inicial = 31
NLSREXPLG	1	Q0	NLS para REXPLG, valor inicial = 31
NLSREXPW	1	Q0	NLS para REXPW, valor inicial = 31
NLSSB	21	Q0	NLS para SB, valor inicial = 16
NLSST	1	Q0	NLS para ST en decodificador
NLSSTATE	11	Q0	NLS para STATELPC, valor inicial = 16
NLSSTTMP	4	Q0	NLS para STTMP, valor inicial = 16
PN	1 a IDIM	Q7	Vector de correlación para la búsqueda de código cifrado
PTAP	1	Q14	Derivación del predictor de tono calculada por el bloque 83
R	1 a 11	BFL	Coeficientes de autocorrelación
RC	1	Q15	Coeficientes de reflexión
RC1	1	Q15	Memoria intermedia temporal para los primeros coeficientes de reflexión
REXP	1 a LPC + 1	BFL	Parte recursiva de autocorrelación, filtro de síntesis
REXPLG	1 a LPCLG + 1	BFL	Parte recursiva de autocorrelación, predictor de ganancia logarítmica
REXPW	1 a LPCW + 1	BFL	Parte recursiva de autocorrelación, filtro de ponderación
RTMP	1 a LPC + 1	BFL	Memoria intermedia temporal para los coeficientes de autocorrelación
S	1 a IDIM	15b Q2	Vector de señal vocal de entrada MIC uniforme
SB	1 a 105	14b BFL	Memoria intermedia para la señal vocal cuantificada anteriormente
SBLG	1 a 34	Q9	Memoria intermedia para la ganancia logarítmica anterior
SBW	1 a 60	Q2	Memoria intermedia para la señal vocal de entrada anterior
SCALE	1	SFL	Factor de escala de posfiltro no filtrado
SCALEFIL	1	Q14	Factor de escala de posfiltro filtrado en paso bajo, valor inicial = 16384
SD	1 a IDIM	Q0	Memoria intermedia de señal vocal decodificada
SPF	1 a IDIM	Q2	Vector de señal vocal posfiltrada
SPFPCFV	1 a 11	Q14	Vector de control de polos del posfiltro de corto plazo
SPFZCFV	1 a 11	Q14	Vector de control de ceros del posfiltro de corto plazo
SO	1	byte	Muestra de señal vocal de entrada MIC de ley A o ley μ
SST (past)	–239 a 0	13b Q0	Memoria intermedia de señal vocal cuantificada
SST (current)	1 a IDIM	15b Q2	Memoria intermedia de señal vocal cuantificada
ST	1 a IDIM	14b BFL	Vector de señal vocal cuantificada
STATELPC	1 a LPC	14b SBFL	Memoria de filtro de síntesis
STLPCI	1 a 10	Q2	Memoria de filtro inverso LPC

CUADRO G.2/G.728 *(fin)*

Variables de procesamiento interno del algoritmo LD-CELP

Nombre	Gama de los índices del ordenamiento	Formato de coma fija	Descripción
STMP	1 a 4 ∗ IDIM	15b Q2	Memoria intermedia para la ventana híbrida del filtro de ponderación perceptual
STTMP	1 a 4 ∗ IDIM	14b SBFL	Memoria intermedia para la ventana híbrida del filtro de síntesis
STPFFIR	1 a 10	Q2	Memoria del posfiltro de corto plazo, sección todos ceros
STPFIIR	1 a 10	Q2	Memoria del posfiltro de corto plazo, sección todos polos
SU	1	Q2	Muestra de señal vocal de entrada MIC uniforme
SUMFIL	1	Q2	Suma de los valores absolutos de la señal vocal posfiltrada
SUMUNFIL	1	Q2	Suma de los valores absolutos de la señal vocal decodificada
SW	1 a IDIM	Q2	Vector de señal vocal ponderada perceptualmente
TARGET	1 a IDIM	BFL	Vector objetivo VQ (con escalamiento de ganancias)
TEMP	1 a IDIM	*	Ordenamiento transitorio para espacio de trabajo temporal
TILTZ	1	Q14	Coeficiente de compensación de inclinación del posfiltro de corto plazo
WFIR	1 a LPCW	Q2	Memoria del filtro de ponderación 4, parte todos ceros
WIIR	1 a LPCW	Q2	Memoria del filtro de ponderación 4, parte todos polos
WNR	1 a 105	Q15	Función de ventana para el filtro de síntesis
WNRLG	1 a 34	Q15	Función de ventana para el predictor de ganancia logarítmica
WNRW	1 a 60	Q15	Función de ventana para el filtro de ponderación
WPCFV	1 a LPCW + 1	Q14	Vector de control de polos del filtro de ponderación perceptual
WS	1 a 105	#	Ordenamiento «espacio de trabajo» para variables intermedias
WZCFV	1 a LPCW + 1	Q14	Vector de control de ceros del filtro de ponderación perceptual
Y	1 a IDIM ∗ NCWD	Q11	Ordenamiento de códigos cifrados de forma
Y2	1 a NCWD	Q5	Energía del vector de código de forma convolucionado
ZIR	1 a IDIM	15b Q2	Respuesta de entrada cero
ZIRWFIR	1 a LPCW	15b Q2	Memoria de filtro de ponderación 10, parte todos ceros
ZIRWIIR	1 a LPCW	15b Q2	Memoria de filtro de ponderación 10, parte todos polos

SFL	Coma flotante escalar (*scalar floating point*)
BFL	Coma flotante de bloque (*block floating point*)
SBFL	Coma flotante de bloque segmentado (*segmented block floating point*)
Qx	Formato Qx
14b o 15b	Indican precisión de 14 ó 15 bits, en los demás casos se supone que la precisión es de 16 bits total
#	Definido por el uso, puede ser BFL o Q fijo ya que se trata de memoria transitoria

* TEMP es un ordenamiento de trabajo temporal y se utiliza en varios bloques; su formato Q puede cambiar de un bloque a otro.

G.5 Tablas de ganancia logarítmica para vectores de código cifrado de ganancia y de forma

Véanse los Cuadros G.3 y G.4.

CUADRO G.3/G.728

Ganancia en coma flotante en dB y representación en coma fija en Q11 para vectores de código cifrado de ganancia

Índice		dB	Coma fija
0	4	–5,7534180	–11783
1	5	–0,8925781	–1828
2	6	3,9682620	8127
3	7	8,8291020	18082

NOTA –Para obtener el valor en coma fija, multiplicar el valor en coma flotante por $2048 = 2^{11}$.

CUADRO G.4/G.728

Ganancia en coma flotante en dB y representación en coma fija en Q11 para vectores de código cifrado de forma

Índice	dB	Coma fija	Índice	dB	Coma fija	Índice	dB	Coma fija
0	–0,1108398	–227	43	1,1064450	2266	85	1,7133790	3509
1	5,0332030	10308	44	7,0932620	14527	86	0,4252930	871
2	3,1977540	6549	45	9,1738280	18788	87	1,0693360	2190
3	3,7856450	7753	46	6,3623050	13030	88	2,7080080	5546
4	3,7094730	7597	47	3,0458980	6238	89	7,4887700	15337
5	8,0874020	16563	48	0,8911133	1825	90	1,8105470	3708
6	3,1279300	6406	49	4,4384770	9090	91	1,1748050	2406
7	5,8266600	11933	50	0,1030273	211	92	2,8076170	5750
8	6,6254880	13569	51	0,9218750	1888	93	3,6806640	7538
9	5,1606450	10569	52	8,8320310	18088	94	1,9101560	3912
10	7,9726560	16328	53	11,0141600	22557	95	1,7299800	3543
11	3,1914060	6536	54	5,3188480	10893	96	–4,9335940	–10104
12	7,7163090	15803	55	8,8652340	18156	97	0,1479492	303
13	5,6997070	11673	56	1,6728520	3426	98	–3,0083010	–6161
14	10,4091800	21318	57	6,5429690	13400	99	–0,5576172	–1142
15	4,4433590	9100	58	–2,1362300	–4375	100	1,8881840	3867
16	5,9790040	12245	59	3,8916020	7970	101	2,8979490	5935
17	5,8681640	12018	60	3,7861330	7754	102	–3,5161130	–7201
18	1,2221680	2503	61	12,3388700	25270	103	–0,3706055	–759
19	7,1728520	14690	62	2,5942380	5313	104	–1,0219730	–2093
20	8,8818360	18190	63	7,6245120	15615	105	–1,3979490	–2863
21	14,0629900	28801	64	–3,0742190	–6296	106	1,0825200	2217
22	8,2045900	16803	65	2,2021480	4510	107	–1,5834960	–3243
23	9,9272460	20331	66	1,0751950	2202	108	3,0083010	6161
24	8,7983400	18019	67	–3,5297850	–7229	109	2,8579100	5853
25	12,1679700	24920	68	1,5361330	3146	110	3,7104490	7599
26	7,8901370	16159	69	–1,3759770	–2818	111	3,2944340	6747
27	8,6025390	17618	70	–1,3056640	–2674	112	–0,9770508	–2001
28	11,2656300	23072	71	–0,7651367	–1567	113	4,9892580	10218
29	13,7085000	28075	72	0,8989258	1841	114	–0,0263672	–54
30	9,3598630	19169	73	2,8334960	5803	115	0,9335938	1912
31	12,5600600	25723	74	3,8203130	7824	116	5,6127930	11495
32	4,2333980	8670	75	0,1557617	319	117	5,1635740	10575
33	4,9165040	10069	76	0,8862305	1815	118	2,2055660	4517
34	0,2456055	503	77	0,8618164	1765	119	2,0893550	4279
35	4,2221680	8647	78	3,3930660	6949	120	0,8852539	1813
36	5,4516600	11165	79	1,2128910	2484	121	0,2763672	566
37	9,0073240	18447	80	1,3710940	2808	122	2,2309570	4569
38	2,0820310	4264	81	4,7431640	9714	123	2,0278320	4153
39	8,4868160	17381	82	–2,0581050	–4215	124	1,6445310	3368
40	1,7241210	3531	83	3,2607420	6678	125	5,4584960	11179
41	5,1479490	10543	84	1,2861330	2634	126	0,8271484	1694
42	–1,1679690	–2392				127	0,3715820	761

NOTA – Para obtener el valor en coma fija, multiplicar el valor en coma flotante por $2048 = 2^{11}$.

G.6 Valores enteros de los ordenamientos relativos a los códigos cifrados de ganancia

Esta subcláusula contiene los valores enteros equivalentes a los de coma flotante de la tabla que figura en el Anexo B/G.728 (véase el Cuadro G.5).

CUADRO G.5/G.728

Valores enteros de los ordenamientos relativos a los códigos cifrados de ganancia

Índice del ordenamiento	1	2	3	4	5	6	7	8
GQ (Q13)	4224	7392	12936	22638	–4224	–7392	–12936	–22638
GB (Q13)	5808	10164	17787	*	–5808	–10164	–17787	*
G2 (Q12)	4224	7392	12936	22638	–4224	–7392	–12936	–22638
GSQ (Q11)	545	1668	5107	15640	545	1668	5107	15640
* Puede ser cualquier valor arbitrario (no utilizado).								

G.7 Seudocódigos del programa principal del codificador y del decodificador

En esta subcláusula se dan los seudocódigos del programa principal del codificador y del decodificador. Lo que se pretende, sobre todo, es mostrar el orden en que se ejecutan los diversos bloques. Por ello sólo se muestra la secuencia de ejecución de bloques y no se describen los detalles de orden menor del paso de parámetros. Se señala que la secuencia de ejecución permitida no es única. Hay muchos órdenes diferentes de ejecución, con todos los cuales se llega a un resultado exacto a nivel de bit. Los seudocódigos que se muestran a continuación son tan sólo dos ejemplos. No obstante, si se utiliza un orden de ejecución de bloques distinto, el implementador deberá asegurar que se llega a resultados exactos a nivel de bit.

A continuación se indica el seudocódigo del programa principal del codificador.

```
Initialize all encoder variables to their initial values.

Initialize y2( ) by executing blocks 14 and 15 with h = [8192, 0, 0, 0, 0]

ILLCOND = .FALSE.
ILLCONDW = .FALSE.
ILLCONDG = .FALSE.
ICOUNT = 0

VEC_LOOP:
        If ICOUNT = 4, set ICOUNT = 0                          | Reiniciar el contador de vectores
        ICOUNT = ICOUNT + 1                                    | Actualizar el contador de vectores
        Get one vector of input speech from the input buffer
        Convert input speech vector to the range [–16384, +16383],
            then assign to S( )                                | Representación en Q2 de [–4096, +4095,75]

                                                               | Comprobar si hay que actualizar los
                                                               | coeficientes del filtro
        If ICOUNT = 3, do the next 4 lines
            If ILLCOND = .FALSE., do block 51
            If ILLCONDW = .FALSE., do block 38
            do block 12
            do blocks 14 and 15

        If ICOUNT = 2 and ILLCONDG = .FALSE., then do block 45

                                                               | Iniciar el procesamiento «una vez por vector»
        do blocks 46, 98, 99, and 48                           | Obtener la ganancia adaptada hacia atrás
                                                               | GSTATE(1:9) desplazada hacia abajo una posición
        do "blockzir" (blocks 9 and 10 during zero-input response calculation)
        do block 4                                             | Filtro de ponderación perceptual
        do block 11                                            | Cálculo del vector objetivo VQ
```

```
do block 16                                  | Normalización del vector objetivo VQ
do block 13                                  | Convolución con inversión del tiempo
do blocks 17 and 18                          | Búsqueda del código cifrado de excitación
put out ICHAN to the communication channel
do blocks 19 and 21                          | Escalar el vector de código de excitación seleccionado
do blocks 9 and 10 during filter memory update | Obtener ST( )
do blocks 93, 94, 96, and 97                 | Actualizar la ganancia logarítmica; obsérvese
                                             | que las 3 unidades de retardo del adaptador
                                             | de ganancia se producen de manera natural en
                                             | el proceso de puesta en bucle y no es necesario
                                             | implementarlas explícitamente
GSTATE(1) =  output of block 97              | Actualizar la memoria del predictor de ganancia
I = (ICOUNT – 1) ∗ IDIM                      | I = dirección de iniciación de STTMP( )
copy ST(1:5) to STTMP(I + 1:I + 5)           | Actualizar STTMP( )
NLSSTTMP(ICOUNT) = NLSST
I = (ICOUNT – 3) ∗ IDIM
If ICOUNT < 3, set I = I + 20                | I = dirección de iniciación de STMP( )
copy S(1:5) to STMP(I + 1:I + 5)             | Actualizar STMP( )
                                             | Final del procesamiento «una vez por vector»

                                             | Iniciar el procesamiento «una vez por trama»
If ICOUNT = 4, do the next 2 lines
    do block 49                              | Bandera de desajuste en
                                             | la salida = ILLCOND
    do block 50                              | Coeficientes del predictor de salida = ATMP( )
                                             | Bandera de desajuste en
                                             | la salida = ILLCOND
If ICOUNT = 2, do the next 2 lines
    do block 36                              | Bandera de desajuste en
                                             | la salida = ILLCONDW
    do block 37                              | Coeficientes del predictor de salida = AWZTMP( )
                                             | Bandera de desajuste en
                                             | la salida = ILLCONDW
If ICOUNT = 1, do the next 6 lines
    GTMP(1) = GSTATE(4)                      | Actualizar GTMP( ) de una sola vez
    GTMP(2) = GSTATE(3)
    GTMP(3) = GSTATE(2)
    GTMP(4) = GSTATE(1)
    do block 43                              | Bandera de desajuste en
                                             | la salida = ILLCONDG
    do block 44                              | Coeficientes del predictor de salida = GPTMP( )
                                             | Bandera de desajuste en
                                             | la salida = ILLCONDG
                                             | Fin del procesamiento «una vez por trama»
Go to VEC_LOOP
```

A continuación se da el seudocódigo del programa principal del decodificador. También aquí se muestra solamente la secuencia de ejecución de bloques y no se describen los detalles de orden menor del paso de parámetros.

```
Initialize all decoder variables to their initial values.

ILLCOND = .FALSE.
ILLCONDG = .FALSE.
ILLCONDP = .FALSE.
ICOUNT = 0

VEC_LOOP:
    If ICOUNT = 4, set ICOUNT = 0            | Reiniciar el contador de vectores
    ICOUNT = ICOUNT + 1                      | Actualizar el contador de vectores
    Get ICHAN of the current vector from the input buffer
    Obtain the shape index IS and gain index IG from ICHAN

                                             | Comprobar si hay que actualizar los
                                             | coeficientes del filtro
```

```
If ICOUNT = 3, do the next line
    If ILLCOND = .FALSE., do block 51

If ICOUNT = 2 and ILLCONDG = .FALSE., then do block 45

                                                    | Iniciar el procesamiento «una vez por vector»
do blocks 46, 98, 99, and 48                        | Obtener la ganancia adaptada hacia atrás
                                                    | GSTATE(1:9) desplazada hacia abajo una posición
do blocks 19 and 21                                 | Escalar el vector de código de excitación seleccionado
do block 32
If ICOUNT = 1, do block 85                          | Actualizar los coeficientes de posfiltro de
                                                    | corto plazo
do block 81
If ICOUNT = 3, do the next 3 lines
    do block 82                                     | Extracción del periodo de tono
    do block 83                                     | Calcular la derivación del predictor
                                                    | de tono
    do block 84                                     | Actualizar los coeficientes de posfiltro de
                                                    | largo plazo
do block 71                                         | Posfiltro de largo plazo
do block 72                                         | Posfiltro de corto plazo
do blocks 73 and 74                                 | Calcular las sumas de los valores absolutos
do block 75                                         | Relación entre las sumas de los valores absolutos
do block 76                                         | Filtro de paso bajo del factor de escalamiento
do block 77                                         | Control de ganancia de la salida de posfiltro

do blocks 93, 94, 96, and 97                        | Actualizar la ganancia logarítmica; obsérvese
                                                    | que las tres unidades de retardo del adaptador
                                                    | de ganancia se producen de manera natural
                                                    | en el proceso de puesta en bucle y no es
                                                    | necesario implementarlas explícitamente
GSTATE(1) =  output of block 97                     | Actualizar la memoria del predictor de ganancia
I = (ICOUNT – 1) * IDIM                             | I = dirección de iniciación de STTMP( )
copy ST(1:5) to STTMP(I + 1:I + 5)                  | Actualizar STTMP( )
NLSSTTMP(ICOUNT) = NLSST                            | Final del procesamiento «una vez por vector»

                                                    | Iniciar el procesamiento «una vez por trama»
If ICOUNT = 4, do the next 5 lines
    do block 49                                     | Bandera de desajuste en
                                                    | la salida = ILLCOND

    do block 50, order 1 to 10                      | Coeficientes del predictor de salida = ATMP( )
                                                    | con NLSATMP
                                                    | Bandera de desajuste en
                                                    | la salida = ILLCONDP
    NLSAPF = NLSATMP                                | Guardar el predictor de décimo orden
    copy ATMP(2:11) to APF(2:11)                    | para su empleo ulterior por el posfiltro
    continue block 50, order 11 to 50               | Continuar hasta acabar el bloque 50
                                                    | Coeficientes del predictor de salida = ATMP( )
                                                    | con NLSATMP
                                                    | Bandera de desajuste en
                                                    | la salida = ILLCOND

If ICOUNT = 1, do the next 6 lines
    GTMP(1) = GSTATE(4)                             | Actualizar GTMP( ) de una sola vez
    GTMP(2) = GSTATE(3)
    GTMP(3) = GSTATE(2)
    GTMP(4) = GSTATE(1)
    do block 43                                     | Bandera de desajuste en
                                                    | la salida = ILLCONDG
    do block 44                                     | Coeficientes del predictor de salida = GPTMP( )
                                                    | Bandera de desajuste en
                                                    | la salida = ILLCONDG
                                                    | Final del procesamiento «una vez por trama»
Go to VEC_LOOP
```