TTC標準
Standard

JT-G728

低遅延符号励振線形予測(LD-CELP)を用いた16kbit/s音声符号化方式

Coding of Speech at 16 kbit/s using Low-Delay Code Excited Linear Prediction (LD-CELP)

第 7.1 版

2007 年 3 月 15 日制定

社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

目　　次

＜参考＞

１．　英文記述の適用レベル

適用レベル：Ｅ３

本標準の付属資料Ｈ，Ｉ，Ｊ，および付録１において、その本文および図表に英文記述を含んでいる。

２．　国際勧告等との関連

本標準の本体、付属資料Ａ，Ｂ，Ｃ，Ｄ，Ｅ，および付録「実現装置の検証」は、加速手続きによる郵便投票により1992年9月に承認されたＩＴＵ－Ｔ勧告Ｇ．７２８に準拠したものである。

本標準の付属資料Ｆは、1994年11月に承認されたＩＴＵ－Ｔ勧告Ｇ．７２８ＡＮＮＥＸ－Ｆに準拠したものである。

本標準の付属資料Ｇは、1994年11月に承認されたＩＴＵ－Ｔ勧告Ｇ．７２８ＡＮＮＥＸ－Ｇ、および2000年2月に承認されたＧ．７２８ＡＮＮＥＸ－Ｇに対するCorrigendum 1に準拠したものである。

本標準の付属資料Ｈは、1997年7月に承認されたＩＴＵ－Ｔ勧告Ｇ．７２８ＡＮＮＥＸ－Ｈ、および1999年5月に承認されたＩＴＵ－Ｔ資料ＣＯＭ１６－Ｒ　４４－ＥのＧ．７２８ＡＮＮＥＸ－Ｈに対するCorrigendumに基づいて改定されたＩＴＵ－Ｔ勧告Ｇ．７２８ＡＮＮＥＸ－Ｈに準拠したものである。

本標準の付属資料Ｉは、1999年5月に承認されたＩＴＵ－Ｔ勧告Ｇ．７２８ＡＮＮＥＸ－Ｉ、および、2004年1月に承認されたＩＴＵ－Ｔ　Implementer's Guide for Ｇ．７２８に準拠したものである。

本標準の付属資料Ｊは、1999年9月に承認されたＩＴＵ－Ｔ勧告Ｇ．７２８ＡＮＮＥＸ－Ｊ、および2006年5月に承認されたＩＴＵ－Ｔ勧告Ｇ．７２８に対するAmendment 1に準拠したものである。

本標準の付録１は、1995年7月に発行されたＩＴＵ－Ｔ勧告Ｇ．７２８ＡＰＰＥＮＤＩＸ－Ｉに準拠したものである。

本標準の付録２は、1995年11月に発行されたＩＴＵ－Ｔ勧告Ｇ．７２８ＡＰＰＥＮＤＩＸ－ＩＩに準拠したものである。

本標準の本体は、2006年4月に承認されたＩＴＵ－Ｔ勧告Ｇ．７２８に対するインプリメンターズガイドに準拠したものである。

３．上記国際勧告等に対する追加項目等

３．１　オプション選択項目

な　し

３．２　ナショナルマター決定項目

な　し

３．３　その他

(1) 本標準の第２版改訂は、1993年3月のＩＴＵ－Ｔ　Ｑ２１／１５のアドホック会合で当課題議長が発行したＧ．７２８の訂正書に準拠したものである。

(2) 本標準は、上記ＩＴＵ－Ｔ勧告に対し、追加した項目はない。

(3) 本標準においては、上記ＩＴＵ－Ｔ勧告に対し、Ａ則ＰＣＭ符号化方式に関する記述を全て削除している。理由は、ＰＣＭ符号化方式としてμ則を採用しているわが国の現状による。

(4) 本標準は、上記ＩＴＵ－Ｔ勧告に対し、変更した項目はない。

(5) 本標準の付属資料Ｇに記述されるアルゴリズムは、内部演算を固定小数点精度で規定したものであり、本文に記述される浮動小数点精度のアルゴリズムと対向接続が可能である。2種類の異なる演算精度のアルゴリズムを相互接続したときの音声品質は受聴試験により保証されている。

３．４　原勧告との章立て構成比較

上記国際勧告との章立ての構成の相違を下表に示す。

TTC 標準	ITU－T 勧告	備　考
本体	本体	
付属資料Ａ～ Ｊ	ANNEX A～J	
付録「実現装置の検証」	APPENDIX　1	
付録 1, 2	APPENDIX　I，II	

４．改版の履歴

版　数	制　定　日	改　版　内　容
第1版	1992年11月26日	制定
第1.1版	1993年 4月27日	工業所有権等に関する注意事項記載
第2版	1993年11月26日	ＩＴＵ－Ｔ　Ｑ２１／１５発行の訂正書による補筆、修正と誤記訂正
第3版	1995年 4月27日	付属資料Ｆ，Ｇの追加および表現の適正化
第4版	1997年11月26日	付属資料Ｈ、付録２，３の追加、および誤記訂正
第5版	1999年11月25日	付属資料Ｉの追加、および付属資料Ｈの記述追加
第6版	2000年 4月20日	付属資料Ｊの追加
第7版	2005年 3月17日	付属資料Ｉの修正、付録１の改定、および付録番号の修正
第7.1版	2007年 3月15日	本体、および付属資料Ｊの修正

５．工業所有権

本標準に関わる「工業所有権の実施の権利に係る確認書」の提出状況は、ＴＴＣホームページでご覧になれます。

６．その他

⑴参照している勧告、標準等

ＴＴＣ標準：　　ＪＴ－Ｇ７１１

ＩＴＵ－Ｔ勧告：（Ｇ．１１３，Ｑ．３５，Ｐ．５０、いずれも付録で参考として引用。）

１．本標準の規定範囲

本標準では、低遅延符号励振線形予測（ＬＤ－ＣＥＬＰ）を用いた16kbit/s音声符号化アルゴリズムについて記述する。本標準は以下の様に構成される。

２章ではＬＤ－ＣＥＬＰアルゴリズムの概要について記述する。３章および４章ではそれぞれＬＤ－ＣＥＬＰ符号器およびＬＤ－ＣＥＬＰ復号器の原理について記述する。５章では各機能ブロックにおける計算方法の詳細について明確にする。付属資料Ａ，Ｂ，ＣおよびＤでは、ＬＤ－ＣＥＬＰアルゴリズムで用いる定数の一覧を示す。付属資料Ｅでは、変数の更新と計算する順序について記述する。付属資料Ｆでは、本標準で使用する略語のアルファベット順リストを示し、付属資料Ｇでは、16kbit/s固定小数点の詳細記述を示す。付属資料Ｈでは、主にDCME用の12.8kbit/s, 9.6kbit/s可変ビットレート動作について記述する。付属資料Ｉでは、ＬＤ－ＣＥＬＰ復号器用フレームおよびパケット消失補償について示す。付録１では、浮動小数点アルゴリズムを実現したものに適用される検証手順に関する情報を示す。また、ＬＤ－ＣＥＬＰアルゴリズムの適用と音声性能ガイドラインについて、それぞれ、付録２と付録３に示す。

２．ＬＤ－ＣＥＬＰの概要

ＬＤ－ＣＥＬＰアルゴリズムは2.1節、2.2節および図２－１／JT-G728にそれぞれ示される符号器および復号器から構成される。

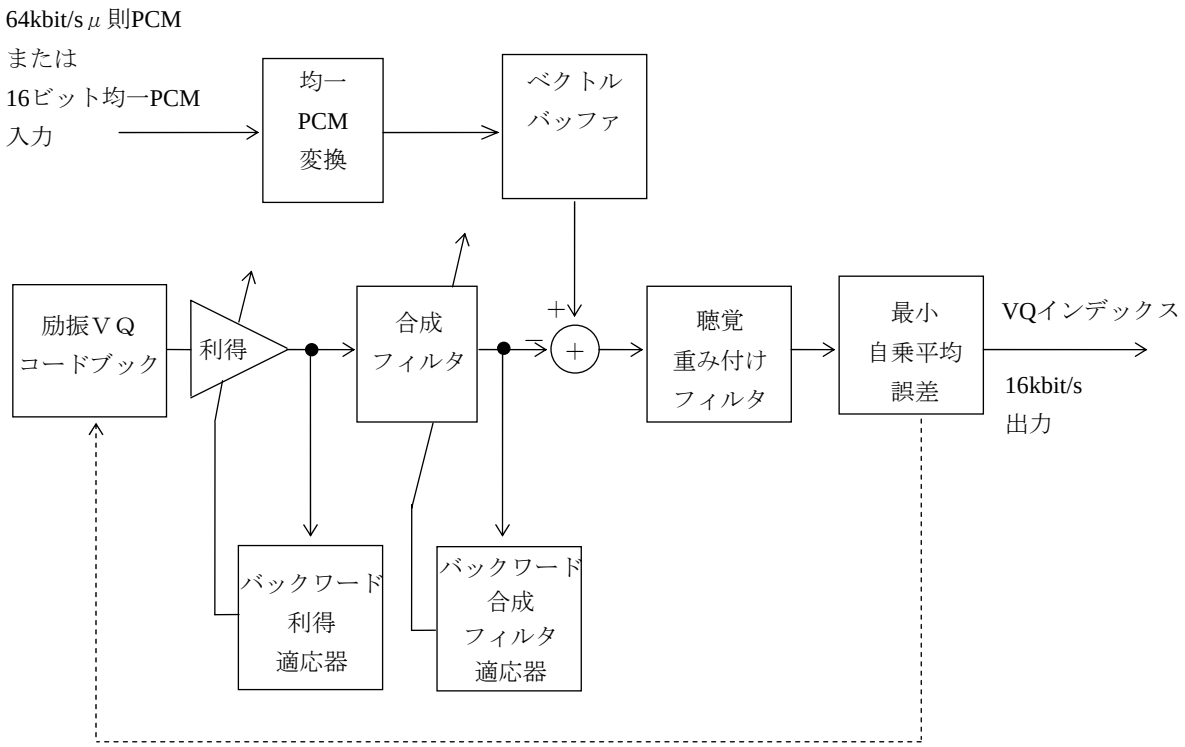
ＬＤ－ＣＥＬＰではＣＥＬＰの基本技術である合成による分析（Ａ－ｂ－Ｓ）法によりコードブックを探索する方法が用いられる。ＬＤ－ＣＥＬＰではアルゴリズムによる遅延を0.625msにするために予測器と利得にバックワード適応を用いている。励振コードブックのインデックスのみが送信される。予測係数は過去の量子化音声をＬＰＣ分析することにより更新される。励振利得は過去の量子化励振信号に含まれている利得情報を用いて更新される。励振ベクトルと利得適応のブロックの大きさはわずか5サンプルである。聴覚重み付けフィルタは非量子化音声をＬＰＣ分析することにより更新される。

２．１　ＬＤ－ＣＥＬＰ符号器

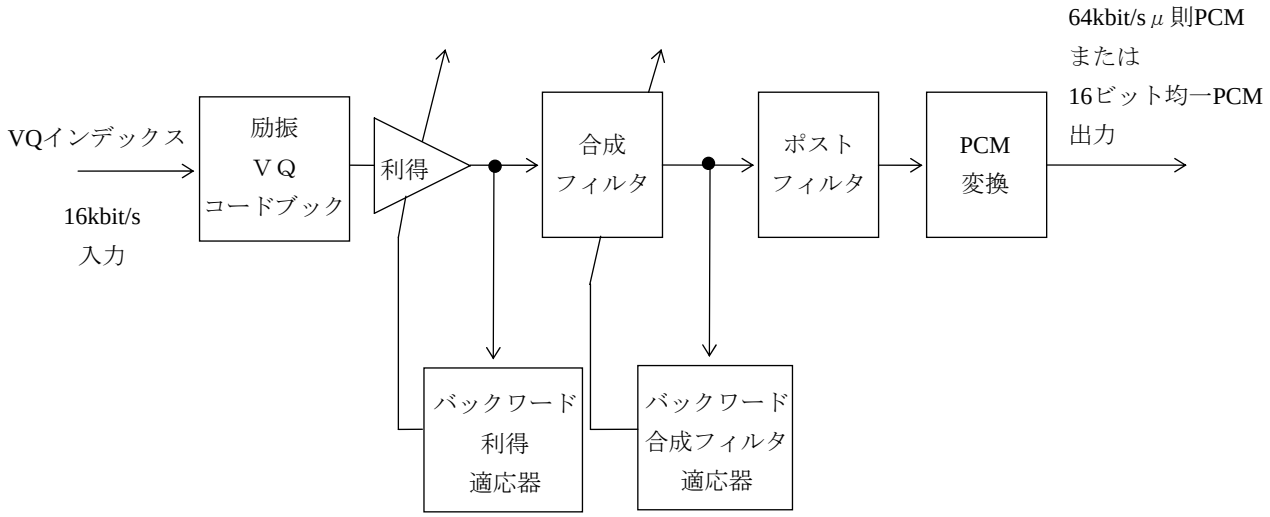
入力信号はμ則ＰＣＭから均一ＰＣＭに変換された後、5個の連続した入力信号サンプルのブロックに分割される。各入力ブロックに対して符号器は1024個のコードブックベクトル候補（励振コードブックに保持されている）を利得調整ユニットおよび合成フィルタに通す。その結果である1024個の量子化信号ベクトル候補の中から、入力信号ベクトルに対して周波数の重み付けをされた自乗平均誤差が最小となるものを決定する。最適量子化信号ベクトルを発生させる最適コードブックベクトル（「コードベクトル」ともいう）に対応した10ビットのコードブックインデックスを復号器に送信する。その後、次の信号ベクトルを符号化する準備としてフィルタメモリを更新するために、最適コードベクトルは利得調整ユニットおよび合成フィルタに通される。合成フィルタの係数および利得は、過去の量子化信号および利得調整された励振信号に基づいたバックワード適応により周期的に更新される。

２．２　ＬＤ－ＣＥＬＰ復号器

復号動作も5サンプルの信号ブロックごとに行われる。受信した10ビットのインデックスに基づき、復号器は励振コードブックから対応するコードベクトルを抽出する。抽出されたコードベクトルはその時点の復号信号ベクトルを発生させるために利得調整ユニットおよび合成フィルタに通される。合成フィルタ係数および利得は符号器と同様の方法で更新される。復号信号ベクトルは聴感上の品質を向上させるために適応ポストフィルタに通される。ポストフィルタ係数は復号器において得られる情報を用いて周期的に更新される。ポストフィルタから出力された5サンプルの信号ベクトルは、5サンプルのμ則ＰＣＭに変換され出力される。



ＬＤ－ＣＥＬＰ符号器



ＬＤ－ＣＥＬＰ復号器

図２－１／JT-G728＊　ＬＤ－ＣＥＬＰの簡略ブロック図
(ITU-T G.728)

３．ＬＤ－ＣＥＬＰ符号器の原理

図３－１／JT-G728は、ＬＤ－ＣＥＬＰ符号器の詳細ブロック図である。図３－１／JT-G728の符号器は、前記の図２－１／JT-G728に示している符号器と数学的に等価であるが、実現する場合、計算する上でより役立つようになっている。

以下の記述の中で

(l) 各変数の記述で、*k*はサンプル番号である。標本化は125μs間隔で行われる。

(2) 与えられた信号の連続する5サンプルのグループを、その信号のベクトルと呼ぶ。
例えば、連続する音声5サンプルは、音声ベクトル、5励振サンプルは、励振ベクトル、等である。

(3) *n*をサンプル番号*k*と区別して、ベクトル番号として用いる。

(4) 4つの連続するベクトルが、1つの適応周期を構成する。また、適応周期をフレームとも記述する。この2つの語を、互換性があるものとして用いる。

励振ベクトル量子化（ＶＱ）コードブックのインデックスだけが、符号器から復号器に伝送される情報である。励振利得、合成フィルタ係数および聴覚重み付けフィルタ係数の3種のパラメータを、周期的に更新する。これらのパラメータは、現在の信号ベクトル以前の信号からバックワード適応方式で導き出す。励振利得は、1ベクトルに1回更新を行うが、合成フィルタ係数と聴覚重み付けフィルタ係数は、4ベクトルに1回更新する（即ち、20サンプル、または2.5msの更新周期である）。このアルゴリズムの処理手順では、更新周期は4ベクトル（20サンプル）であるが、それでもなおバッファサイズは、1ベクトル（5サンプル）だけであることに注意すべきである。この小さなバッファサイズが、片方向遅延が2ms以下を実現することを可能にしている。

符号器の各ブロックについて以下に記述する。ＬＤ－ＣＥＬＰ符号器は、主として音声の符号化に用いられるので、記述の都合上、以後の入力信号は音声と仮定している。しかし、実際には非音声信号も同様に扱える。

３．１　入力ＰＣＭフォーマット変換

このブロックは、入力信号 $s_o(k)$ を均一ＰＣＭ信号 $s_u(k)$ に変換する。

３．１．１　内部均一ＰＣＭレベル

入力信号がμ則を均一ＰＣＭに変換する場合、デバイスにより異なる内部表現が可能である。 例えば、μ則ＰＣＭの標準変換表は、均一で-4015.5から+4015.5の範囲を規定している。この表は、0.5の端数を持った出力値を記載している。これらの端数は、表全体に2を掛けて、全ての数値を整数にしない限り、整数型のデバイスでは表現できない。実際、これは固定小数点ディジタル信号処理（ＤＳＰ）チップで最も普通に行われていることである。他方、浮動小数点ＤＳＰチップでは、表に載せられている数値そのものを表現することができる。この標準の中では、入力信号は-4095から+4095の範囲であると仮定している。固定小数点プロセッサでμ則ＰＣＭ入力信号を変換した場合は、-8031から+8031であるので、符号化を始める前に、1/2しなければならないことを意味する。代案として、これらの数値操作をQ1フォーマットで扱うことができる。即ち小数点の右に1ビットあるものとする。全ての演算は、このビット操作を考慮したデータを用いる。

入力信号が16ビット均一ＰＣＭ信号の場合、全ダイナミックレンジは-32768から+32767である。この入力値はQ3フォーマットで考慮しなければならない。このことは、入力値は8で除算しなければならないことを意味する。復号器出力では、これらの信号に8を乗算し、元に戻す必要がある。

３．２　ベクトルバッファ

このブロックは、5つの連続する音声サンプル $s_u(5n), s_u(5n+1), \ldots, s_u(5n+4)$ をバッファし、5次元の音声ベクトル $s(n)=[s_u(5n), s_u(5n+1), \ldots, s_u(5n+4)]$ を構成する。

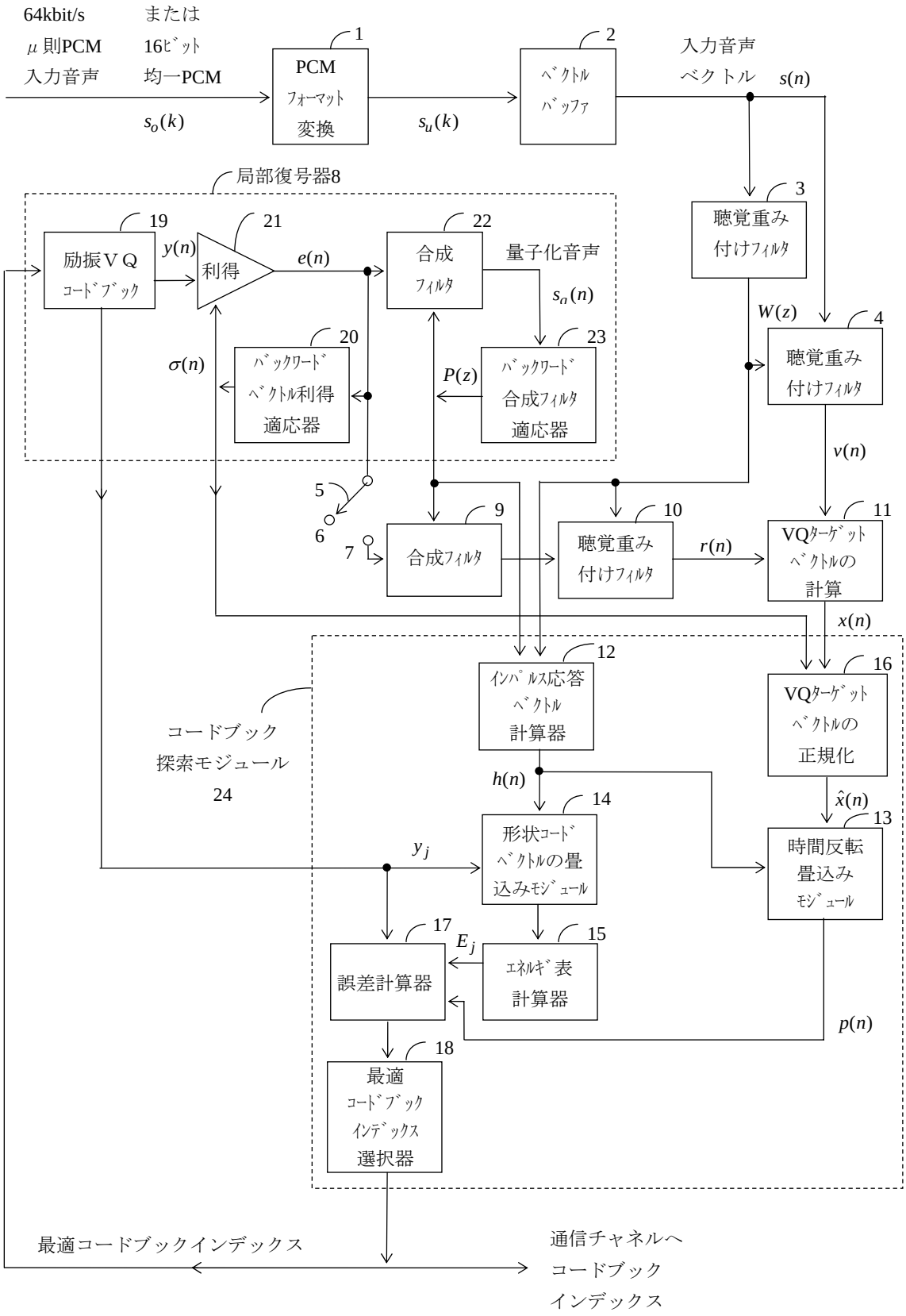


図３－１／JT-G728＊　ＬＤ－ＣＥＬＰ符号器ブロック図
(ITU-T G.728)

３．３　聴覚重み付けフィルタ適応器

聴覚重み付けフィルタ適応器（図３－１／JT-G728のブロック３）の詳細な動作を図３－２、図３－３／JT-G728に示す。この適応器は、非量子化音声の線形予測分析（しばしばＬＰＣ分析と呼ばれる）に基づき、4個の音声ベクトルごとに1回、聴覚重み付けフィルタの係数を計算する。係数の更新は、4ベクトルからなる適応周期の3番目の音声ベクトルで行い、次の更新まで一定に保たれる。

図３－２／JT-G728を説明する。計算を次のように行う。まず、入力（非量子化）音声ベクトルがハイブリッド窓かけモジュール（ブロック３６）を通る。ハイブリッド窓かけモジュールは、過去の音声ベクトルに窓をかけ、窓関数がかけられた音声信号の最初の11個の自己相関係数を計算し、出力する。レビンソン・ダービン再帰法モジュール（ブロック３７）は、これらの自己相関係数を予測係数に変換する。これらの予測係数に基づき、重み付けフィルタ係数計算器（ブロック３８）は、重み付けフィルタの係数を算出する。これら3個のブロックについて、以下で詳細に説明する。

まず、ハイブリッド窓の原理を示す。このハイブリッド窓技法は、3種類のＬＰＣ分析で使用されるので、最初にこの技法の一般的な説明を行い、次に各々の場合について述べる。ここで、ＬＰＣ分析をＬ個の信号サンプルごとに1回行うものとする。現ＬＤ－ＣＥＬＰ適応周期に相当する信号サンプルを、一般的に $s_u(m), s_u(m+1), s_u(m+2), \ldots, s_u(m+L-1)$ とする。バックワード適応ＬＰＣ分析の場合、ハイブリッド窓はmより小さいインデックスを有する全ての信号サンプルに対して適用される（図３－３／JT-G728参照）。ハイブリッド窓関数にＮ個の非巡回サンプルがあるとすると、信号サンプル $s_u(m-1), s_u(m-2), \ldots, s_u(m-N)$ は窓の非巡回部分によって全て重み付けがなされる。 $s_u(m-N-1)$ で始まり、このサンプル（を含む）より過去の全てのサンプルは、窓の巡回部分で重み付けされる。巡回部分の窓関数の値は、 $b, b\alpha, b\alpha^2, \ldots$ である。ここで、 $0<b<1, \quad 0<\alpha<1$ である。

時刻mのとき、ハイブリッド窓関数 $w_m(k)$ を次のように定義する。

$$w_m(k)=\begin{cases} f_m(k)=b\alpha^{-[k-(m-N-1)]}, & k \le m-N-1 \\ g_m(k)=-\sin[c(k-m)], & m-N \le k \le m-1 \\ 0, & k \ge m \end{cases} \tag{1a}$$

また、この窓がかけられた後の信号は次式となる。

$$\begin{aligned} s_m(k) &= s_m(k)w_m(k) \\ &= \begin{cases} s_u(k)f_m(k)=s_u(k)b\alpha^{-[k-(m-N-1)]}, & k \le m-N-1 \\ s_u(k)g_m(k)=-s_u(k)\sin[c(k-m)], & m-N \le k \le m-1 \\ 0, & k \ge m \end{cases} \end{aligned} \tag{1b}$$

種々のハイブリッド窓に対する非巡回部分 $g_m(k)$ および巡回部分 $f_m(k)$ の先頭部分の数値例を付属資料Ａに示す。M次のＬＰＣ分析の場合、i=0,1,2,...,Mに対するM+1個の自己相関係数 $R_m(i)$ を計算する必要がある。現適応周期におけるi番目の自己相関係数は、次式で表現できる。

$$R_m(i)=\sum_{k=-\infty}^{m-1} s_m(k)s_m(k-i)=r_m(i)+\sum_{k=m-N}^{m-1} s_m(k)s_m(k-i) \tag{1c}$$

ここで、

$$r_m(i)=\sum_{k=-\infty}^{m-N-1} s_m(k)s_m(k-i)=\sum_{k=-\infty}^{m-N-1} s_u(k)s_u(k-i)f_m(k)f_m(k-i) \tag{1d}$$

である。

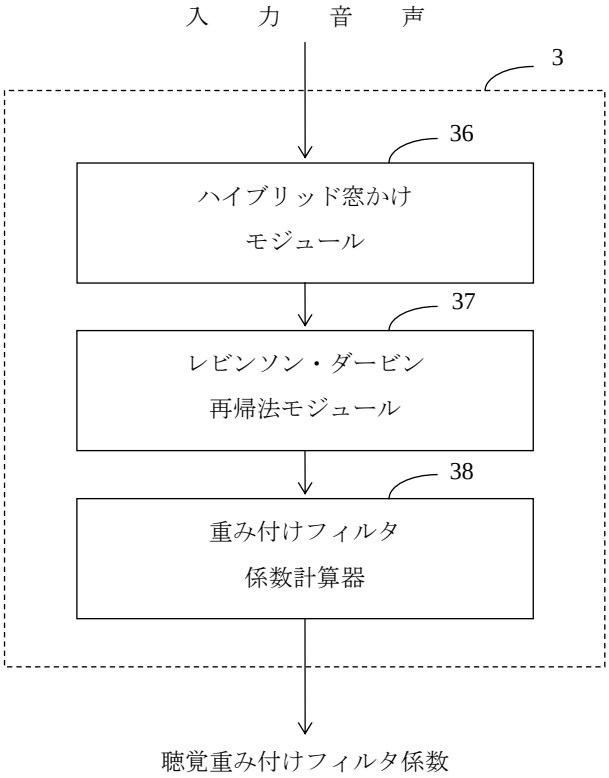


図３－２／JT-G728　聴覚重み付けフィルタ適応器
(ITU-T G.728)

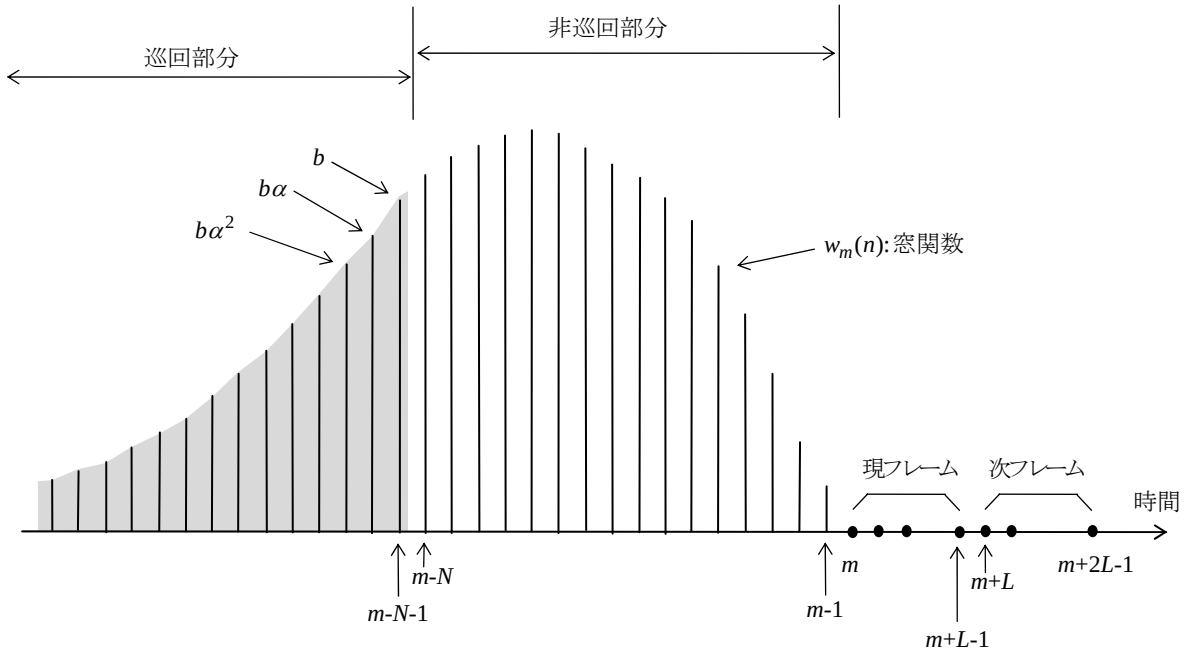


図３－３／JT-G728　　ハイブリッド窓の図解
(ITU-T G.728)

式(1c) の右辺の初項 $r_m(i)$ は $R_m(i)$ の「巡回成分」であり、第2項は「非巡回成分」である。各適応周期において、非巡回成分は有限個の総和を計算するのに対し、巡回成分は再帰的に算出される。その説明を以下で行う。

現適応周期に対する全ての $r_m(i)$ が算出されて格納されており、サンプル $s_u(m+L)$ で始まる次の適応周期に継続して動作するものとする。ハイブリッド窓をL個のサンプルだけ右にシフトさせると、次の適応周期における新しい窓がかけられた信号は次式となる。

$$
\begin{aligned}
&s_{m+L}(k) = s_u(k) w_{m+L}(k) \\
&= \begin{cases} s_u(k) f_{m+L}(k) = s_u(k) f_m(k) \alpha^L, & k \le m+L-N-1 \\ s_u(k) g_{m+L}(k) = -s_u(k) \sin[c(k-m-L)], & m+L-N \le k \le m+L-1 \\ 0, & k \ge m+L \end{cases}
\end{aligned}
\tag{1e}
$$

$R_{m+L}(i)$ の巡回成分は、次式のように書ける。

$$
\begin{aligned}
r_{m+L}(i) &= \sum_{k=-\infty}^{m+L-N-1} s_{m+L}(k) s_{m+L}(k-i) \\
&= \sum_{k=-\infty}^{m-N-1} s_{m+L}(k) s_{m+L}(k-i) + \sum_{k=m-N}^{m+L-N-1} s_{m+L}(k) s_{m+L}(k-i) \\
&= \sum_{k=-\infty}^{m-N-1} s_u(k) f_m(k) \alpha^L s_u(k-i) f_m(k-i) \alpha^L + \sum_{k=m-N}^{m+L-N-1} s_{m+L}(k) s_{m+L}(k-i)
\end{aligned}
\tag{1f}
$$

すなわち、

$$
r_{m+L}(i) = \alpha^{2L} r_m(i) + \sum_{k=m-N}^{m+L-N-1} s_{m+L}(k) s_{m+L}(k-i) \tag{1g}
$$

である。

したがって、$r_{m+L}(i)$ は式(1g)を用いて $r_m(i)$ から再帰的に算出される。この新しく計算された $r_{m+L}(i)$ は、次の適応周期で利用するためメモリに格納される。自己相関係数 $R_{m+L}(i)$ は次式で計算される。

$$
R_{m+L}(i) = r_{m+L}(i) + \sum_{k=m+L-N}^{m+L-1} s_{m+L}(k) s_{m+L}(k-i) \tag{1h}
$$

これまで、ハイブリッド窓計算手順の一般的な原理を説明した。図３－２／JT-G728のハイブリッド窓かけモジュール３６のパラメータ値は、M=10，L=20，N=30，そして、$\alpha = \left(\frac{1}{2}\right)^{\frac{1}{40}} = 0.982820598$，（$\alpha^{2L} = \frac{1}{2}$ より）である。

11個の自己相関係数 $R(i)$，i=0,1,...,10を、上記したハイブリッド窓かけ手順で計算し、次に「白色雑音補正」法を適用する。これは、式(1i)に示すようにエネルギ $R(0)$ を少量増加させることである。

$$
R(0) \leftarrow \left(\frac{257}{256}\right) R(0) \tag{1i}
$$

これは、白色雑音でスペクトルの谷間を埋めることによりスペクトルのダイナミックレンジを減少させ、次段のレビンソン・ダービン再帰法モジュールを良好に動作させる効果がある。257/256の白色雑音補正係数（ＷＮＣＦ）は、平均音声電力から約24dBだけ下がった白色雑音のレベルに相当する。

次に、白色雑音補正後の自己相関係数を用いて、レビンソン・ダービン再帰法モジュール３７は、1次から10次

までの予測係数を再帰的に算出する。i次の予測器のj番目の係数を$a_j^{(i)}$とする。このとき、再帰演算は次のようになる。

$$E(0) = R(0) \tag{2a}$$

$$k_i = -\frac{R(i) + \sum_{j=1}^{i-1} a_j R(i-j)}{E(i-1)} \tag{2b}$$

$$a_i^{(i)} = k_i \tag{2c}$$

$$a_j^{(i)} = a_j^{(i-1)} + k_i a_{i-j}^{(i-1)}, \qquad 1 \le j \le i-1 \tag{2d}$$

$$E(i) = (1 - k_i^2) E(i-1) \tag{2e}$$

式(2b) から式(2e) は、i=1,2,...,10に対して繰り返し計算され、最終的な解は、次式で与えられる。

$$q_i = a_i^{(10)}, \qquad 1 \le i \le 10 \tag{2f}$$

$q_0 = 1$と定義した場合、10次の「予測誤差フィルタ」（「分析フィルタ」とも呼ばれる）は、次式の伝達関数を有する。

$$\tilde{Q}(z) = \sum_{i=0}^{10} q_i z^{-i} \tag{3a}$$

また、対応する10次の線形予測器は、次式の伝達関数で定義される。

$$Q(z) = -\sum_{i=1}^{10} q_i z^{-i} \tag{3b}$$

重み付けフィルタ係数計算器（ブロック３８）は、次式により聴覚重み付けフィルタ係数を計算する。

$$W(z) = \frac{1 - Q(z/\gamma_1)}{1 - Q(z/\gamma_2)}, \qquad 0 < \gamma_2 < \gamma_1 \le 1 \tag{4a}$$

$$Q(z/\gamma_1) = -\sum_{i=1}^{10} (q_i \gamma_1^i) z^{-i} \tag{4b}$$

$$Q(z/\gamma_2) = -\sum_{i=1}^{10} (q_i \gamma_2^i) z^{-i} \tag{4c}$$

聴覚重み付けフィルタは、式(4a) の伝達関数$W(z)$で定義される10次の極零フィルタである。γ_1とγ_2の値は、それぞれ0.9と0.6である。

ここで図３－１／JT-G728を参照されたい。聴覚重み付けフィルタ適応器（ブロック３）は、式(2)から式(4) に従い$W(z)$の係数を周期的に更新し、その係数をインパルス応答ベクトル計算器（ブロック１２）と聴覚重み付け

フィルタ（ブロック４と１０）に与える。

３．４　聴覚重み付けフィルタ

図３－１／JT-G728において、現在の入力音声ベクトル $s(n)$ は、聴覚重み付けフィルタ（ブロック４）を通過し、重み付けされた音声ベクトル $v(n)$ となる。初期化中を除きこのフィルタメモリ（すなわち、内部状態変数、またはフィルタの遅延ユニットに格納された変数）は、いかなるときにも零にリセットしてはならない。一方、聴覚重み付けフィルタ（ブロック１０）のメモリは後述されるような特別な取り扱いが必要とされる。

３．４．１　非音声動作

モデム信号あるいは、他の非音声信号に関して、ＩＴＵ－Ｔでの試験結果では聴覚重み付けフィルタを切ることが望ましい。

このことは、$W(z)=1$ に設定することに等しい。これは、式(4a) の γ_1 と γ_2 を零に設定することで最も簡単に実現できる。音声モードにおけるそれらの変数の通常値は、それぞれ0.9と0.6である。

３．５　合成フィルタ

図３－１／JT-G728において、同一の係数を持つ2つの合成フィルタ（ブロック９と２２）がある。両方のフィルタは、バックワード合成フィルタ適応器（ブロック２３）によって更新される。個々の合成フィルタは、フィードバックのパスに50次のＬＰＣ予測器を持つフィードバックループから構成される50次の全極フィルタである。合成フィルタの伝達関数は $F(z)=1/[1-P(z)]$ である。但し、$P(z)$ は50次ＬＰＣ予測器の伝達関数である。

重み付けされた音声ベクトル $v(n)$ が得られた後、合成フィルタ（ブロック９）と聴覚重み付けフィルタ（ブロック１０）を用いて、零入力応答ベクトル $r(n)$ が生成される。これを実現するために、まずスイッチ５を開く、即ちノード６に接続する。これは、ノード７から合成フィルタ９に行く信号が零となることを意味する。

そして合成フィルタ９と聴覚重み付けフィルタ１０を5サンプル（1ベクトル）の間だけ動作させる。これは、ノード７に零信号を与えながら5サンプル期間フィルタ動作を継続させることを意味する。この結果、聴覚重み付けフィルタ１０の出力は、零入力応答ベクトル $r(n)$ となる。

初期化直後のベクトルを除き、フィルタ９と１０のメモリは通常、零でないことに注意すべきである。従って、ノード７からのフィルタ入力が零であっても、その出力ベクトル $r(n)$ も同様に通常零でない。事実上、このベクトル $r(n)$ は過去の利得調整された励振ベクトル $e(n-1), e(n-2), \ldots$ に対する2つのフィルタの応答である。このベクトルは実際に時刻 $(n-1)$ までのフィルタメモリによる効果に相当する。

３．６　ＶＱターゲットベクトルの計算

このブロックはＶＱコードブック探索ターゲットベクトル $x(n)$ を得るために重み付けされた音声ベクトル $v(n)$ から零入力応答ベクトル $r(n)$ を減算する。

３．７　バックワード合成フィルタ適応器

この適応器２３は、合成フィルタ９と２２の係数を更新する。それは、入力として、量子化された（合成された）音声を取り込み、出力として1組の合成フィルタ係数を生成する。この動作は、聴覚重み付けフィルタ適応器３と極めて類似している。

この適応器の詳細を図３－４／JT-G728に示す。ハイブリッド窓かけモジュール４９とレビンソン・ダービン再帰法モジュール５０の動作は、次に示す3つの相違点を除き図３－２／JT-G728の対応する部分（３６と３７）と全く同一である。

(1)　入力信号は、入力音声そのものではなく、ここでは量子化された音声である。

(2)　予測次数は、10ではなく、50である。

(3)　ハイブリッド窓のパラメータが異なる。 $N=35, \quad \alpha=(3/4)^{1/40}=0.992833749$

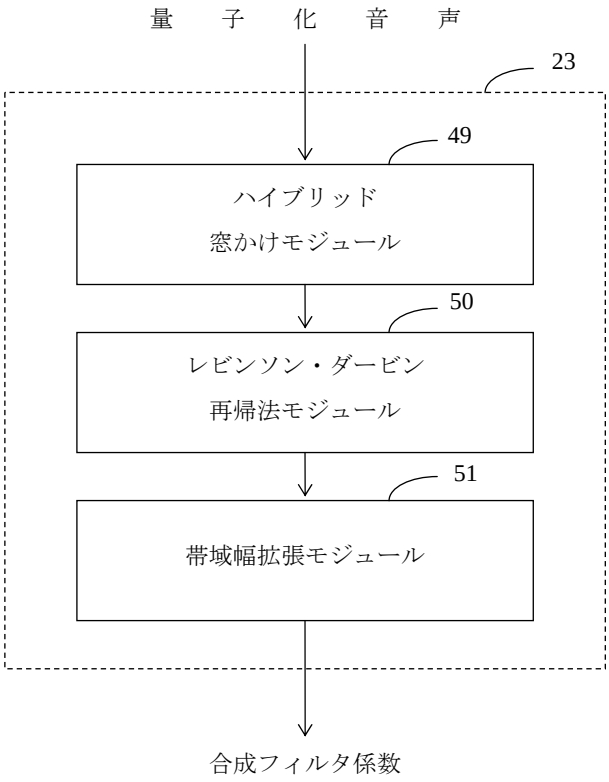


図３－４／JT-G728　バックワード合成フィルタ適応器
(ITU-T G.728)

更新周期はやはりＬ＝20であり、白色雑音補正係数はやはり257/256=1.00390625であることに注意すること。

50次ＬＰＣ予測器の伝達関数を $\hat{P}(z)$ とすると、 $\hat{P}(z)$ は次式で示される。

$$\hat{P}(z) = -\sum_{i=1}^{50} \hat{a}_i z^{-i} \tag{5}$$

但し、$\hat{a}_i$ は予測係数である。伝送誤りに対する耐性を向上させるためにＬＰＣスペクトラムのピークがわずかに広い帯域幅を持つようにこれらの係数は修正される。帯域幅拡張モジュール５１は以下の方法で帯域幅拡張の手順を実行する。ＬＰＣ予測係数 $\hat{a}_i$ が与えられると、新しい1組の係数 a_i が次式により計算される。

$$a_i = \lambda^i \hat{a}_i, \qquad i = 1, 2, \ldots, 50 \tag{6}$$

但し、λ は次式で与えられる。

$$\lambda = 253/256 = 0.98828125 \tag{7}$$

これは、合成フィルタの全ての極を原点に向かって半径方向に λ 倍だけ動かす効果を持つ。極が単位円から離れるため、周波数応答のピークが広げられる。

この様な帯域幅拡張のあと、修正されたＬＰＣ予測器は次の伝達関数を持つ。

$$P(z) = -\sum_{i=1}^{50} a_i z^{-i} \tag{8}$$

修正された係数は、それから合成フィルタ９と２２に与えられる。これらの係数はインパルス応答ベクトル計算器１２にも与えられる。

合成フィルタ９と２２は両方共、次の伝達関数をもつ。

$$F(z) = \frac{1}{1 - P(z)} \tag{9}$$

聴覚重み付けフィルタと同様に合成フィルタ９と２２もまた4ベクトルからなる適応周期毎に1回更新され、その更新時期は4ベクトル中の第3音声ベクトルである。しかしながら、更新は前適応周期の最後のベクトルまでの量子化された音声に基づいている。言い換えれば、更新が行われるまでに2ベクトルの遅延が生じている事になる。これは、レビンソン・ダービン再帰法モジュール５０とエネルギ表計算器１５（後述）に計算の負荷が集中しているためである。結果として、前フレームの量子化された音声の自己相関が各4ベクトル周期の第1ベクトルで求まったとしても、計算は1ベクトル以上の時間を必要とすることが考えられる。従って、1ベクトルの基本バッファサイズを保つため（符号化遅延を短くするため）そして、実時間動作を保持するために、フィルタ更新での2ベクトル分の遅延を導入して装置の実現を容易にしている。

３．８　バックワードベクトル利得適応器

本適応器は、ベクトル番号n毎に励振利得 $\sigma(n)$ を更新する。励振利得 $\sigma(n)$ は、選択された励振ベクトル $y(n)$ のスケーリングファクタ（利得調整係数）である。適応器２０は、利得調整された励振ベクトル $e(n)$ を入力とし、励振利得 $\sigma(n)$ を出力する。基本的に、 $e(n-1), e(n-2), \ldots$ の利得を基に対数領域で適応線形予測を用いることにより $e(n)$ の利得の「予測」が行われる。このバックワードベクトル利得適応器２０の詳細を図３－５／JT-G728に示す。

図３－５／JT-G728を参照し、本利得適応器の動作を以下に示す。1ベクトル遅延ユニット６７は、直前の利得調整された励振ベクトル $e(n-1)$ を保持する。実効値（ＲＭＳ）計算器３９は、ベクトル $e(n-1)$ のＲＭＳ値を計算する。次に対数計算器４０は、常用対数を計算した後、20を乗算することにより $e(n-1)$ のＲＭＳ値のdB値を計算する。

図３－５／JT-G728の対数利得オフセット値保持器４１には、対数利得オフセット値である32dBが格納される。この値は、有声音での平均励振利得レベル(dB値)におおよそ等しい。加算器４２は、対数計算器４０より出力された対数利得からこの対数利得オフセット値を減算する。この結果得られるオフセットが除去された対数利得 $\delta(n-1)$ は、ハイブリッド窓かけモジュール４３とレビンソン・ダービン再帰法モジュール４４で使用される。また、モジュール４３と４４は、ハイブリッド窓のパラメータの値が異なること、及び分析される信号が入力音声ではなくオフセットが除去された対数利得であることを除いて、聴覚重み付けフィルタ適応モジュール（図３－２／JT-G728）中のブロック３６と３７と全く同じ動作をする（5音声サンプル毎に1個の利得値だけが求められることに注意）。ブロック４３のハイブリッド窓のパラメータは、 $M = 10, N = 20, L = 4,$ $\alpha = (3/4)^{1/8} = 0.96467863$ である。

レビンソン・ダービン再帰法モジュール４４の出力は、次式で伝達関数が与えられる10次の線形予測器の係数である。

$$\hat{R}(z) = -\sum_{i=1}^{10} \hat{\alpha}_i z^{-i} \tag{10}$$

帯域幅拡張モジュール４５は、図３－４／JT-G728のモジュール５１と同様な方法で、この多項式の根を単位円の半径に沿って放射状にｚ平面の原点方向に移動させる。この結果、帯域幅拡張された利得予測器の伝達関数は、次のようになる。

$$R(z) = -\sum_{i=1}^{10} \alpha_i z^{-i} \tag{11}$$

ここで、係数 α_i は、次式で計算される。

$$\alpha_i = (29/32)^i \hat{\alpha}_i = (0.90625)^i \hat{\alpha}_i \tag{12}$$

このような帯域幅拡張は、利得適応器（図３－１／JT-G728のブロック２０）の伝送誤りに対する耐性を向上させる。これらの係数 α_i は、対数利得線形予測器（図３－５／JT-G728のブロック４６）の係数として使用される。

この予測器４６は、4音声ベクトル毎に更新され、更新は4ベクトルの適応周期の第2音声ベクトルで行われる。予測器は、$\delta(n-1), \delta(n-2), \ldots, \delta(n-10)$ の線形結合を基に $\delta(n)$ を予測する。予測された $\delta(n)$ を $\hat{\delta}(n)$ とすると、$\hat{\delta}(n)$ は次式で与えられる。

$$\hat{\delta}(n) = -\sum_{i=1}^{10} \alpha_i \delta(n-i) \tag{13}$$

対数利得線形予測器４６により $\hat{\delta}(n)$ が算出された後、ブロック４１に格納されている対数利得オフセット値32dBを $\hat{\delta}(n)$ に加え戻す。対数利得リミッタ４７は、その結果得られた対数利得値を調べ、値が異常に大きかったり、小さかったりした場合、その値を制限する。リミッタの下限は0dBに、上限は60dBに各々設定されている。利得リミッタの出力は、対数計算器４０と逆の作用をする逆対数計算器４８に入力され、対数値（dB値）から線形領域に変換される。利得リミッタは、利得が線形領域で1から1000の間の値をとる事を保証する。

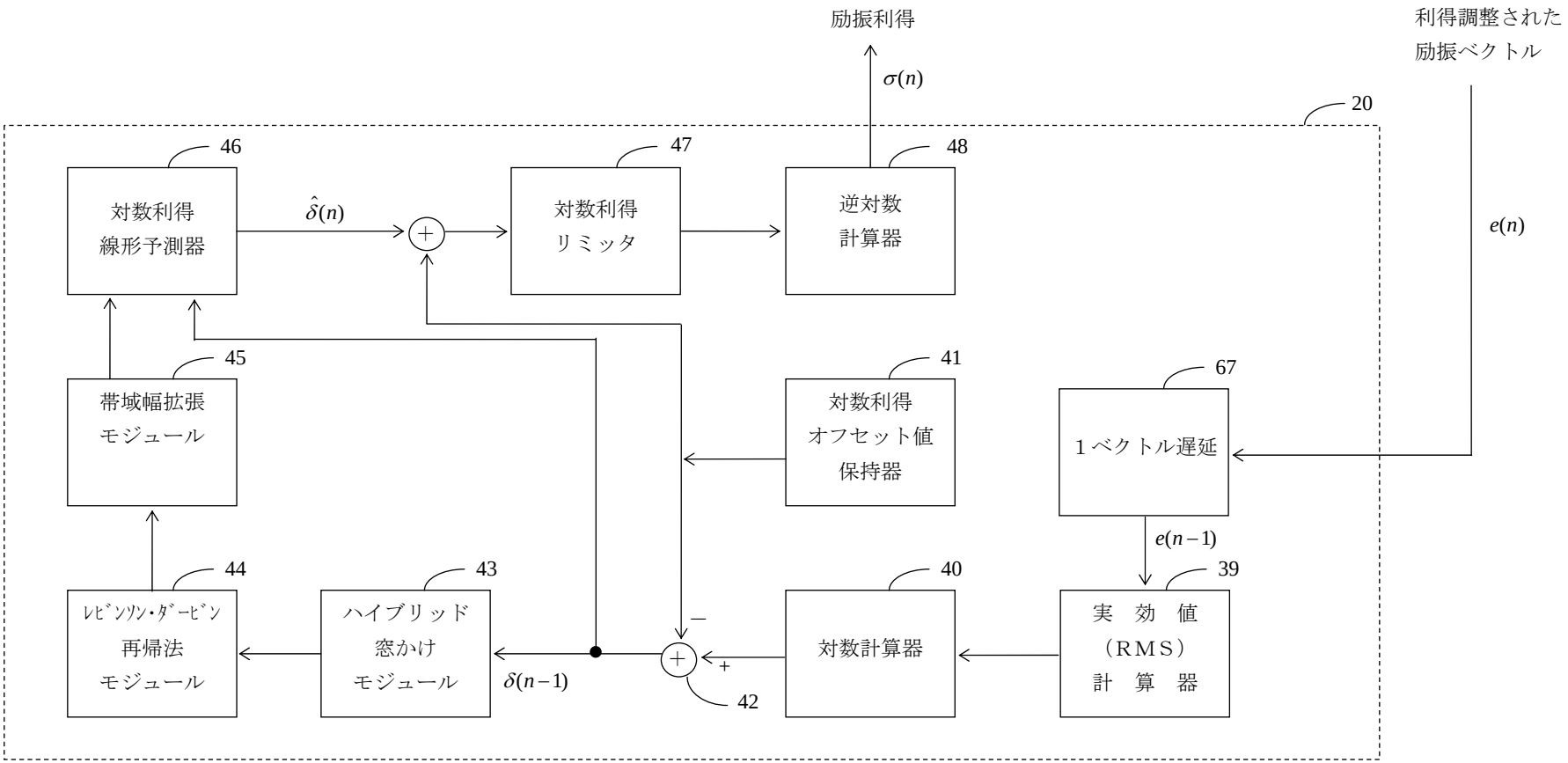


図３－５／JT-G728　バックワードベクトル利得適応器
(ITU-T G.728)

３．９　コードブック探索モジュール

図３－１／JT-G728のブロック１２から１８は、コードブック探索モジュール２４を構成している。本モジュールは、励振ＶＱコードブック１９に格納されている1024個の候補コードベクトルを探索し、入力音声ベクトルに最も近い量子化音声ベクトルを与える最適コードベクトルのインデックスを確定する。

コードブック探索に要する演算量を低減するため、10ビット、1024ベクトルを有するコードブックは、2つの小さなコードブック、即ち128個の独立したコードベクトルを有する7ビットの「形状コードブック」と、0を中心に対称な8個のスカラー値を有する3ビット（即ち、1ビットは正、負の符号を示し、残りの2ビットで絶対値を表す）の「利得コードブック」に分割される。最終的な出力コードベクトルは、最適形状コードベクトル（7ビットの形状コードブックから選定される）と最適利得レベル（3ビットの利得コードブックから選定される）の積である。7ビット形状コードブックの表、及び3ビット利得コードブックの表を付属資料Ｂに示す。

３．９．１　コードブック探索の原理

原理的には、コードブック探索モジュール２４は、1024個の候補コードベクトル各々を現時点の励振利得 $\sigma(n)$ によりスケーリングし、その結果得られた1024個の利得調整されたベクトルを1個ずつ、合成フィルタ $F(z)$ と聴覚重み付けフィルタ $W(z)$ からなる縦続接続されたフィルタによりフィルタリングする。伝達関数 $H(z) = F(z)W(z)$ で表される縦続接続されたフィルタのフィルタメモリは、本モジュールにコードベクトルが入力される度に逐一零に初期化される。

ＶＱコードベクトルのフィルタリングは、行列とベクトルの乗算によって表現できる。y_j を7ビット形状コードブックの中のj番目のコードベクトル、g_i を3ビット利得コードブック中のi番目のレベルとする。$\{h(n)\}$ を縦続接続されたフィルタのインパルス応答列とする。そのとき、コードブックインデックスiとjによって与えられるコードベクトルが縦続接続されたフィルタ $H(z)$ に入力された場合、フィルタ出力は、次のように表される。

$$\tilde{x}_{ij} = \mathbf{H}\sigma(n) g_i y_j \tag{14}$$

ここで、

$$\mathbf{H} = \begin{bmatrix} h(0) & 0 & 0 & 0 & 0 \\ h(1) & h(0) & 0 & 0 & 0 \\ h(2) & h(1) & h(0) & 0 & 0 \\ h(3) & h(2) & h(1) & h(0) & 0 \\ h(4) & h(3) & h(2) & h(1) & h(0) \end{bmatrix} \tag{15}$$

である。

コードブック探索モジュール２４は、次に示す自乗平均誤差（ＭＳＥ）を最小にするインデックスiとjの最適な組合せを探索する。

$$D = \left\| x(n) - \tilde{x}_{ij} \right\|^2 = \sigma^2(n) \left\| \hat{x}(n) - g_i \mathbf{H} y_j \right\|^2 \tag{16}$$

ここで、$\hat{x}(n) = x(n)/\sigma(n)$ は、利得で正規化されたＶＱターゲットベクトルである。上式を展開すると、次式が得られる。

$$D = \sigma^2(n)[\left\| \hat{x}(n) \right\|^2 - 2 g_i \hat{x}^T(n) \mathbf{H} y_j + g_i^2 \left\| \mathbf{H} y_i \right\|^2] \tag{17}$$

$\left\| x(n) \right\|^2$ の値と $\sigma^2(n)$ の値は、コードブック探索の間固定されているので、Dの i, j に関する最小化は、次式を i, j に関して最小化することに等しい。

$$\hat{D} = -2g_i p^T(n) y_j + g_i^2 E_j \tag{18}$$

ここで、

$$p(n) = \mathbf{H}^T \hat{x}(n) \tag{19}$$

$$E_j = \left\| \mathbf{H} y_j \right\|^2 \tag{20}$$

である。

ここで留意すべきことは、E_j は縦続接続されたフィルタを通過したj番目の形状コードベクトルのエネルギであり、ＶＱターゲットベクトル $\hat{x}(n)$ には依存しないことである。また、形状コードベクトル y_j は定数であり、行列Ｈは合成フィルタ及び聴覚重み付けフィルタのみに依存し、この2つのフィルタ係数は、1適応周期（4音声ベクトル周期）の間は一定である。従って、E_j も4音声ベクトル周期の間は一定である。以上の事実に基づき、2つのフィルタが更新される時、128個全てのエネルギ項 E_j, $j = 0,1,2,\ldots,127$ （128個の形状コードベクトルに対応する）を計算し、格納しておくことが可能であり、次の4音声ベクトル周期の間、コードブック探索のために繰り返しこれらのエネルギ項を使用することができる。この方法により、コードブック探索に要する演算量が低減される。

さらに演算量を低減するために、次の2つの配列についてもあらかじめ計算し、格納しておくことが可能である。

$$b_i = 2g_i \tag{21}$$

$$c_i = g_i^2 \tag{22}$$

$$i = 0,1,\ldots,7$$

g_i が定数であるため、この2つの配列も定数となる。b_i と c_i を用いて $\hat{D}$ は次のように表される。

$$\hat{D} = -b_i P_j + c_i E_j \tag{23}$$

ここで、$P_j = p^T(n) y_j$ である。

一度、E_j, b_i 及び c_i の表があらかじめ計算され、格納されていれば、jのみに依存する内積項 $P_j = p^T(n) y_j$ の計算が $\hat{D}$ を求めるために必要な演算の大部分を占める。従って、コードブック探索の処理手順としては、先ず形状コードブックに関する処理を行い P_j と E_j を計算し、次の各形状コードベクトル y_j に対する最適な利得インデックス i の確定を行う。

与えられた形状コードベクトル y_j に対する最適利得インデックス i の探索方法には、以下の数通りの方法がある。

(l)　最初に最も明白な方法として、8通りの i に対応する8通りの $\hat{D}$ を求め、$\hat{D}$ を最小化するインデックス i を選ぶという方法がある。しかし、この方法は、各 i に対し2回の乗算を必要とする。

(2)　第2の方法は、先ず最適な利得 $\hat{g} = P_j / E_j$ を最初に計算し、次にこの $\hat{g}$ を3ビットの利得コードブック中の8個の利得レベル $\{g_0,\ldots,g_7\}$ のうちの1個に量子化するという方法である。最適なインデックス i は、$\hat{g}$ に最も近い利得レベルを与える g_i のインデックスである。しかし、この方法は、各128個の形状コードベク

トルに対し、1回の除算処理を必要とする。この除算は、ＤＳＰチップを使用して実現する際に効率を落とす典型的な演算である。

(3)　第3の方法は、第2の方法を僅かに変更したものであり、ＤＳＰチップによる実現に対しては特に効率的な方法である。$\hat{g}$ の量子化は、隣接する利得レベル間の中間点である「量子化セルの境界」と、$\hat{g}$ との大小比較を繰り返すことによって実現できる。d_i を同じ符号を有する利得レベル g_i と g_{i+1} の中間点とする。そのとき、「$\hat{g} < d_i$ ？」をテストすることは、「$P_j < d_i E_j$ ？」をテストすることに等しい。従って、後者のテスト方法を使用することにより、除算処理を避けることができ、各インデックス i 毎に必要な演算は乗算1回のみとなる。これが、コードブック探索で使用される方法である。利得量子化セルの境界 d_i は定数であるため、あらかじめ計算し、表として格納しておくことが可能である。8個の利得レベルに対しては、実際には、$d_0, d_1, d_2, d_4, d_5, d_6$ の6個の境界値が用いられる。

最適インデックス i と j が確定した後、それらは連結されて、コードブック探索モジュールの出力である単一の10ビット最適コードブックインデックスとなる。

３．９．２　コードブック探索モジュールの演算

図３－１／JT-G728を参照し、先に示したコードブック探索の原理に基づくコードブック探索モジュール２４の動作について以下に述べる。合成フィルタ９と聴覚重み付けフィルタ１０が更新される毎にインパルス応答ベクトル計算器１２は、縦続接続されたフィルタ $F(z)W(z)$ のインパルス応答の最初の5サンプルを計算する。インパルス応答ベクトルを計算するため、最初に、縦続接続されたフィルタのメモリに零を設定し、次に入力列 $\{1, 0, 0, 0, 0\}$ でフィルタを励振する。この時のフィルタ出力の5サンプルが、ここで求めようとしているインパルス応答ベクトル成分 $h(0), h(1), h(2), h(3), h(4)$ に相当する。フィルタ９と１０が再び更新されるまで、計算されたインパルス応答ベクトルは定数として保持され、後続する4音声ベクトルのコードブック探索において使用される。

次に形状コードベクトル畳込みモジュール１４は、128個のベクトル $\mathbf{H}y_j,\quad j = 0, 1, 2, \ldots, 127$ を計算する。言い換えると、それぞれの形状コードベクトル $y_j,\quad j = 0, 1, 2, \ldots, 127$ とインパルス応答列 $h(0), h(1), \ldots, h(4)$ の畳込みを行う。ここでの畳込みは、最初の5サンプルに対してのみ行われる。その結果得られた128個のベクトルのエネルギは、エネルギ表計算器１５により式(20) を用いて計算され、格納される。ベクトルのエネルギは、ベクトルの各成分の自乗和として定義される。

ここで留意すべきことは、まずコードベクトル探索モジュールにおいて、ブロック１２，１４及び１５における計算は、4音声ベクトル周期に1回だけ実行されるのに対し、他のブロックでは、1音声ベクトル毎に計算が実行されることである。さらに、E_j テーブルの更新は、合成フィルタ係数の更新に同期して行われることである。即ち、新しい E_j テーブルは、それぞれの適応周期の第3音声ベクトルから使用が開始される（３．７節を参照）。

ＶＱターゲットベクトル正規化モジュール１６は、利得で正規化されたＶＱターゲットベクトル $\hat{x}(n) = x(n) / \sigma(n)$ を計算する。ＤＳＰチップを使用して実現する場合は、最初に $1/\sigma(n)$ を計算し、次に $\hat{x}(n)$ の各成分に $1/\sigma(n)$ を乗算すれば、より効率的である。

次に、時間反転畳込みモジュール１３は、ベクトル $p(n) = \mathbf{H}^T \hat{x}(n)$ を計算する。

この演算は、最初に $\hat{x}(n)$ の成分の順番を反転し、その結果とインパルス応答ベクトルを畳込んだ後、出力されたベクトルの成分の順番を再び反転することに等しい（これを「時間反転畳込み」という）。

一度、E_j, b_i 及び c_i のテーブルがあらかじめ計算されて格納され、かつ、ベクトル $p(n)$ もまた計算されたならば、誤差計算器１７と最適コードブックインデックス選択器１８は、次に示す効率的なコードブック探索アルゴリズムを互いに実行する。

(1)　$\hat{D}_{\min}$ を $\hat{D}$ のとりうる最大値より大きな値に初期設定する（または、ＤＳＰチップで表現できる最大の値を使用する）。

(2)　形状コードブックのインデックス $j = 0$ と設定する。

(3)　内積 $P_j = p^T(n) y_j$ を計算する。

(4)　$P_j < 0$ ならば、負の利得を探索するためステップ（8）に移る。そうでなければ、正の利得を探索するためステップ(5) に進む。

(5)　$P_j < d_0 E_j$ ならば、$i = 0$ とし、ステップ(11)に移る。そうでなければ、ステップ(6)に進む。

(6) $P_j < d_1 E_j$ ならば、$i = 1$ とし、ステップ(11)に移る。そうでなければ、ステップ(7)に進む。

(7) $P_j < d_2 E_j$ ならば、$i = 2$ とし、ステップ(11)に移る。そうでなければ、$i = 3$ とし、ステップ(11)に進む。

(8) $P_j > d_4 E_j$ ならば、$i = 4$ とし、ステップ(11)に移る。そうでなければ、ステップ(9)に進む。

(9) $P_j > d_5 E_j$ ならば、$i = 5$ とし、ステップ(11)に移る。そうでなければ、ステップ(10)に進む。

(10) $P_j > d_6 E_j$ ならば、$i = 6$ とする。そうでなければ、$i = 7$ とする。

(11) $\hat{D} = -b_i P_j + c_i E_j$ を計算する。

(12) $D < \hat{D}_{\min}$ ならば、$\hat{D}_{\min} = D$，$i_{\min} = i$，及び $j_{\min} = j$ とする。

(13) $j < 127$ ならば、$j = j + 1$ とし、ステップ(3) に移る。そうでなければ、ステップ(14) へ進む。

(14) アルゴリズムがここまで進行した時、利得レベルと形状コードベクトル全ての組合せ1024通りについて探索が終了したことになる。その結果得られた $i_{\min}$ と $j_{\min}$ は、それぞれ、利得と形状についての最適伝送インデックスである。符号器が出力する最適コードブックインデックス（10ビット）は、これら2つのインデックスを連結したものであり、これに対応する最適な励振コードベクトルは、$y(n) = g_{i\min} y_{j\min}$ である。選択された10ビットのコードブックインデックスは、通信チャネルを介して復号器に伝送される。

３．１０　局部復号器

前述までの処理で、符号器が最適コードブックインデックスを選択し送信したが、次の音声ベクトルの符号化の準備としてさらに以下の処理が実行される。

最初に、最適コードブックインデックスは、相当する最適励振コードベクトル $y(n) = g_{i\min} y_{j\min}$ を取り出すために励振ＶＱコードブックに供給される。この最適励振コードベクトルは、利得ブロック２１で現時点の励振利得 $\sigma(n)$ によりスケーリングされる。この結果、利得調整された励振ベクトルは、$e(n) = \sigma(n) y(n)$ となる。

この励振ベクトル $e(n)$ は、現時点の量子化音声ベクトル $s_q(n)$ を得るため、合成フィルタ２２に通される。ブロック１９から２３までが、局部復号器８を形成していることに注意すること。従って、伝送誤りがない場合、量子化音声ベクトル $s_q(n)$ は、実際に復号された音声ベクトルと等しくなる。図３－１／JT-G728において、バックワード合成フィルタ適応器２３は、合成フィルタ係数を更新するため、この量子化音声ベクトル $s_q(n)$ を必要とする。同様に、バックワード利得適応器２０は、対数利得線形予測係数を更新するため、利得調整された励振ベクトル $e(n)$ を必要とする。

次の音声ベクトルの符号化を開始する前処理として、合成フィルタ９と聴覚重み付けフィルタ１０のメモリ内容を更新する。この手順として、まず３．５節で記述した零入力応答計算の結果として残っているフィルタ９及び、１０のメモリ内容を保存する。次にフィルタ９及び、１０のメモリを零に設定しスイッチ５を閉じる。すなわち、ノード７に接続する。そして、利得調整された励振ベクトル $e(n)$ を、メモリ内容が零のフィルタ９及び１０に通す。$e(n)$ はわずか5サンプルであり、フィルタのメモリ内容が零であるので、積和演算回数は、5サンプル期間でわずか0回から4回増加するだけである。フィルタのメモリ内容が零でなければ、サンプル当たり70回の積和演算が増加することを考慮すると、計算量の大幅な削減になっている。次に、$e(n)$ をフィルタリングした後の新たなフィルタのメモリ内容に、保存していた元のフィルタのメモリ内容を加算する。これは、実質的にフィルタ９及び、１０の零入力応答に零状態応答を加えたものになる。結果として、次の音声ベクトル符号化の零入力応答計算過程で使用する望ましいフィルタのメモリ内容となる。

フィルタのメモリ内容の更新後、合成フィルタ９の先頭から5個のメモリ内容は所望の量子化音声ベクトル $s_q(n)$ と完全に一致している。従って、合成フィルタ２２を実際に省略し、合成フィルタ９の更新されたメモリ内容から $s_q(n)$ が得られる。これは、更にサンプル当たり50回の積和演算の削減を意味している。

単一の入力音声ベクトルの符号化方法として、符号器の動作を記述した。音声波形全体に対する符号化は、総ての音声ベクトルに対しこれまでに記述した処理を繰り返すことにより達成される。

３．１１　同期並びに帯域内シグナリング

符号器の記述において、復号器では10ビットの受信コードブックインデックスの境界及び、合成フィルタと対数利得予測器の更新時期（4ベクトル毎に更新される）が既知であると仮定している。実際には、この同期化情報は、送信される16kbit/sのビットストリームの先頭に付加的な外部同期ビットを加えれば、復号器で使用できる。

しかしながら、多くのアプリケーションにおいては、16kbit/sのビットストリーム内に、同期ビットまたは、帯域内シグナリングビットの挿入が要求される。これに対しては、以下のように実現できる。同期ビットをN音声ベクトル毎に挿入することを想定する。総てのN番目毎の入力音声ベクトルに対しては、形状コードブックの半分を探索し、6ビットの形状コードブックインデックスを生成する。このようにして、総てのN番目毎に送信するコードブックインデックスから1ビットをスチールし、そのかわり同期ビットまたは、シグナリングビットを挿入できる。

ここで重要なことは、既に選択された7ビットの形状コードブックインデックスから任意に1ビットをスチールすることはできないということである。このため符号器は、どの音声ベクトルから1ビットをスチールするかを知っており、そのベクトルに対しては、コードブックの半分だけを探索する。そうでなければ、復号器は、それらの音声ベクトルに対し、同じ復号化励振コードベクトルを得ることができない。

符号化アルゴリズムは、4ベクトル毎の基本適応周期をもっているので、復号器が符号器の適応周期の境界を容易に決定できるように、Nを4の倍数にするのは妥当である。妥当なN（例えば、10msのビットスチール期間に相当する16）に対して生じる音声品質の劣化は基本的には無視し得る。特に、$N=16$では歪みがほとんど増加しないことが分かっている。このビットスチールレートは、わずか100bit/s である。

上記の処理において、同期ビットが0の場合、コードベクトルのインデックスが0から63をもつ形状コードブックの前半部を、同期ビットが1の場合、コードベクトルのインデックスが64から127をもつ形状コードブックの後半部を探索する。この選択は、形状コードベクトルを表す7ビットが、符号、利得コードブックを表す3ビットに先行するので、同期ビット位置が符号語において最も左になることを意味している。さらに、同期ビットを、4ベクトル周期の最後のベクトルからスチールする。いったん、同期が検出されると、受信した次の符号語はコードベクトルの新しい周期となる。

同期化による歪みは、殆ど生じないと述べたが、この同期手法を含んだハードウェアで正式な試験を行っていない。従って、歪み量は測定していない。

しかしながら、符号器の動作、非動作を繰り返すシステムにおける同期方法として、ビットスチールによる同期ビットの使用は適していない。例えば、音声のない区間は符号器の動作を止める音声検出器を使用するシステムでは、符号器が動作するたびに、復号器は、同期を取る必要がある。100bit/s では、数100msを要するであろう。さらに、復号器状態が符号器状態を追跡するための許容時間を見込まねばならない。この結果、音声発声の頭部が欠落する話頭切断と言われる現象が生じる。もし、符号器、復号器とも音声の立ち上がりと同時に動作すれば、音声欠落は生じない。これは、始動時刻を示す外部信号や外部同期を使用するシステムにおいてのみ可能となる。

４．ＬＤ－ＣＥＬＰ復号器の原理

図４－１／JT-G728にＬＤ－ＣＥＬＰ復号器のブロック図を示す。各ブロックの機能の説明は以下の節で示す。

４．１　励振ＶＱコードブック

このブロックは、ＬＤ－ＣＥＬＰ符号器のコードブック１９と全く同一の励振ＶＱコードブック（形状および利得のコードブックを含む）により構成されている。受信した最適コードブックインデックスを用いて、ＬＤ－ＣＥＬＰ符号器で選ばれた最適コードベクトル $y(n)$ を求める。

４．２　利得調整ユニット

このブロックは、$y(n)$ の各要素と利得 $\sigma(n)$ を掛け、利得調整された励振ベクトル $e(n)$ を求める。

４．３　合成フィルタ

このフィルタは、ＬＤ－ＣＥＬＰ符号器の合成フィルタと同じ伝達関数を持っている（但し、伝送路をエラー無しと仮定している）。利得調整された励振ベクトル $e(n)$ をフィルタに通し、復号された音声ベクトル $s_d(n)$ を出力する。復号による丸め誤差の累積を避けるためには、符号器で $s_q(n)$ を得るのに用いられている手順と同じ方法を正確に行う必要がある。もしこの場合に、符号器で $s_q(n)$ を合成フィルタ９の更新されたメモリから得ているならば、復号器も符号器で行なわれているように、合成フィルタ３２の零入力応答と零状態応答の合計として、$s_d(n)$ を求めるべきである。

４．４　バックワードベクトル利得適応器

このブロックの機能は３．８節に記述されている。

４．５　バックワード合成フィルタ適応器

このブロックの機能は３．７節に記述されている。

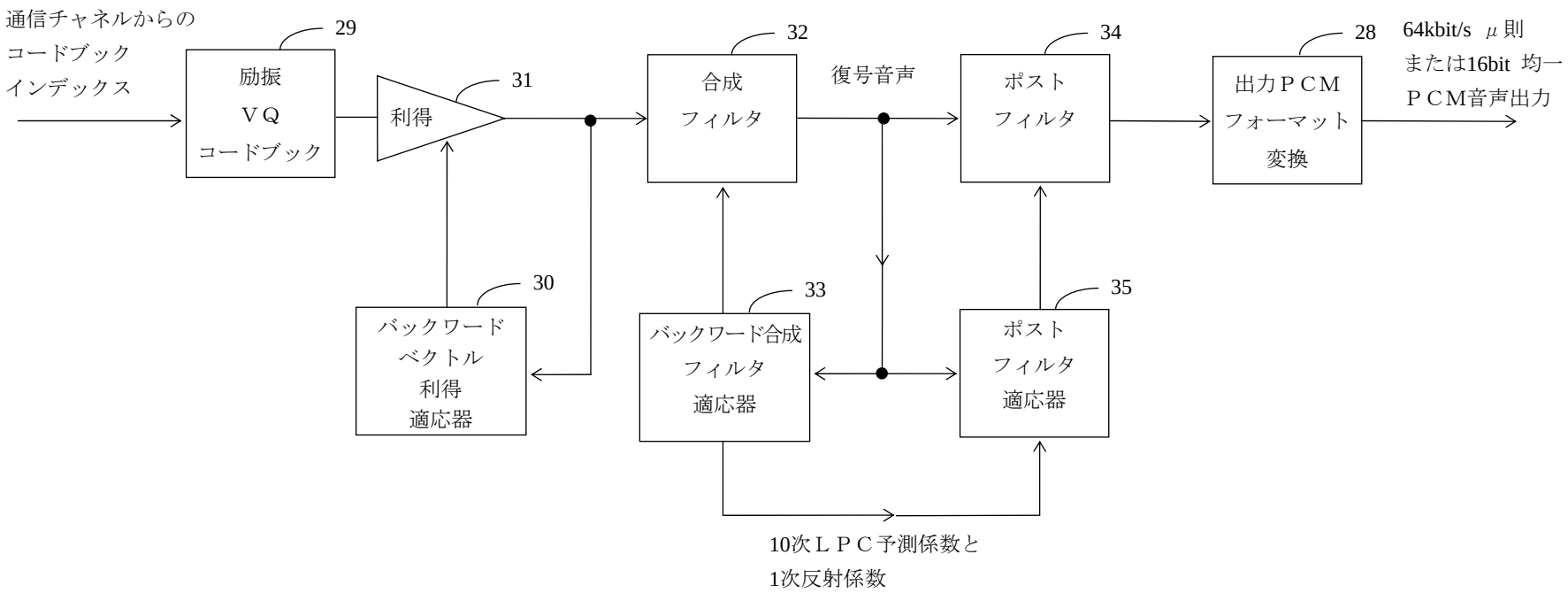


図４－１／JT-G728＊　ＬＤ－ＣＥＬＰ復号器ブロック図
(ITU-T G.728)

４．６　ポストフィルタ

このブロックは聴覚上の品質を向上させるために復号された音声をフィルタ処理する。図４－２／JT-G728に、このブロックの詳細を示す。図４－２／JT-G728を参照。ポストフィルタは基本的に3つの大きな部分：(1) 長期ポストフィルタ７１、(2) 短期ポストフィルタ７２、(3) 出力利得調整ユニット７７、から成り立っている。図４－２／JT-G728の他の4つのブロックは、出力利得調整ユニット７７で用いられる適切なスケーリングファクタを計算するためのものである。

長期ポストフィルタ７１は、しばしばピッチポストフィルタと呼ばれ、ポストフィルタを通る音声の基本周波数（またはピッチ周波数）の倍数にスペクトルのピークを持った櫛形フィルタである。基本周波数の逆数はピッチ周期と呼ばれる。ピッチ周期は、復号された音声からピッチ検出器（またはピッチ抽出器）を用いて抽出することができる。ピッチ検出器で得られる基本ピッチ周期（サンプル数）をpとすれば、長期ポストフィルタの伝達関数は次のように表される。

$$H_l(z) = g_l(1+bz^{-p}) \tag{24}$$

ここで、係数 g_l， b およびピッチ周期 p は1音声ベクトル（1適応周期）ごとに1回更新され、実際の更新は各適応周期の3番目の音声ベクトルで行われる。便宜上、以後適応周期をフレームと呼ぶ事にする。 g_l， b， p の求め方については４．７節に記述されている。

短期ポストフィルタ７２は、10次極零フィルタに1次全零フィルタが直列に接続された構成である。10次極零フィルタはホルマントピーク間の周波数成分を減衰させ、一方1次全零フィルタは、10次極零フィルタの周波数応答のスペクトラムの傾きを補正する。

復号音声をバックワードＬＰＣ分析することにより得られる10次ＬＰＣ予測係数を $\tilde{a}_i, i=1,2,\ldots,10$ とし、同じＬＰＣ分析によって得られる1次反射係数を k_1 とする。ここで、$\tilde{a}_i$ と k_1 はともに、50次バックワードＬＰＣ分析（図３－４／JT-G728のブロック５０）の付随的な結果として得ることができる。これらを得るのに必要な事は、50次レビンソン・ダービン再帰法を10次でやめ、この時の k_1 と $\tilde{a}_1,\tilde{a}_2,\tilde{a}_3,\ldots,\tilde{a}_{10}$ をコピーする。そしてレビンソン・ダービン再帰法を11次から50次まで再び始めることである。短期ポストフィルタの伝達関数は、

$$H_s(z) = \frac{1-\sum_{i=1}^{10}\bar{b}_i z^{-i}}{1-\sum_{i=1}^{10}\bar{a}_i z^{-i}}[1+\mu z^{-1}] \tag{25}$$

ここで、

$$\bar{b}_i = \tilde{a}_i(0.65)^i, i=1,2,\ldots,10 \tag{26}$$

$$\bar{a}_i = \tilde{a}_i(0.75)^i, i=1,2,\ldots,10 \tag{27}$$

そして、

$$\mu = (0.15)k_1 \tag{28}$$

係数 $\bar{a}_i$， $\bar{b}_i$ と μ も1フレームに1回更新されるが、更新は各フレームの最初のベクトルで（即ち、$\tilde{a}_i$ が得られたらすぐ）行われる。

一般に、復号された音声が長期ポストフィルタと短期ポストフィルタを通った後では、このフィルタを通った音声と、復号された（フィルタを通っていない）音声とのパワーレベルは同じではない。時折利得が大きく外れることがあり、これを避けるために、ポストフィルタを通った音声をフィルタを通っていない音声とおおよそ同じパワーにさせる自動利得制御が必要である。これはブロック７３から７７で行われる。

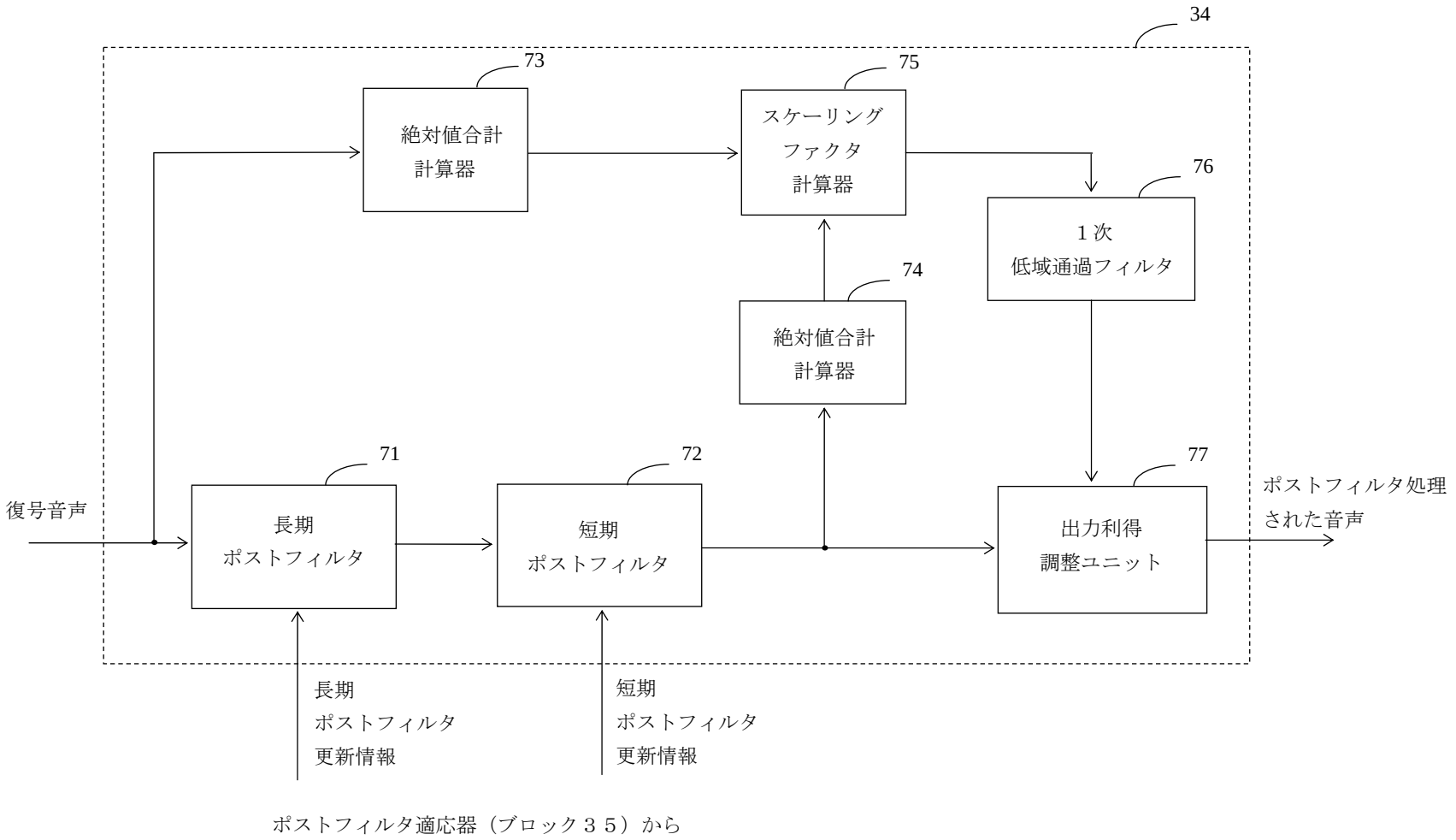


図４－２／JT-G728　ポストフィルタブロック図
(ITU-T G.728)

絶対値合計計算器７３は1ベクトル毎に動作する。現在の復号された音声ベクトル $s_d(n)$ を取り出し、このベクトルの5つの要素の絶対値の合計を計算する。同様に、絶対値合計計算器７４は、短期ポストフィルタの現在の出力ベクトル $s_f(n)$ に対して、同じ計算を行う。スケーリングファクタ計算器７５は、ブロック７３の出力値をブロック７４の出力値で割り、現在の $s_f(n)$ ベクトルに対するスケーリングファクタを求める。そしてこのスケーリングファクタは1次の低域通過フィルタ７６を通り、 $s_f(n)$ の各5つの要素に対して別々のスケーリングファクタが得られる。1次の低域通過フィルタ７６の伝達関数は $0.01/(1-0.99z^{-1})$ である。低域通過フィルタを通ったスケーリングファクタは、出力利得調整ユニット７７により、短期ポストフィルタの出力のサンプル毎のスケーリングを行うのに用いられる。スケーリングファクタ計算器７５は1ベクトル毎に1つのスケーリングファクタを生成するので、もし低域通過フィルタ７６が無ければ、ブロック７７のサンプルごとのスケーリング動作に階段状の影響を及ぼす。低域通過フィルタ７６は効果的に階段状の影響を滑らかにするものである。

４．６．１　非音声信号に対する動作

ＩＴＵ－Ｔでの客観試験結果は、いくつかの非音声信号に対して、適応ポストフィルタを切ったとき、コーデックの性能が改善される事を指摘している。適応ポストフィルタの入力が合成フィルタの出力であるので、この標準ではポストフィルタを無効にするようにスイッチが設定されたときには、単純にこのポストフィルタを通っていない合成フィルタ信号を出力する。

４．７　ポストフィルタ適応器

このブロックは1フレームに一度ポストフィルタの係数を計算し、更新する。ポストフィルタ適応器の詳細を図４－３／JT-G728に示す。

以下、図４－３／JT-G728を参照しながら説明する。10次のＬＰＣ逆フィルタ８１とピッチ周期抽出器８２はともに復号された音声からピッチ周期を抽出する。事実、合理的な動作（付加的な遅延が生じない）をするピッチ抽出器であればどのようなものがここで用いられてもよい。ここで述べられることは、ピッチを抽出する1つの実現法に過ぎない。

10次のＬＰＣ逆フィルタ８１は式(29) の伝達関数で示される。

$$\tilde{A}(z) = 1 - \sum_{i=1}^{10} \tilde{a}_i z^{-i} \tag{29}$$

ここで、係数 $\tilde{a}_i$ はレビンソン・ダービン再帰法（図３－４／JT-G728のブロック５０）から供給され、各フレームの第1ベクトルの時点で更新される。この、ＬＰＣ逆フィルタは入力として復号音声を用い出力としてＬＰＣ予測残差数列 $d(k)$ を発生する。ピッチ分析における窓長は100サンプルとし、ピッチ周期の範囲は20から140サンプルとする。ピッチ周期抽出器８２はこのＬＰＣ予測残差の最も新しい240サンプルを保持するための大きなバッファをもっている。表記上都合が良いようにバッファに蓄えられた240のＬＰＣ残差サンプルに $d(-139), d(-138), \ldots, d(100)$ というインデックスをつける。

ピッチ周期抽出器８２は1フレームに一度ピッチ周期を抽出し、また抽出は各フレームの第3ベクトルにおいて行われる。それゆえ、ＬＰＣ逆フィルタの出力ベクトルはある特定の順序でＬＰＣ残差バッファに蓄えなければならない。直前のフレームの第4ベクトルに対応するＬＰＣ残差ベクトル $d(81), d(82), \ldots, d(85)$ 、現在のフレームの第1ベクトルに対応するＬＰＣ残差は $d(86), d(87), \ldots, d(90)$ に蓄積される。現在のフレームの第2ベクトルのＬＰＣ残差は $d(91), d(92), \ldots, d(95)$ に蓄積される。現在のフレームの第3ベクトルのＬＰＣ残差は $d(96), d(97), \ldots, d(100)$ に蓄積される。サンプル $d(-139), d(-138), \ldots, d(80)$ は、単純に正しい時間順序で並んだＬＰＣ残差サンプル列である。

ＬＰＣ残差バッファの準備が整い次第ピッチ周期抽出器８２は以下の手順で動く。最初に、ＬＰＣ残差バッファの最後の20サンプル（ $d(81)$ から $d(100)$ まで）を3次の楕円フィルタ（係数は付属資料Ｄで与えられる）によって1kHz以下に帯域制限し、次にこれを4:1に間引きする（すなわち4要素に一回のダウンサンプル）。この結果、低域通過フィルタを通過し、間引きされたＬＰＣ残差サンプル列は $\bar{d}(21), \bar{d}(22), \ldots, \bar{d}(25)$ と表され、間引きされたＬＰＣ残差バッファに最も新しい5サンプルとして蓄えられる。更にこれらの5サンプルとは別にＬＰＣ残

差バッファの他の55サンプルである $\bar{d}(-34), \bar{d}(-33), \ldots, \bar{d}(20)$ は間引きされたＬＰＣ残差サンプル列の過去のフレームをシフトして得られる。

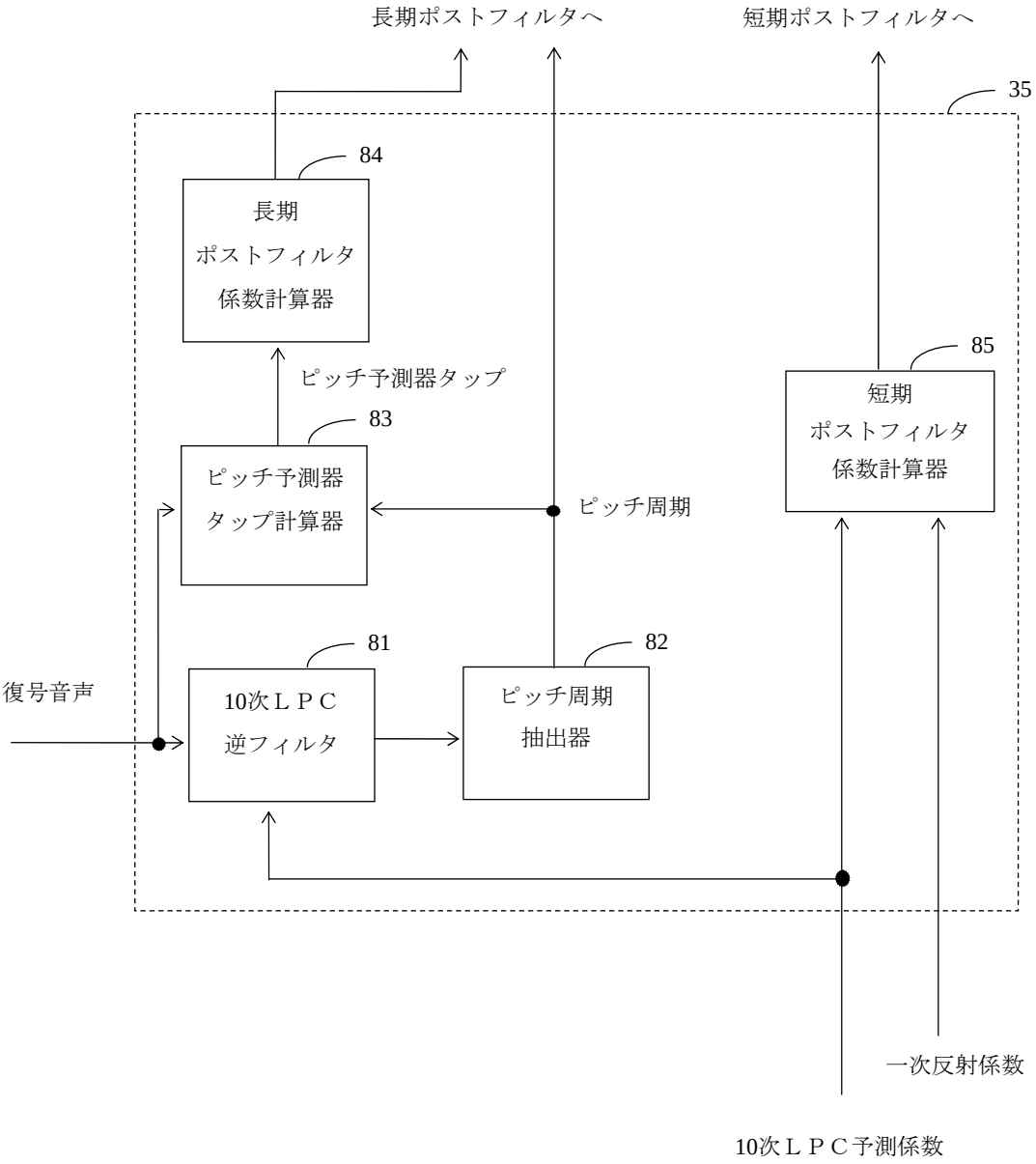


図４－３／JT-G728　ポストフィルタ適応器ブロック図
(ITU-T G.728)

間引きされたＬＰＣ残差サンプル列の i 番目の自己相関は以下により計算される。

$$p(i) = \sum_{n=1}^{25} \bar{d}(n)\bar{d}(n-i) \tag{30}$$

時間遅延は $i = 5, 6, 7, \ldots, 35$ （20から140のサンプルのピッチ周期に対応する）である。計算された31個の自己相関のうち最大のものを与える時間遅延 τ が求まる。この時間遅延 τ は、4:1に間引きされた残差数列におけるものであるから、間引きされる前の残差数列において、最大の自己相関関数を与える時間遅延は、$4\tau - 3$ と $4\tau + 3$ の間に存在するはずである。もとの時間分解能を得るために間引きされる前のＬＰＣ残差バッファを用いて、以下の式で表される間引きされる前のＬＰＣ残差数列の自己相関を計算する。

$$C(i) = \sum_{k=1}^{100} d(k)d(k-i) \tag{31}$$

7個の時間遅延は $i = 4\tau - 3, 4\tau - 2, \ldots, 4\tau + 3$ である。この7個の時間遅延の中から最大の自己相関を与える時間遅延 p_0 が求められる。

このようにして求められた時間遅延 p_0 は本来の基本ピッチ周期の倍数である可能性がある。長期ポストフィルタに必要なものは本来の基本ピッチ周期であって、その倍数ではない。したがって、基本ピッチ周期を見つけるためにはさらに処理を行う必要がある。これには20音声サンプル毎という極めて頻繁にピッチ周期の推定を行っていることを利用する。ピッチ周期は通常20サンプルから140サンプルの間にあるので、頻繁にピッチ抽出を行っていると次のようなことがいえる。つまり音声の始まりの時点では、以上で述べてきた自己相関のピークを見つけ出す過程で、ピッチ周期の倍数が現れる前にまず基本ピッチ周期を見つけたとする。その後は直前のフレームのピッチ周期の近くに自己相関のピークがあるかどうかを調べることにより基本ピッチ周期を得ることができる。

$\hat{p}$ を直前のフレームピッチ周期とする。もし、先に述べた方法で求めた時間遅延 p_0 が $\hat{p}$ の近傍に無ければ式(31)を $i = \hat{p} - 6, \hat{p} - 5, \ldots, \hat{p} + 5, \hat{p} + 6$ にして再び評価する。この13個の時間遅延の中で、最大の自己相関を与える時間遅延 p_1 が求められる。この新しい時間遅延 p_1 を現在のフレームの出力ピッチ周期として使えるかどうか検証する。まず次式を計算する。

$$\beta_0 = \frac{\sum_{k=1}^{100} d(k)d(k - p_0)}{\sum_{k=1}^{100} d(k - p_0)d(k - p_0)} \tag{32}$$

β_0 は p_0 サンプルの遅延を持ったシングルタップピッチ予測器の最適タップ係数である。β_0 の値は0と1の範囲に制限される。次に次式を計算する。

$$\beta_1 = \frac{\sum_{k=1}^{100} d(k)d(k - p_1)}{\sum_{k=1}^{100} d(k - p_1)d(k - p_1)} \tag{33}$$

β_1 は p_1 サンプルの遅延を持ったシングルタップピッチ予測器の最適なタップ係数である。β_1 の値は0と1の範囲に制限される。ブロック８２の出力ピッチ周期 p は次式で与えられる。

$$p = \begin{cases} p_0 & \text{if} \quad \beta_1 \le 0.4\beta_0 \\ p_1 & \text{if} \quad \beta_1 > 0.4\beta_0 \end{cases} \tag{34}$$

ピッチ周期抽出器８２がピッチ周期 p を抽出した後、ピッチ予測器タップ計算器８３は復号音声に対するシング

ルタップピッチ予測器の最適タップ係数を計算する。 ピッチ予測器タップ計算器８３と長期ポストフィルタ７１は復号音声サンプル用の大きなバッファを共有する。このバッファには復号音声 $s_d(-239)$, $s_d(-238)$, $s_d(-237)$, ... , $s_d(4)$, $s_d(5)$ が納められており、 $s_d(1)$ から $s_d(5)$ までは復号音声の現在のベクトルに対応する。長期ポストフィルタ７１はこのバッファをフィルタの遅延器として使用する。 一方、ピッチ予測器タップ計算器８３はこのバッファを次式の計算に使用する。

$$\beta = \frac{\sum_{k=-99}^{0} s_d(k)s_d(k-p)}{\sum_{k=-99}^{0} s_d(k-p)s_d(k-p)} \tag{35}$$

長期ポストフィルタ係数計算器８４はピッチ周期 p とピッチ予測器タップ係数 β から長期ポストフィルタ係数 b と g_l を以下により計算する。

$$b = \begin{cases} 0, & \beta < 0.6 \\ 0.15\beta, & 0.6 \le \beta \le 1 \\ 0.15, & \beta > 1 \end{cases} \tag{36}$$

$$g_l = \frac{1}{1+b} \tag{37}$$

一般的に、 β が1に近くなるほど音声波形は周期的になる。式(36) 、(37)によると、 $\beta < 0.6$ であれば、概略的に無声音かあるいは遷移領域であり、 $b = 0$, $g_l = 1$ となって長期ポストフィルタの伝達関数は $H_L(z) = 1$ となる。このことは長期ポストフィルタのフィルタ動作が無効になることを意味する。一方、 $0.6 < \beta < 1$ であれば、長期ポストフィルタは機能し、 β の値によって櫛形フィルタの度合いが決まる。音声が周期的になるほど櫛形フィルタの度合いは大きくなる。最後に、 $\beta > 1$ であれば、 b を0.15に制限するが、これは櫛形フィルタが過剰に動作するのを避けるためである。係数 g_l は有声音領域が無声音や遷移領域よりも増幅されてしまうことの無いように長期ポストフィルタを利得調整するためのものである（もし、 g_l が1に固定されているとすると、長期ポストフィルタリングのあと、有声音領域はほぼ $1+b$ 倍に増幅される。そうすると、無声音や遷移領域のいくつかの子音は、不明瞭になったり極めて弱々しくなったりする）。

短期ポストフィルタ係数計算器８５は式(26), (27), (28) により、各フレームの第1ベクトルの時点で短期ポストフィルタ係数 $\overline{a}_i, \overline{b}_i$ および μ を計算する。

４．８　出力ＰＣＭフォーマット変換

本ブロックは復号音声ベクトルの5つの要素をそれに対応する5つの μ 則ＰＣＭサンプル列に変換し、この5つのＰＣＭサンプル列を125 μ s間隔で順次出力する。

もし、内部均一ＰＣＭフォーマットが３．１．１節で述べられているようにスケールされているならば μ 則に変換する前に逆スケールを行う必要がある。

５．演算の詳細

この章ではそれぞれＬＤ－ＣＥＬＰ符号器と復号器の演算の詳細を示す。５．１節と５．２節では後述するコーデックのパラメータと内部処理変数について説明する。図３－１／JT-G728から図３－５／JT-G728、図４－１／JT-G728の各ブロックは５．３節以降５章で詳しく定義する。入力音声ベクトルを符号化及び復号するため、ほぼ５．３節以降５章に記述する順番で符号器及び復号器の各ブロックの処理を実行する。

５．１　コーデックの基本的なパラメータ

表５－１／JT-G728にコーデックの基本的なパラメータを定義する。表５－１／JT-G728の1列目に後述する詳細なＬＤ－ＣＥＬＰアルゴリズムで使用するコーデックのパラメータ名を示す。３章あるいは４章で異なる変数名で引用された同一のパラメータに対しては参照のため2列目にその変数名を示す。各々のコーデックのパラメータはコーデックの設計段階で決められた値であり、3列目にその決められた値を、4列目にコーデックのパラメータの簡単な説明を示す。

表５－１／JT-G728　ＬＤ－ＣＥＬＰコーデックの基本的なパラメータ
(ITU-T G.728)

名　称	シンボル	値	説　　明
AGCFAC		0.99	ＡＧＣ適応速度補正係数
FAC	λ	253/256	合成フィルタの帯域幅拡張係数
FACGP	λ_g	29/32	対数利得予測器の帯域幅拡張係数
DIMINV		0.2	ベクトル次元数の逆数
IDIM		5	ベクトル次元数（励振ブロックサイズ）
GOFF		32	対数利得オフセット値
KPDELTA		6	過去のピッチ周期からの許容差
KPMIN		20	最小ピッチ周期（サンプル数）
KPMAX		140	最大ピッチ周期（サンプル数）
LPC		50	合成フィルタの次数
LPCLG		10	対数利得予測器の次数
LPCW		10	聴覚重み付けフィルタの次数
NCWD		128	形状コードブックの大きさ（コードベクトルの数）
NFRSZ		20	フレームの大きさ（適応周期のサンプル数）
NG		8	利得コードブックの大きさ（利得のレベル数）
NONR		35	合成フィルタの非巡回窓サンプルの数
NONRLG		20	対数利得予測器の非巡回窓サンプルの数
NONRW		30	聴覚重み付けフィルタの非巡回窓サンプルの数
NPWSZ		100	ピッチ分析窓長（サンプル数）
NUPDATE		4	予測器更新周期（ベクトル数）
PPFTH		0.6	長期ポストフィルタの有無の閾値
PPFZCF		0.15	長期ポストフィルタの零点補正係数
SPFPCF		0.75	短期ポストフィルタの極補正係数
SPFZCF		0.65	短期ポストフィルタの零点補正係数
TAPTH		0.4	基本ピッチの置換に関する閾値
TILTF		0.15	スペクトル傾斜補正係数
WNCF		257/256	白色雑音補正係数
WPCF	γ_2	0.6	聴覚重み付けフィルタの極補正係数
WZCF	γ_1	0.9	聴覚重み付けフィルタの零点補正係数

５．２　内部変数

表５－２／JT-G728にＬＤ－ＣＥＬＰの内部処理変数のリストを示す。なお、表５－２／JT-G728の書式は表５－１／JT-G728と同様である。2列目は配列変数のインデックスの範囲を示す。4列目に変数の推奨される初期値を示す。いくつかの配列の初期値を付属資料Ａ，Ｂ，Ｃ，Ｄに示す。符号器や復号器が動作を開始する時、あるいはＤＣＭＥに適用される場合のように、コーデックの状態のリセットが必要な場合は、常に内部変数を初期値に設定すること。これらの初期設定により動作開始直後あるいはリセット直後のグリッチを除去できる。

いくつかの配列変数はメモリ領域を節約するため、物理的には同一のメモリ領域を共有し得るが、明確化のため表中では異なる変数名を用いる。

前の章で述べたように、処理シーケンスは4音声ベクトル毎の基本適応周期を持っている。変数ICOUNTはベクトル番号として使われる。言い換えれば、符号器あるいは復号器が適応周期でn番目の音声ベクトルを処理している時、ICOUNT=nとなる。

以降の章で、計算式中の＊は算術乗算を示す。

表５－２／JT-G728（１／２）　ＬＤ－ＣＥＬＰ内部処理変数

(ITU-T G.728)

名　称	配列のインデックスの範囲	シンボル	初　期　値	説　　明
A	1 ~ LPC+1	$-a_{i-1}$	1,0,0,...	合成フィルタの係数
AL	1 ~ 3		付属資料D	1kHz 低域通過フィルタの分母の係数
AP	1 ~ 11	$-\bar{a}_{i-1}$	1,0,0,...	短期ポストフィルタの分母の係数
APF	1 ~ 11	$-\tilde{a}_{i-1}$	1,0,0,...	10次ＬＰＣフィルタの係数
ATMP	1 ~ LPC+1	$-a_{i-1}$		合成フィルタの係数の一時記憶
AWP	1 ~ LPCW+1		1,0,0,...	聴覚重み付けフィルタの分母の係数
AWZ	1 ~ LPCW+1		1,0,0,...	聴覚重み付けフィルタの分子の係数
AWZTMP	1 ~ LPCW+1		1,0,0,...	聴覚重み付けフィルタの係数の一時記憶
AZ	1 ~ 11	$-\bar{b}_{i-1}$	1,0,0,...	短期ポストフィルタの分子の係数
B	1	b	0	長期ポストフィルタの係数
BL	1 ~ 4		付属資料D	1kHz 低域通過フィルタの分子の係数
DEC	-34 ~ 25	$\bar{d}(n)$	0,0,...,0	4 : 1に間引きしたＬＰＣ予測残差
D	-139 ~ 100	$d(k)$	0,0,...,0	ＬＰＣ予測残差
ET	1 ~ IDIM	$e(n)$	0,0,...,0	利得調整された励振ベクトル
FACV	1 ~ LPC+1	λ^{i-1}	付属資料C	合成フィルタの帯域幅拡張ベクトル
FACGPV	1 ~ LPCLG+1	λ_g^{i-1}	付属資料C	利得予測器の帯域幅拡張ベクトル
G2	1 ~ NG	b_i	付属資料B	利得コードブックの利得の2倍
GAIN	1	$\sigma(n)$		励振利得
GB	1 ~ NG-1	d_i	付属資料B	隣接利得値間の中間点
GL	1	g_l	1	長期ポストフィルタのスケーリングファクタ
GP	1 ~ LPCLG+1	$-\alpha_{i-1}$	1,-1,0,0,...	対数利得線形予測器の係数
GPTMP	1 ~ LPCLG+1	$-\hat{\alpha}_{i-1}$		対数利得線形予測器の係数の帯域拡張前の一時記憶配列
GQ	1 ~ NG	g_i	付属資料B	利得コードブックの利得値
GSQ	1 ~ NG	c_i	付属資料B	利得コードブックの利得値の自乗値
GSTATE	1 ~ LPCLG	$\delta(n)$	-32,-32,...,-32	対数利得線形予測器のメモリ
GTMP	1 ~ 4		-32,-32,-32,-32	対数利得の一時記憶
H	1 ~ IDIM	$h(n)$	1,0,0,0,0	$F(z)W(z)$ のインパルス応答ベクトル
ICHAN	1			伝送される最適なコードブックインデックス
ICOUNT	1			音声ベクトルカウンタ（1~4）
IG	1	i		3bit の最適な利得コードブックインデックス
IP	1		IPINIT [b]	ＬＰＣ予測残差のアドレスポインタ
IS	1	j		7bit の最適な形状コードブックインデックス
KP	1	p		現フレームのピッチ周期
KP1	1	$\hat{p}$	50	前フレームのピッチ周期
PN	1 ~ IDIM	$p(n)$		コードブック探索のための相関ベクトル
PTAP	1	β		ブロック８３で算出するピッチ予測器のタップ係数
R	1 ~ NR+1 [a]			自己相関係数
RC	1 ~ NR [a]			反射係数
RCTMP	1 ~ LPC			反射係数の一時記憶
REXP	1 ~ LPC+1		0,0,...,0	自己相関の巡回計算用（合成フィルタ）
REXPLG	1 ~ LPCLG+1		0,0,...,0	自己相関の巡回計算用（対数利得予測器）
REXPW	1 ~ LPCW+1		0,0,...,0	自己相関の巡回計算用（重み付けフィルタ）

[a] NR=Max(LPCW,LPCLG)>IDIM.　[b] IPINIT=NPWSZ-NFRSZ+IDIM.

表５－２／JT-G728（２／２）＊　ＬＤ－ＣＥＬＰ内部処理変数
(ITU-T G.728)

名　称	配列の インデックス の範囲	シンボル	初　期　値	説　　明
RTMP	1 ~ LPC+1			自己相関係数の一時記憶
S	1 ~ IDIM	$s(n)$	0,0,...,0	均一ＰＣＭ入力音声ベクトル
SB	1 ~ 105		0,0,...,0	過去の量子化音声のバッファ
SBLG	1 ~ 34		0,0,...,0	過去の対数利得のバッファ
SBW	1 ~ 60		0,0,...,0	過去の入力音声のバッファ
SCALE	1			低域通過フィルタを通していないポスト フィルタのスケーリングファクタ
SCALEFIL	1		1	低域通過フィルタを通ったポストフィルタの スケーリングファクタ
SD	1 ~ IDIM	$s_d(k)$		復号音声のバッファ
SPF	1 ~ IDIM			ポストフィルタを通った音声ベクトル
SPFPCFV	1 ~ 11	$SPFPCF^{i-1}$	付属資料Ｃ	短期ポストフィルタの極補正ベクトル
SPFZCFV	1 ~ 11	$SPFZCF^{i-1}$	付属資料Ｃ	短期ポストフィルタの零点補正ベクトル
SO	1	$s_o(k)$		入力音声サンプル
SU	1	$s_u(k)$		均一ＰＣＭ入力音声サンプル
ST	-239 ~ IDIM	$s_q(n)$	0,0,...,0	量子化音声ベクトル
STATELPC	1 ~ LPC		0,0,...,0	合成フィルタのメモリ
STLPCI	1 ~ 10		0,0,...,0	ＬＰＣ逆フィルタのメモリ
STLPF	1 ~ 3		0,0,0	1kHz 低域通過フィルタのメモリ
STMP	1 ~ 4＊IDIM		0,0,...,0	聴覚重み付けフィルタのハイブリット窓のバッファ
STPFFIR	1 ~ 10		0,0,...,0	短期ポストフィルタのメモリ（全零の部分）
STPFIIR	1 ~ 10		0,0,...,0	短期ポストフィルタのメモリ（全極の部分)
SUMFIL	1			ポストフィルタを通った音声の絶対値和
SUMUNFIL	1			復号音声の絶対値和
SW	1 ~ IDIM	$v(n)$		聴覚重み付けした音声ベクトル
TARGET	1 ~ IDIM	$\hat{x}(n), x(n)$		正規化したＶＱターゲットベクトル
TEMP	1 ~ IDIM			作業用領域
TILTZ	1	μ	0	短期ポストフィルタの傾斜補正係数
WFIR	1 ~ LPCW		0,0,...,0	聴覚重み付けフィルタ４のメモリ(全零の部分)
WIIR	1 ~ LPCW		0,0,...,0	聴覚重み付けフィルタ４のメモリ(全極の部分)
WNR	1 ~ 105	$w_m(k)$	付属資料Ａ	合成フィルタの窓関数
WNRLG	1 ~ 34	$w_m(k)$	付属資料Ａ	対数利得予測器の窓関数
WNRW	1 ~ 60	$w_m(k)$	付属資料Ａ	聴覚重み付けフィルタの窓関数
WPCFV	1 ~ LPCW+1	γ_2^{i-1}	付属資料Ｃ	聴覚重み付けフィルタの極補正ベクトル
WS	1 ~ 105			中間変数の作業用領域
WZCFV	1 ~ LPCW+1	γ_1^{i-1}	付属資料Ｃ	聴覚重み付けフィルタの零点補正ベクトル
Y	1 ~ IDIM＊NCWD	y_j	付属資料Ｂ	形状コードブックの配列
Y2	1 ~ NCWD	E_j	y_j のエネルギ	畳込んだ形状コードブックのエネルギ
YN	1 ~ IDIM	$y(n)$		量子化励振ベクトル
ZIRWFIR	1 ~ LPCW		0,0,...,0	聴覚重み付けフィルタ１０のメモリ(全零の部分)
ZIRWIIR	1 ~ LPCW		0,0,...,0	聴覚重み付けフィルタ１０のメモリ(全極の部分)

（注）レビンソン・ダービン再帰法の表記上、A, ATMP, AWP, AWZ, GPの各配列の最初の要素は常に1であり、$i \geq 2$ に対しては、i 番目の要素は３章の対応する配列の $(i-1)$ 番目の要素である。

５．３　入力ＰＣＭフォーマット変換（ブロック１）

入力：SO

出力：SU

機能：μ則、または16ビット均一入力ＰＣＭサンプルを均一ＰＣＭサンプルに変換する。

このブロックの動作はＴＴＣ標準ＪＴ－Ｇ７１１で完全に定義されているので、ここでの説明は省略する。但し、３．１．１節で述べたように、入力範囲を-4095から+4095とする仕様に合わせるために、利得調整が必要である。

５．４　ベクトルバッファ（ブロック２）

入力：SU

出力：S

機能：5個の連続する均一ＰＣＭ音声サンプルを格納し、1個の5次元音声ベクトルを作る。

５．５　聴覚重み付けフィルタ適応器（図３－１／JT-G728のブロック３）

以下に、図３－２／JT-G728の3つのブロック（３６，３７，３８）を詳しく説明する。

(1)　ハイブリッド窓かけモジュール（ブロック３６）

入力：STMP

出力：R

機能：入力音声へハイブリッド窓を掛けて自己相関係数を計算する。

このモジュールの動作について、行頭の字下がりによってループ境界を示し、「|」の右側を注釈とする表現を用いて以下に説明する。以下のアルゴリズムを適応周期（20サンプル）毎に使用する。配列STMPは、現適応周期の2番目の音声ベクトルまでの4個の連続する入力音声ベクトルである。即ち、STMP(1)からSTMP(5)は前適応周期の3番目の入力音声ベクトル（初期値0）、STMP(6)からSTMP(10)は前適応周期の4番目の入力音声ベクトル（初期値0）、STMP(11)からSTMP(15)は現適応周期の1番目の入力音声ベクトル、STMP(16)からSTMP(20)は現適応周期の2番目の入力音声ベクトルである。

```
N1=LPCW+NFRSZ                          | 定数の計算（あらかじめ計算して、
N2=LPCW+NONRW                          | メモリに格納しておくこともできる）
N3=LPCW+NFRSZ+NONRW
N=1,2,...,N2について、以下の1行を実行する。
    SBW(N)=SBW(N+NFRSZ)                | 過去の信号バッファのシフト
N=1,2,...,NFRSZについて、以下の1行を実行する。
    SBW(N2+N)=STMP(N)                  | 新しい信号のシフト
                                       | SBW(N3) は最も新しいサンプル
K=1
N=N3,N3-1,...,3,2,1について、以下の2行を実行する。
    WS(N)=SBW(N)＊WNRW(K)              | 窓関数をかける。
    K=K+1
I=1,2,...,LPCW+1について、以下の4行を実行する。
    TMP=0.
    N=LPCW+1,LPCW+2,...,N1について、以下の1行を実行する。
        TMP=TMP+WS(N)＊WS(N+1-I)
    REXPW(I)=(1/2)＊REXPW(I)+TMP       | 巡回成分の更新
I=1,2,...,LPCW+1について、以下の３行を実行する。
    R(I)=REXPW(I)
    N=N1+1,N1+2,...,N3について、以下の1行を実行する。
        R(I)=R(I)+WS(N)＊WS(N+1-I)     | 非巡回成分の加算
R(1)=R(1)＊WNCF                        | 白色雑音補正
```

(2)　レビンソン・ダービン再帰法モジュール（ブロック３７）

入力：R（ブロック３６の出力）

出力：AWZTMP

機能：自己相関係数を線形予測係数に変換する。

このブロックを4ベクトル適応周期毎に実行する。ブロック３６の処理完了後ICOUNT=3で行う。レビンソン・ダービン再帰法はよく知られている技術であるので、アルゴリズムのみ以下に示し説明は省略する。

```
R(LPCW+1)=0.ならば、LABELへジャンプする。          | 0ならばスキップ
R(1)≦0.ならば、LABELへジャンプする。               | 0信号ならばスキップ
RC(1)=-R(2)/R(1)
AWZTMP(1)=1.                                      |
AWZTMP(2)=RC(1)                                   | 1次予測器
ALPHA=R(1)+R(2)＊RC(1)                            |
ALPHA≦0.ならば、LABELへジャンプする。              | 異常状態ならばアボート
MINC=2,3,4,...,LPCWについて、以下を実行する。
    SUM=0.
    IP=1,2,3,...,MINCについて、以下の2行を実行する。
        N1=MINC-IP+2
        SUM=SUM+R(N1)＊AWZTMP(IP)
    RC(MINC)=-SUM/ALPHA                           | 反射係数
    MH=MINC/2+1                                   |
    IP=2,3,4,...,MHについて、以下の4行を実行する。
        IB=MINC-IP+2
        AT=AWZTMP(IP)+RC(MINC)＊AWZTMP(IB)        |
        AWZTMP(IB)=AWZTMP(IB)+RC(MINC)＊AWZTMP(IP)  | 予測係数
        AWZTMP(IP)=AT                             |
    AWZTMP(MINC+1)=RC(MINC)                       |
    ALPHA=ALPHA+RC(MINC)＊SUM                     | 予測残差エネルギ
    ALPHA≦0.ならば、LABELへジャンプする。          | 異常状態ならばアボート
次のMINCに対しても、上記の演算を繰り返す。          |
プログラム正常終了                                | 正常終了
LABEL: ここに来た場合、異常状態であるので、ブロック３８をスキップし、聴覚重み付けフィルタ係
       数は更新しない。（前適応周期の聴覚重み付けフィルタ係数を用いる。）
```

(3) 聴覚重み付けフィルタ係数計算器（ブロック３８）

入力：AWZTMP

出力：AWZ, AWP

機能：入力音声の線形予測係数から聴覚重み付けフィルタ係数を計算する。

このブロックを適応周期毎に実行する。ブロック３７の処理完了後ICOUNT=3で行う。

```
I=2,3,...,LPCW +1について、以下の1行を実行する。
    AWP(I)=WPCFV(I)＊AWZTMP(I)                    | 分母係数
I=2,3,...,LPCW+1について、以下の1行を実行する。
    AWZ(I)=WZCFV(I)＊AWZTMP(I)                    | 分子係数
```

５．６　バックワード合成フィルタ適応器（図３－４／ＪＴ－Ｇ７２８のブロック２３）

図３－４／ＪＴ－Ｇ７２８における3種のブロック（４９，５０，５１）の仕様を以下に示す。

(1) ハイブリッド窓かけモジュール（ブロック４９）

入力：STTMP

出力：RTMP

機能：量子化音声にハイブリッド窓をかけ自己相関係数を計算する。

このブロックの処理は、パラメータや変数、自己相関係数を求めるサンプリング時点を除き、基本的にはブロック３６と同一である。第３章で記述したように、自己相関係数は、過去の4ベクトル毎の適応周期の最後の量子化音声ベクトルをもとに計算される。すなわち、現在の適応周期において使用される自己相関係数は、過去の適応周期において現在から20サンプル過去までの量子化音声に含まれる情報に基づいている。（これは実際、適応周期をどう定義するかによる。）配列STTMPには、過去の適応周期における4つの量子化音声ベクトルが格納されている。

```
N1=LPC+NFRSZ                                  | 定数の計算（あらかじめ計算して
N2=LPC+NONR                                   | メモリに格納しておくこともできる）
N3=LPC+NFRSZ+NONR
N=1,2,...,N2について、以下の1行を実行する。
    SB(N)=SB(N+NFRSZ)                         | 過去の信号バッファのシフト
N=1,2,...,NFRSZについて、以下の1行を実行する。
    SB(N2+N)=STTMP(N)                         | 新しい信号のシフト
                                              | SB(N3)は最も新しいサンプル
K=1
N=N3,N3-1,...,3,2,1について、以下の1行を実行する。
    WS(N)=SB(N)＊WNR(K)                       | 窓関数をかける
    K=K+1
I=1,2,...,LPC+1について、以下の4行を実行する。
    TMP=0.
    N=LPC+1,LPC+2,...,N1について、以下の1行を実行する。
        TMP=TMP+WS(N)＊WS(N+1-I)
    REXP(I)=(3/4)＊REXP(I)+TMP                | 巡回成分の更新
I=1,2,...,LPC+1について、以下の3行を実行する。
    RTMP(I)=REXP(I)
    N=N1+1,N1+2,....N3について、以下の1行を実行する。
        RTMP(I)=RTMP(I)+WS(N)＊WS(N+1-I)      | 非巡回成分の加算
RTMP(1)=RTMP(1)＊WNCF                         | 白色雑音補正
```

(2) レビンソン・ダービン再帰法モジュール（ブロック５０）

入力：RTMP

出力：ATMP

機能：自己相関係数を合成フィルタの係数に変換する。

このブロックの処理は、いくつかのパラメータや変数を除き、ブロック３７と同一である。しかしながら、このブロックを実現するには特別の注意が必要である。第３章で記述したように、自己相関係数の格納された配列RTMPは、各適応周期の最初のベクトルの時点で利用可能であるが、合成フィルタの係数が実際に更新されるのは第3番目のベクトルの時点である。このように更新を意識的に遅らせるのは、ハードウエアの実時間処理において、この処理モジュールの計算を各適応周期内の最初の3ベクトル区間に分散させるためである。この処理モジュールが各周期の最初の2ベクトル区間で実行されている間は、前の周期に求められた古い係数（配列A）が合成フィルタにおいて使用される。このように、別の配列ATMPを設けているのは、古い係数の格納された配列Aに上書きされるのを防ぐ必要があるためである。同様に、配列RTMP, RCTMP, ALPHATMP他は、別のレビンソン・ダービン再帰法モジュール（ブロック３７と４４）による影響を防ぐために設けてある。

```
RTMP(LPCW+1)=0.ならば、LABELへジャンプする。   | 0ならばスキップ
RTMP(1)≦0.ならば、LABELへジャンプする。        | 0信号ならばスキップ
RCTMP(1)=-RTMP(2)/RTMP(1)
```

```
ATMP(1)=1.
ATMP(2)=RCTMP(1)                                    | 1次予測器
ALPHATMP=RTMP(1)+RTMP(2)＊RCTMP(1)
ALPHATMP≦0.ならば、LABELへジャンプする。            | 異常状態ならばアボート
MINC=2,3,4,...,LPCについて、以下を実行する。
    SUM=0.
    IP=1,2,3,...,MINCについて、以下の2行を実行する。
        N1=MINC-IP+2
        SUM=SUM+RTMP(N1)＊ATMP(IP)
    RCTMP(MINC)=-SUM/ALPHATMP                       | 反射係数
    MH=MINC/2+1
    IP=2,3,4,...,MHについて、以下の4行を実行する。
        IB=MINC-IP+2
        AT=ATMP(IP)+RCTMP(MINC)＊ATMP(IB)
        ATMP(IB)=ATMP(IB)+RCTMP(MINC)＊ATMP(IP)    | 予測係数の更新
        ATMP(IP)=AT
    ATMP(MINC+1)=RCTMP(MINC)
    ALPHATMP=ALPHATMP+RCTMP(MINC)＊SUM              | 予測残差エネルギ
    ALPHATMP≦0.ならば、LABELへジャンプする。        | 異常状態ならばアボート
次のMINCに対しても上記の演算を繰り返す。
プログラム正常終了                                  | 正常終了
LABEL: ここに来た場合、異常状態であるので、ブロック５１をスキップし、合成フィルタ係数は更新
       しない。（前適応周期の合成フィルタ係数を使用する。）
```

(3) 帯域幅拡張モジュール（ブロック５１）

入力：ATMP

出力：A

機能：スペクトルピークの帯域幅を拡張するために合成フィルタ係数をスケーリングする。

このブロックは、適応化周期毎に1回実行される。ブロック５０の処理が終了した後に実行が開始され、ICOUNT=3でブロック９と１０の処理が開始される前に終了される。このモジュールの処理が終了しICOUNT=3であるときは、フィルタ係数を更新するために、配列ATMPを配列Aにコピーする。

```
I=2,3,...,LPC+1について、以下の1行を実行する。
    ATMP(I)=FACV(I)＊ATMP(I)                        | スケール係数
ICOUNT=3まで待機、それから
I=2,3,...,LPC+1について、以下の1行を実行する。      | 各周期の3番目のベクトル
    A(I)=ATMP(I)                                    | で係数更新
```

５．７　バックワードベクトル利得適応器（図３－５／JT-G728のブロック２０）

図３－５／JT-G728の各ブロックについて述べる。　ただし、図３－５／JT-G728では概念を説明するために分割して書かれているブロックを、ここでは実現を容易にするために、まとめて一つのブロックとして示しているものもある。

図３－５／JT-G728の全ブロックは、ICOUNT=2の時のみ実行するブロック４３，４４，４５を除いて、各音声ベクトル毎に1回実行される。

(1) 1ベクトル遅延、ＲＭＳ計算器と対数計算器（ブロック６７，３９，４０）

入力：ET

出力：ETRMS

機能：利得調整された一つ前の励振ベクトルの実効値（ＲＭＳ）のdB値を計算する。

ＶＱコードブック探索前にこれら3ブロックを実行する時、配列ETには前の音声ベクトルで決定され利

得調整された励振ベクトルが格納されている。従って、1ベクトル遅延ユニット（ブロック６７）は自動的に実行される（そのことを図３－５／JT-G728では明示している）。対数計算器がＲＭＳ計算器の直後に続くので、ＲＭＳ計算器の平方根計算は対数計算器の出力を2で割ることで処理できる。したがって、対数計算器の出力（dB値）は$10*\log_{10}$(ETのエネルギ / IDIM) となる。ET=0（システムの初期化またはリセット後）の時、対数値がオーバフローするのを避けるため、対数計算器の引数が小さすぎる場合には1に制限する。またETRMSはブロック４２で直ちに処理される一時的な値なので、通常アキュムレータに保存しておく。

```
ETRMS=ET(1)＊ET(1)
K=2,3,...,IDIMについて、以下の1行を実行する。
    ETRMS=ETRMS+ET(K)＊ET(K)          | ETのエネルギを計算する。
ETRMS=ETRMS＊DIMINV                   | IDIMで割る。
ETRMS<1.ならば、ETRMS=1.              | 対数のオーバフローを避ける。
ETRMS=10＊log10(ETRMS)                | dB値を算出する。
```

(2) 対数利得オフセットの減算器（ブロック４２）

入力：ETRMS, GOFF

出力：GSTATE(1)

機能：ブロック４０の出力（dB利得値）から、ブロック４１に保持されている対数利得オフセット値を引き算する。

```
GSTATE(1)=ETRMS-GOFF
```

(3) ハイブリッド窓かけモジュール（ブロック４３）

入力：GTMP

出力：R

機能：オフセットを引いた対数利得系列にハイブリッド窓をかけ、自己相関係数を計算する。

いくつかのパラメータと変数、及び自己相関係数を求めるサンプリング時刻が異なる点を除いて、このブロックの演算はブロック３６と同様である。

ブロック３６とこのブロックの大きな違いは、このブロックが実行されるたびに4利得サンプル（20ではない）だけ、入力として与えられることである。

対数利得予測係数は、各適応周期の2番めのベクトルで更新される。配列GTMPには、前適応周期の２番めのベクトルから現適応周期の最初のベクトルに対応する4つの対数利得値から、オフセット値を引いた値を格納しておく。GTMP(1)が前適応周期の2番めのベクトルの対数利得値からオフセット値を引いた値であり、GTMP(4)が現適応周期の最初のベクトルの対数利得値からオフセット値を引いた値である。

```
N1=LPCLG+NUPDATE                      | 定数の計算（あらかじめ計算して、
N2=LPCLG+NONRLG                       | メモリに格納しておくこともできる)
N3=LPCLG+NUPDATE+NONRLG
N=1,2,...,N2について、以下の1行を実行する。
    SBLG(N)=SBLG(N+NUPDATE)           | 過去の信号バッファのシフト
N=1,2,...,NUPDATEについて、以下の1行を実行する。
    SBLG(N2+N)=GTMP(N)                | 新しい信号のシフト
                                      | SBLG(N3)は最も新しいサンプル
K=1
N=N3,N3-1,...,3,2,1について、以下の2行を実行する。
    WS(N)=SBLG(N)＊WNRLG(K)           | 窓関数を掛ける。
    K=K+1
I=1,2,...,LPCLG+1について、以下の4行を実行する。
```

```
    TMP=0.
    N=LPCLG+1,LPCLG+2,...,N1について、以下の1行を実行する。
        TMP=TMP+WS(N)＊WS(N+1-I)
    REXPLG(I)=(3/4)＊REXPLG(I)+TMP                    | 巡回成分を更新
I=1,2,...,LPCLG+1について、以下の3行を実行する。
    R(I)=REXPLG(I)
    N=N1+1,N1+2,...,N3について、以下の1行を実行する。
        R(I)=R(I)+WS(N)＊WS(N+1-I)                    | 非巡回成分を加算
R(1)=R(1)＊WNCF                                       | 白色雑音補正
```

(4) レビンソン・ダービン再帰法モジュール（ブロック４４）

入力：R（ブロック４３の出力）

出力：GPTMP

機能：自己相関係数を対数利得予測係数に変換する。

このブロックの演算は、次に示すパラメータと変数の置換を除いて、ブロック３７と全く同じである。すなわちLPCWをLPCLGに、AWZをGPに置き換えたものである。このブロックはブロック４３が実行された後、ICOUNT=2の時のみ実行する。第一段階としてR(LPCLG+1)の値をチェックし、もし0であれば、ブロック４４と４５をスキップし対数利得予測係数を更新せずに、前の適応周期で決められた対数利得予測係数を用いる。この特別な手順は、システムの初期化またはリセットのすぐ後に生じるかもしれない非常に小さなグリッチを避けるために設定する。また、配列の内容が異常な場合にもブロック４５をスキップし、前の古い値を用いる。

(5) 帯域幅拡張モジュール（ブロック４５）

入力：GPTMP

出力：GP

機能：スペクトルのピーク帯域幅を広げるため、対数利得予測係数をスケーリングする。

このブロックは、ブロック４４を実行した後ICOUNT=2の時のみ行う。

```
I=2,3,...,LPCLG+1について、以下を実行する。
    GP(I)=FACGPV(I)＊GPTMP(I)                         | スケーリング係数
```

(6) 対数利得線形予測器（ブロック４６）

入力：GP, GSTATE

出力：GAIN

機能：オフセットを引いた対数利得の現在値を予測する。

```
GAIN=0.
I=LGLPC,LPCLG-1,...,3,2について、以下の2行を実行する。
    GAIN=GAIN-GP(I+1)＊GSTATE(I)
    GSTATE(I)=GSTATE(I-1)
GAIN=GAIN-GP(2)＊GSTATE(1)
```

(7) 対数利得オフセット加算器（ブロック４６とブロック４７の間）

入力：GAIN, GOFF

出力：GAIN

機能：対数利得値をもとに戻すため、オフセット値を対数利得予測器の出力に加算する。

```
GAIN=GAIN+GOFF
```

(8) 対数利得リミッタ（ブロック４７）

入力：GAIN

出力：GAIN

機能：予測対数利得の変動幅を制限する。

```
GAIN<0.ならば、GAIN=0.                    | 線形利得を1にする。
GAIN>60.ならば、GAIN=60.                  | 線形利得を1000にする。
```

(9) 逆対数計算器（ブロック４８）

入力：GAIN

出力：GAIN

機能：予測対数利得（dB値）を線形領域へもどす。

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

５．８ 聴覚重み付けフィルタ

(1) 聴覚重み付けフィルタ（ブロック４）

入力：S, AWZ, AWP

出力：SW

機能：入力の音声ベクトルに聴覚重み付けをするためフィルタをかける。

```
K=1,2,...,IDIMについて、以下を実行する。
    SW(K)=S(K)
    J=LPCW,LPCW-1,...,3,2について、以下の2行を実行する。
        SW(K)=SW(K)+WFIR(J)＊AWZ(J+1)      | フィルタの全零部分
        WFIR(J)=WFIR(J-1)
    SW(K)=SW(K)+WFIR(1)＊AWZ(2)            | 最後の1サンプルの処理
    WFIR(1)=S(K)
    J=LPCW,LPCW-1,...,3,2について、次の2行を実行する。
        SW(K)=SW(K)-WIIR(J)＊AWP(J+1)      | フィルタの全極部分
        WIIR(J)=WIIR(J-1)
    SW(K)=SW(K)-WIIR(1)＊AWP(2)            | 最後の1サンプルの処理
    WIIR(1)=SW(K)
次のKに対しても、上記の演算を繰り返す。
```

５．９ 零入力応答ベクトルの計算

３．５節は、「零入力応答ベクトル」 $r(n)$ がブロック９と１０によって、どのように計算されるかを説明している。この場合のこれら2つのブロックの動作を、以下に示す。メモリ更新中のこれら2つのブロックの動作については、５．１３節で述べる。

(l) 零入力応答計算中の合成フィルタ（ブロック９）

入力：A, STATELPC

出力：TEMP

機能：合成フィルタの零入力応答ベクトルの計算

```
K=1,2,...,IDIMについて、以下を実行する。
    TEMP(K)=0.
    J=LPC,LPC-1,...,3,2について、以下の2行を実行する。
        TEMP(K)=TEMP(K)-STATELPC(J)＊A(J+1)  | 積和
        STATELPC(J)=STATELPC(J-1)            | メモリシフト
```

TEMP(K)=TEMP(K)-STATELPC(1)＊A(2)　　| 最後の1サンプルの処理
STATELPC(1)=TEMP(K)
次のKに対しても、上記の演算を繰り返す。

(2)　零入力応答計算中の聴覚重み付けフィルタ（ブロック１０）
入力：AWZ, AWP, ZIRWFIR, ZIRWIIR, 上記において計算されたTEMP
出力：ZIR
機能：聴覚重み付けフィルタの零入力応答ベクトルの計算

K=1,2,...,IDIMについて、以下を実行する。
TMP=TEMP(K)
J=LPCW,LPCW-1,...,3,2について、以下の2行を実行する。
TEMP(K)=TEMP(K)+ZIRWFIR(J)＊AWZ(J+1)　　| フィルタの全零部分
ZIRWFIR(J)=ZIRWFIR(J-1)
TEMP(K)=TEMP(K)+ZIRFIR(1)＊AWZ(2)　　| 最後の1サンプルの処理
ZIRWFIR(1)=TMP
J=LPCW,LPCW-1,...,3,2について、以下の2行を実行する。
TEMP(K)=TEMP(K)-ZIRWIIR(J)＊AWP(J+1)　　| フィルタの全極部分
ZIRWIIR(J)=ZIRWIIR(J-1)
ZIR(K)=TEMP(K)-ZIRWIIR(1)＊AWP(2)　　| 最後の1サンプルの処理
ZIRWIIR(1)=ZIR(K)
次のKに対しても、上記の演算を繰り返す。

５．１０　ＶＱターゲットベクトルの計算

(1)　ＶＱターゲットベクトルの計算（ブロック１１）
入力：SW, ZIR
出力：TARGET
機能：重み付けられた音声ベクトルから、零入力応答ベクトルを減ずる。
（注）上記のブロック１０により、ZIR(K)=ZIRWIIR(IDIM+1-K)であるので、ZIR(K)の為の新たな蓄積領域を必要としない。

K=1,2,...,IDIMについて、以下の1行を実行する。
TARGET(K)=SW(K)-ZIR(K)

５．１１　コードブック探索モジュール（ブロック２４）

コードブック探索モジュール（ブロック２４）に含まれる7個のブロックを以下に示す。また便宜と実現を容易にする為、複数のブロックを1ブロックとして表現する。ブロック１２、１４と１５はICOUNT=3の時、適応周期毎に1回実行される。一方、その他のブロックは音声ベクトル毎に1回実行される。

(1)　インパルス応答ベクトル計算器（ブロック１２）
入力：A, AWZ, AWP
出力：H
機能：合成フィルタと聴覚重み付けフィルタを縦続接続したフィルタのインパルス応答ベクトルを計算する。
このブロックは、ICOUNT=3のとき、そしてブロック２３と３の実行が終了した後に実行される。（すなわちA, AWZ, AWP係数の新しいセットが用意されている時）

TEMP(1)=1.　　| TEMP= 合成フィルタメモリ
RC(1)=1.　　| RC=W(z) 全極部分のメモリ

```
K=2,3,...,IDIMについて、以下を実行する。
    A0=0.
    A1=0.
    A2=0.
    I=K,K-1,...3,2について、以下の5行を実行する。
        TEMP(I)=TEMP(I-1)
        RC(I)=RC(I-1)                          |
        A0=A0-A(I)＊TEMP(I)                    | フィルタリング
        A1=A1+AWZ(I)＊TEMP(I)                  |
        A2=A2-AWP(I)＊RC(I)
    TEMP(1)=A0
    RC(1)=A0+A1+A2
次のKに対しても、上記の演算を繰り返す。
ITMP=IDIM+1                                    | h(n)は、W(z)の全極部分
K=1,2,...,IDIMについて、以下の1行を実行する。  | のメモリの順序を逆に
    H(K)=RC(ITMP-K)                            | することにより得られる。
```

(2) 形状コードベクトル畳込みモジュールとエネルギ表計算器（ブロック１４と１５）

入力：H, Y

出力：Y2

機能：ブロック１２において得られたインパルス応答とそれぞれの形状コードベクトルを畳込む。得られたベクトルのエネルギを計算し、蓄積する。

このブロックもまたブロック１２の実行終了後、ICOUNT=3の時に、実行される。

```
J=1,2,...,NCWDについて、以下を実行する。      | ループあたり1コードベクトル
    J1=(J-1)＊IDIM
    K=1,2,...,IDIMについて、以下の4行を実行する。
        K1=J1+K+1
        TEMP(K)=0.
        I=1,2,...,Kについて、以下の1行を実行する。
            TEMP(K)=TEMP(K)+H(I)＊Y(K1-I)      | 畳込み
    Y2(J)=0.
    K=1,2,...,IDIMについて、以下の1行を実行する。  |
        Y2(J)=Y2(J)+TEMP(K)＊TEMP(K)           | エネルギ計算
次のJに対しても、上記の演算を繰り返す。
```

(3) ＶＱターゲットベクトル正規化（ブロック１６）

入力：TARGET, GAIN

出力：TARGET

機能：予測励振利得を用いて、ＶＱターゲットベクトルを正規化する。

```
TMP=1./GAIN
K=1,2,...,IDIMについて、以下の1行を実行する。
    TARGET(K)=TARGET(K)＊TMP
```

(4) 時間反転畳込みモジュール（ブロック１３）

入力：H, TARGET（ブロック１６からの出力）

出力：PN

機能：インパルス応答ベクトルと正規化されたＶＱターゲットベクトルとの時間反転畳込みを行い、

ベクトル $p(n)$ を得る。

（注）ベクトルPNは、一時的な領域に確保することができる。

```
K=1,2,...,IDIMについて、以下を実行する。
    K1=K-1
    PN(K)=0.
    J=K,K+1,...,IDIMについて、以下の1行を実行する。
        PN(K)=PN(K)+TARGET(J)＊H(J-K1)
次のKに対しても、上記の演算を繰り返す。
```

(5) 誤差計算器と最適コードブックインデックス選択器（ブロック１７と１８）

入力：PN, Y, Y2, GB, G2, GSQ

出力：IG, IS, ICHAN

機能：利得コードブックインデックスと形状コードブックインデックスの最適な組合わせを決定する為、利得コードブックと形状コードブックを探索し、2つを連結して最適な10ビットのコードブックインデックスを求める。

（注）以下で使用される変数CORは、メモリ内よりも、むしろアキュムレータに通常保持するべきである。変数IDXGとJは、一時的なレジスタに保持する。一方IGとISは、メモリ内に保持する。

```
DISTMをハードウェアにおいて表現できる最も大きな数に初期化する。
N1=NG/2
J=1,2,...,NCWDについて、以下を実行する。
    J1=(J-1)＊IDIM
    COR=0.
    K=1,2,...,IDIMについて、以下の1行を実行する。          |
        COR=COR+PN(K)＊Y(J1+K)                          | 内積Pjを計算
    COR>0.ならば、以下の5行を実行する。
        IDXG=N1
        K=1,2,...,N1-1について、以下の3行を実行する。
            COR<GB(K)＊Y2(J)ならば、以下の2行を実行する。
                IDXG=K                                  | 最適な正の利得が求められる。
                LABELへジャンプする。
    COR≦0.ならば、以下の5行を実行する。
        IDXG=NG
        K=N1+1,N1+2,...,NG-1について、以下の3行を実行する。
            COR>GB(K)＊Y2(J)ならば、以下の2行を実行する。
                IDXG=K                                  | 最適な負の利得が求められる。
                LABELへジャンプする。
LABEL:
    D=-G2(IDXG)＊COR+GSQ(IDXG)＊Y2(J)                   | 歪 D̂ を計算する。
    D<DISTMならば、以下の3行を実行する。
        DISTM=D                                         | ここまでの最小の歪と
        IG=IDXG                                         | 最適なコードブック
        IS=J                                            | インデックスを保存する。
次のJに対しても、上記の演算を繰り返す。
ICHAN=(IS-1)＊NG+(IG-1)                                 | 形状と利得コードブック
                                                        | インデックスを結合する。

通信チャネルを通して、ICHANを伝送する。
```

ビットシリアル伝送に対して，ICHANの最上位ビット（ＭＳＢ）が最初に伝送される。もしICHANが10ビットのワード$b_9\ b_8\ b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$,で表現されるならば、伝送されるビット順序は、$b_9$から始まり$b_8, b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$（$b_9$は最上位ビット（ＭＳＢ））となる。

５．１２　局部復号器（ブロック８）

ブロック２０，２３については先に述べたので、以下には、ブロック１９，２１，２２について示す。

(1)　励振ＶＱコードブック（ブロック１９）

入力：IG, IS

出力：YN

機能：最適なコードベクトルと利得を求めるためのテーブルサーチと、量子化励振ベクトルを求めるための乗算を行う。

```
NN=(IS-1)＊IDIM
K=1,2,...,IDIMについて、以下の1行を実行する。
      YN(K)=GQ(IG)＊Y(NN+K)
```

(2)　利得調整ユニット（ブロック２１）

入力：GAIN, YN

出力：ET

機能：励振利得と量子化励振ベクトルの乗算を行う。

```
K=1,2,...,IDIMについて以下の1行を実行する。
      ET(K)=GAIN＊YN(K)
```

(3)　合成フィルタ（ブロック２２）

入力：ET, A

出力：ST

機能：利得調整された励振ベクトルを合成フィルタに通し、量子化音声ベクトルを求める。

３章で説明したようにこのブロックは省略することが可能で、量子化音声ベクトルは、以下に示すメモリ更新の際の副産物として求められる。もし、何らかの方法で、このブロックを実行したいならば、STATELPCでないフィルタメモリを用意し、この全極合成フィルタに使用しなければならない。

５．１３　ブロック９，１０のフィルタメモリ更新

以下に示すブロック９，１０におけるフィルタメモリ更新の方法は、メモリ更新の副産物として、量子化音声ベクトルSTが求められることを前提としている。信号レベルのオーバロードを避けるために、フィルタメモリをMAX，MINで制限する振幅リミッタを設ける。ここでMAX，MINは正、負の飽和レベルであり、それぞれ+4095, – 4095 の値をとる。

(1)　フィルタメモリ更新（ブロック９，１０）

入力：ET, A, AWZ, AWP, STATELPC, ZIRWFIR, ZIRWIIR

出力：ST, STATELPC, ZIRWFIR, ZIRWIIR

機能：ブロック９、ブロック１０のフィルタメモリを更新し、量子化音声ベクトルを計算する。

```
ZIRWFIR(1)=ET(1)                    | ZIRWFIR は、ここでは単に作業領域
                                    | として使用する。
TEMP(1)=ET(1)
K=2,3,...,IDIMについて、以下を実行する。
      A0=ET(K)
```

```
    A1=0.
    A2=0.
    I=K,K-1,...,2について、以下の5行を実行する。
        ZIRWFIR(I)=ZIRWFIR(I-1)
        TEMP(I)=TEMP(I-1)
        A0=A0-A(I)＊ZIRWFIR(I)                    | 縦続接続されたフィルタの各段階
        A1=A1+AWZ(I)＊ZIRWFIR(I)                  | の零状態応答を計算する。
        A2=A2-AWP(I)＊TEMP(I)                     |
    ZIRWFIR(1)=A0                                 |
    TEMP(1)=A0+A1+A2
次のKの値に対しても、上記の演算を繰り返す。
                                                  | 零入力応答に零状態応答を加算し
                                                  | フィルタメモリの更新を行う。
K=1,2,...,IDIMについて、以下の4行を実行する。
    STATELPC(K)=STATELPC(K)+ZIRWFIR(K)
    STATELPC(K)>MAXならば、STATELPC(K)=MAXとする。    | 範囲の制限
    STATELPC(K)<MINならば、STATELPC(K)=MINとする。    |
    ZIRWIIR(K)=ZIRWIIR(K)+TEMP(K)
I=1,2,...,LPCWについて、以下の1行を実行する。    | ZIRWFIR に正しい値を
    ZIRWFIR(I)=STATELPC(I)                        | セットする。
I=IDIM+1
K=1,2,...,IDIMについて、以下の1行を実行する。    | 合成フィルタメモリの順
    ST(K)=STATELPC(I-K)                           | 番を逆にして量子化音声
                                                  | 信号を求める。
```

５．１４　復号器（図４－１／JT-G728）

復号器（図４－１／JT-G728）のブロックについて動作を以下に示す。出力ＰＣＭフォーマット変換ブロックを除き全てのブロックは図３－１／JT-G728の局部復号器（ブロック８）のブロックと全く同じである。

復号器では、表５－２／JT-G728の変数のサブセットのみを使用する。もし、復号器と符号器が1個のＤＳＰチップで実現されるなら、符号器の局部復号器ブロックにおいて使用される変数によって上書きされることを避けるために、復号器の変数は、異なる名前をつけなければならない。例えば、復号器の変数名として、表５－２／JT-G728における変数名に「d」を接頭語としてつけることができる。もし復号器が符号器と独立してスタンドアロンで構成されるならば、変数名を変更する必要はない。

以下ではスタンドアロンの復号器として話を進める。また、各ブロックは以下に示されるのと同じ順序で実行される。

(1) 復号器バックワード合成フィルタ適応器（ブロック３３）

入力：ST

出力：A

機能：過去の復号音声から合成フィルタの係数を周期的に計算する。

本ブロックの動作は符号器のブロック２３と全く同じである。

(2) 復号器バックワードベクトル利得適応器（ブロック３０）

入力：ET

出力：GAIN

機能：過去の利得調整された励振ベクトルより励振利得を計算する。

本ブロックの動作は符号器のブロック２０と全く同じである。

(3) 復号器励振ＶＱコードブック（ブロック２９）

入力：ICHAN

出力：YN

機能：励振ベクトルを計算するために、受信した最適コードブックインデックス（伝送インデックス）を復号する。

本ブロックでは、受信した10ビットの伝送インデックスからまず、3ビットの利得コードブックインデックスIGと7ビットの形状コードブックインデックスISを求める。 以降の動作は符号器のブロック１９と全く同じである。

```
ITMP=(ICHAN/NG)の整数部                          | (IS-1)を復号する。
IG=ICHAN-ITMP＊NG+1                              | IGを復号する。
NN=ITMP＊IDIM
K=1,2,...,IDIMについて、以下の1行を実行する。
    YN(K)=GQ(IG)＊Y(NN+K)
```

(4) 復号器利得調整ユニット（ブロック３１）

入力：GAIN, YN

出力：ET

機能：励振利得を励振ベクトルに乗じる。

本ブロックの動作は符号器のブロック２１と全く同じである。

(5) 復号器合成フィルタ（ブロック３２）

入力：ET, A, STATELPC

出力：ST

機能：復号された音声ベクトルを計算するために利得調整された励振ベクトルを合成フィルタに通す。

本ブロックは簡単な全極フィルタとして実現できる。しかし、４．３節で述べたように、もし、演算量を節約するために符号器がフィルタメモリ更新の副産物として量子化音声信号を求めていたり、丸めによる誤差の累積が問題となるようであれば、このブロックは符号器の局部復号器とまったく同じ方法で、復号音声を計算すべきである。すなわち、復号音声ベクトルは合成フィルタの零入力応答ベクトルと零状態応答ベクトルの和として計算しなければならない。これは以下の方法による。

```
K=1,2,...,IDIMについて、以下の6行を実行する。
    TEMP(K)=0.
    J=LPC,LPC-1,...,3,2まで以下の2行を実行する。
        TEMP(K)=TEMP(K)-STATELPC(J)＊A(J+1)      | 零入力応答計算
        STATELPC(J)=STATELPC(J-1)
    TEMP(K)=TEMP(K)-STATELPC(1)＊A(2)            | 最後の1サンプルの処理
    STATELPC(1)=TEMP(K)
TEMP(1)=ET(1)
K=2,3,...,IDIMについて、以下の5行を実行する。
    A0=ET(K)
    I=K,K-1,...,2について、以下の2行を実行する。
        TEMP(I)=TEMP(I-1)
        A0=A0-A(I)＊TEMP(I)                      | 零状態応答計算
    TEMP(1)=A0
                                  | 零入力応答と零状態応答を加算する
                                  | ことによってフィルタメモリを更新する。
K=1,2,...,IDIMについて、以下の3行を実行する。
    STATELPC(K)=STATELPC(K)+TEMP(K)              | ZIR+ZSR
    STATELPC(K)>MAXならば、STATELPC(K)=MAXとする。  | 範囲制限
    STATELPC(K)<MINならば、STATELPC(K)=MINとする。  |
```

```
I=IDIM+1
K=1,2,...,IDIMについて、以下の1行を実行する。                    | 合成フィルタメモリの順
    ST(K)=STATELPC(I-K)                                        | 番を逆にして量子化音声
                                                               | を計算する。
```

(6) 10次ＬＰＣ逆フィルタ（ブロック８１）

本ブロックは、1つのベクトルに対して1回実行され、出力ベクトルはＬＰＣ予測残差信号バッファの最後の20サンプル分（すなわちD(81)～D(100))に順次書き込まれる。ポインタIPは書き込まれるサンプルD(k)のアドレスを示すために使用する。このポインタIPは、このブロックが最初の適応周期（フレーム）の最初の音声ベクトルの復号処理を開始する以前にNPWSZ-NFRSZ+IDIMに初期化され、その後IPは以下に示す方法で更新される。４．６節で示した様に、ブロック５０により、10次のＬＰＣ予測係数APF(I)は、レビンソン・ダービン再帰法の途中の演算で求められる。本ブロックの処理を開始する以前に、既に復号器合成フィルタ（図４－１／JT-G728のブロック３２）が、現在の復号音声ベクトルをST(1)～ST(IDIM)に書き込んでいることを前提にしている。

入力：ST, APF

出力：D

機能：現在の復号音声ベクトルに対してＬＰＣ予測残差を計算する。

```
IP=NPWSZならば、IP＝NPWSZ-NFRSZ                    | IPのチェックと更新
K=1,2,...,IDIMについて、以下の7行を実行する。
    ITMP=IP+K
    D(ITMP)=ST(K)
    J=10,9,...,3,2について、以下の2行を実行する。
        D(ITMP)=D(ITMP)+STLPCI(J)＊APF(J+1)            | FIR フィルタリング
        STLPCI(J)=STLPCI(J-1)                          | メモリシフト
    D(ITMP)=D(ITMP)+STLPCI(1)＊APF(2)                  | 最後の1サンプルの処理
    STLPCI(1)=ST(K)                                    | 入力シフト
IP=IP+IDIM                                             | IP更新
```

(7) ピッチ周期抽出器（ブロック８２）

本ブロックは各フレーム毎に1回ずつ第3番目のベクトルにおいて、第3の復号音声ベクトルを生成した後で実行する。

入力：D

出力：KP

機能：ＬＰＣ予測残差からピッチ周期を抽出する。

本ブロックはICOUNT=3のときのみ実行する。

```
                                                       | 低域通過ﾌｨﾙﾀﾘﾝｸﾞと４：１の間引き
K=NPWSZ-NFRSZ+1,...,NPWSZについて、以下の7行を実行する。
    TMP=D(K)-STLPF(1)＊AL(1)-STLPF(2)＊AL(2)-STLPF(3)＊AL(3)    | IIR フィルタ
    Kが４で割り切れれば、以下の２行を実行する。
        N=K/4                                          | 必要な場合のみFIR ﾌｨﾙﾀﾘﾝｸﾞの実行
        DEC(N)=TMP＊BL(1)+STLPF(1)＊BL(2)+STLPF(2)＊BL(3)+STLPF(3)＊BL(4)
    STLPF(3)=STLPF(2)
    STLPF(2)=STLPF(1)                                  | 低域通過フィルタメモリのシフト
    STLPF(1)=TMP
M1=KPMIN/4                                             | 間引きされたLPC 残差領域での相関
M2=KPMAX/4                                             | のピークの計算を開始
CORMAX=装置で表現できる負の最小値
```

```
J=M1,M1+1,...,M2について、以下の6行を実行する。
    TMP=0.
    N=1,2,...,NPWSZ/4について、以下の1行を実行する。
        TMP=TMP+DEC(N)＊DEC(N-J)                | TMP=間引きされた領域での相関
    TMP>CORMAXならば、以下の2行を実行する。
        CORMAX=TMP                              | 相関の最大値と対応する時間遅延を
        KMAX=J                                  | 探索
N=-M2+1,-M2+2,...,(NPWSZ-NFRSZ)/4について、以下の1行を実行する。
    DEC(N)=DEC(N+IDIM)                          | 間引きされたLPC残差用ﾊﾞｯﾌｧをｼﾌﾄ
M1=4＊KMAX -3                                   | 間引きされてない領域での相関の
M2=4＊KMAX+3                                    | ピークの計算を開始
M1<KPMINならば、M1=KPMINとする。                | M1が範囲外かどうかを確認
M2>KPMAXならば、M2=KPMAXとする。                | M2が範囲外かどうかを確認
CORMAX=装置で表現できる負の最小値
J=M1,M1+1,...,M2について、以下の6行を実行する。
    TMP=0.
    K=1,2,...,NPWSZについて、以下の1行を実行する。
        TMP=TMP+D(K)＊D(K-J)                    | 間引きされてない領域での相関
    TMP>CORMAXならば、以下の2行を実行する。
        CORMAX=TMP                              | 相関の最大値と対応する時間遅延を
        KP=J                                    | 探索
M1=KP1 - KPDELTA                                | 前回フレームのピッチ周期の周辺に
M2=KP1+KPDELTA                                  | サーチ範囲を決定
KP<M2+1ならば、LABELへジャンプする。            | 真ならKPはピッチの倍数になりえない。
M1<KPMINならば、M1=KPMINとする。                | M1が範囲外かどうかを確認
M2>KPMAXならば、M2=KPMAXとする。                | M2が範囲外かどうかを確認
CMAX=装置で表現できる負の最小値
J=M1,M1+1,...,M2について、以下の6行を実行する。
    TMP=0.
    K=1,2,...,NPWSZについて、以下の1行を実行する。
        TMP=TMP+D(K)＊D(K-J)                    | 間引きされてない領域での相関
    TMP>CMAXならば、以下の2行を実行する。
        CMAX=TMP                                | 相関の最大値と対応する時間遅延を
        KPTMP=J                                 | 探索
SUM=0.
TMP=0.                                          | タップ係数の計算を開始
K=1,2,...,NPWSZについて、以下の2行を実行する。
    SUM=SUM+D(K-KP)＊D(K-KP)
    TMP=TMP+D(K-KPTMP)＊D(K-KPTMP)
SUM=0.ならば、TAP=0、そうでなければ、TAP=CORMAX/SUMとする。
TMP=0.ならば、TAP1=0、そうでなければ、TAP1=CMAX/TMPとする。
TAP>1ならば、TAP=1とする。                     | TAPを0から1の範囲に制限
TAP<0ならば、TAP=0とする。
TAP1>1ならば、TAP1=1とする。                   | TAP1を0から1の範囲に制限
TAP1<0ならば、TAP1=0とする。
                                                | TAP1が充分大きければ、KPを基本
                                                | ピッチに置換
TAP1>TAPTH＊TAPならば、KP=KPTMPとする。
LABEL:  KP1=KP                                  | 前回フレームのピッチ周期を更新
```

K=-KPMAX+1,-KPMAX+2,...,NPWSZ-NFRSZについて、以下の1行を実行する。
　　D(K)=D(K+NFRSZ)　　| LPC 残差用バッファをシフト

(8)　ピッチ予測器タップ計算器（ブロック８３）

本ブロックも各フレーム毎に1回ずつ第3番目のベクトルにおいて、ブロック８２の実行直後に実行する。
本ブロックは長期ポストフィルタ部（ブロック７１）と復号音声バッファ（配列ST(K)）を共有している。
長期ポストフィルタ部では以下のように配列のシフトを行っている。

(a)　ST(1)からST(IDIM)が現在の復号音声ベクトル
(b)　ST(-KPMAX-NPWSZ+1)からST(0)までが以前の復号音声ベクトル

入力：ST, KP
出力：PTAP
機能：復号音声の1次ピッチ予測器の最適なタップ係数を計算する。
本ブロックはICOUNT=3のときのみ実行する。

SUM=0.
TMP=0.
K=-NPWSZ+1,-NPWSZ+2,...,0について、以下の2行を実行する。
　　SUM=SUM+ST(K-KP)＊ST(K-KP)
　　TMP=TMP+ST(K)＊ST(K-KP)
SUM=0.ならば、PTAP=0とし、そうでなければ、PTAP=TMP/SUMとする。

(9)　長期ポストフィルタ係数計算器（ブロック８４)

本ブロックも各フレーム毎に1回ずつ第3番目のベクトルにおいて、ブロック８３の実行直後に実行する。
入力：PTAP
出力：B, GL
機能：長期ポストフィルタの係数 b とスケーリングファクタ（利得調整用係数） g_l を計算する。
本ブロックはICOUNT=3のときのみ実行する。

PTAP>1ならば、PTAP=1とする。　　| PTAPを1に制限
PTAP<PPFTHならば、PTAP=0とする。　　| PTAPがしきい値より小さければ
　　| 長期ポストフィルタをオフ
B=PPFZCF＊PTAP
GL=1/(1+B)

(10)　短期ポストフィルタ係数計算器（ブロック８５）

本ブロックも各フレーム毎に1回ずつ実行するが、各フレームの最初のベクトルで実行する。
入力：APF, RCTMP(1)
出力：AP, AZ, TILTZ
機能：短期ポストフィルタの係数を計算する。
本ブロックはICOUNT=1のときのみ実行する。

I=2,3,...,11について、以下の2行を実行する。
　　AP(I)=SPFPCFV(I)＊APF(I)　　| 分母の係数の大きさの補正
　　AZ(I)=SPFZCFV(I)＊APF(I)　　| 分子の係数の大きさの補正
TILTZ=TILTF＊RCTMP(1)　　| 傾斜補正フィルタ係数

(11)　長期ポストフィルタ(ブロック７１）

本ブロックはベクトル毎に1回実行する。
入力：ST, B, GL, KP

出力：TEMP
機能：長期ポストフィルタのフィルタリングを行う。

K=1,2,...,IDIMについて、以下の1行を実行する。
　　TEMP(K)=GL＊(ST(K)+B＊ST(K-KP))　　| 長期ポストフィルタリング
K=-NPWSZ-KPMAX+1,...,-2,-1,0について、以下の1行を実行する。
　　ST(K)=ST(K+IDIM)　　| 復号音声バッファをシフト

(12)　短期ポストフィルタ（ ブロック７２）
本ブロックはベクトル毎に1回、ブロック７１の直後に実行する。
入力：AP, AZ, TILTZ, STPFFIR, STPFIIR, TMP（ブロック７１の出力）
出力：TEMP
機能：短期ポストフィルタのフィルタリングを行う。

K=1,2,...,IDIMについて、以下を実行する。
　　TMP=TEMP(K)
　　J=10,9,...,3,2について、以下の2行を実行する。
　　　　TEMP(K)=TEMP(K)+STPFFIR(J)＊AZ(J+1)　　| フィルタの全零部分
　　　　STPFFIR(J)=STPFFIR(J-1)
　　TEMP(K)=TEMP(K)+STPFFIR(1)＊AZ(2)　　| 最後の乗算
　　STPFFIR(1)=TMP
　　J=10,9,...,3,2について、以下の2行を実行する。
　　　　TEMP(K)=TEMP(K)-STPFIIR(J)＊AP(J+1)　　| フィルタの全極部分
　　　　STPFIIR(J)=STPFIIR(J-1)
　　TEMP(K)=TEMP(K)-STPFIIR(1)＊AP(2)　　| 最後の乗算
　　STPFIIR(1)=TEMP(K)
　　TEMP(K)=TEMP(K)+STPFIIR(2)＊TILTZ　　| スペクトル傾斜補正フィルタ
次のKに対しても上記の演算を繰り返す。

(13)　絶対値合計計算器（ブロック７３）
本ブロックはベクトル毎に1回、ブロック３２の後に実行する。
入力：ST
出力：SUMUNFIL
機能：復号音声ベクトルの成分の絶対値の合計を計算する。

SUMUNFIL=0.
K=1,2,...,IDIMについて、以下の1行を実行する。
　　SUMUNFIL=SUMUNFIL+〔ST(K)の絶対値〕

(14)　絶対値合計計算器（ブロック７４）
本ブロックはベクトル毎に1回、ブロック７２の後に実行する。
入力：TEMP（ブロック７２の出力）
出力：SUMFIL
機能：短期ポストフィルタの出力ベクトルの成分の絶対値の合計を計算する。

SUMFIL=0.
K=1,2,...,IDIMについて、以下の1行を実行する。
　　SUMFIL=SUMFIL+〔TEMP(K)の絶対値〕

(15)　スケーリングファクタ計算器（ブロック７５）

本ブロックはベクトル毎に1回、ブロック７３と７４の後に実行する。

入力：SUMUNFIL, SUMFIL

出力：SCALE

機能：ポストフィルタの総合スケーリングファクタを計算する。

SUMFIL>1ならば、SCALE=SUMUNFIL/SUMFIL、そうでなければSCALE=1とする。

(16)　１次低域通過フィルタ（ブロック７６）と出力利得調整ユニット（ブロック７７）

本２ブロックはベクトル毎に1回、ブロック７２と７５の後に実行する。この2ブロックは一緒に記述する。

入力：SCALE, TEMP（ブロック７２の出力）

出力：SPF

機能：１ベクトルに１回のスケーリングファクタの低域通過フィルタリング、及び、短期ポストフィルタの出力ベクトルの利得調整のためにフィルタを通したスケーリングファクタを使用する。

K=1,2,...,IDIMについて、以下の2行を実行する。

SCALEFIL=AGCFAC＊SCALEFIL+(1-AGCFAC)＊SCALE	| 低域通過フィルタリング
SPF(K)=SCALEFIL＊TEMP(K)	| 出力の調整
SPF(K)>MAXならば、SPF(K)=MAXとする。	| 範囲の制限
SPF(K)<MINならば、SPF(K)=MINとする。	

1次低域通過フィルタ（ブロック７６）と出力利得調整ユニット（ブロック７７）によって生成されるポストフィルタの出力信号のビット数は、16ビットに制限されねばならない。（適応ポストフィルタの入力信号に対してと同様の飽和レベルで制限を行なう。）

(17)　出力ＰＣＭフォーマット変換（ブロック２８）

入力：SPF

出力：SD

機能：復号音声ベクトルの5つの要素をそれに対応する5つのμ則ＰＣＭサンプル列に変換し、125μsの間隔で順次出力する。

均一ＰＣＭからμ則ＰＣＭへの変換則はＴＴＣ標準ＪＴ－Ｇ７１１で規定されている。

付属資料Ａ
（標準ＪＴ－Ｇ７２８に対する）
ＬＤ－ＣＥＬＰにおける各ＬＰＣ分析に用いるハイブリッド窓関数

ＬＤ－ＣＥＬＰ符号器では3つの独立したＬＰＣ分析を、合成フィルタ、対数利得予測器、聴覚重み付けフィルタの3つのフィルタ係数を更新するために用いる。これらのＬＰＣ分析では別々のハイブリッド窓を使用する。ハイブリッド窓かけ計算に用いるそれぞれの窓関数を以下に示す。これらの窓関数は、まず浮動小数点演算で算出され、次に小数部15ビットの16ビット表現で正確に表せるように量子化されている。以下ではそれぞれの窓について、16ビット精度の値に等しい浮動小数点表現の表、対応する16ビットの整数表現の表の順で示している。

Ａ．１　合成フィルタ用ハイブリッド窓

合成フィルタ用窓関数の最初の105サンプルを付表Ａ－１／JT-G728に示す。初めから35サンプルは非巡回部分で、残りは巡回部分である。表は1行目を左から右へ、次に2行目を左から右へという順に（走査線のように）読む。

付表Ａ－１／JT-G728　合成フィルタ用ハイブリッド窓関数（浮動小数点表現）
(ITU-T G.728)

0.047760010	0.095428467	0.142852783	0.189971924	0.236663818
0.282775879	0.328277588	0.373016357	0.416900635	0.459838867
0.501739502	0.542480469	0.582000732	0.620178223	0.656921387
0.692199707	0.725891113	0.757904053	0.788208008	0.816680908
0.843322754	0.868041992	0.890747070	0.911437988	0.930053711
0.946533203	0.960876465	0.973022461	0.982910156	0.990600586
0.996002197	0.999114990	0.999969482	0.998565674	0.994842529
0.988861084	0.981781006	0.974731445	0.967742920	0.960815430
0.953948975	0.947082520	0.940307617	0.933563232	0.926879883
0.920227051	0.913635254	0.907104492	0.900604248	0.894134521
0.887725830	0.881378174	0.875061035	0.868774414	0.862548828
0.856384277	0.850250244	0.844146729	0.838104248	0.832092285
0.826141357	0.820220947	0.814331055	0.808502197	0.802703857
0.796936035	0.791229248	0.785583496	0.779937744	0.774353027
0.768798828	0.763305664	0.757812500	0.752380371	0.747009277
0.741638184	0.736328125	0.731048584	0.725830078	0.720611572
0.715454102	0.710327148	0.705230713	0.700164795	0.695159912
0.690185547	0.685241699	0.680328369	0.675445557	0.670593262
0.665802002	0.661041260	0.656280518	0.651580811	0.646911621
0.642272949	0.637695313	0.633117676	0.628570557	0.624084473
0.619598389	0.615142822	0.610748291	0.606384277	0.602020264

付表Ａ－２／JT-G728は付表Ａ－１／JT-G728に対応する16ビット整数表現である。付表Ａ－２／JT-G728の各要素を $2^{15} = 32768$ で割ったものが付表Ａ－１／JT-G728である。

付表Ａ－２／JT-G728　合成フィルタ用ハイブリッド窓関数（整数表現）
(ITU-T G.728)

1565	3127	4681	6225	7755
9266	10757	12223	13661	15068
16441	17776	19071	20322	21526
22682	23786	24835	25828	26761
27634	28444	29188	29866	30476
31016	31486	31884	32208	32460
32637	32739	32767	32721	32599
32403	32171	31940	31711	31484
31259	31034	30812	30591	30372
30154	29938	29724	29511	29299
29089	28881	28674	28468	28264
28062	27861	27661	27463	27266
27071	26877	26684	26493	26303
26114	25927	25742	25557	25374
25192	25012	24832	24654	24478
24302	24128	23955	23784	23613
23444	23276	23109	22943	22779
22616	22454	22293	22133	21974
21817	21661	21505	21351	21198
21046	20896	20746	20597	20450
20303	20157	20013	19870	19727

Ａ．２　対数利得予測器用ハイブリッド窓

対数利得予測器用窓関数の最初の34サンプルを付表Ａ－３／JT-G728に示す。初めから20サンプルは非巡回部分で、残りは巡回部分である。表は付表Ａ－１／JT-G728と同様に読む。

付表Ａ－３／JT-G728　対数利得予測器用ハイブリッド窓関数（浮動小数点表現）
(ITU-T G.728)

0.092346191	0.183868408	0.273834229	0.361480713	0.446014404
0.526763916	0.602996826	0.674072266	0.739379883	0.798400879
0.850585938	0.895507813	0.932769775	0.962066650	0.983154297
0.995819092	0.999969482	0.995635986	0.982757568	0.961486816
0.932006836	0.899078369	0.867309570	0.836669922	0.807128906
0.778625488	0.751129150	0.724578857	0.699005127	0.674316406
0.650482178	0.627502441	0.605346680	0.583953857	

付表Ａ－４／JT-G728は付表Ａ－３／JT-G728に対応する16ビット整数表現である。付表Ａ－４／JT-G728の各要素を $2^{15} = 32768$ で割ったものが付表Ａ－３／JT-G728である。

付表Ａ－４／JT-G728　対数利得予測器用ハイブリッド窓関数（整数表現）
(ITU-T G.728)

3026	6025	8973	11845	14615
17261	19759	22088	24228	26162
27872	29344	30565	31525	32216
32631	32767	32625	32203	31506
30540	29461	28420	27416	26448
25514	24613	23743	22905	22096
21315	20562	19836	19135	

Ａ．３　聴覚重み付けフィルタ用ハイブリッド窓

聴覚重み付けフィルタ用窓関数の最初の60サンプルを付表Ａ－５／JT-G728に示す。初めから30サンプルは非巡回部分で、残りは巡回部分である。表は付表Ａ－１／JT-G728と同様に読む。

付表Ａ－５／JT-G728　聴覚重み付けフィルタ用ハイブリッド窓関数（浮動小数点表現）
(ITU-T G.728)

0.059722900	0.119262695	0.178375244	0.236816406	0.294433594
0.351013184	0.406311035	0.460174561	0.512390137	0.562774658
0.611145020	0.657348633	0.701171875	0.742523193	0.781219482
0.817108154	0.850097656	0.880035400	0.906829834	0.930389404
0.950622559	0.967468262	0.980865479	0.990722656	0.997070313
0.999847412	0.999084473	0.994720459	0.986816406	0.975372314
0.960449219	0.943939209	0.927734375	0.911804199	0.896148682
0.880737305	0.865600586	0.850738525	0.836120605	0.821746826
0.807647705	0.793762207	0.780120850	0.766723633	0.753570557
0.740600586	0.727874756	0.715393066	0.703094482	0.691009521
0.679138184	0.667480469	0.656005859	0.644744873	0.633666992
0.622772217	0.612091064	0.601562500	0.591217041	0.581085205

付表Ａ－６／JT-G728は付表Ａ－５／JT-G728に対応する16ビット整数表現である。付表Ａ－６／JT-G728の各要素を $2^{15} = 32768$ で割ったものが付表Ａ－５／JT-G728である。

付表Ａ－６／JT-G728　聴覚重み付けフィルタ用ハイブリッド窓関数（整数表現）
(ITU-T G.728)

1957	3908	5845	7760	9648
11502	13314	15079	16790	18441
20026	21540	22976	24331	25599
26775	27856	28837	29715	30487
31150	31702	32141	32464	32672
32763	32738	32595	32336	31961
31472	30931	30400	29878	29365
28860	28364	27877	27398	26927
26465	26010	25563	25124	24693
24268	23851	23442	23039	22643
22254	21872	21496	21127	20764
20407	20057	19712	19373	19041

付属資料Ｂ

（標準ＪＴ－Ｇ７２８に対する）

励振形状コードブックと利得コードブックの表

付表Ｂ－１／JT-G728に、7ビットの励振形状ＶＱコードブックを示す。付表Ｂ－１／JT-G728において、それぞれの行は128個の形状コードベクトルの中の1つを示している。この表の第1列はそれぞれのコードベクトル（グレイコード割当アルゴリズムによって得られたもの）の伝送インデックスを示している。第2列目から第6列目までが16ビット固定小数点形式で表された128個の形状コードベクトルの第1要素から第5要素である。この固定小数点形式の値を浮動小数点形式の値に変換するためにはそれぞれの値を2048で割ればよい。これは 2^{-11} を乗算するか、あるいは小数点位置を11ビット左にシフトすることでも得られる。

付表Ｂ－１／JT-G728（１／４） 励振形状ＶＱコードブック
(ITU-T G.728)

伝送 インデックス	コードベクトル要素				
0	668	-2950	-1254	-1790	-2553
1	-5032	-4577	-1045	2908	3318
2	-2819	-2677	-948	-2825	-4450
3	-6679	-340	1482	-1276	1262
4	-562	-6757	1281	179	-1274
5	-2512	-7130	-4925	6913	2411
6	-2478	-156	4683	-3873	0
7	-8208	2140	-478	-2785	533
8	1889	2759	1381	-6955	-5913
9	5082	-2460	-5778	1797	568
10	-2208	-3309	-4523	-6236	-7505
11	-2719	4358	-2988	-1149	2664
12	1259	995	2711	-2464	-10390
13	1722	-7569	-2742	2171	-2329
14	1032	747	-858	-7946	-12843
15	3106	4856	-4193	-2541	1035
16	1862	-960	-6628	410	5882
17	-2493	-2628	-4000	-60	7202
18	-2672	1446	1536	-3831	1233
19	-5302	6912	1589	-4187	3665
20	-3456	-8170	-7709	1384	4698
21	-4699	-6209	-11176	8104	16830
22	930	7004	1269	-8977	2567
23	4649	11804	3441	-5657	1199
24	2542	-183	-8859	-7976	3230
25	-2872	-2011	-9713	-8385	12983
26	3086	2140	-3680	-9643	-2896
27	-7609	6515	-2283	-2522	6332
28	-3333	-5620	-9130	-11131	5543
29	-407	-6721	-17466	-2889	11568
30	3692	6796	-262	-10846	-1856
31	7275	13404	-2989	-10595	4936

付表Ｂ－１／JT-G728（２／４）　励振形状ＶＱコードブック
(ITU-T G.728)

伝送インデックス	コードベクトル要素				
32	244	-2219	2656	3776	-5412
33	-4043	-5934	2131	863	-2866
34	-3302	1743	-2006	-128	-2052
35	-6361	3342	-1583	-21	1142
36	-3837	-1831	6397	2545	-2848
37	-9332	-6528	5309	1986	-2245
38	-4490	748	1935	-3027	-493
39	-9255	5366	3193	-4493	1784
40	4784	-370	1866	1057	-1889
41	7342	-2690	-2577	676	-611
42	-502	2235	-1850	-1777	-2049
43	1011	3880	-2465	2209	-152
44	2592	2829	5588	2839	-7306
45	-3049	-4918	5955	9201	-4447
46	697	3908	5798	-4451	-4644
47	-2121	5444	-2570	321	-1202
48	2846	-2086	3532	566	-708
49	-4279	950	4980	3749	452
50	-2484	3502	1719	-170	238
51	-3435	263	2114	-2005	2361
52	-7338	-1208	9347	-1216	-4013
53	-13498	-439	8028	-4232	361
54	-3729	5433	2004	-4727	-1259
55	-3986	7743	8429	-3691	-987
56	5198	-423	1150	-1281	816
57	7409	4109	-3949	2690	30
58	1246	3055	-35	-1370	-246
59	-1489	5635	-678	-2627	3170
60	4830	-4585	2008	-1062	799
61	-129	717	4594	14937	10706
62	417	2759	1850	-5057	-1153
63	-3887	7361	-5768	4285	666

付表Ｂ－１／JT-G728（３／４）　励振形状ＶＱコードブック
(ITU-T G.728)

伝送 インデックス	コードベクトル要素				
64	1443	-938	20	-2119	-1697
65	-3712	-3402	-2212	110	2136
66	-2952	12	-1568	-3500	-1855
67	-1315	-1731	1160	-558	1709
68	88	-4569	194	-454	-2957
69	-2839	-1666	-273	2084	-155
70	-189	-2376	1663	-1040	-2449
71	-2842	-1369	636	-248	-2677
72	1517	79	-3013	-3669	-973
73	1913	-2493	-5312	-749	1271
74	-2903	-3324	-3756	-3690	-1829
75	-2913	-1547	-2760	-1406	1124
76	1844	-1834	456	706	-4272
77	467	-4256	-1909	1521	1134
78	-127	-994	-637	-1491	-6494
79	873	-2045	-3828	-2792	-578
80	2311	-1817	2632	-3052	1968
81	641	1194	1893	4107	6342
82	-45	1198	2160	-1449	2203
83	-2004	1713	3518	2652	4251
84	2936	-3968	1280	131	-1476
85	2827	8	-1928	2658	3513
86	3199	-816	2687	-1741	-1407
87	2948	4029	394	-253	1298
88	4286	51	-4507	-32	-659
89	3903	5646	-5588	-2592	5707
90	-606	1234	-1607	-5187	664
91	-525	3620	-2192	-2527	1707
92	4297	-3251	-2283	812	-2264
93	5765	528	-3287	1352	1672
94	2735	1241	-1103	-3273	-3407
95	4033	1648	-2965	-1174	1444

付表Ｂ－１／JT-G728（４／４）　励振形状ＶＱコードブック
(ITU-T G.728)

伝送 インデックス	コードベクトル要素				
96	74	918	1999	915	-1026
97	-2496	-1605	2034	2950	229
98	-2168	2037	15	-1264	-208
99	-3552	1530	581	1491	962
100	-2613	-2338	3621	-1488	-2185
101	-1747	81	5538	1432	-2257
102	-1019	867	214	-2284	-1510
103	-1684	2816	-229	2551	-1389
104	2707	504	479	2783	-1009
105	2517	-1487	-1596	621	1929
106	-148	2206	-4288	1292	-1401
107	-527	1243	-2731	1909	1280
108	2149	-1501	3688	610	-4591
109	3306	-3369	1875	3636	-1217
110	2574	2513	1449	-3074	-4979
111	814	1826	-2497	4234	-4077
112	1664	-220	3418	1002	1115
113	781	1658	3919	6130	3140
114	1148	4065	1516	815	199
115	1191	2489	2561	2421	2443
116	770	-5915	5515	-368	-3199
117	1190	1047	3742	6927	-2089
118	292	3099	4308	-758	-2455
119	523	3921	4044	1386	85
120	4367	1006	-1252	-1466	-1383
121	3852	1579	-77	2064	868
122	5109	2919	-202	359	-509
123	3650	3206	2303	1693	1296
124	2905	-3907	229	-1196	-2332
125	5977	-3585	805	3825	-3138
126	3746	-606	53	-269	-3301
127	606	2018	-1316	4064	398

付表Ｂ－２／JT-G728に利得コードブックを示す。付表Ｂ－２／JT-G728には、GQ, GB, G2およびGSQの値を示している。16ビット演算で正確にGQとGBを表すにはQ13フォーマット（小数点以下が13ビット）を用いる。G2の固定小数点形式による値はGQと同じ値であるが、フォーマットはQ12フォーマット（小数点以下が12ビット）である。GSQの値はQ12フォーマットの固定小数点形式における最も近い整数値で近似すれば十分である。

付表Ｂ－２／JT-G728　利得コードブックに関連する配列の値
(ITU-T G.728)

配列名	1	2	3	4
GQ [b)]	0.515625	0.90234375	1.579101563	2.763427734
GB	0.708984375	1.240722656	2.171264649	[a)]
G2	1.03125	1.8046875	3.158203126	5.526855468
GSQ	0.26586914	0.814224243	2.493561746	7.636532841

配列名	5	6	7	8
GQ [b)]	-GQ(1)	-GQ(2)	-GQ(3)	-GQ(4)
GB	-GB(1)	-GB(2)	-GB(3)	[a)]
G2	-G2(1)	-G2(2)	-G2(3)	-G2(4)
GSQ	GSQ(1)	GSQ(2)	GSQ(3)	GSQ(4)

[a)] 使用しないため、任意の値でよい。

[b)] GQ(1)=33/64,　GQ(*i*)=(7/4)GQ(*i*-1),　*i*=2, 3, 4 である。

付属資料Ｃ
（標準ＪＴ－Ｇ７２８に対する）
帯域幅拡張に用いる値

付表Ｃ－１／JT-G728は表５－２／JT-G728に示されている極、零点の補正および帯域幅拡張用のベクトルの整数値である。この値を浮動小数点形式に変換するには16384で割ればよい。付表Ｃ－１／JT-G728の値は浮動小数点形式での値を、2以下の数を16ビット固定小数点演算で計算するときに最もよく用いられるQ14フォーマット（小数点以下が14ビット）で示したものである。

付表Ｃ－１／JT-G728（１／２）　極、零点の補正および帯域幅拡張に用いる値
(ITU-T G.728)

i	FACV	FACGPV	WPCFV	WZCFV	SPFPCFV	SPFZCFV
1	16384	16384	16384	16384	16384	16384
2	16192	14848	9830	14746	12288	10650
3	16002	13456	5898	13271	9216	6922
4	15815	12195	3539	11944	6912	4499
5	15629	11051	2123	10750	5184	2925
6	15446	10015	1274	9675	3888	1901
7	15265	9076	764	8707	2916	1236
8	15086	8225	459	7836	2187	803
9	14910	7454	275	7053	1640	522
10	14735	6755	165	6347	1230	339
11	14562	6122	99	5713	923	221
12	14391					
13	14223					
14	14056					
15	13891					
16	13729					
17	13568					
18	13409					
19	13252					
20	13096					
21	12943					
22	12791					
23	12641					
24	12493					
25	12347					
26	12202					
27	12059					
28	11918					
29	11778					
30	11640					
31	11504					
32	11369					
33	11236					

付表Ｃ－１／JT-G728（２／２）　極、零点の補正および帯域幅拡張に用いる値
(ITU-T G.728)

i	FACV
34	11104
35	10974
36	10845
37	10718
38	10593
39	10468
40	10346
41	10225
42	10105
43	9986
44	9869
45	9754
46	9639
47	9526
48	9415
49	9304
50	9195
51	9088

付属資料Ｄ
（標準ＪＴ－Ｇ７２８に対する）
ピッチ周期抽出器（ブロック８２）に用いる1kHz 低域通過楕円フィルタの係数

ピッチ周期抽出器（ブロック８２）に用いる1kHz 低域通過フィルタは、3次の極零フィルタであり、その伝達関数は、

$$L(z)=\frac{\sum_{i=0}^{3} b_i z^{-i}}{1+\sum_{i=1}^{3} a_i z^{-i}}$$

である。その係数 a_i と b_i を以下の表に示す。

付表Ｄ－１／JT-G728　1kHz 低域通過楕円フィルタの係数
(ITU-T G.728)

i	a_i	b_i
0	―	0.0357081667
1	-2.34036589	-0.0069956244
2	2.01190019	-0.0069956244
3	-0.614109218	0.0357081667

付属資料Ｅ
（標準ＪＴ－Ｇ７２８に対する）
計算順序の時間割り当て

符号器と復号器における全ての計算は、2つの組に分類できる。第1の組に含まれるのは、ベクトル毎に1回行われる計算である。３章から５．１４節迄に、これらの計算が記述されている。一般に、これらは励振信号の実際の量子化や出力信号の合成を伴うものか、もしくはこれを導くものである。特に図３－１／JT-G728のブロック番号で示すならば、１，２，４，９，１０，１１，１３，１６，１７，１８，２１および２２のブロックがこの組に属する。図４－１／JT-G728では、２８，２９，３１，３２および３４のブロックがこの組に属する。図３－５／JT-G728では、３９，４０，４１，４２，４６，４７，４８および６７のブロックがこの組に属する。（注：図３－５／JT-G728は、図３－１／JT-G728のブロック２０と図４－１／JT-G728のブロック３０の両方に適用できる。図３－５／JT-G728の４３，４４と４５のブロックは、この組ではない。従って、ブロック２０と３０は両方の組に属する部分である。）

もう一方の組は、4ベクトル毎に1回だけ実行される計算である。再び図３－１／JT-G728から図４－３／JT-G728までを参照すると、この組に属するのは、３，１２，１４，１５，２３，３３，３５，３６，３７，３８，４３，４４，４５，４９，５０，５１，８１，８２，８３，８４，そして８５の各ブロックである。第2の組に属する全ての計算は、符・復号器にいくつかある適応フィルタないし予測器の更新に関連したものである。符号器ではそのような適応動作部が3つあり、50次のＬＰＣ合成フィルタ、ベクトル利得予測器、そして聴覚重み付けフィルタである。復号器ではそのような適応動作部が4つあり、合成フィルタ、利得予測器、そして長期および短期の適応ポストフィルタである。３章から５．１４節において、これら5種類の適応動作部のそれぞれに対する計算時間と入力信号が記述されている。重複にはなるが、読者の利便を図るために、本付属資料ではこのタイミング情報の全てを1つにまとめて明確に示す。以下に示す表は、5種類の適応動作部について、その入力信号、計算時間、そして更新された値を最初に用いる時刻、をまとめたものである。ちなみに同表の4列目の欄は、これらの計算を参照するための、図や３章から５章迄で用いたブロック番号を示すものである。

付表Ｅ－１／JT-G728　各種適応器の更新タイミング
(ITU-T G.728)

適応器の更新タイミング			
適　応　器	入力信号	変数の更新時期[a)]	参照ブロック
バックワード合成フィルタ適応器	第4ベクトルまでの合成フィルタの出力音声（ST）	符号器、復号器の第3ベクトル時	２３，３３（４９，５０，５１）
バックワードベクトル利得適応器	第1ベクトルまでの対数利得	符号器、復号器の第2ベクトル時	２０，３０（４３，４４，４５）
聴覚重み付けフィルタ適応器、および高速コードブック探索	第2ベクトルまでの入力音声（S）	符号器の第3ベクトル時	３（３６，３７，３８）１２，１４，１５
長期ポストフィルタの適応器	第3ベクトルまでの合成フィルタ出力音声（ST）	ポストフィルタ出力の第3ベクトル合成時	３５（８１から８４まで）
短期ポストフィルタの適応器	第4ベクトルまでの合成フィルタ出力音声（ST）	ポストフィルタ出力の第1ベクトル合成時	３５（８５）

[a)] 更新されたパラメータ値を最初に用いる時刻を示す。

50次の合成フィルタの更新には、かなりの計算時間を要する。これに必要な入力信号は、合成フィルタ出力音声（ST）である。前の周期の第4ベクトルを復号したら、直ちに自己相関係数の計算のためのハイブリッド窓かけ（ブロック４９）の演算を開始することができる。それが完了したならば、予測係数を得るためのレビンソン・

ダービン再帰法（ブロック５０）を開始して良い。経験的に、この計算には1ベクトル周期以上の時間が必要であることが判っている。ここでは、第1ベクトルの受信が完了しないうちにハイブリッド窓の計算を開始する。レビンソン・ダービン再帰法を計算し終える前に、第1ベクトルを符号化するための割り込みを掛けなければならない。レビンソン・ダービン再帰法は、第2ベクトルになって完了する。最後に、予測係数に対して帯域幅拡張（ブロック５１）の処理を適用する。この計算の結果は、第3ベクトルの符号化ないし復号までは使用しない。何故ならば、符号器において、この更新された値を聴覚重み付けフィルタおよびコードベクトルエネルギの更新に連動させる必要があるからである。これらの更新は、第3ベクトルまでは不可能である。

利得の適応は2つのやり方で実行される。適応予測器は4ベクトル毎に1回更新される。しかし、この適応予測器は1ベクトルごとに新しい利得値を算出する。本節では、予測器の更新タイミングについて記述する。この計算には、まず最初に過去の対数利得に基づくハイブリッド窓かけ（ブロック４３）を実行し、次にレビンソン・ダービン再帰法（ブロック４４）、そして帯域幅拡張（ブロック４５）を実行することが要求される。これら全ては、第1ベクトルまでの対数利得を使用し、第2ベクトルの間に実行を完了することが可能である。レビンソン・ダービン再帰法の結果に特異性がなければ、新しい利得予測器が第2ベクトルの符号化において直ちに使用される。

聴覚重み付けフィルタの更新は、第3ベクトルの間に計算される。この更新の最初の部分は第2ベクトルまでの入力音声に基づくＬＰＣ分析の実行である。この計算は第2ベクトルを符号化した直後、または第3ベクトルの受信が完了する以前に開始することができる。この更新は、ハイブリッド窓かけ（ブロック３６）、レビンソン・ダービン再帰法（ブロック３７）、そして重み付けフィルタ係数の算出（ブロック３８）の実行からなる。次に、インパルス応答ベクトル計算器（ブロック１２）の計算のために、聴覚重み付けフィルタを合成フィルタの更新に連動させる必要がある。また、コードベクトルのエネルギを知るために、形状コードベクトル毎にこのインパルス応答との畳込みを行わなければならない（ブロック１４と１５）。これらの計算が終了すれば、第3ベクトルの符号化において全ての更新した値を直ちに使用可能である。（注：コードベクトルエネルギの計算は負荷が大きいので、たとえ利得予測器の更新を他の時間に移しても、第2ベクトルの期間中の演算の一部として聴覚重み付けフィルタの更新を完了させる事は不可能であった。これが第3ベクトルまで更新を延長した理由である。）

長期ポストフィルタは、合成フィルタ出力音声（ST）をその入力とする高速ピッチ抽出アルゴリズムに基づいて更新される。ポストフィルタは復号器でのみ使用されるので、この計算を実行するための時間割り当ては、復号器における他の計算負荷に基づくものである。復号器は聴覚重み付けフィルタとコードベクトルエネルギを更新する必要がなく、そのため第3ベクトルの時間が使用可能である。第3ベクトルに対する符号語が復号されると、合成フィルタ出力音声がそれ以前の合成出力ベクトルと共に使用可能となる。これらは適応器の入力となり、そこで、適応器は新規のピッチ周期（ブロック８１と８２）と長期ポストフィルタ係数（ブロック８３と ８４）を算出する。これらの新しい値は、第3ベクトルに対するポストフィルタ出力の算出に直ちに使用される。

短期ポストフィルタは、合成フィルタの更新の副産物として更新される。レビンソン・ダービン再帰法を10次までで停止し、ポストフィルタの更新のために予測係数を格納する。レビンソン・ダービン再帰法の計算は常に第1ベクトルの期間中に開始されるので、短期ポストフィルタの更新は、第1出力ベクトルのポストフィルタリングの時間に終了する。

付属資料Ｆ
（標準ＪＴ－Ｇ７２８に対する）
本標準で使用する略語のアルファベット順リスト

ＣＥＬＰ	Code excited linear prediction 符号励振線形予測
ＤＣＭＥ	Digital circuit multiplication equipment ディジタル回線多重化装置
ＤＳＰ	Digital signal processing ディジタル信号処理
ＬＤ－ＣＥＬＰ	Low-delay code excited linear prediction 低遅延符号励振線形予測
ＬＰＣ	Linear prediction coding 線形予測符号化
ＭＳＥ	Mean-square error 自乗平均誤差
ＰＣＭ	Pulse code modulation パルス符号変調
ＲＭＳ	Root-mean square 実効値
ＶＱ	Vector quantization ベクトル量子化
ＷＮＣＦ	White noise correction factor 白色雑音補正係数

付属資料Ｇ
（標準ＪＴ－Ｇ７２８に対する）
16kbit/s固定小数点の詳細記述

１．本付属資料の規定範囲

本資料の目的はＴＴＣ標準ＪＴ－Ｇ７２８の16kbit/sＬＤ－ＣＥＬＰを固定小数点演算のデバイスで実現する方法について詳細に記述することである。本資料に基づき固定小数点で実現された装置は、ＪＴ－Ｇ７２８の浮動小数点版のものと完全に相互接続ができ、音声や帯域内データ信号について同等の品質の信号を出力できるようにしなければならない。ここでは固定小数点演算は16ビット語長の演算を意味する。ほとんどの16ビットデバイスは16ビット語長以外のレジスタも持っている。たとえば2つの16ビットワードの積は32ビットワードである。したがって16ビットデバイスの乗算レジスタは通常32ビット語長である。アキュムレータは積和の結果を保持するので、同様に少なくとも32ビット以上の語長でなければならない。したがって“16ビット実現”とはいってもいくつかの内部状態変数は16ビットよりも大きな精度を持っている。

本資料では、32ビットの乗算レジスタと少なくとも2つの32ビット（あるいはそれ以上の）アキュムレータを備える16ビット固定小数点ディジタル信号処理プロセッサで、標準ＪＴ－Ｇ７２８を実現するのに必要なすべての演算の、完全なビット対応の記述を与える。本資料中の数多くの項目のなかには、全く同等の結果が得られるような代用可能な処理方法がある。そのような項目については代用の方法で置き換えられるかも知れない。しかしながら、起こりうるすべての入力に対して同等の結果が得られるのでなければそのような置き換えを行うべきではない。代用可能な方法は極めて数多くあるので、それらの多くは示されていない。

本資料は７章から構成される。１章では導入のほか固定小数点信号処理や本資料の中で使用される規約について記述する。２章では特にＪＴ－Ｇ７２８の固定小数点による実現のために行ったアルゴリズムの変更について記述する。３章ではコーデックに使用されるその他のモジュールの固定小数点擬似コードを示す。４章では固定小数点コーデックの状態変数表現の総覧を示す。５章と６章ではバックワードベクトル利得適応器に関する表を示す。７章では符号器と復号器のメインプログラム擬似コードを示す。

１．１　一般的方針

本資料はＴＴＣ標準ＪＴ－Ｇ７２８の付属資料である。したがってＪＴ－Ｇ７２８の全てにわたって再び詳しく記述したり議論したりする必要はない。いくつかの参考になる箇所については再度記述する。ＪＴ－Ｇ７２８では、浮動小数点で実現するための計算について、全て詳細に記述している。浮動小数点演算を固定小数点演算に置き換えただけの部分については、本資料で計算の詳細をいっさい記述していない。

浮動小数点版コーデックから固定小数点版コーデックへの最も大きな変更点は、

(1)状態変数の様々な種類の算術演算と精度について記述したこと、

(2)バックワードベクトル利得適応を変更したこと、（ただし数学的に等価である。）

(3)レビンソン・ダービン再帰法で予測係数を計算する際の変数精度について記述したこと、

である。

１章の残りの部分ではいろいろな数値の表現方法と固定小数点演算について詳しく述べる。

２章では前に述べた2つの主なアルゴリズムの変更点、すなわちバックワードベクトル利得適応部およびレビンソン・ダービン再帰法の変更点について詳細に記述する。

３章ではハイブリッド窓かけモジュールすなわちＪＴ－Ｇ７２８のブロック４９の擬似コードを記述する。ハイブリッド窓かけモジュールのアルゴリズムは変更しないが、固定小数点演算を用いることによって実現装置が複雑になる。ハイブリッド窓かけモジュールの擬似コードはコーデックのその他の多くの部分で行わなければならない類の変更の良い一例である。

４章では符号器および復号器で使われるすべての状態変数の数値表現を示した表５－２／JT-G728に対応した表を掲載する。

本資料では一貫してすべての数値表現は2の補数表現であると仮定している。コーデックを実現する際には、数学的に等価な結果になるような別の数値表現を使うことができる。

１．２　数値表現

16ビット固定小数点実現装置の基本構成単位は16ビットワードである。純粋な整数を表現する場合には-32768

から+32767の範囲となる。1は0000000000000001で表現され-32768は1000000000000000で表現される。ここで最も右のビットは最下位ビット（ＬＳＢ）であり、最も左のビットは最上位ビット（ＭＳＢ）である。2の補数表現では、ＭＳＢが0ならば正の数であり、1ならば負の数である。ＬＳＢがビット0、ＭＳＢがビット15となるようにすべてのビットに0から15までの番号を付ける。

小数部分を持つような数を表現する場合には、2つのビットの間に小数点を付けなければならない。 たとえば-1.0と+1.0の間にある数を表現する場合には、ビット14とビット15との間に小数点をつける。このフォーマットは小数点の右に15ビットあるのでQ15フォーマットと呼ぶ。Q*n*フォーマットとは小数点の右に*n*ビットあるものと定義する。純粋な整数はQ0フォーマットで表現される。

いくつかのデータは16ビットワードによる数値表現よりも大きな精度を要求する。そのようなデータのために倍精度フォーマットを定義する。このフォーマットは32ビットの情報を持っている。16ビット語長では 2^{15} 分の1の精度でデータを表現できるが、市販で入手できるほとんどのＤＳＰチップの乗算レジスタやアキュムレータのような32ビットレジスタは 2^{31} 分の1の精度でデータを表現できる。これらを倍精度と呼ぶ。倍精度の場合もダイナミックレンジを示すために小数点を持たなければならない。

いくつかのデータは、いかなる固定小数点16ビットフォーマットでも表現できないような大きな範囲を持っている。精度は16ビットでよいが、値のスケーリングをダイナミックに行わなくてはならない。このようなデータの場合は単精度浮動小数点で表現できる。単精度浮動小数点ではデータは2ワードで表現される。第1の16ビットワードは、値の絶対値が16384から32767の間にそろえられた数である。これは値の仮数部を表す。このように仮数部の値の範囲をそろえた表現を正規化フォーマットと呼ぶ。もし値が正ならば仮数部のビット14は1になる。第2の16ビットワードは値を正規化フォーマットにするために行った左シフトの数（ＮＬＳ）を表す。これは仮数部のQフォーマットを特定するものである。このフォーマットが単一の値に使用された場合はスカラ浮動小数点と呼ぶ。

同様にブロック浮動小数点を用いて、*n*要素の配列を*n*+1ワードで表現することができる。このフォーマットを用いる場合は、配列の中で絶対値最大の値をスカラ浮動小数点の所で述べた方法と同様の方法で表現する。配列のその他のすべての値は同じＮＬＳを用いる。それらの仮数部は正規化フォーマットである必要はない。ブロック浮動小数点の拡張として分割ブロック浮動小数点がある。この場合*mn*要素の配列は*m*(*n*+1)ワードで表現できる。配列は*m*個の大きさ*n*の副配列に分割され、それぞれの副配列は*n*ワードで大きさを表現し、1ワードで　ＮＬＳを表現する。

その他、使用される表現方法の類には倍精度浮動小数点がある。この表現では仮数部に倍精度整数を用い、1つの単精度の数をＮＬＳを表現するのに用いる。使用される数値表現方法の種類をまとめると、単精度固定小数点フォーマット、アキュムレータや乗算レジスタのための倍精度固定小数点フォーマット、スカラ単精度浮動小数点フォーマット、および単精度と倍精度のブロック浮動小数点フォーマットである。

１．３　算術演算

2つの16ビットワードを乗算すると結果は32ビットの数値になる。このため乗算レジスタはふつう倍精度である。乗算レジスタはアキュムレータに加算される可能性があるのでアキュムレータも同様に少なくとも32ビット長でなければならない。畳込みやＦＩＲフィルタリングのような積和計算の場合、アキュムレータがオーバフローするかも知れない。このようなオーバフローの問題は市販のＤＳＰチップによって、それぞれ異なった扱いをしている。

ＩＩＲフィルタリングでは、積和や累積演算の結果はフィルタ用メモリの一部となり次回のフィルタリング処理が行われる際に再び使用される。とくに出力の上位16ビットは乗算器の入力として使用される。正の絶対値の大きな値を負の絶対値の大きな値に変換したり、あるいは負の絶対値の大きな値を正の絶対値の大きな値に変換するようなオーバフローはラップアラウンドとして知られており、フィルタ出力に大きな差異を生じさせる。このことを防止するには、すべてのＩＩＲフィルタおよび積和の結果が後に乗算器の入力として使用されるすべての場合について飽和モードの算術演算を使用する。飽和モードは上位のワードが32767よりも大きくなるかあるいは-32768よりも小さくなる場合、ラップアラウンドを避けるためにそれらの値にクリップすることを意味する。

１．３．１　シフトと丸め

算術演算の議論は最初にシフトと丸めについて行う。Q*n*フォーマットの数値とQ*m*フォーマットの数値を乗算すると、乗算レジスタの中の結果はQ(*n*+*m*)の倍精度フォーマットになる。この乗算結果が異なった精度で蓄積さ

れるか加算される必要のあるとき、妥当な精度になるようにシフトか丸め、あるいはその両方を行わなければならない。

シフトは2つのタイプが考えられる。それは左シフトと右シフトである。市販ＤＳＰチップでは、シフトは通常アキュムレータの中で行われる。またアキュムレータの中に加算したり蓄積したりする前に、乗算レジスタの中で乗算結果をシフトすることも通常可能である。左シフトではビットは左にシフトされ、右シフトでは右にシフトされる。いま数値をkビット右シフトすると、その数値の下位kビットは失われてしまう。また左シフトするとオーバフローの可能性を調べなければならない。変数TMPをkビット右シフトすることを

TMP=TMP>>k

と記述し、左シフトすることを

TMP=TMP<<k

と記述する。

ある場合にはkが変数であることもあり、また負の数であることもある。kが負の場合にはkビットの左シフトは$-k$ ビットの右シフトと定義される。同様にkビットの右シフトは$-k$ ビットの左シフトと等しい。kが負である可能性がある場合、擬似コードでは負の可能性を調べて、kが負であるときは$-k$ ビットの逆シフトをおこなう。負のシフトは数学的には上記のように定義されているが、ほとんどのデバイスやいくつかのコンピュータ言語では実現することができない。

2の補数表現では、右シフトする場合に注意すべき点がある。いま、3を1ビット右シフトすることを考える。3は16ビット表現で0000000000000011となる。これを1ビット右シフトすると0000000000000001 = 1となる。つぎに-3を右シフトしたとする。-3は1111111111111101で表現される。これを右シフトすると1111111111111110 = -2となる。右シフトにおいてまず注意すべきことは、符号ビットが拡張されるということである。また上記の2つの例ではシフトした結果の絶対値が一致しない。もし符号絶対値表現を用いたならばシフトした結果の絶対値は一致する。実現の際にはこの違いについて注意すること。

さらにコンパイラに依存した捕らえ難い違いがある。アルゴリズム処理中に、語長より大きいビット数の右シフト命令を実行する可能性がある。たとえば16ビットワードを18ビットシフトする場合などである。1ビット右シフトを18回実行することで実現されるような処理では、その結果はもとのデータの符号に依存して、0か-1になる。しかし、コンパイラによっては18ビットシフトを不当な命令として誤った結果を出力することが知られている。実現の際には目的のハードウェアおよび言語コンパイラがこれらの場合についてどのように処理するのか確認するべきである。

丸めはアキュムレータにおいて倍精度から単精度に変換する処理である。通常丸めは16ビットワードとしてメモリに値を格納する直前に行われる。アキュムレータは上位ワードと下位ワードから構成される（さらに上位ワードの左側にいくつかの付加ビットを備える場合もある）。通常は、上位ワードでも下位ワードでもどちらでもメモリに転送することができる。また2つの連続した命令を使って両方を転送することもできる。アキュムレータの上位ワードと下位ワードとの間に小数点があるものとすると、丸めはアキュムレータの上位および下位ワードに記憶されている非整数をそれに最も近い整数に変換する処理である。ふつう2の補数において行われる丸めの方法は、下位ワードのＭＳＢで判断するものである。下位ワードのＭＳＢが1ならば上位ワードに1を加算する。そして下位ワードをすべて0にする。たとえばアキュムレータ上の数が1.5であったとすると、上位ワードは0000000000000001であり下位ワードは1000000000000000である。この場合下位ワードのＭＳＢが1であるから上位ワードに1を加算して下位ワードをすべて0にする。結果は上位ワードが0000000000000010、すなわち2となる。またアキュムレータ上の数が-1.5であったとすると、上位ワードは1111111111111110であり下位ワードは1000000000000000である。下位ワードのＭＳＢは1であるから上位ワードに1を加算して下位ワードを0にする。結果は1111111111111111 = -1となる。この結果については、右シフトの所で述べたことと同様の注意を払わなければならない。

丸めの処理を行う場合にはオーバフローの可能性に注意する必要がある。たとえば上位ワードの値が0111111111111111（=32767）で下位ワードのＭＳＢが1のとき通常の丸めの方法に従えば結果はオーバフローする。プロセッサによっては出力されるワードは1000000000000000となり、これは-32768である。このような場合通常

の方法では対応できない。その代わりに表現不可能な値になるのを避けるために値を飽和させる。

擬似コードの例について記述する際、以上で述べた丸めの処理はRND(.)と表現する。

VSCALE擬似コード

本箇所で導入する擬似コードの新モジュールでは、ブロック浮動小数点表現のためのベクトルスケーリングを行う。本モジュールの名前はVSCALEである。その目的は、ベクトルの要素の絶対値最大の値を要求どおり左づめに揃えるように、ベクトルスケーリングすることである。すなわち、正規化フォーマットで表現することである。本モジュールは第1要素が絶対値最大の値である事が既知であるベクトルで、あるいは絶対値最大の値の要素の位置が未知であるベクトルで使用する。VSCALEの入力は、スケーリングされる入力ベクトルIN、入力ベクトルの長さLEN、絶対値最大の値探索の探索長SLEN、左シフトの最大許容値MLSである。VSCALEの出力は、出力ベクトルOUT、入力ベクトルをスケーリングするために使用した左シフトの数NLSである。入力と出力ベクトルは同じ型であると仮定し、単精度ブロック浮動小数点（16ビット整数）あるいは倍精度ブロック浮動小数点（32ビット整数）のいずれかとする。単精度ベクトルの場合にはMLS=14、倍精度ベクトルの場合にはMLS=30である。時には、変数を表現するのに16ビットあるいは32ビットより小さな値を使用する事がある。例えば、14あるいは15ビットの精度をもついくつかの変数があるとする。このような場合、それぞれMLS=12あるいは13とする。この可能性のため、変数を正規化するのに左シフトでなく、右シフトを必要とする。このような場合、戻り値のNLSは負となる。例えば、NLS＝-1ならば、1ビットの右シフトが必要となる。本モジュールでは、アキュムレータ（AA0）はシフト演算でき少なくとも32ビットの精度をもつことを仮定している。もし、絶対値最大の要素が第1要素であるという事が既知であるならば、SLEN=1とする。そうでなければ、SLEN=LENとし、入力ベクトル全体に対して絶対値最大の値を探索する。

以下の擬似コードは、データが2の補数形式表現である。それぞれ、絶対値最大の値が正と負の場合を扱う。

```
        サブルーチンVSCALE(IN,LEN,SLEN,MLS,OUT,NLS)
        AA0=IN(1)                                   | 入力の正の絶対値最大の値を探す。
        AA1=IN(1)                                   | 入力の負の絶対値最大の値を探す。
        SLEN=1ならば、以下の3行をスキップする。
            I=2,3,...,SLENについて、以下の2行を実行する。
                IN(I)>AA0ならば、AA0=IN(I)とする。
                IN(I)<AA1ならば、AA1=IN(I)とする。
                                                    | ケース1:零入力ベクトル
        AA0=0かつ、AA1=0ならば、以下の3行を実行する。
            I=1,2,...,LENについて、OUT(I)=0とする。
            NLS=MLS+1                               | 0の場合は、1よりもさらに1ビット
            サブルーチン正常終了。                  | 左シフトできるものとする。
        NLS=0                                       | NLSを初期化する。
                                                    | ケース2あるいはケース3の決定
        AA0<0あるいは、AA1<-AA0ならば、以下の字下げ行を実行する。
                                                    | ケース2、負数が絶対値最大の値である。
            MAXI=-2^MLS                             | シフト後の仮数部の下限値
            MINI=2＊MAXI
            AA1<MINIならば、必要な右シフト量を求め要素をスケーリングするため、
            以下の2重字下げ行を実行する。
LOOP1R:         AA1=AA1>>1
                NLS=NLS-1                           | 負のNLS ==> 右シフト
                AA1 < MINIならば、LOOP1Rへジャンプする。
                I=1,2,3,...,LENについて、以下の1行を実行する。
                    OUT(I)=IN(I)>>-NLS
                サブルーチン正常終了。
```

```
LOOP1L:   AA1<MAXIならば、SCALE1へジャンプする。          | 左シフトの数を探す。
          AA1=AA1<<1
          NLS=NLS+1
          LOOP1Lへジャンプする。
SCALE1:   I=1,2,3,...,LENについて、以下の1行を実行する。
              OUT(I)=IN(I)<<NLS
          サブルーチン正常終了。
      そうでなければ、以下の字下げ行を実行する。
                                                | ケース3、正数が絶対値最大の値である。
          MINI=2^MLS                            | シフト後の仮数部の下限値
          MAXI=MINI-1                           | MLS=30ならば、2＊MINはオーバフローする。
          MAXI=MAXI+MINI                        | 仮数部の上限値
          AA0>MAXIならば、必要な右シフト量を求め要素をスケーリングするため、
          以下の2重字下げ行を実行する。
LOOP2R:       AA0=AA0>>1
              NLS=NLS-1
              AA0>MAXIならば、LOOP2Rへジャンプする。
              I=1,2,3,...,LENについて、以下の1行を実行する。
                  OUT(I)=IN(I)>>-NLS
              サブルーチン正常終了。
LOOP2L:   AA0≧MINIならば、SCALE2へジャンプする。
          AA0=AA0<<1
          NLS=NLS+1
          LOOP2Lへジャンプする。
SCALE2:   I=1,2,3,...,LENについて、以下の1行を実行する。
              OUT(I)=IN(I)<<NLS
          サブルーチン正常終了。
```

データのスケーリングを実際には行わずに、スケーリングに必要な左シフトのビット数だけを求めたい場合がある。以下のルーチンは、LENを除いてVSCALEと同じ入力を使用し、出力としてNLSのみを得る。それは入力ベクトルのスケーリングを省略する、しかし、他の点ではVSCALEと同じである。

```
      サブルーチンFINDNLS(IN,SLEN,MLS,NLS)
      AA0=IN(1)                                 | 入力の正の絶対値最大の値を探す。
      AA1=IN(1)                                 | 入力の負の絶対値最大の値を探す。
      SLEN=1ならば、以下の3行をスキップする。
          I=2,3,...,SLENについて、以下の2行を実行する。
              IN(I)>AA0ならば、 AA0=IN(I)とする。
              IN(I)<AA1ならば、 AA1=IN(I)とする。
                                                | ケース1:零入力ベクトル
      AA0=0かつ、AA1=0ならば、以下の2行を実行する。
          NLS=MLS+1                             | 0の場合は、1よりもさらに1ビット
          サブルーチン正常終了。                | 左シフトできるものとする。
      NLS=0                                     | NLSを初期化する。
                                                | ケース2あるいはケース3の決定
      AA0<0あるいは、AA1<-AA0ならば、以下の字下げ行を実行する。
                                                | ケース2、負数が絶対値最大の値である。
          MAXI=−2^MLS                           | シフト後の仮数部の下限値
          MINI=2＊MAXI
```

```
        AA1<MINIならば、必要な右シフト量を求めるため、以下の2重字下げ行を実行する。
LOOP1R:     AA1=AA1>>1
            NLS=NLS-1                                   | 負のNLS==>右シフト
            AA1<MINIならば、 LOOP1Rへジャンプする。
            サブルーチン正常終了。
LOOP1L:     AA1<MAXIならば、サブルーチン正常終了。          | NLSを探す。
            AA1=AA1<<1
            NLS=NLS+1
            LOOP1Lへジャンプする。
        そうでなければ、以下の字下げ行を実行する。
                                                        | ケース3、正数が絶対値最大の値である。
            MINI=2MLS                                   | シフト後の仮数部下限値
            MAXI=MINI-1                                 | MLS=30ならば、2＊MINはオーバフロー
                                                        |する。
            MAXI=MAXI+MINI                              | 仮数部の上限値
            AA0>MAXIならば、必要な右シフト量を求めるため、以下の2重字下げ行を実行する。
LOOP2R:     AA0=AA0>>1
            NLS=NLS-1
            AA0>MAXIならば、LOOP2Rへジャンプする。
            サブルーチン正常終了。
LOOP2L:     AA0≧MINIならば、サブルーチン正常終了。          | NLSを探す。
            AA0=AA0<<1
            NLS=NLS+1
            LOOP2Lへジャンプする。
```

１．３．２　乗算

2つの固定小数点数の乗算は32ビットとなる、通常、ＤＳＰの乗算レジスタに格納する。2つの固定小数点数がQnとQmフォーマットならば、乗算レジスタ内の結果はQ(n+m)フォーマットとなる。アキュムレータにそれを加える前に、前章の説明どおり、結果をシフトすることがある。

2つの浮動小数点数の乗算は、2つの仮数部の固定小数点乗算と2つのＮＬＳの加算となる。上記に述べたとおり、乗算結果はQ(n+m)フォーマットの32ビットワードである。乗算結果を浮動小数点数に逆変換をするならば、乗算結果を再正規化する必要があるかもしれない。例えば、2つの正の浮動小数点数を乗算する際、乗算結果のビット29あるいは30のどちらかは1でなければならない。もし、ビット30が0であるならば、再正規化が必要である。これは、さらに1ビットの左シフトが必要となる事を意味している。左シフトの後、乗算結果はQ(n+m+1)フォーマット表現となる。もし、乗算結果をスカラ浮動小数点で格納するならば、格納前に丸めをしなければならない。もし、乗算結果をブロック浮動小数点数で格納するならば、絶対値最大の値にしたがって全要素の再正規化が必要となる。

倍精度変数をこのコーデックの一部で使用するが、倍精度乗算はない。時々、倍精度変数の乗算をする。しかし、その場合は、上位16ビットのみ使用する。このことは擬似コードに記述してある。

１．３．３　加算

固定小数点どうしの加算では両方が同じQフォーマットでなくてはならない。一般に、どちらのダイナミックレンジが大きいかによって、どちらの値を適切なQフォーマットに変換するかが決まる。例えば、Q9とQ11フォーマットで格納してある値を加算するならば、Q11フォーマットの値は、Q9フォーマットの値と加算する前に2ビット右シフトしなければならない。

スカラ浮動小数点数の加算も同様である。2つの値は同じＮＬＳでなければならない。より大きなＮＬＳを持つ値は、もう一方の値のＮＬＳに合わせるため、右にシフトする必要がある。もし、加算結果の表現に17ビットを必要とするならば、アキュムレータの加算結果を1ビット右シフトし、16ビットに丸める。そして、新しいＮＬＳは前のフォーマットより1小さくなる。例として、ＮＬＳが5と7である2つの値を加算する場合を考える。ＮＬＳ

が7である値は、もう一方の値に加算する前に2ビット右シフトしなければならない。もし、両方の値が同じ符号を持つならば、2つの仮数の和は32767より大きな値を持つかも知れない。この場合、アキュムレータ内部の値を1ビットシフトし丸めを行う。加算結果のＮＬＳは4となる。もし、2つの値が異符号であるならば、アキュムレータの結果は値が16384より小さい仮数部となることがある。この場合、結果は16384以上になるまで左シフトで正規化し、ＮＬＳを左シフトの数だけ増加する。加算前のＮＬＳが5と7の例では、最終的にはＮＬＳは4以上6以下となる。

ブロック浮動小数点数の加算は絶対値最大の値に依存して複雑である。この場合、2つのベクトルが5と7のＮＬＳをもつならば、ＮＬＳが7のベクトルは2ビット右シフトしなければならない。次に対になる2つの値をそれぞれ加算する。最後に加算結果の絶対値最大の値で再正規化が必要かどうかを決定する。

１．３．４　除算

除算は加算や乗算のように頻繁には使用しない。唯一使用する除算はスカラ浮動小数点除算である。分子と分母はそれぞれ、正規化フォーマット表現とし、商も同様である。商のＮＬＳは分子のＮＬＳから分母のＮＬＳを減じ、14を加算することで求まる。この14を説明するために、分子は分母よりわずかに大きく両方のＮＬＳが0の場合を考える。この場合、商はＮＬＳが14であるとすれば、商は適切に正規化されていることがわかる。もし、分子の仮数部が分母の仮数部より小さいならば、分子を1ビット左シフトするべきである。そして、商のＮＬＳを計算するためそのＮＬＳに1をたす。これは、商の仮数部が正規化フォーマットになっている事を保証する。

除算結果の仮数部は17ビット結果を丸める完全16ビット精度を持たなければならない。そのような除算に対する擬似コードを以下に示す。

浮動小数点除算擬似コード

16ビットの固定小数点デバイスの浮動小数点除算の計算に、このルーチンを使用する。利用できる32ビットのアキュムレータが少なくとも1つあると仮定する。入力及び出力は全て16ビットワードである。

入力： NUM, NUMNLS, DEN, DENNLS

出力： QUO, QUONLS

機能：商を計算する。NUMとNUMNLSは、それぞれ分子の仮数部及びQフォーマットである。DENとDENNLSは、それぞれ分母の仮数部及びQフォーマットである。QUOとQUONLSは、それぞれ商の仮数部及びQフォーマットである。全てが正規化フォーマットであると仮定する。
DENは、0でないと仮定しておいて、0かどうかのテストはしない。

```
        サブルーチン DIVIDE(NUM,NUMNLS,DEN,DENNLS,QUO,QUONLS)
        SIGN=1                                  | 最初に商の符号ビットを
        P=NUM＊DEN                              | 決定する。
        P<0ならば、SIGN=−1とする。
        QUONLS=NUMNLS−DENNLS+14                 | 次にQUONLSを計算する。
        A0=| NUM|                               | A0は32ビットのアキュムレータである。
                                                | |NUM|は下位16ビットにある。
        A1=|DEN|                                | A1は16ビットか32ビットのレジスタである。
                                                | もし32ビットならば、|DEN|は下位16ビットにある。
        A0<A1ならば、以下の2行を実行する。
            QUONLS=QUONLS+1
            A0=A0<<1
        QUO=0                                   | 商初期化
        I=0                                     | ループカウンタ初期化
LOOP:   ここへ来た場合QUO=QUO<<1                | ロング除算ループ
        A0≧A1ならば、以下の2行を実行する。
            QUO=QUO+1
            A0=A0−A1
```

A0=A0<<1
I=I+1
I<15ならば、LOOPへジャンプする。
A0≧A1ならば、QUO=QUO+1とする。　　| 丸め処理
SIGN<0ならば、QUO=−QUOとする。　　| 符号処理

２．アルゴリズムの変更

２．１　バックワードベクトル利得適応器（ブロック２０）の変更

注：この節はＴＴＣ標準ＪＴ－Ｇ７２８の３．８節を参照している。読者はこの節を理解しようとする前に、３．８節をよく知っていなくてはならない。この節で述べる変更は、バックワードベクトル利得適応器での1ベクトルあたり1回の計算に関するものである。可能な限り、ＪＴ－Ｇ７２８と同じ表記をここでも用いている。

この節は、浮動小数点で実行されるような、ＪＴ－Ｇ７２８での1ベクトルあたり1回のバックワードベクトル利得適応器操作を簡単に説明する。さらに固定小数点プロセッサでもっと簡単にまた正確に実行されうる、数学的に等価な手法を説明する。この代替手法に必要な値の表は、この付属資料の５章に載っている。

浮動小数点操作は以下のように簡単に説明できる。変数 δ で表される内部状態変数配列GSTATEには、過去の10個のオフセットを除去した対数利得が含まれている。変数 $\delta(n)$ は、ベクトルnのオフセットを除去した対数利得を意味している。ベクトルnの対数利得予測器出力（ $\delta(n)$ の予測バージョン）は以下のようになる。

$$\hat{\delta}(n) = -\sum_{i=1}^{10} \alpha_i \delta(n-i) \tag{1}$$

図３－５／JT-G728で示されるように、 $\delta(n)$ を線形領域に変換する前に、32dBの利得オフセットが加えられ、 その結果について以下のことを確認する。

$$0 \le \hat{\delta}(n) + 32 \le 60 \tag{2}$$

これは次式と等価である。

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

線形領域での推定利得は以下のようになる。

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

$\sigma(n)$ の値は最初に、励振ＶＱターゲットベクトルを正規化するために使われる。コードブック探索が終了した後、選択された最適コードベクトルのスケーリングに $\sigma(n)$ が使われる。利得コードブックインデックス i と形状コードブックインデックス j がベクトルnで選択されたとすると、励振ベクトル $e(n)$ は以下のようになる。

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

ここで、 y_j はj番目の形状コードベクトルであり、 g_i は利得コードブックの i 番目の利得レベルである。励振ベクトル $e(n)$ は、 $\delta(n)$ の計算に用いられる。まず最初に、 $e(n)$ のＲＭＳ値の2乗（即ち $e(n)$ の“パワ”）を計算する。それは以下のようになる。

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

任意のベクトル x についても、記号 $P[x]$ を使って x のパワを表し、それはxのベクトルの次元で割ったxのエネルギであると定義される。 $P[e(n)]$ の対数領域のdB値に変換する前に、それが1より小さい場合 $P[e(n)]$ を1にクリップする。従って、 $P[e(n)]$ の許容範囲は、以下のようになる。

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

これは、対数変換におけるオーバフローもしくは非常に小さなdB値を避けるためである。このような範囲の限定は図３－５／JT-G728では明示されてはないが、ＪＴ－Ｇ７２８の５．７節の”擬似コード”では記述されている。ベクトルnでのオフセット除去対数利得（dB値で）は、以下のようになる。

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

式 (7) は以下のことを示している。

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

次に式(8) で求めた $\delta(n)$ は、以降の励振利得を予測し、対数利得予測器係数の更新に使用される。これでバックワードベクトル利得適応器での浮動小数点操作の簡単な説明を終える。

ここからは固定小数点実現における数学的に等価な手法を説明する。 y_{jk} を形状コードブックのj番目のコードベクトルのk番目の要素とする。式(5) と(6) より以下のようになる。

$$P[e(n)] = \frac{1}{5}\sum_{k=1}^{5} (\delta(n) g_i y_{jk})^2 \tag{10}$$

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

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

式(12) を式(8) に代入すると、以下のようになる。

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

ここで、式(4) を使って $\delta(n)$ を以下のように表すことができる。

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

言い替えると、 $\delta(n)$ とは単に予測された対数利得 $\hat{\delta}(n)$ に、次の2つの”補正項”を加えたものである。

(1) $20\log_{10}|g_i|$ は、利得コードブックから選択された最適利得レベルのdB値である。

(2) $10\log_{10} P[y_j]$ は、形状コードブックから選択された最適形状コードベクトルのパワのdB値である。

（ある意味では、これは利得に関する予測符号器であるが、対数領域で動作する。）

付図Ｇ－１／JT-G728に、この数学的に等価な手法のブロック図を示す。 $|g_i|$ は4個、 $P[y_j]$ は128個しかないので、それらのdB値を前もって計算し、それらを対数利得表（付図Ｇ－１／JT-G728のブロック９３，９４）に格納できる。これらの2つの表はこの付属資料の５章に載っている。

遅延ユニット９１，９２で、前のベクトルの励振コードブック探索において選択された最適利得と形状コード

ブックインデックスが利用可能になる。これら2つのインデックスは、ブロック９３，９４の対数利得表から、$20\log_{10}|g_i|$ と $10\log_{10} P[y_j]$ の値を読み出すために使われる。1サンプル遅延ユニット９５は、直前に予測された（範囲限定の）対数利得 $\delta(n-1)$ を保持している。加算器９６はブロック９３，９４，９５の出力を加え、式(14)に従って、クリップされいてない $\delta(n-1)$ を作り出す。そしてリミッタ９７は、それが-32dB以下である場合に-32dBで加算器９６の出力をクリップすることにより、不等式(9) を満たす。

リミッタ９７の出力は、図３－５／JT-G728の加算器４２の出力に数学的に等価である。故に付図Ｇ－１／JT-G728のブロック４３～４６は、図３－５／JT-G728のそれに相当するものと同一である。対数利得リミッタ９８の操作は、許容範囲が32dBまでシフトダウンされたことを除いては、図３－５／JT-G728のリミッタ４７に等しい。加算器９９はブロック４１に保持されている32dBの対数利得オフセット値を、対数利得リミッタ９８の出力に加算する。その結果生じる対数利得の値は、図３－５／JT-G728でのものと同一の逆対数計算器４８により、線形領域に変換される。ここまでで、固定小数点実現での数学的に等価な手法の説明を終える。

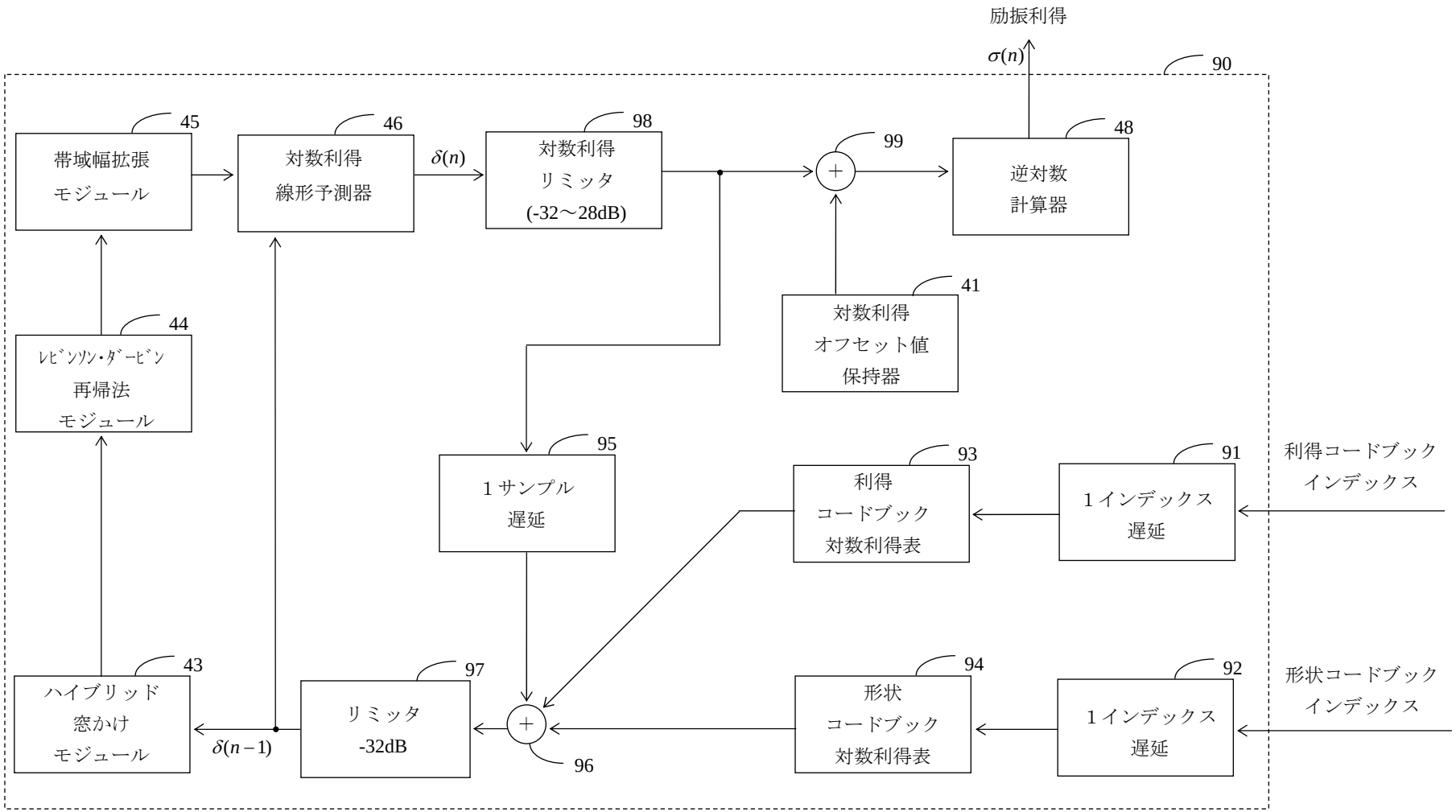
付図Ｇ－１／JT-G728で示される等価な手法には、図３－５／JT-G728の元の手法と比べて、次の2つの重要な利点がある。

(1) 対数関数（図３－５／JT-G728のブロック４０）を計算する必要がなくなる。ＤＳＰによる実現では、対数関数は、普通、べき級数展開を用いて計算されるので、一般的に多数の命令サイクルを要する。故に対数計算をテーブルルックアップに置き換えることは、ＤＳＰサイクルのかなりの節約になる。テーブルの内容もまた必要な最大精度まで前もって計算することができる。

(2) 固定小数点プロセッサが使用されるときには、元の手法よりも正確な値が期待できる。バックワード適応のために、利得適応プロセスにはフィードバックループが存在する。図３－５／JT-G728では、このフィードバックループは非常に長い。それは逆対数計算器４８から利得調整ユニット２１（図３－１／JT-G728）まで進み、その後ブロック６７，３９，４０，４２～４８まで戻る。このループ内の計算が多ければ多いほど、有限精度による演算誤差がフィードバックループに、ますます蓄積される。これは特に、固定小数点プロセッサが対数関数で最大可能精度を達成できない場合に顕著である。それとは対照的に、付図Ｇ－１／JT-G728のフィードバックループはできるだけ小さくしてある。利得調整ユニット、$e(n)$ のエネルギ及びパワ計算、対数計算器、対数利得オフセットを再蓄積するための加算器でさえも、ここではフィードバックループ外である。両手法とも共通のブロック４３～４６を除いて、フィードバックループには、リミッタが2つしかない、従って固定小数点プロセッサによって非常に高精度に実現可能である。

この変更によって、ＪＴ－Ｇ７２８の固定及び浮動小数点実現装置同士の相互接続性が高められる。この新しい手法の主なデメリットは、ＲＯＭメモリの付加ワードが必要であるということである。形状ベクトルが128個、利得ベクトルが4個なので、必要な付加メモリは128+4=132ワードである。これは、ＪＴ－Ｇ７２８で既に必要とされたＲＯＭ領域の中で非常に僅かなものである。

この新しい手法は、バックワードベクトル利得適応器（ブロック２０，３０）への入力を、$e(n)$ 利得及び形状コードブックインデックス i, j に変更している。この事実を反映するために、図２－１／JT-G728、図３－１／JT-G728、図４－１／JT-G728は、バックワード利得適応器が励振ＶＱコードブックブロックから入力を得ることができるように、ここで書き直されるべきである。しかしながら、必要な変更は些細であり、以上の記述で明白なのでそのような修正図は省略した。



付図Ｇ－１／JT-G728　固定小数点実現用バックワードベクトル利得適応器
(ITU-T G.728)

２．２　レビンソン・ダービン再帰法モジュールに対する変更

この節では、ＴＴＣ標準ＪＴ－Ｇ７２８に用いられたレビンソン・ダービン再帰法モジュールに対する変更について記述する。ブロック３７，４４および５０の3つのモジュールがあり、それぞれ、聴覚重み付けフィルタ、対数利得線形予測器、合成フィルタとして用いる。より詳細な情報を得るためにＪＴ－Ｇ７２８の５．５節および５．６節を参照すること。この節では５．６節にあるブロック５０に関する擬似コードを例として用い、固定小数点化のための変更について説明する。同様の変更は、ブロック３７（聴覚重み付けフィルタ）と４４（対数利得予測器）にも行う必要がある。まず、浮動小数点擬似コードのリストを示す。

```
RTMP(LPC+1)=0 ならば、LABELへジャンプする。          | 0ならばスキップ
RTMP(1)≤0ならば、LABELへジャンプする。               | 0信号ならばスキップ
RC1=-RTMP(2)/RTMP(1)
ATMP(1)=1.                                          | 1次予測器
ATMP(2)=RC1
ALPHATMP=RTMP(1)+RTMP(2)＊RC1
ALPHATMP≤0 ならば、LABELへジャンプする。            | 異常状態ならばアボート
MINC=2,3,4,...,LPCについて、以下を実行する。
    SUM=0.
    IP=1,2,3,...,MINC について、以下の2行を実行する。
        N1=MINC-IP+2
        SUM=SUM+RTMP(N1)＊ATMP(IP)
    RC=-SUM/ALPHATMP                                | 反射係数
    MH=MINC/2+1                                     |
    IP=2,3,4,...,MH について、以下の4行を実行する。
        IB=MINC-IP+2
        AT=ATMP(IP)+RC＊ATMP(IB)                    |
        ATMP(IB)=ATMP(IB)+RC＊ATMP(IP)              | 予測係数の更新
        ATMP(IP)=AT                                 |
    ATMP(MINC+1)=RC
    ALPHATMP=ALPHATMP+RC＊SUM                       | 予測残差エネルギ
    ALPHATMP≤0 ならば、LABELへジャンプする。        | 異常状態ならばアボート
次のMINCに対しても、上記の演算を繰り返す。
プログラム正常終了。                                | ここに来た場合、正常終了
LABEL: ここに来た場合、異常状態であるので、ブロック５１をスキップし、合成フィルタ係数は更新
       しない。(前適応周期の合成フィルタ係数を使用する。)
```

この擬似コードで参照されている浮動小数点変数について考慮することから始める。浮動小数点変数としてはRC, RC1, RTMP, SUM, ALPHATMPとATMPがある。（コード内の他の変数、MINC, IP, IBとN1は整数型のインデックスである。）

RCは、反射係数を示していて、このモジュールの中間変数として計算される。反射係数は次のような性質を持っている。安定なＬＰＣフィルタのRCは1を越えない。そこでRCは1ビットの符号ビットと15ビットの小数部を持ったQ15フォーマットによって表現する。

RCはそれぞれの繰り返し毎に計算され、その繰り返しの中では再び使われるが、その後は使われない。唯一の例外は復号器の合成フィルタのＬＰＣ分析で、後でポストフィルタに使うためにRC1としてセーブされる。記憶領域を節約するため、RCの他の全ての変数は、次の繰り返しで上書きされるよう同じ位置に書かれる。これは元の浮動小数点擬似コードからの変更を表している。しかし、出力に対しては何の影響も与えないし、浮動小数点実現と同様に用いることができる。

RTMPは自己相関関数値である。RTMPの値は非常に大きなダイナミックレンジを持っているため、ブロック浮動小数点として保持する必要がある。このことは、全ての変数が同一の2のべき乗の値で正規化されることを意味している。理論的には、RTMP(1)が絶対値最大で、しかも正値であることが知られている。RTMPの絶対値最大の

値を0.5から1.0の間の値に表現すれば良い。このため、RTMPは全てQ15フォーマットによって表現可能である。ハイブリッド窓かけでは、RTMPはブロック浮動小数点を用いて表現されるが、ダービン再帰法では仮数だけが必要である。

RTMP(LPC+1)について他に注意が必要である。第1行に示されている通り、もし、この変数の値が0の場合、このモジュールは終了しなければならない。

もし、RTMP(LPC+1)が16ビットの整数で表現されているとき、この条件は、同じRTMP(LPC+1)が、浮動小数点実現における32ビットの浮動小数点型で表現された場合より多く発生すると思われる。これは、相互接続性に問題が生じることを示している。RTMP(LPC+1)の計算においては、前段のモジュール（ハイブリッド窓かけ、ブロック４９）において、この値は、どのような固定小数点ＤＳＰでも備わっている、少なくとも32ビットのアキュムレータによって累算される。計算が完了したときにアキュムレータの32ビットの値が0かどうかテストすることを提案する。この変更によって、精度不足によって0判定が早く起こるために生じる誤終了（と、先に述べた相互接続性の問題）は避けられる。新しいコードにある論理変数ILLCONDは真か偽かをテストされる。この変数はハイブリッド窓かけモジュールのRTMP(LPC+1)のテスト結果に依存する。ILLCONDは後でこのブロックの出力値を用いるべきか無視すべきかを示す出力変数として用いられる。

10回目の繰り返しの後、異常状態が生じる可能性がある。このような場合、短期適応ポストフィルタの新しい予測係数は確定され、かつ有効であるが、50次の合成フィルタ係数は無効である。第2の論理変数ILLCONDPはポストフィルタの係数の状態を示している。

次にSUMとALPHATMPについて説明する。両方とも累算された値であるが、決して乗算された値ではない。この値は32ビットのアキュムレータに保持される。しかしながら、この2つの変数はRCを計算するために除算される。SUMとALPHATMPは両方とも除算のために16ビットの固定小数点に変換される。除算結果は16ビットのQ15固定小数点フォーマットで表現され、RCに割り当てられる。変数SUMは固定小数点擬似コードには明示的には現れない。これは、32ビットフォーマットにおいてはアキュムレータAA0、16ビットフォーマットでは変数SIGNである。

ALPHATMPはアキュムレータで累算された32ビットの数であることに注意すること。しかしながら、実際のＤＳＰによる実現においては、ALPHATMPが再び更新される前に、アキュムレータが他の計算のために必要とされるため、次の高次の繰り返しのためにALPHATMPをメモリに保存しておく必要がある。それゆえ、丸められた上位16ビットだけをセーブ・ロードすれば、32ビット全部を扱うよりＤＳＰの実行サイクルを節約できる。実際、ALPHATMPの更新毎に上位16ビットだけをセーブしてもコーデックの性能を劣化させないことが判明した。従って、ALPHATMPの更新の後、ALPHATMPの上位16ビットだけをセーブすることにした。ALPHATMPが16ビット表現か32ビット表現かを区別するため、16ビットのときだけ、ALPHATMPという名前を擬似コード上で用いる。32ビットのときはアキュムレータを直接参照する。

予測器の係数を表すベクトル変数ATMPの説明が残っている。全てのシミュレーションにおいて、ATMPの絶対値最大の値は4未満であることが観測されている。これはQ13フォーマットを使えば良いことを示唆している。しかしながら、ダービン再帰法ではQ13フォーマットでは浮動小数点実現との間での相互接続性を満足しないことがシミュレーションによって示されている。固定小数点、浮動小数点実現間のより優れた相互接続性を得るために、オーバフローが生じる場合を除きQ15フォーマットを使う方が良いことがわかった。

ＤＳＰチップ上では、ATMP(IB)あるいはATMP(IP)の計算結果は初期状態ではアキュムレータにある。ほとんどの16ビット固定小数点ＤＳＰのアキュムレータは少なくとも32ビット幅である。ATMP(IB)とATMP(IP)の値は16ビット精度に丸められる。固定小数点コードではATMP(IB)あるいはATMP(IP)が計算された後、オーバフローが生じたかどうかを検出するために、アキュムレータがガードビットを持っているかオーバフローフラグが提供されていることが重要である。

ATMPのフォーマットを選択するに当たっては以下の方法を用いる。繰り返しはMINCの値にしたがって番号付けされる。まず、MINC=2としQ15フォーマットのATMPとして開始する。もし、オーバフローが生じなければ、つまり、全てのIPに対し|ATMP(IP)|<1であれば、ATMPの最終的な表現方法はQ15とする。そうで無い場合としては、ある繰り返しの中でオーバフローを生じる場合がある。例えばK番目の繰り返しでオーバフローが生じたとすると、K番目とK-1番目のどちらかの繰り返しの間に計算されたATMPの値を全て右シフトによってQ14フォーマットに変換する。K番目の繰り返しはQ14フォーマットで再度実行される。それに続くK+1番目の繰り返しは同様にQ14フォーマットで計算される。Q14フォーマットを用いてもオーバフローが生じる可能性があるが、この場合は、同様の手順によって、Q13フォーマットで計算を継続する。経験的には、このようなオーバフローはQ13

フォーマットでは決して生じないことがわかっている。オーバフローを避けることにより正確な結果を得ることが他のフォーマットを用いる理由である。このブロックを出る前の最終的なATMPの表現はQ13, Q14あるいはQ15のいずれかになる。

帯域幅拡張動作の後、帯域幅拡張モジュール（ブロック３８，４５，５１と８５）の出力におけるフィルタ係数は、常にQ14フォーマットで表現されることがわかっている。Q14フォーマットを用いて得られる相互復号SNRは、Q15フォーマットを用いることが可能であったとしても、改善されないことがわかっているため、それらの3つの帯域幅拡張モジュールは常に出力の係数配列を、入力の係数配列（つまり、レビンソン・ダービン再帰法モジュールの出力）がQ13, Q14あるいはQ15であるかに関わらず、Q14に変換する。このことは、レビンソン・ダービン再帰法モジュールがQ13あるいはQ15の係数配列を出力したときは、対応する帯域幅拡張モジュールに知らせ、追加のシフト処理を配列に対して実行しQ14に変換する必要があることを意味している。この理由から、固定小数点によるレビンソン・ダービン再帰法モジュールは以下のようになる。このモジュールの出力の一つとしてフラグNLSATMPを追加した。

復号器では、レビンソン・ダービン再帰法は、10次の予測係数が得られた時点で中断される。これらの値は適応ポストフィルタのために保存される。その結果、2つの開始条件が考えられる。通常は、再帰計算はMINC0=1から開始される。復号器に限れば、他にMINC0=10という可能性がある。後者の場合NRSとALPHATMPの値は必ずセーブすること。また、NLSTMPの値は、帯域幅拡張モジュールが実行されるときICOUNT=3までは保存しておく必要がある。

最終的には、このルーチンは3つのアキュムレータを用いる。第3のアキュムレータ、AA2、はRCを17ビットの精度で保持し、最新の予測係数を更新するために用いる。

以下レビンソン・ダービン再帰法の固定小数点擬似コードを示す。

```
        MINC0>1 ならば、RECURSIONへジャンプする。
        MINC0=1                                    | 復号器のみの初期化
        ILLCONDP=偽                                |
        ILLCOND=真ならば、FAILEDへジャンプする。   | RTMP(LPC+l)が0ならば
                                                   | スキップ
        RTMP(1)≤0 ならば、FAILEDへジャンプする。   | 0信号ならばスキップ
        NRS=0                                      | 初期はQ15フォーマット
        DEN=RTMP(1)                                | 1次予測計算
        NUM=RTMP(2)
        NUM<0 ならば、NUM=-NUMとする。
        SIMPDIV(NUM,DEN,AA0)をコールする。         | |RTMP(2)|/RTMP(1)
        AA0=AA0<<15
        RC1=RND(AA0)
        RTMP(2)>0 ならば、RC1=-RC1とする。         | 符号情報を加える。
        RC=RC1                                     | 1次予測係数
        ATMP(2)=RC1
        AA0=RTMP(1)<<16
        P=RTMP(2)＊RC
        AA0=AA0+(P<<1)
        ALPHATMP=RND(AA0)                          | DSPアキュムレータの上位16
                                                   | ビットをメモリにセーブ
RECURSION:
        MINC=MINC0+1,MINC0+2,...,LPCについて、以下の字下げ行を実行する。
            AA0= 0
            IP=2,3,...,MINC について、以下の3行を実行する。
                N1=MINC-IP+2
                P=RTMP(N1)＊ATMP(IP)
                AA0=AA0+P                          | SUMに対しては32ビット
```

```
AA0=AA0<<1
AA0=AA0<<NRS
AA1=RTMP(MINC+1)<<16
AA0=AA0+AA1
SIGN=RND(AA0)                                   | 上位ワードSIGNをセーブ
NUM=SIGN
NUM<0 ならば、NUM=-NUMとする。
NUM≧ALPHATMPならば、FAILEDへジャンプする。
SIMPDIV(NUM,ALPHATMP,AA0) をコールする。       | RCを得るため除算
AA2=AA0<<15                                     | 17ビットのRCをAA2にセーブ
RC=RND(AA2)
SIGN>0ならば、RC=-RCとする。
                                                | ALPHATMPの更新
AA1=ALPHATMP<<16
P=RC＊SIGN
AA1=AA1+(P<<1)
AA1≦0ならば、FAILEDへジャンプする。
ALPHATMP=RND(AA1)
MH=MINC/2+1                                     | MINC/2の小数部の切り捨て; MHは整数型
                                                | 予測器の係数の更新を開始。
IP=2,3,4,....,MHについて、以下の2重字下げ行を実行する。
    IB=MINC-IP+2
    AA0=ATMP(IP)<<16                            | AA0の上位ワードをロード
    P=RC＊ATMP(IB)                              | Q15 RC1ビット左シフト
    AA0=AA0+(P<<1)
    AA0 がオーバフローならば、以下の3重字下げ行を実行する。
        NRS=NRS+1
        LP=2,3,...,MINC について、ATMP(LP)=ATMP(LP)>>1とする。
        AA0=ATMP(IP)<<16                        | まずATMPを再スケール
        P=RC＊ATMP(IB)                          | 次にオーバフローした
        AA0=AA0+(P<<1)                          | AA0の再計算
    AA1=ATMP(IB)<<16
    P=RC＊ATMP(IP)
    AA1=AA1+(P<<1)
    AA1 がオーバフローならば、以下の3重字下げ行を実行する。
        NRS=NRS+1
        LP=2,3,...,MINC について、ATMP(LP)=ATMP(LP)>>1とする。
        AA0=ATMP(IP)<<16                        | まずATMP(IP)の再スケール
        P=RC＊ATMP(IB)                          |
        AA0=AA0+(P<<1)                          | 次のAA0の再計算
        AA1=ATMP(IB)<<16                        | 次のATMP(IB)再スケール
        P=RC＊ATMP(IP)                          | オーバフローしたAA1の
        AA1=AA1+(P<<1)                          | 再計算
    ATMP(IP)=RND(AA0)
    ATMP(IB)=RND(AA1)
                                                | ATMP(MINC+1)の更新
AA0=AA2>>NRS                                    | 17ビットのRCをAA2に保持
AA0=RND(AA0)                                    | AA0の下位ワードを出力
SIGN>0ならば、AA0=-AA0とする。
```

```
        ATMP(MINC+1)=AA0                                    | ATMPに下位ワードを記憶
    次のMINCに対しても、上記の演算を繰り返す。
    NLSATMP=15-NRS
    NLSATMP<13ならば、FAILEDにジャンプする。
    プログラム正常終了。
                                                            | 再帰法の終了
                                                            | ここに来た場合、
                                                            | 正常終了
FAILED: ILLCOND=真とする。
    MINC≤10ならば、ILLCONDP=真とする。
    ここに来た場合、異常状態であるので、ブロック５１をスキップし、合成フィルタ係数は更新しない。
    (前適応周期の合成フィルタ係数を使用する。)
```

参照しやすくするため、次の表に上記の擬似コード中の全ての変数とそれらの表現フォーマットを示す。

付表Ｇ－２－１／JT-G728
(ITU-T G.728)

変数	フォーマット	サイズ	一時／常置	旧変数／新変数
AA0, AA1, AA2, P	DP-int	1	一時	新変数
ALPHATMP	SFL	1	一時	旧変数
ATMP	Q13/Q14/Q15	51	常置	旧変数
IB	int	1	一時	旧変数
ILLCOND	論理	1	常置	新変数
ILLCONDP	論理	1	常置	新変数
IP	int	1	一時	旧変数
LP	int	1	一時	新変数
MH	int	1	一時	旧変数
MINC	int	1	一時	旧変数
NLSATMP	int	1	一時	新変数
NRS	int	1	一時	新変数
NUM	int	1	一時	新変数
RC	Q15	1	一時	新変数
RC1	Q15	1	一時	新変数
RTMP	Q15	51	常置	旧変数
SIGN	int	1	一時	新変数

（注）SFL=16ビットスカラ浮動小数点、DP-int＝アキュムレータまたは乗算レジスタ(AA0, AA1, AA2, P)のような32ビットレジスタ、int=16ビット整数、Q14/Q15＝これらの表現のうち1つを用いた16ビット整数。

上記のコードはブロック５０について記述されたものであり、ブロック５０と関連した変数名を使用した。しかし、ブロック３７，４４にもそれを使用することができる。次の表では、ブロック５０に特定化した変数名を他の各ブロックに特定化した変数名に書き換えている。

ブロック５０	ブロック３７	ブロック４４
ATMP	AWZTMP	GPTMP
ILLCOND	ILLCONDW	ILLCONDG
NLSATMP	NLSAWZTMP	NLSGPTMP
RTMP	R	R

上記のコードでは、他のアルゴリズムで使用される除算アルゴリズムとは異なる、より簡単な除算アルゴリズムを用いている。それは上記でSIMPDIVとして表されている。SIMPDIVの擬似コードを以下に示す。入力は、NUMおよびDENであり、両方とも16ビット整数である。出力は、下位17ビットに結果を持つAA0である。

```
        サブルーチン SIMPDIV(NUM,DEN,AA0)
        AA0=0
        AA1=NUM
        K=0
LOOP:   AA0=AA0<<1
        AA1=AA1<<1
        AA1≥DENならば、AA1=AA1-DEN、AA0=AA0+1とする。
        K=K+1
        K<16ならば、LOOPへジャンプする。
```

３．ＪＴ－Ｇ７２８の他のモジュールについての擬似コード

この章では、ＪＴ－Ｇ７２８の他のモジュールの擬似コードを記述する。レビンソン・ダービン再帰法の擬似コードは、バックワードベクトル利得適応器のアルゴリズムの変更と共に前章で示した。おのおののモジュールに関して、浮動小数点擬似コードを最初に記述し、次に注釈と固定小数点擬似コードを記述する。次の表は、符号器における特定のモジュールの擬似コードを探すための参照用として使用できる。

全てのブロックの擬似コードは、ビットイグザクトな仕様を規定している。そして、この擬似コードが、固定小数点版ＪＴ－Ｇ７２８の最終的な定義である。この擬似コードを変更した場合には、誤ったシミュレーション結果や実現結果を生じることがある。

付表Ｇ－３－１／JT-G728（１／２）　固定小数点ＪＴ－Ｇ７２８ブロック番号、説明
(ITU-T G.728)　および擬似コード名

ブロック	説　　明	擬似コード
1	入力ＰＣＭフォーマット変換	不要
2	ベクトルバッファ	不要
3	聴覚重み付けフィルタ適応器	ブロック４５使用
4	聴覚重み付けフィルタ	ブロック４
5-7	ＺＩＲ／メモリ更新切替器	不要
8	局部復号器	下記の詳細ブロック参照
9	ＺＩＲ計算中の合成フィルタ	ｂｌｏｃｋｚｉｒ
10	ＺＩＲ計算中の聴覚重み付けフィルタ	ｂｌｏｃｋｚｉｒ
9,10	メモリ更新中のブロック９，１０	ブロック９
11	ＶＱターゲットベクトルの計算	ブロック１１
12	インパルス応答ベクトルの計算	ブロック１２
13	時間反転畳込み	ブロック１３
14	形状コードベクトルの畳込み	ブロック１４
15	コードブックエネルギ表計算	ブロック１４
16	ＶＱターゲットベクトルの正規化	ブロック１６
17	ＶＱ探索誤差計算器	ブロック１７
18	最適コードブックインデックス選択器	ブロック１７
19	励振ＶＱコードブック	ブロック１９
20	バックワードベクトル利得適応器	下記の詳細ブロック参照
21	利得調整ユニット	ブロック１９
22	合成フィルタ	ブロック９使用
23	合成フィルタ適応器	ブロック４９- ５１参照
24	コードブック探索モジュール	ブロック１２- １８参照
28	出力ＰＣＭフォーマット変換	不要
29	復号器励振コードブック	ブロック１９使用
30	復号器バックワード利得適応器	ブロック２０と同じ

表Ｇ－３－１／JT-G728（２／２）　固定小数点ＪＴ－Ｇ７２８ブロック番号、説明
(ITU-T G.728)　　および擬似コード名

ブロック	説　　明	擬似コード
31	復号器利得調整ユニット	ブロック１９使用
32	復号器合成フィルタ	ブロック３２
33	復号器合成フィルタ適応器	ブロック２３と同じ
34	ポストフィルタ	ブロック７１- ７７参照
35	ポストフィルタ適応器	ブロック８１- ８５参照
36	$W(z)$ 用ハイブリッド窓かけ	ブロック３６
37	$W(z)$ 用ダービン再帰法	２章参照
38	$W(z)$ 係数計算器	ブロック３８
43	$GP(z)$ 用ハイブリッド窓かけ	ブロック４３
44	$GP(z)$ 用ダービン再帰法	２章参照
45	$GP(z)$ 帯域幅拡張	ブロック４５
46	対数利得線形予測器	ブロック４６
48	逆対数計算器	ブロック４６
49	$A(z)$ 用ハイブリッド窓かけ	ブロック４９
50	$A(z)$ 用ダービン再帰法	２章参照
51	$A(z)$ 係数計算器	ブロック５１
71-77	ポストフィルタ内のブロック	対応するブロック
81-85	ポストフィルタ適応器内のブロック	対応するブロック
91	利得コードブックインデックス遅延ユニット	不要
92	形状コードブックインデックス遅延ユニット	不要
93	利得コードブック対数利得表	付表Ｇ－５－１／JT-G728 参照
94	形状コードブック対数利得表	付表Ｇ－５－２／JT-G728 参照
95	対数利得の1サンプル遅延	不要
96	対数利得更新用加算器	ブロック４６
97	対数利得リミッタ -32dB	ブロック４６
98	対数利得リミッタ：-32 から28dB	ブロック４６
99	利得オフセット復元用加算器	ブロック４６

３．１　ブロック４－聴覚重み付けフィルタ

以下に、聴覚重み付けフィルタによる入力音声のフィルタリング、ブロック４の浮動小数点擬似コードを示す。

```
K=1,2,...,IDIMについて、以下を実行する。
    SW(K)=S(K)
    J=LPCW,LPCW-1,...,3,2 について、以下の2行を実行する。
        SW(K)=SW(K)+WFIR(J)＊AWZ(J+1)          | フィルタの全零部分
        WFIR(J)=WFIR(J-1)                      |
    SW(K)=SW(K)+WFIR(1)＊AWZ(2)                | 最後の1サンプルの処理
    WFIR(1)=S(K)                               |
    J=LPCW,LPCW-1,...,3,2 について、以下の2行を実行する。
        SW(K)=SW(K)-WIIR(J)＊AWP(J+1)          | フィルタの全極部分
        WIIR(J)=WIIR(J-1)                      |
    SW(K)=SW(K)-WIIR(1)＊AWP(2)                | 最後の1サンプルの処理
    WIIR(1)=S(K)                               |
次のKに対しても、上記の演算を繰り返す。
```

この擬似コードの固定小数点版に対しては、ＮＬＳ値が、WFIR, WIIRの両者に関連する。演算は入力音声がWFIRと同じＮＬＳを、計算結果はWIIRと同じＮＬＳを持つように行わなければならない。すなわち、WFIRとWIIRのＮＬＳ値は、入力音声と同じ値で固定される。AWZとAWPはQ14である。入力音声に対するＮＬＳの値は2である。異なる入力フォーマット（16ビット均一、μ則）はQ2フォーマットで表わされる範囲［-4096, +4095.75］に変換するものと仮定する。

Kビットの均一入力に対しては、データは16ビットワードK_BIT_SAMPLEの下位Kビットにおかれると仮定する。適切な表現は以下で与えられる。

NLS=15-K
S=K_BIT_SAMPLE<<NLS

16ビット均一入力信号（16_BIT_SAMPLE）に対しては、1ビットの右シフトが要求される。

S=16_BIT_SAMPLE>>1

μ則ＰＣＭ（MULAW_SAMPLE）に対しては、最大サンプル値が4015.5であり、これを8031となるようQ1フォーマットで表現されていると仮定する。Q2フォーマットに変換するために1ビットの左シフトが必要になる。

S=MULAW_SAMPLE<<1

以下に、聴覚重み付けフィルタによる入力音声のフィルタリング、ブロック４の固定小数点擬似コードを示す。

```
K=1,2,...,IDIMについて、以下を実行する。
    AA0=S(K)
    AA0=AA0<<14
    J=LPCW,LPCW-1,...,3,2 について、以下の2行を実行する。
        AA0=AA0+WFIR(J)＊AWZ(J+1)          | フィルタの全零部分
        WFIR(J)=WFIR(J-1)                  |
    AA0=AA0+WFIR(1)＊AWZ(2)                | 最後の1サンプルの処理
    WFIR(1)=S(K)                           |
    J=LPCW,LPCW-1,...,3,2 について、以下の2行を実行する。
        AA0=AA0-WIIR(J)＊AWP(J+1)          | フィルタの全極部分
        WIIR(J)=WIIR(J-1)                  |
    AA0=AA0-WIIR(1)＊AWP(2)                | 最後の1サンプルの処理
    AA0=AA0>>14                            |
    AA0>32767ならば、AA0=32767とする。     | 後段の乗算器入力に対する
    AA0<-32768ならば、AA0=-32768とする。   | 飽和モード
    WIIR(1)=AA0                            | 16ビット下位ワードをセーブ
    SW(K)=AA0                              | SWはQ2フォーマット
次のKに対しても、上記の演算を繰り返す。
```

３．２　Blockzirー零入力応答計算中の合成フィルタ、聴覚重み付けフィルタ

以下に、零入力応答計算中のブロック９（合成フィルタ）の浮動小数点擬似コードを示す。

```
K=1,2,...,IDIMについて、以下を実行する。
    TEMP(K)=0.
    J=LPC,LPC-1,...,3,2について、以下の2行を実行する。
        TEMP(K)=TEMP(K)-STATELPC(J)＊A(J+1)  | 積和
        STATELPC(J)=STATELPC(J-1)            | メモリシフト
```

```
        TEMP(K)=TEMP(K)-STATELPC(1)＊A(2)          | 最後の1サンプルの処理
        STATELPC(1)=TEMP(K)                       |
    次のKに対しても、上記の演算を繰り返す。
```

以下に、零入力応答計算中のブロック１０（聴覚重み付けフィルタ）の浮動小数点擬似コードを示す。

```
K=1,2,...,IDIMについて、以下を実行する。
    TMP=TEMP(K)
    J=LPCW,LPCW-1,...,3,2について、以下の2行を実行する。
        TEMP(K)=TEMP(K)+ZIRWFIR(J)＊AWZ(J+1)   | フィルタの全零部分
        ZIRWFIR(J)=ZIRWFIR(J-1)                |
    TEMP(K)=TEMP(K)+ZIRWFIR(1)＊AWZ(2)         | 最後の1サンプルの処理
    ZIRWFIR(1)=TMP                             |
    J=LPCW,LPCW-1,...,3,2 について、以下の2行を実行する。
        TEMP(K)=TEMP(K)-ZIRWIIR(J)＊AWP(J+1)   | フィルタの全極部分
        ZIRWIIR(J)=ZIRWIIR(J-1)                |
    ZIR(K)=TEMP(K)-ZIRWIIR(1)＊AWP(2)          | 最後の1サンプルの処理
    ZIRWIIR(1)=ZIR(K)                          |
次のKに対しても、上記の演算を繰り返す。
```

固定小数点擬似コードでは、STATELPCは、分割ブロック浮動小数点表現であり、これに関連したNLSSTATE値を有する。零入力のため、NLSSTATE値を入力のＮＬＳ値に一致させる必要はない。A( ), AWZ( ), AWP( )の値は常にQ14フォーマットで表現されている。

以下に、零入力応答計算中のブロック９（合成フィルタ）の固定小数点擬似コードを示す。

```
NLSSTATE(11)=NLSSTATE(1)
K=2,3,4,...,10について、以下の1行を実行する。          | 最小のNLSSTATEを見つける。
    NLSSTATE(K)<NLSSTATE(11)ならば、NLSSTATE(11)=NLSSTATE(K)とする。
K=1,2,...,IDIMについて、以下を実行する。
    I=1
    L=6-K
    J=LPC
    AA0=0
    LL=1,...,Lについて、以下の3行を実行する。
        AA0=AA0-STATELPC(J)＊A(J+1)            | 積和
        STATELPC(J)=STATELPC(J-1)              | メモリシフト
        J=J-1
    NLS=NLSSTATE(I)-NLSSTATE(11)
    AA1=AA0>>NLS
    I=2,...,10 について、以下の8行を実行する。
        AA0=0
        LL=1,2,..,IDIM について、以下の3行を実行する。
            AA0=AA0-STATELPC(J)＊A(J+1)
            STATELPC(J)=STATELPC(J-1)          | J=1 の時STATELPC(0)は
                                               | 不要データである。
            J=J-1
        NLS=NLSSTATE(I)-NLSSTATE(11)
        AA0=AA0>>NLS                           | 位置合わせシフト
        AA1=AA1+AA0
```

```
        K=1 ならば、SHIFT2へジャンプする。
        L=K-1
        AA0=0
        LL=1,2,...,Lについて、以下の3行を実行する。
            AA0=AA0-STATELPC(J)＊A(J+1)
            STATELPC(J)=STATELPC(J-1)          | Ｊ=1 の時STATELPC(0)
                                               | は不要データである。
            J=J-1
        AA1=AA1+AA0                            | このとき、シフトは不要
SHIFT2: AA1=AA1>>14                            | A( )はQ14フォーマット
                                               | AA1のNLSはNLSSTATE(11)
        AA1>32767ならば、AA1=32767とする。     | STATELPC(1)は乗算器の入力なので
        AA1<-32768ならば、AA1=-32768とする。    | 必要あれば16ビットにクリップ
        STATELPC(1)=AA1                        | STATELPCに下位16ビットをセーブ
        IR =NLSSTATE(11)-2                     | TEMPをQ2フォーマットにする。
        IR>0ならば、AA1=AA1>>IRとする。        |
        IR<0 ならば、AA1=AA1<<-IRとする。      |
        TEMP(K)=AA1
    次のKに対しても、上記の演算を繰り返す。
    VSCALE(STATELPC,IDIM,IDIM,13,STATELPC,NLS)をコールする。
    NLSSTATE(11)=NLSSTATE(11)+NLS              | 新しいSTATELPCを
                                               | 15ビットに再正規化
    L=1,2,...,10について、以下の1行を実行する。 | NLSSTATEの更新
        NLSSTATE(L)=NLSSTATE(L+1)
```

ブロック１０に対する固定小数点擬似コードでは、TEMP, ZIRWFIRとZIRWIIRはQ2である。前のブロックで、TEMPは明白にこのフォーマットとなる。従って、これらの加算では正規化の必要はない。

以下に、零入力応答計算中のブロック１０（聴覚重み付けフィルタ）の固定小数点擬似コードを示す。

```
    K=1,2,...,IDIMについて、以下を実行する。
        AA0=TEMP(K)<<14                        | AWZはQ14のため
        J=LPCW,LPCW-1,...,3,2について、以下の2行を実行する。
            AA0=AA0+ZIRWFIR(J)＊AWZ(J+1)       | フィルタの全零部分
            ZIRWFIR(J)=ZIRWFIR(J-1)            |
        AA0=AA0+ZIRWFIR(1)＊AWZ(2)             | 最後の1サンプルの処理
        ZIRWFIR(1)=TEMP(K)                     |
        J=LPCW,LPCW-1,...,3,2 について、以下の2行を実行する。
            AA0=AA0-ZIRWIIR(J)＊AWP(J+1)       | フィルタの全極部分
            ZIRWIIR(J)=ZIRWIIR(J-1)            |
        AA0=AA0-ZIRWIIR(1)＊AWP(2)             | 最後の1サンプルの処理
        AA0=AA0>>14                            |
        AA0>32767ならば、AA0=32767とする。     | ZIRとZIRWIIRは乗算器の
        AA0<-32768ならば、AA0=-32768とする。    | 入力なのでクリップする。
        ZIR(K)=AA0                             | ZIRとZIRWIIRへ
        ZIRWIIR(1)=AA0                         | 下位16ビットをセーブする
    次のKに対しても、上記の演算を繰り返す。
```

３．３　ブロック９，１０ー合成および聴覚重み付けフィルタメモリ更新

以下はブロック９，１０、フィルタメモリ更新の浮動小数点擬似コードである。

```
ZIRWFIR(1)=ET(1)                                    | ZIRWFIRは、ここでは単に作業領域
                                                    | として使用する。
TEMP(1)=ET(1)
K=2,3,...,IDIMについて、以下を実行する。
    A0=ET(K)
    A1=0.
    A2=0.
    I=K,K-1,...,2について、以下の5行を実行する。
        ZIRWFIR(I)=ZIRWFIR(I-1)
        TEMP(I)=TEMP(I-1)
        A0=A0-A(I)＊ZIRWFIR(I)                      | 縦続接続されたフィルタの各段階
        A1=A1+AWZ(I)＊ZIRWFIR(I)                    | の零状態応答を計算する。
        A2=A2-AWP(I)＊TEMP(I)                       |
    ZIRWFIR(1)=A0                                   |
    TEMP(1)=A0+A1+A2
次のKに対しても、上記の演算を繰り返す。
                                                    | 零入力応答に零状態応答を加算し
                                                    | フィルタメモリの更新を行う。
K=1,2,...,IDIMについて、以下の4行を実行する。
    STATELPC(K)=STATELPC(K)+ZIRWFIR(K)
    STATELPC(K)>MAXならば、STATELPC(K)=MAXとする。        | 範囲の制限
    STATELPC(K)<MINならば、STATELPC(K)=MINとする。        |
    ZIRWIIR(K)=ZIRWIIR(K)+TEMP(K)
I=1,2,...,LPCWについて、以下の1行を実行する。      | ZIRWFIRに正しい値を
    ZIRWFIR(I)=STATELPC(I)                          | セットする。
I=IDIM+1
K=1,2,...,IDIMについて、以下の1行を実行する。      | 合成フィルタメモリの順番
ST(K)=STATELPC(I-K)                                 | を逆にして量子化音声信号
                                                    | を求める。
```

以下、固定小数点擬似コードを示す。STATELPCは、NLSSTATE(1), ... ,NLSSTATE(10)にストアされる10個の指数部をもつ。配列ETに対応するものはNLSETである。ZIRWIIRおよびZIRWFIRは、更新の後Q2となる。ZIRWFIRは最初は作業領域として使用される。このコードにおいては、ETおよびSTATELPCの初めの5つの要素（STATELPC(1)からSTATELPC(5)）はともに15ビットのブロック浮動小数点配列である。ETが、メモリをもたないＬＰＣ合成フィルタにかけられると、その出力（即ちＬＰＣフィルタの零状態応答）は15ビットの範囲を超えることがある。このときには、ETを1ビット右シフトして、出力が15ビットに収まるまでこの計算を繰り返す。経験的に、繰り返し回数は多くても3回（初回を数えると4回）である。繰り返し計算の各積和計算の回数は10回だけである。これは、重み付けフィルタの零状態応答の計算が分割されたループとなっているためである。この方法で計算されるＬＰＣフィルタの零状態応答は、つねに15ビットまたはそれ以下で表現できる。STATELPCを更新するため、この応答が15ビットのSTATELPCに加算されると、その結果は16ビットで表現できることが保証されている。このコードを終了する前にSTATELPCは、後の零入力応答の計算でのオーバフローを避けるために14ビットにスケーリングされる。

```
LABEL1: ZIRWFIR(1)=ET(1)                            | 最初にLPC合成フィルタの
                                                    | 零状態応答を計算する。
    K=2,3,...,IDIMについて、以下の字下げ行を実行する。
        AA0=ET(K)<<14                               | Q14=16384においてA(1)=1であるため。
        I=K,K-1,...,2について、以下の3行を実行する。
```

```
            ZIRWFIR(I)=ZIRWFIR(I-1)
            P=A(I)＊ZIRWFIR(I)                          | Q14乗算
            AA0=AA0-P                                   | 零状態応答を計算する。
        AA1=AA0<<3
        上の行でAA1がオーバフローしたら、以下の4行を実行する。      | 後のAA0>>14
            I=1,2,...,IDIMについて以下の1行を実行する。          | の結果が15
                ET(I)=ET(I)>>1                          | ビットを超えないことを確認する。
            NLSET=NLSET-1                               | 超えたならば、ET>>1とし、越え
            LABEL1へジャンプする。                          | なくなるまで計算を繰り返す。
        AA0=AA0>>14                                     | A( )についてQ14の補償をする。
        ZIRWFIR(1)=AA0                                  | 下位16ビットを保持する。
    次のKに対しても、上記の演算を繰り返す。
    N=IDIM+1                                            | 重み付けフィルタの零状態応答
    TEMP(1)=ZIRWFIR(IDIM)                               | をここで計算する。
    K=2,3,...,IDIMについて、以下の字下げ行を実行する。
        AA1=ZIRWFIR(N-K)<<14            | Q14=16384において、AWZ(1)=1であるため。
         M=IDIM-K
         I=K,K-1,...,2について、以下の5行を実行する。
            TEMP(I)=TEMP(I-1)           | フィルタメモリの全極部分をシフトする。
            P=AWZ(I)＊ZIRWFIR(I+M)      | 重み付けフィルタの全零部分
            AA1=AA1+P
            P=AWP(I)＊TEMP(I)           | 重み付けフィルタの全極部分
            AA1=AA1-P
        AA1=AA1>>14
        AA1>32767ならば、AA1=32767とする。                  | TEMP(1)が乗算器への16ビット入力
        AA1<-32768ならば、AA1=-32768とする。                | なので、必要ならばクリップする。
        TEMP(1)=AA1                                     | 下位16ビットを保持する。
    次のKに対しても、上記の演算を繰り返す。
    IR=NLSET-2
    K=1,2,...,IDIMについて、以下の2行を実行する。          | ZIRWIIRと同様ここでTEMPを
                                                        | Q2にシフトする。
        IR>0ならば、TEMP(K)=TEMP(K)>>IRとする。
        IR<0ならば、TEMP(K)=TEMP(K)<<-IRとする。
                                                | ここで、零状態応答と零入力応答とを
                                                | 加算してフィルタメモリを更新する。
                                                | 初めにZIRWFIRとSTATELPCのNLSが
                                                | マッチしていなければならない。
    NLSET=NLSSTATE(10)ならば、LABEL2へジャンプする。        | 変更不要
    NLSET<NLSSTATE(10)ならば、以下の5行を実行する。         | NLSDビット分だけ
        NLSD=NLSSTATE(10)-NLSET                         | STATELPCの精度を落とす。
        K=1,2,...,IDIMについて、以下の1行を実行する。
            STATELPC(K)=STATELPC(K)>>NLSD
        NLSSTATE(10)=NLSET
        LABEL2へジャンプする。
    NLSD=NLSET-NLSSTATE(10)                             | NLSET>NLSSTATEの場合のみ。
    K=1,2,...,IDIMについて、以下の1行を実行する。           | NLSDビット分だけ
        ZIRWFIR(K)=ZIRWFIR(K)>>NLSD                     | ZIRWFIRの精度を落とす。
LABEL2:
        AA1=4095                                        | 4095＝STATELPCクリップレベル
```

```
NLSSTATE(10)≥0ならば、AA1=AA1<<NLSSTATE(10)とする。          | STATELPCに合わせて
NLSSTATE(10)<0ならば、AA1=AA1>>-NLSSTATE(10)とする。          | クリップレベルシフト
K=1,2,...,IDIMについて以下の字下げ行を実行する。
    AA0=STATELPC(K)+ZIRWFIR(K)                    | LPCフィルタメモリを更新する。
    AA0>AA1ならば、AA0=AA1とする。                 | 必要なら、浮動小数点JT-G728に
    AA0<-AA1ならば、AA0=-AA1とする。               | 規定したクリッピングを行う。
                                                  | これらの値はスケーリングされて
    AA0>32767ならば、AA0=32767とする              | いる。後でSTATELPC(K)が乗算器の
    AA0<-32768ならば、AA0=-32768とする。           | 16ビット入力となるので、もし
    STATELPC(K)=AA0                               | 32767<|AA0|<AA1なら、AA0を16ビッ
                                                  | トにクリップする必要がある。
    AA0=ZIRWIIR(K)+TEMP(K)                        | W(z)メモリの全極部分を更新する。
    AA0>32767ならば、AA0=32767とする。            | ZIRWIIR(K)もまた乗算器の16ビット
    AA0<-32768ならば、AA0=-32768とする。           | 入力となるので、必要ならば
    ZIRWIIR(K)=AA0                                | 16ビットにクリップする。
次のKに対しても、上記の演算を繰り返す。
VSCALE(STATELPC,IDIM,IDIM,12,STATELPC,NLS)をコールする。      | 後の零入力応答の
NLSSTATE(10)=NLSSTATE(10)+NLS                     | 計算でのオーバフローを避ける
                                                  | ために、STATELPCを14ビットに
                                                  | スケーリングする。
IR=NLSSTATE(10)-2
I=1,2,...,5について、以下の4行を実行する。                | ここで、W(z)メモリの全零部分
    AA0=STATELPC(I)                               | ZIRWFIRを、Q2フォーマットの
    IR>0ならば、AA0=AA0>>IRとする。               | 正しい値に設定する。
    IR<0ならば、AA0=AA0<<-IRとする。              |
    ZIRWFIR(I)=AA0                                |
IR=NLSSTATE(9)-2                                  |
I=6,7,...,10について、以下の4行を実行する。               |
    AA0=STATELPC(I)                               |
    IR>0ならば、AA0=AA0>>IRとする。               |
    IR<0ならば、AA0=AA0<<-IRとする。              |
    ZIRWFIR(I)=AA0                                |
I=IDIM+1
K=1,2,...,IDIMについて、以下の1行を実行する。             | 合成フィルタメモリの先頭の
    ST(K)=STATELPC(I-K)                           | 5要素の順番を反転させるこ
NLSST=NLSSTATE(10)                                | とにより、量子化音声を得る。
                                                  | NLSSTは復号器のみで使用さ
                                                  | れる。
```

３．４　ブロック１１－ＶＱターゲットベクトル計算

以下はブロック１１、ＶＱターゲットベクトル計算の浮動小数点擬似コードである。

```
K=1,2,...,IDIMについて、以下の1行を実行する。
    TARGET(K)=SW(K)-ZIR(K)
```

固定小数点コードについて、SWおよびZIRはともにQ2フォーマットであり、入力音声と同じである。したがってNLSTARGET=2となる。以下、固定小数点擬似コードを示す。

```
NLSTARGET=2とする。
K=1,2,...,IDIMについて、以下の6行を実行する。
    AA0=SW(K)
    AA1=ZIR(K)
    AA0=AA0-AA1
    AA0>32767ならば、AA0=32767とする。          | 必要ならばクリップする。
    AA0<-32768ならば、AA0=-32768とする。         |
    TARGET(K)=AA0
```

３．５　ブロック１２－インパルス応答ベクトル計算

以下にブロック１２の浮動小数点擬似コードを示す。

```
TEMP(1)=1.                                    | TEMP=合成フィルタメモリ
WS(1)=1.                                      | WS=W(z)全極部分のメモリ
K=2,3,...,IDIMについて、以下を実行する。
    A0=0.
    A1=0.
    A2=0.
    I=K,K-1,...,3,2について、以下の5行を実行する。
        TEMP(I)=TEMP(I-1)
        WS(I)=WS(I-1)                         |
        A0=A0-A(I)＊TEMP(I)                   | フィルタリング
        A1=A1+AWZ(I)＊TEMP(I)                 |
        A2=A2-AWP(I)＊WS(I)
    TEMP(1)=A0
    WS(1)=A0+A1+A2
次のKに対しても、上記の演算を繰り返す。
ITMP=IDIM+1                                   | h(n)は、W(z)の全極部分の
K=1,2,...,IDIMについて、以下の1行を実行する。  | メモリの順序を逆にするこ
    H(K)=WS(ITMP-K)                           | とにより得られる。
```

予測係数の値、A( ), AWZ( ), AWP( )は全てQ14フォーマットで記憶している。以降の固定小数点擬似コードでは、AA0とAA1の2つのみが32ビットのアキュムレータである。浮動小数点擬似コードにおいて、別々に用いられていたアキュムレータA1とA2は1つにまとめられている。また、ガードビットは必要ない。出力配列H( )はQ13フォーマットで記憶しており、以下に固定小数点擬似コードを示す。

```
TEMP(1)=8192                                  | TEMP=合成フィルタメモリ
WS(1)=8192                                    | WS=W(z)全極部分のメモリ
                                              | WSとTEMPは16ビットワードのQ13
K=2,3,...,IDIMについて、以下を実行する。
```

```
AA0=0
AA1=0
I=K,K-1,...,3,2について、以下の5行を実行する。
    TEMP(I)=TEMP(I-1)
    WS(I)=WS(I-1)                          |
    AA0=AA0-A(I)＊TEMP(I)                  | フィルタリング
    AA1=AA1+AWZ(I)＊TEMP(I)                |
    AA1=AA1-AWP(I)＊WS(I)
AA1=AA0+AA1
AA0=AA0>>14                                | A( ), AWZ( ), AWP( )はQ14フォー
AA1=AA1>>14                                | マットであるため14ビット右シフト
TEMP(1)=AA0
WS(1)=AA1
次のKに対しても、上記の演算を繰り返す。
ITMP=IDIM+1                                | h(n)は、W(z)の全極部分の
K=1,2,...,IDIMについて、以下の1行を繰り返す。  | メモリの順序を逆にするこ
    H(K)=WS(ITMP-K)                        | とにより得られる。
```

３．６　ブロック１３－時間反転畳込み

このモジュールではコードブック探索の準備のため時間反転畳込みを行う。浮動小数点擬似コードは以下の通りである。

```
K=1,2,...,IDIMについて、以下を実行する。
    K1=K-1
    PN(K)=0.
    J=K,K+1,...,IDIMについて、以下の1行を実行する。
        PN(K)=PN(K)+TARGET(J)＊H(J-K1)
次のKについても、上記の演算を繰り返す。
```

固定小数点版では、H( )はQ13フォーマット、TARGETはブロック浮動小数点フォーマットである。NLSTARGETはブロック１６で決められる。NLSPNは7固定である。

```
K=1,2,...,IDIMについて、以下を実行する。
    K1=K-1
    AA0=0                                  | アキュムレータの初期化
    J=K,K+1,...,IDIMについて、以下の2行を実行する。
        P=TARGET(J)＊H(J-K1)
        AA0=AA0+P
    AA0=AA0>>13+(NLSTARGET-7)              | 右シフトしQ7フォーマット化する。
    AA0>32767ならば、AA0=32767とする。      | PNは飽和モードにおける
    AA0<-32768ならば、AA0=-32768とする。     | 乗算入力AA0となるため
    PN(K)=AA0                              | AA0を16ビットにクリップする。
次のKに対しても、上記の演算を繰り返す。
```

３．７　ブロック１４－形状コードベクトル畳込みとエネルギ計算

ここではブロック１４，１５のコードベクトルのエネルギ計算の擬似コードを記述する。

```
J=1,2,...,NCWDについて、以下を実行する。    | ループあたり1コードベクトル
    J1=(J-1)＊IDIM
```

```
K=1,2,...,IDIMについて、以下の4行を実行する。
    K1=J1+K+1
    TEMP(K)=0.
    I=1,2,...,Kについて、以下の1行を実行する。
        TEMP(K)=TEMP(K)+H(I)＊Y(K1-I)          | 畳込み
次のKに対しても、上記の4行を繰り返す。
Y2(J)=0.
K=1,2,...,IDIMについて、次の1行を実行する。
    Y2(J)=Y2(J)+TEMP(K)＊TEMP(K)               | エネルギ計算
次のJに対しても、上記の演算を繰り返す。
```

固定小数点擬似コードでは、H( )はQ13フォーマットで、Y( )はQ11フォーマットで表現されている。H( )とY( )の畳込みの結果、アキュムレータにおけるオーバフロー、つまり 2^{31} より大きい値とならないことが経験的に分かっている。このため、以下の擬似コードでは、アキュムレータにおけるオーバフローのテストを行わない。これにより、対応するＤＳＰの実行を高速にすることが出来る。引き続く14ビットのシフトは、Q10フォーマットのTEMP( )の表現を許容するために必要である。使用されている表現は種々の場合において動作することが分かっている。NLSY2は(NLSH+NLSY-14)＊2-15である。NLSYは11、NLSHは13であるためNLSY2は5である。

```
J=1,2,...,NCWDについて、以下を実行する。          | ループあたり1コードベクトル
    J1=(J-1)＊IDIM
    K=1,2,...,IDIMについて、以下の7行を実行する。
        K1=J1+K+1
        AA0=0
        I=1,2,...,K について、以下の2行を実行する。  |
            P=H(I)＊Y(K1-I)                    | 畳込み
            AA0=AA0+P                          |
        AA0=AA0>>14
        TEMP(K)=AA0                            | 下位16ビットのみ
    次のKに対しても、上記の演算を繰り返す。
    AA0=0
    K=1,2,...,IDIMについて、以下の2行を実行する。
        P=TEMP(K)＊TEMP(K)                     | エネルギ計算
        AA0=AA0+P
    AA0=AA0>>15
    Y2(J)=AA0                                  | 下位16ﾋﾞｯﾄのみ
次のJに対しても、上記の演算を繰り返す。
```

３．８　ブロック１６－ＶＱターゲットベクトル正規化

最初にこのモジュールの浮動小数点擬似コードを示す。

```
TMP=1./GAIN
K=1,2,...,IDIMについて、以下の1行を実行する。
    TARGET(K)=TARGET(K)＊TMP
```

固定小数点擬似コードの場合、利得のＮＬＳとTARGETのＮＬＳを考慮する必要がある。初めはNLSTARGETは2であり、処理中にTMPに対するＮＬＳを算出する。

```
                                               | 除算の分子は16384
                                               | 分子のNLSは14
```

```
                                                    | 分母のNLSはNLSGAIN
                                                    | NLSGAINはブロック46で
                                                    | 決める。
    DIVIDE(16384,14,GAIN,NLSGAIN,TMP,NLSTMP)をコールする。
    K=1,2,...,IDIMについて、以下の2行を実行する。
        AA0=TMP＊TARGET(K)                          | AA0は32ビット
        TARGET(K)=AA0>>15                           | 下位16ビットのみを保持
                                                    | TARGETはこの時点でQ2と
                                                    | なる。
    NLSTARGET=2+NLSTMP-15                           | TARGETをブロック浮動小数
                                                    | 点にする。
    VSCALE(TARGET,IDIM,IDIM,14,TARGET,NLS)をコールする。
    NLSTARGET=NLSTARGET+NLS                         | NLSはVSCALEで変更される。
```

３．９　ブロック１７－ＶＱ探索誤差計算器、最適コードブックインデックス選択器

以下ではブロック１７、誤差計算器とブロック１８、最適コードブックインデックス選択器に対する浮動小数点擬似コードを示す。

```
    DISTMをハードウエアで表現できる正の絶対値最大の値に初期化する。
    N1=NG/2
    J=1,2,...,NCWDについて、以下を実行する。
        J1=(J-1)＊IDIM
        COR=0.
        K=1,2,...,IDIMについて、次の1行を実行する。
            COR=COR+PN(K)＊Y(J1+K)                  | 内積Pjを計算
        COR>0ならば、以下の5行を実行する
            IDXG=N1
            K=1,2,...,N1-1について、以下を実行する。
                COR<GB(K)＊Y2(J)ならば、以下の2行を実行する。
                    IDXG=K                          | 最適な正の利得が求められる。
                    LABELへジャンプする。
        COR≦0ならば、以下の5行を実行する
            IDXG=NG
            K=N1+1,N1+2,...,NG-1について、以下の3行を実行する。
                COR>GB(K)＊Y2(J)ならば、以下の2行を実行する。
                    IDXG=K                          | 最適な負の利得が求められる。
                    LABELへジャンプする。
LABEL:  D=-G2(IDXG)＊COR+GSQ(IDXG)＊Y2(J)           | 歪みD̂を計算する。
        D<DISTMならば、以下の3行を実行する。
            DISTM=D                                 | ここまでの最小の歪と最適な
            IG=IDXG                                 | コードブックインデックスを
            IS=J                                    | 保存する。
    次のJに対しても、上記の演算を繰り返す。
    ICHAN=(IS-1)＊NG+(IG-1)
```

以下が誤差計算器と最適コードブックインデックス選択器（ブロック１７，１８）の固定小数点擬似コードである。ターゲット値との相関の絶対値AA0が利得探索のために使用され、後でその符号が再決定されることが以下のコードに記述されている。このことは、余分な処理のように見えるかもしれないが、探索ループの途中での分岐を避けているのである。ほとんどのＤＳＰによる実現では、分岐を避けることによって命令を節約している。

あるいは、符号を保持して再計算を避けることもできるが、これも通常、探索ループに余分な命令を要する。

```
DISTM=2147483647
J=1,2,...,NCWDについて、以下を実行する。
    J1=(J-1)＊IDIM
    AA0=0
    K=1,2,...,IDIMについて、以下の2行を実行する。
        P=PN(K)＊Y(J1+K)                    | 内積Pjを計算する。
        AA0=AA0+P                           | AA0のNLSは7+11=18
    AA0<0ならば、AA0=-AA0とする。           | 絶対値をとる。
    IDXG=1
    P=GB(1)＊Y2(J)                          | PのNLSは13+5=18
    AA0≧Pならば、IDXG=IDXG+1とする。
    P=GB(2)＊Y2(J)
    AA0≧Pならば、IDXG=IDXG+1とする。
    P=GB(3)＊Y2(J)
    AA0≧Pならば、IDXG=IDXG+1とする。
    AA0=AA0>>14                             | AA0のNLS=4
    AA0>32767ならば、AA0=32767とする。      | AA0の制限;AA0が飽和モード
    AA1=GSQ(IDXG)＊Y2(J)                    |NLSGSQ=11, NLSY2=5,即ちNLSAA1=16
    P=G2(IDXG)＊AA0                         | NLSG2=12, NLSAA0=4,即ちNLSP=16
    AA1=AA1-P
    AA1<DISTMならば以下の3行を実行する。
        DISTM=AA1                           | 倍精度DISTM
        IG=IDXG
        IS=J
次のJに対しても、上記の演算を繰り返す。
AA0=0                                       | 符号ビットを探す。
J1=(IS-1)＊IDIM
K=1,2,...,IDIMについて、以下の2行を実行する。
    P=PN(K)＊Y(J1+K)                        | 内積を計算する。
    AA0=AA0+P
AA0≦0ならば、IG=IG+4とする。
ICHAN=(IS-1)＊NG+(IG-1)
```

上記のコードのなかで、以下の4行を用いた。

```
AA0=AA0>>14                                 | AA0のNLS=4
AA0>32767ならば、AA0=32767とする。          | AA0の制限
AA1=GSQ(IDXG)＊Y2(J)                        | NLSGSQ=11,NLSY2=5,即ちNLSAA1=16
P=G2(IDXG)＊AA0                             | NLSG2=12, NLSAA0=4,即ちNLSP=16
```

“クリッピング”機能を持つＤＳＰチップでは、これらの行は以下のコードに置き換えて、全く同じ結果を得ることができる。

```
AA0=AA0<<2                                  | AA0のNLS=20
AA0=CLIP(AA0)                               | AA0が飽和モード
AA0=AA0>>16                                 | 上位ワードをとる;AA0のNLS=4
AA1=GSQ(IDXG)＊Y2(J)                        | NLSGSQ=11,NLSY2=5,即ちNLSAA1=16
```

```
P=G2(IDXG)＊AA0                                    | NLSG2=12, NLSAA0=4,即ちNLSP=16
```

クリップ機能と飽和モードは<<2操作が実行された時、AA0をオーバフローさせないようにという考えに基づいている。オーバフローする代わりに、その元の符号に依存してAA0は正または負の絶対値最大の値にセットされる。このモジュールでは、AA0は常に正である。後者のコードを選択するためには、ＤＳＰがその機能を持っていること、及びアキュムレータのビット長が32ビットより大きいことが必要である。前者のメインに記述されている擬似コードは、（ＤＳＰの機能にかかわらず）無条件で実現することができる。

３．１０　ブロック１９－励振ＶＱコードブックとブロック２１－利得調整ユニット

以下にブロック１９励振ＶＱコードブックの浮動小数点版擬似コードを示す。

```
NN=(IS-1)＊IDIM
K=1,2,...,IDIMについて、以下の1行を実行する。
    YN(K)=GQ(IG)＊Y(NN+K)
```

以下にブロック２１利得調整ユニットの浮動小数点版擬似コードを示す。

```
K=1,2,...,IDIMについて、以下の1行を実行する。
    ET(K)=GAIN＊YN(K)
```

固定小数点擬似コードでは、ブロック１９とブロック２１を組み合わせて、一つのモジュールにする。YとGQのどちらも固定のQフォーマットで、それぞれQ11, Q13とする。GAINの値はNLSGAINと対応する。最大の精度を得るために、GQ(IG)＊GAINは上位16ビットが丸められる前に32ビットに正規化される。NNGQ(I)を（1+Q13のGQ(I)の正規化に必要な左シフト数）とすると、I=1, 2, 5, 6のときNNGQ(I)=3とし、I=3, 7のとき、NNGQ(I)=2、そしてI=4, 8のときNNGQ(I)=1となる。そこで、この擬似コードは以下のようになる。

```
AA0=GQ(IG)＊GAIN          | AA0には上位にNNGQ(IG)個の0がある。
AA0=AA0<<NNGQ(IG)         | AA0を正規化するためにNNGQ(IG)ビットだけ左にシフトする。
TMP=RND(AA0)              | 上位16ビットを丸めてTMPに入れる。
NLSAA0=13+NLSGAIN                                  | GQ(IG)＊GAINのQフォーマット
NLSTMP=NLSAA0+NNGQ(IG)-16                          | TMPのQフォーマット、NNGQ(IG)ビッ
                                                   | トだけAA0の左シフトを行い、丸めて上
                                                   | 位16ビットを取る。
NN=(IS-1)＊IDIM                                    | 選択された波形コードベク
VSCALE(Y(NN+1),IDIM,IDIM,14,TEMP,NLS)をコールする。 | トルを16ビットに正規化し
                                                   | てTEMPに入れる。
K=1,2,...,IDIMについて、以下の2行を実行する。      | TMPとTEMPの両方とも16ビットに正
    AA0=TMP＊TEMP(K)                               | 規化されるためその結果には上位
    ET(K)=RND(AA0)                                 | に1個の0がある。上位ワードを直
                                                   | 接丸め、15ビット配列ETに入れる。
NLSET=NLSTMP+11+NLS-16                             | ETのNLSを計算
```

３．１１　ブロック３２－復号器合成フィルタ

以下にブロック３２復号器合成フィルタの浮動小数点擬似コードを示す。

```
K=1,2,...,IDIMについて、以下の6行を実行する。
    TEMP(K)=0.
    J=LPC,LPC-1,...,3,2について、次の2行を実行する。
        TEMP(K)=TEMP(K)-STATELPC(J)＊A(J+1)        | 零入力応答計算
```

```
        STATELPC(J)=STATELPC(J-1)
    TEMP(K)=TEMP(K)-STATELPC(1)＊A(2)          | 最後の1サンプルの処理
    STATELPC(1)=TEMP(K)
次のKに対しても、上記の演算を繰り返す。
TEMP(1)=ET(1)
K=2,3,...,IDIMについて、以下の5行を実行する。
    A0=ET(K)
    I=K,K-1,...,2について、以下の2行を実行する。
        TEMP(I)=TEMP(I-1)
        A0=A0-A(I)＊TEMP(I)                   | 零状態応答計算
    TEMP(1)=A0
次のKに対しても、上記の演算を繰り返す。
                                              | 零入力応答と零状態応答に加算する
                                              | ことによって、フィルタメモリを
                                              | 更新する。
K=1,2,...,IDIMについて、次の3行を実行する。
    STATELPC(K)=STATELPC(K)+TEMP(K)                  | ZIR+ZSR
    STATELPC(K)>MAXならば、STATELPC(K)=MAXとする。    | 範囲制限
    STATELPC(K)<MINならば、STATELPC(K)=MINとする。    |
I=IDIM+1
K=1,2,...,IDIMについて、以下の1行を実行する。    | 合成フィルタメモリの順序を逆
    ST(K)=STATELPC(I-K)                       | にして、量子化音声を計算する。
```

ブロック３２の固定小数点擬似コードは聴覚重み付けフィルタのメモリ更新が無いことを除きブロック９で用いた方法と同様である。

```
NLSSTATE(11)=NLSSTATE(1)
K=2,3,4,...,10について以下の１行を実行する。    | 最小のNLSSTATEを見つける。
    NLSSTATE(K)<NLSSTATE(11)ならば、NLSSTATE(11)=NLSSTATE(K)とする。
K=1,2,...,IDIMについて、以下を実行する。
    I=1
    L=6-K
    J=LPC
    AA0=0
    LL=1,...,Lについて、以下の3行を実行する。
        AA0=AA0-STATELPC(J)＊A(J+1)           | 積和
        STATELPC(J)=STATELPC(J-1)             | メモリシフト
        J=J-1
    NLS=NLSSTATE(I)-NLSSTATE(11)
    AA1=AA0>>NLS
    I=2,...,10について、以下の8行を実行する。
        AA0=0
        LL=1,2,...,IDIMについて以下の3行を実行する。
            AA0=AA0-STATELPC(J)＊A(J+1)
            STATELPC(J)=STATELPC(J-1)         | J=1の時STATELPC(0)は不要データである。
            J=J-1
        NLS=NLSSTATE(I)-NLSSTATE(11)
        AA0=AA0>>NLS                          | 位置合わせシフト
        AA1=AA1+AA0
```

```
        K=1ならば、SHIFT2へジャンプする。
        L=K-1
        AA0=0
        LL=1,2,...,Lについて、以下の3行を実行する。
            AA0=AA0-STATELPC(J)＊A(J+1)
            STATELPC(J)=STATELPC(J-1)      | J=1の時STATELPC(0)は不要データである。
            J=J-1
        AA1=AA1+AA0                        | この時シフトは、不要
SHIFT2: AA1=AA1>>14                                | A()はQ14フォーマット
                                                   | AA1のNLSはNLSSTATE(11)
        AA1>32767ならば、AA1=32767とする。         | STATELPC(1)は乗算器の入力なので
        AA1<-32768ならば、AA1=-32768とする。       | 必要があれば16ビットにクリップ
        STATELPC(1)=AA1                            | STATELPCに下位16ビットをセーブ
        IR=NLSSTATE(11)-2                          | TEMPをQ2フォーマットにする。
        IR>0ならば、AA1=AA1>>IRとする。            |
        IR<0ならば、AA1=AA1<<-IRとする。           |
        TEMP(K)=AA1
    次のＫに対しても上記の演算を繰り返す。
    VSCALE(STATELPC,IDIM,IDIM,13,STATELPC,NLS)をコールする。
    NLSSTATE(11)=NLSSTATE(11)+NLS          | 新しいSTATELPCを15ビットに再正規化
    L=1,2,...,10について、以下の1行を実行する。       | NLSSTATEの更新
        NLSSTATE(L)=NLSSTATE(L+1)
                                                   | LPC合成フィルタの最初の零状態応
LABEL1:TEMP(1)=ET(1)                               | 答を計算する。
    K=2,3,...,IDIMについて、以下の字下げ行を実行する。
        AA0=ET(K)<<14                  |A(1)がQ14で16384の値をとるときA(1)=1であるため
        I=K,K-1,...,2 について、以下の3行を実行する。
            TEMP(I)=TEMP(I-1)
            P=A(I)＊TEMP(I)                        | Q14乗算
            AA0=AA0-P                              | 零状態応答を計算
        AA1=AA0<<3
        AA1が上記でオーバフローしたならば、以下の4行を実行する。
            I=1,2,...,IDIMについて、以下の1行を実行する。
                ET(I)=ET(I)>>1         | AA0>>14の後その結果が15ビットを越えていない
            NLSET=NLSET-1              | か確認する。もし越えていたらET>>1を行い条件
            LABEL1にジャンプする。     | に合うまでその計算を繰り返す。
        AA0=AA0>>14                                | A()をQ14になるようにする。
        TEMP(1)=AA0                                | 下位16ビットを保持する。
    次のKに対しても、上記の演算を繰り返す。
    NLSET=NLSSTATE(10)ならば、LABEL2にジャンプする。  | この場合変更不要
    NLSET<NLSSTATE(10)ならば、以下の5行を実行する。   | NLSDビットだけSTATELPCは
        NLSD=NLSSTATE(10)-NLSET                    | 精度を落とす。
        K=1,2,...,IDIMについて、以下の1行を実行する。
            STATELPC(K)=STATELPC(K)>>NLSD
        NLSSTATE(10)=NLSET
        LABEL2にジャンプする。                     | 左辺がNLSET>NLSSTATEのときのみ
    NLSD=NLSET-NLSSTATE(10)                        | TEMPはNLSDビットだけ精度を落とす。
    K=1,2,...,IDIMについて、以下の1行を実行する。
        TEMP(K)=TEMP(K)>>NLSD
```

```
LABEL2:                                                    | ここで準備完了
    AA1=4095                                               | 4095=STATELPCの範囲制限レベル
    NLSSTATE(10)≥0ならば、AA1=AA1<<NLSSTATE(10)とする。      | STATELPCに対し調整す
    NLSSTATE(10)<0ならば、AA1=AA1>>-NLSSTATE(10)とする。     | るための範囲制限レベ
                                                           | ルをシフトする。
    K=1,2,...,IDIMについて、以下の字下げ行を実行する。
        AA0=STATELPC(K)+TEMP(K)                            | LPCフィルタメモリを更新。
        AA0>AA1ならば、AA0=AA1とする。                      | 必要なら、浮動小数点JT-G728で定
        AA0<-AA1ならば、AA0=-AA1とする。                    | 義されたように範囲制限する。
                                                           | これらの値はスケーリングされて
                                                           | いることに注意する。
        AA0>32767ならば、AA0=32767とする。                  | 従って、もし32767<|AA0|<AA1なら
        AA0<-32768ならば、AA0=-32768とする。                | ば、AA0を16ビットに制限する必要
        STATELPC(K)=AA0                                    | がある。何故なら、STATELPC(K)は
    次のKに対しても、上記の演算を繰り返す。                  | 後に乗算器に入力される16ビット
                                                           | となるからである。
    VSCALE(STATELPC,IDIM,IDIM,12,STATELPC,NLS)をコールする。
    NLSSTATE(10)=NLSSTATE(10)+NLS                          | STATELPCを14ビットにスケーリングして、後
                                                           | の零入力応答でのオーバフローを回避する。
    I=IDIM+1
    K=1,2,...,IDIMについて、以下の1行を実行する。            | 合成フィルタメモリの先頭5
        ST(K)=STATELPC(I-K)                                | 要素の順番を反転させること
    NLSST=NLSSTATE(10)                                     | により量子化音声を得る。
                                                           | NLSSTは復号器で後に使用される。
```

３．１２　ブロック３６－$W(z)$用ハイブリッド窓かけモジュール

この節では、ブロック３６に対する浮動小数点擬似コードと固定小数点擬似コード、両方について示す。以下、浮動小数点擬似コードを示す。

```
    N1=LPCW+NFRSZ                                          | 定数の計算（あらかじめ計算してメモリに格
    N2=LPCW+NONRW                                          | 納しておくこともできる）
    N3=LPCW+NFRSZ+NONRW
    N=1,2,...,N2について、以下の1行を実行する。
        SBW(N)=SBW(N+NFRSZ)                                | 過去の信号バッファのシフト
    N=1,2,...,NFRSZ について、以下の1行を実行する。
        SBW(N2+N)=STMP(N)                                  | 新しい信号のシフト
                                                           | SBW(N3) は最も新しいサンプル
    K=1
    N=N3,N3-1,...,3,2,1 について、以下の2行を実行する。
        WS(N)=SBW(N)＊WNRW(K)                              | 窓関数を掛ける。
        K=K+1
    I=1,2,...,LPCW+1について、以下の4行を実行する。
        TMP=0.
        N=LPCW+1,LPCW+2,...,N1について、以下の1行を実行する。
            TMP=TMP+WS(N)＊WS(N+1-I)
        REXPW(I)=(1/2)＊REXPW(I)+TMP                       | 巡回成分の更新
    I=1,2,...,LPCW+1について、以下の3行を実行する。
        R(I)=REXPW(I)
        N=N1+1,N1+2,...,N3について、以下の1行を実行する。
```

```
        R(I)=R(I)+WS(N)＊WS(N+1-I)                    | 非巡回成分の加算
R(1)=R(1)＊WNCF                                      | 白色雑音補正
```

同じモジュールの固定小数点版を示す。ここでいくつかの新しい変数を導入する。NLSREXPWはグローバル変数であり、REXPWを正規化するための左シフト数を示している。この変数は31に初期化されている。

```
N1=LPCW+NFRSZ (=10+20)                  | 定数の計算（あらかじめ計算してメモリに格
N2=LPCW+NONRW (=10+30)                  | 納しておくこともできる）
N3=LPCW+NFRSZ+NONRW (=10+20+30)
N=1,2,...,N2について、以下の1行を実行する。
    SBW(N)=SBW(N+NFRSZ)                 | 過去の信号バッファのシフト
N=1,2,...,NFRSZ について、以下の1行を実行する。
    SBW(N2+N)=STMP(N)                   | SBW(N3)は最も新しいサンプル。
                                        | 全てのSBWはQ2であり
                                        | 15ビット精度で表されている。
FINDNLS(SBW,N3,14,NLS)をコールする。    | ヘッドルームの2ビットを得るための
                                        | 以下のループで必要とされる左シフト量
                                        | を見つける。
                                        | 実際にスケーリングを行う必要は無い。
                                        | NLSを使用するのみ。
NLSTMP=NLS-1
K=1
N=60,59,...,1 について、以下の4行を実行する。
    P=SBW(N)＊WNRW(K)                   | WNRWはQ15であり、NLSTMPビットの
    AA0=P<<NLSTMP                       | 左シフトにより配列WS(N)の中の
    WS(N)=RND(AA0)                      | 絶対値最大の値を14ビットにする。
    K=K+1                               | （ヘッドルームの2ビットは後の計算のための
                                        | ビット）
NLSATTW=15
HWMCORE(LPCW,N1,N3,NLSATTW,WS,NLSTMP,REXPW,NLSREXPW,R,ILLCONDW)をコールする。
NLSREXPW>41 ならば、NLSREXPW=41とする。  | 長い周期のゼロ信号入力による
                                        | REXPW( )とR( )の精度が落ちること
                                        | を避ける。
```

サブルーチンHWMCOREは３．１８節に示されている。

上記プログラムにおいて、サブルーチンFINDNLS( )は60サンプルのSBWバッファ全体を探す。しかし、計算を減らすために、2ワード以上のメモリを使用するビットイグザクトなサブルーチンを使用することもできる。SBWは、常に過去の40サンプルと、新しい20サンプルからなるバッファである。このSBWバッファは20サンプル毎の3ベクトルに分割できる。3ベクトルのそれぞれに対し、ＮＬＳの値を記憶しておき、この中で最も小さい値を選び、ハイブリッド窓に使用する事ができる。2つのベクトルは過去のサンプルから成っているため、それぞれのＮＬＳは既知である。従って、最も新しい1ベクトルのみを探す事によりＮＬＳを求めることができる。最も新しいベクトルのＮＬＳを格納し、次の計算のために古い2ベクトルの値を更新する。この方法は、上記の擬似コードで示された手順と全く同じＮＬＳを選ぶ事ができる。

付表Ｇ－３－２／JT-G728は上記の擬似コードにおける全ての変数に対応するフォーマットとサイズについて示している。また、表ではそれぞれの変数が、一時的な変数なのか（すなわちモジュールがそのプログラムを終了した後に格納しておく必要の無い変数）、常置の変数なのか（現在の計算の後にも同様に使用される変数）をそれぞれ（一時）／（常置）として示している。さらに表では、以前の浮動小数点擬似コードでは含まれていなかった変数なのか否かも（新変数）／（旧変数）として示している。

変　数	フォーマット	サイズ	一時／常置	新／旧変数
NLS	int	1	一時	新変数
NLSREXPW	int	1	常置	新変数
NLSTMP	int	1	一時	新変数
REXPW	BFL	11	常置	旧変数
R	BFL	11	常置	旧変数
SBW	Q2	60	常置	旧変数
STMP	Q2	20	常置	旧変数
WS	BFL	60	一時	旧変数

注：BFL＝ブロック浮動小数点形式、int＝16ビット整数。

３．１３　ブロック３８－聴覚重み付けフィルタ係数計算器

以下、浮動小数点擬似コードを示す。

```
ICOUNTが3でなければ、本ブロックをスキップする。
そうでなければ、以下を実行する。
I=2,3,...,11について、以下の1行を実行する。
    AWP(I)=WPCFV(I)＊AWZTMP(I)                    | 分母係数
I=2,3,...,11について、以下の1行を実行する。
    AWZ(I)=WZCFV(I)＊AWZTMP(I)                    | 分子係数
```

固定小数点擬似コードにおいては、ダービン再帰法における異常状態あるいは、AWZTMPがQ13ですらオーバフローが発生する可能性について注意が必要である。（Q13で不十分な場合が観測されている訳ではないが、オーバフローの可能性はまだ考慮にいれなければならない。）変数ILLCONDWはブロック３７から与えられるフラグであり、このフラグはブロック３７の結果が有効であるか否かを示している。

ＪＴ－Ｇ７２８では、仮にILLCONDWが真であった場合、ダービン法の結果は使用されない事を暗黙の仮定としている。これはすなわち、AWZTMPからAWZと、AWPを更新しない事である。ここでは同様の仮定を用い、ILLCONDWが真ならばAWPとAWZは更新しない。これを実行する必要はなく、以前の値を使用し続けるだけでよい。

次に、ダービン再帰法により得られる係数AWZTMP( )がQ13, Q14および、Q15になる可能性について考慮しなければならない。NLSAWZTMPはAWZTMPの左シフト数を示す。分子と分母の係数AWZとAWPは結果出力としてQ14に納める必要がある。AWZがQ14に納まらないならば、AWZとAWPは更新しない。以下、固定小数点擬似コードを示す。

```
ICOUNTが3でなければ、本ブロックをスキップする。
そうでなければ、以下を実行する。
                                    | ILLCONDWが真である事を確認する。
ILLCONDW=真ならば、このブロックをスキップする。
そうでなければ、以下を実行する。
                                    | 分子係数を確認しQ14でオーバフローするな
                                    | らば、AWZとAWPは更新しない。
                                    | オーバフローする場合、一時記憶配列WS
                                    | を使用する。
                                    | これによりAWZは保護される。
I=2,3,...,7 について、以下の6行を実行する。
    AA0=WZCFV(I)＊AWZTMP(I)                    | WZCFVはQ14,
```

```
        NLSAWZTMP=13ならば、AA0=AA0<<3とする。        | AA0は14+NLSAWZTMP
        NLSAWZTMP=14ならば、AA0=AA0<<2とする。        | 3ケース全てについて適当な
        NLSAWZTMP=15ならば、AA0=AA0<<1とする。        | シフト数でAA0をQ30にする。
        AA0がオーバフローならば、LABELへジャンプする。 | 真ならばQ14で
        WS(I)=RND(AA0)                                | オーバフロー；上位ワードを
                                                      | 丸める。それ以外の場合にオーバ
                                                      | フローが発生する事は無い。
                                                      | ここまで来たならば確認無しに
                                                      | WSをAWZにコピーする。
    I=8,9,10,11 について、以下の5行を実行する。
        AA0=WZCFV(I)＊AWZTMP(I)
        NLSAWZTMP=13ならば、AA0=AA0<<3とする。
        NLSAWZTMP=14ならば、AA0=AA0<<2とする。
        NLSAWZTMP=15ならば、AA0=AA0<<1とする。
        WS(I)=RND(AA0)
    I=2,3,...,11について、以下の1行を実行する。    | オーバフローがなければWS
        AWZ(I)=WS(I)                                 | をAWZへコピーする。
                                          | 分母係数を求める。
                                          | 分子係数がオーバフローしていなければ
                                          | 分母係数もオーバフローしない。
    I=2,3,...,11について、以下の5行を実行する。
        AA0=WPCFV(I)＊AWZTMP(I)                       | WPCFVはQ14,
        NLSAWZTMP=13ならば、AA0=AA0<<3とする。        | AA0は14+NLSAWZTMP
        NLSAWZTMP=14ならば、AA0=AA0<<2とする。        | 3ケース全てについて適当な
        NLSAWZTMP=15ならば、AA0=AA0<<1とする。        | シフト数でAA0をQ30にする。
        AWP(I)=RND(AA0)                               | AWPの上位ワードを丸める。
    サブルーチン正常終了。
LABEL: | ここに来た場合、AWZをQ14で表そうとするとオーバフローがあるので、
    | 聴覚重み付けフィルタ係数を更新しない。（すなわち、以前の適応周期の
    | 聴覚重み付けフィルタ係数を使用し続ける。）
```

３．１４　ブロック４３－$GP(z)$ 用ハイブリッド窓かけモジュール

この節では、ブロック４３の浮動小数点擬似コードと固定小数点擬似コードについて記述する。まず、浮動小数点擬似コードについて記す。

```
    N1=LPCLG+NUPDATE                              | 定数の計算（あらかじめ計算して、
    N2=LPCLG+NONRLG                               | メモリに格納しておくこともできる）
    N3=LPCLG+NUPDATE+NONRLG
    N=1,2,...,N2について、以下の1行を実行する。
        SBLG(N)=SBLG(N+NUPDATE)                   | 過去の信号バッファのシフト
    N=1,2,...,NUPDATE について、以下の1行を実行する。
        SBLG(N2+N)=GTMP(N)                        | 新しい信号のシフト
                                                  | SBLG(N3)は最も新しいサンプル
    K=1
    N=N3,N3-1,...3,2,1について、以下の2行を実行する。
        WS(N)=SBLG(N)＊WNRLG(K)                   | 窓関数を掛ける。
        K=K+1
    I=1,2,...,LPCLG+1 について、以下の4行を実行する。
        TMP=0
```

```
        N=LPCLG+1,LPCLG+2,...,N1について、以下の1行を実行する。
            TMP=TMP+WS(N)＊WS(N+1-I)
        REXPLG(I)=(3/4)＊REXPLG(I)+TMP                    | 巡回成分を更新
    I=1,2,...,LPCLG+1 について、以下の3行を実行する。
        R(I)=REXPLG(I)
        N=N1+1,N1+2,...,N3について、以下の1行を実行する。
            R(I)=R(I)+WS(N)＊WS(N+1-I)                    | 非巡回成分を加算
    R(1)=R(1)＊WNCF                                       | 白色雑音補正
```

このルーチンが呼び出される前に、GTMP( )は、次のように割り当てられている。

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

GSTATE( )の初期値は、浮動小数点形式の-32であり、固定小数点形式のQ9フォーマットの-16384である。次に、これと同じモジュールの固定小数点版を記す。このコードでは、新しい変数をつけ加えている。NLSREXPLGは、REXPLGを正規化するための左シフト数を保持するグローバル変数である。この変数は、31に初期化されている。

```
N1=LPCLG+NUPDATE (=10+4)                              | 定数の計算（あらかじめ計算して、
N2=LPCLG+NONRLG (=10+20)                              | メモリに格納しておくこともできる）
N3=LPCLG+NUPDATE+NONRLG (=10+4+20)
N=1,2,...,N2について、以下の1行を実行する。
    SBLG(N)=SBLG(N+NUPDATE)                           | 過去の信号バッファのシフト
N=1,2,...,NUPDATE について、以下の1行を実行する。
    SBLG(N2+N)=GTMP(N)                                | SBLG(N3)は最も新しいサンプル
                                                      | SBLGはすべてQ9フォーマットで、
                                                      | 16ビット精度表現である。
FINDNLS(SBLG,N3,14,NLS)をコールする。                  | ヘッドルームの2ビットを得るため
NLSTMP=NLS-1                                          | の以下のループで必要とされる左
                                                      | シフト量を見つける。
K=1
N=34,33,...,1 について、以下の5行を実行する。
    P=SBLG(N)＊WNRLG(K)                               | WNRLGはQ15フォーマット
    NLSTMP=-1ならば、AA0=P>>1とする。
    NLSTMP>-1ならば、AA0=P<<NLSTMPとする。
    WS(N)=RND(AA0)                                    | WS(N)は14ビットまたはそれ以下
    K=K+1
NLSATTLG=14
HWMCORE(LPCLG,N1,N3,NLSATTLG,WS,NLSTMP,REXPLG,NLSREXPLG,R,ILLCONDG)を
コールする。
```

サブルーチンHWMCOREは、３．１８節に記述されている。

付表Ｇ－３－３／JT-G728は上記の擬似コードにおける全ての変数に対応するフォーマットとサイズについて示している。又、表ではそれぞれの変数が、一時的な変数なのか（すなわちモジュールがそのプログラムを終了した後に格納しておく必要のない変数）、常置の変数なのか（現在の計算の後にも同様に使用される変数）をそれぞれ（一時) ／（常置）として示している。さらに表では、以前の浮動小数点擬似コードでは含まれていなかった変数か否かも（新変数）／（旧変数）として示している。

変　数	フォーマット	サイズ	一時／常置	新／旧変数
GTMP	Q9	4	常置	旧変数
NLS	int	1	一時	新変数
NLSREXPLG	int	1	常置	新変数
NLSTMP	int	1	一時	新変数
REXPLG	BFL	11	常置	旧変数
R	BFL	11	常置	旧変数
SBLG	Q9	34	常置	旧変数
WS	BFL	34	一時	旧変数

注：BFL＝ブロック浮動小数点、int＝16ビット整数。

３．１５　ブロック４５－*GP*(*z*) 用帯域幅拡張モジュール

以下ブロック４５、帯域幅拡張モジュールの浮動小数点擬似コードを示す。

```
ICOUNTが2でなければ、このブロックをスキップする。
I=2,3,...,LPCLG+1について、以下の1行を実行する。
    GP(I)=FACGPV(I)＊GPTMP(I)                    | スケーリング係数
```

FACGPVのテーブルはQ14フォーマットであり、他の帯域幅拡張係数のテーブルと同様である。入力GPTMP配列の値は、Q13, Q14またはQ15フォーマットである。先の固定小数点版レビンソン・ダービン再帰法モジュールの記述で吟味したように、どのフォーマットがGPTMPで使われるかということを示すために、NLSGPTMPがレビンソン・ダービン再帰法モジュールによって与えられる。FACGPV(I)＊GPTMP(I)の乗算後、相当量の左シフトが要求される。

GPの最終的な値は、常にQ14フォーマットである。経験的に、ブロック４５の出力係数配列は、Q14で表現できなくなる程（すなわち、Q13フォーマットまたは、それ以下を要する程）大きくならない。しかし、安全のため、帯域幅拡張ブロックの出力で、Q14でのオーバフローを扱う準備をしなければならない。以下に示す擬似コードでは、Q14でのオーバフローの可能性をチェックする。もし、それが検出されたなら、レビンソン・ダービン再帰法モジュールと同様の措置をとる。すなわち、予測係数を更新せずに前適応周期の古い係数を使用し続ける。他の手段として、2つのうちのどちらのQフォーマットが使用されるかというフィルタリングモジュールへの合図となるフラグ付きの切り替え可能なQ13／Q14フォーマットを使用する事が可能であろう。しかし、これは不必要にＤＳＰコードの複雑さと実行時間を増大させてしまう。帯域幅拡張モジュールの出力では、Q14でのオーバフローは、一度も検出されなかったので、以下で実行されるような簡単な安全チェックで十分である。
以下ブロック４５、固定小数点擬似コードを示す。

```
ICOUNTが2でなければ、このブロックをスキップする。
そうでなければ、以下を実行する。
                                    | まず、ILLCONDGが真かどうかのチェック
ILLCONDG=真ならば、このブロックをスキップする。
そうでなければ、以下を実行する。
    GPTMP(1)=16384
    I=2,3,4,...,LPCLG+1 について、以下の6行を実行する。
        AA0=FACGPV(I)＊GPTMP(I)                  | AA0は、Q27､Q28､または､Q29
        NLSGPTMP=13ならば、AA0=AA0<<3とする。    | 3ケース全てについて、適当な
        NLSGPTMP=14ならば、AA0=AA0<<2とする。    | シフトによりAA0をQ30にする。
        NLSGPTMP=15ならば、AA0=AA0<<1とする。
```

```
        AA0がオーバフローしたならば、LABELへジャンプする。        | 真でないならば、
        GPTMP(I) = RND(AA0)                                  | GPの上位ワードへ丸める。
    I=2,3,4,...,LPCLG+1 について、以下の1行を実行する。| すべてが正規化済みな
        GP(I)=GPTMP(I)                                       | ら、GPTMPをGPへコピー
                                                             | し、終了
        プログラム正常終了。
LABEL: | ここに来た場合、GPをQ14フォーマットで表そうとするとオーバフローがあ
       | るので、対数利得予測係数を更新しない。（すなわち、以前の適応周期の対
       | 数利得予測係数を使用し続ける。
```

３．１６　ブロック４６－対数利得線形予測

以下、ブロック４６、対数利得線形予測の浮動小数点擬似コードを示す。

```
LOGGAIN=0.
I=LPCLG,LPCLG-1,...,3,2 について、以下の2行を実行する。
    LOGGAIN=LOGGAIN-GP(I+1)＊GSTATE(I)
    GSTATE(I)=GSTATE(I-1)
LOGGAIN=LOGGAIN-GP(2)＊GSTATE(1)
```

LOGGAINとGSTATEは、符号器を通してQ9フォーマットである。GPは、Q14フォーマット表現である。以下固定小数点擬似コードを示す。

```
AA0=0
I=LPCLG,LPCLG-1,...,3,2 について、以下の3行を実行する。
    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
```

以下ブロック９８、対数利得リミッタの浮動小数点擬似コードを示す。このコードは、固定小数点版の修正に基づいているので、ＴＴＣ標準ＪＴ－Ｇ７２８浮動小数点版には記載されていない。ここでは、以下に示す固定小数点擬似コードとの比較のために記述する。

```
LOGGAIN>28ならば、LOGGAIN=28とする。
LOGGAIN<-32 ならば、LOGGAIN=-32とする。
```

LOGGAINは、Q9フォーマットで表現されるので、最大、最小の閾値は、512を乗じたものである。これらの値は、以下に記述する固定小数点擬似コードで使用される。

```
LOGGAIN>14336 ならば、LOGGAIN=14336とする。
LOGGAIN<-16384ならば、LOGGAIN=-16384とする。
```

以下ブロック９９、対数利得オフセット加算器の浮動小数点擬似コードを示す。

```
Z=LOGGAIN+GOFF
```

GOFFの浮動小数点形式での値は32であり、固定小数点形式での値は16384であり、これは、512＊32に相当する。LOGGAINは、-32と+28の範囲にあるため、Zは、0から60の範囲にあることになる。固定小数点コードは、浮動小数点コードと同じである。

以下ブロック４８、逆対数計算の浮動小数点コードを示す。

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

目的とする値は、2の逆対数によって表現される。

$$10^{0.05Z} = 2^{0.05\log_2(10)Z} = 2^{0.1660964Z}$$

X=0.1660964Zとする。Xは、0から9.97の値である。さらに、X=[X]+xとする。[X]は、Xをこえない、または、Xと等しい最大整数である。xは、小数部である。 $2^{[x]}$ の値は正確で、指数の形で表現される必要があるだけである。残ったのは、小数部の値の計算である。

Xの計算において0.1660964をQ21フォーマットで表すものとする。これは、0.1660964という数を上位16ビットが10、下位15ビットが20649の数で表すことに相当する。Xの精度を上げるために、Zを前述の上位側、下位側の両方に乗じ、その後[X]とxに分ける。小数部分の指数計算において、 $0 < x < 1$ であるので $1 < 2^x < 2$ であることがわかっている。従って、xはQ15、 2^x はQ14の固定小数点表現を使用することができる。 2^x の計算には、テーラー級数展開を用いて

cの値は、Q14又はQ15で以下の様に与えられる。

$c4 = 323 = 0.0098571$ (Q15)
$c3 = 1874 = 0.0571899$ (Q15)
$c2 = 7866 = 0.2400512$ (Q15)
$c1 = 22702 = 0.6928100$ (Q15)
$c0 = 16384 = 1.0$ (Q14)

以下に、2つの32ビットアキュムレータをもつ16ビットＤＳＰで、 $10^{0.05GAIN}$ を計算するための擬似コードを示す。ここでは、GAINはQ9フォーマットで、32dBのオフセットがあらかじめ加えられているものとする。

```
AA0=10＊Z                | ZはQ9、10はQ6、よってAAOはQ15
AA1=20649＊Z             | 20649はQ21、よってAA1はQ30
AA1=AA1<<1               | AA1をQ31にする。
AA1=RND(AA1)             | AA1を丸め、下位ワードをQ15とする。
AA0=AA0+AA1              | Q15でAA0=[X]+x
AA1=AA0>>15              | [X]をQ0、xをQ15とする。
NLS=AA1                  | NLS=[X]
AA1=AA1<<15
x=AA0-AA1                | xはQ15の小数部分
                         | 2^xの計算
AA0=c4＊x                | Q15＊Q15→AA0はQ30
AA0=AA0<<1               | AA0をQ31とする。
AA1=c3<<16               | AA1はQ31
```

```
AA0=AA0+AA1
TMP=RND(AA0)                    | 丸めて、TMPをQ15とする。
AA0=TMP＊x                      | Q15＊Q15→AA0はQ30
AA0=AA0<<1                      | AA0をQ31とする。
AA1=c2<<16                      | AA1はQ31
AA0=AA0+AA1
TMP=RND(AA0)                    | 丸めて、TMPをQ15とする。
AA0=TMP＊x                      | Q15＊Q15→AA0はQ30
AA0=AA0<<1                      | AA0をQ31とする。
AA1=c1<<16                      | AA1はQ31
AA0=AA0+AA1
TMP=RND(AA0)                    | 丸めて、TMPをQ15とする。
AA0=TMP＊x                      | Q15＊Q15→AA0はQ30
                                | このときは左シフトしない。
AA1=c0<<16                      | AA1はQ30
AA0=AA0+AA1
GAIN=RND(AA0)                   | GAINはQ14で2^xを含む。
                                | NLSGAINは2^xに関して14-NLS
NLSGAIN=14-NLS                  | 結果のQの値
```

以下に、GSTATE(1)に関する新しい値の計算を行う、ブロック９６と９７の浮動小数点擬似コードを示す。本ブロックは、ＴＴＣ標準ＪＴ－Ｇ７２８浮動小数点版には記載されておらず、以下に与えられる固定小数点擬似コードと比較するためにここで与えられる。
入力変数は、ブロック９８の出力（GAIN）、過去の励振ベクトルに対する利得コードブック入力のdB値（GCBLG）、過去の励振ベクトルに対する形状コードブックのdB値（SHAPELG）である。これらの値は、付表Ｇ－５－１／JT-G728、付表Ｇ－５－２／JT-G728でそれぞれ与えられる。これらの表は各々の値の浮動小数点表示と固定小数点表示を示す。固定小数点表示は、Q11フォーマットである。
以下浮動小数点擬似コードを示す。

```
GSTATE(1)=LOGGAIN+GCBLG(IG)+SHAPELG(IS)
GSTATE(1)<-32.ならば、GSTATE(1)=-32.とする。
```

以下固定小数点擬似コードを示す。

```
AA0=LOGGAIN<<7                  | 小数点位置をアキュムレータの上位
AA0=AA0+(GCBLG(IG)<<5)          | ワードと下位ワードの間にそろえる。
AA0=AA0+(SHAPELG(IS)<<5)        |
AA0=AA0>>7                      | Q9フォーマットとするため右シフトを行う。
AA0<-16384ならば、AA0=-16384とする。 | 下限値チェック
GSTATE(1)=AA0                   | 下位ワード16ビットを格納
```

３．１７　ブロック４９ー合成フィルタ $A(z)$ 用ハイブリッド窓かけモジュール

以下ハイブリッド窓かけモジュールの浮動小数点擬似コードを示す。

```
N1=LPC+NFRSZ                    | 定数の計算（あらかじめ計算し、
N2=LPC+NONR                     | メモリに格納しておくこともできる。）
N3=LPC+NFRSZ+NONR
N=1,2,...,N2について、以下の1行を実行する。
    SB(N)=SB(N+NFRSZ)                   | 過去の信号バッファをシフト
```

```
N=1,2,...,NFRSZ について、以下の1行を実行する。
    SB(N2+N)=STTMP(N)                          | 新しい信号でシフト
                                               | SB(N3)が最も新しいサンプル
K=1
N=N3,N3-1,...,3,2,1 について、以下の2行を実行する。
    WS(N)=SB(N)＊WNR(K)                        | 窓関数の乗算
    K=K+1
I=1,2,...,LPC+1 について、以下の4行を実行する。
    TMP=0.
    N=LPC+1,LPC+2,...,N1について、以下の1行を実行する。
        TMP=TMP+WS(N)＊WS(N+1-I)
    REXP(I)=(3/4)＊REXP(I)+TMP                 | 巡回成分の更新
I=1,2,...,LPC+1 について、以下の3行を実行する。
    RTMP(I)=REXP(I)
    N=N1+1,N1+2,...,N3について、以下の1行を実行する。
        RTMP(I)=RTMP(I)+WS(N)＊WS(N+1-I)       | 非巡回成分を加算
RTMP(1)=RTMP(1)＊WNCF                          | 白色雑音補正
```

ハイブリッド窓かけモジュール（ブロック４９）の固定小数点擬似コードは、浮動小数点版よりもずっと複雑である。これは、十分な精度を維持するために必要な分割ブロック浮動小数点（ＳＢＦＬ）フォーマットの特別な操作を行うためである。

配列STTMPは、過去の適応周期での4つの量子化音声ベクトルより成る。4つの量子化音声ベクトル（配列ST）各々が計算された時、それらは14ビット精度のＢＦＬフォーマットで表された。4つの量子化ベクトルに対する左シフトの数（ＮＬＳ）は一般に異なっている。このため、配列STTMPは、4つのＢＦＬフォーマットSTベクトルの連続からなるＳＢＦＬフォーマットで格納されている。同様に、配列SBは、21個のＢＦＬフォーマットSTベクトルの連続からなる14ビット精度のＳＢＦＬフォーマットで格納されている。STTMPを構成する4つのベクトルには、それぞれに対するＮＬＳ値がある。これらは配列NLSSTTMP( )に格納されている。SBを構成する21個のベクトルに対するＮＬＳ値は、配列NLSSB( )に格納されている。

次に、ハイブリッド窓かけモジュールの固定小数点擬似コードを示す。

```
N1=LPC+NFRSZ(=70)                  | 定数の計算（あらかじめ計算し、
N2=LPC+NONR(=85)                   | メモリに格納しておくこともできる。）
N3=LPC+NFRSZ+NONR(=105)
N4=N3/IDIM(=21)
N5=NFRSZ/IDIM(=4)
N6=N4-N5(=17)
N=1,2,...,N2について、以下の1行を実行する。
    SB(N)=SB(N+NFRSZ)                  | バッファSBの古い部分をシフト
N=1,2,...,N6について、以下の1行を実行する。
    NLSSB(N)=NLSSB(N+N5)               | 古いNLSSBをシフト
N=1,2,...,NFRSZ について、以下の1行を実行する。
    SB(N2+N)=STTMP(N)                  | SBの新しい部分でシフト
N=1,2,...,N5について、以下の1行を実行する。
    NLSSB(N6+N)=NLSSTTMP(N)            | 新しいNLSSBでシフト
                                       | NLSSBの最小値を抽出―これがNLSTMPを決定
NLSTMP=Min[NLSSB(1),NLSSB(2),...,NLSSB(N4)]
K=1                                    | ハイブリッド窓かけ関数とSBの乗算
N=N3                                   |
J=1,2,...,N4について、以下の8行を実行する。
```

```
        NRSH=NLSSB(J)-NLSTMP-1                          | Q15の乗算のため-1補正
        M=1,2,...,IDIMについて、以下の6行を実行する。
            P=SB(K)＊WNR(N)                             | WNRはQ15の乗算
            NRSH=-1 ならば、AA0=P<<1とする。
            NRSH>-1 ならば、AA0=P>>NRSH とする。
            WS(K)=RND(AA0)                              | 上位ワードを丸め、WSに格納
            N=N-1
            K=K+1
    NLSATT50=14
    HWMCORE(LPC,N1,N3,NLSATT50,WS,NLSTMP,REXP,NLSREXP,RTMP,ILLCOND)をコールする。
```

（注）： 付表Ｇ－３－４／JT-G728は、簡単に参照できるように上記の擬似コードの全変数と、それらのフォーマットおよびサイズを示している。また、表ではそれぞれの変数が、モジュールがそのプログラムを終了した後に格納しておく必要のない一時変数（一時）なのか、現在の計算の後にも同様に使用される常置変数（常置）なのかを示している。さらに表では、以前の浮動小数点擬似コードでは含まれていなかった変数なのか否かも（旧変数／新変数）として示している。

付表Ｇ－３―４／JT-G728
(ITU-T G.728)

変　数	フォーマット	サイズ	一時／常置	新／旧変数
NLSSB	int	21	常置	新変数
NLSREXP	int	1	常置	新変数
NLSSTTMP	int	4	常置	新変数
NLSTMP	int	1	一時	新変数
NRSH	int	1	一時	新変数
REXP	BFL	51	常置	旧変数
RTMP	BFL	51	常置	旧変数
SB	SBFL	105	常置	旧変数
STTMP	SBFL	20	常置	旧変数
WS	BFL	105	一時	旧変数

注：BFL＝ブロック浮動小数点、int＝16ビット整数
SBFL＝14ビット精度分割ブロック浮動小数点
WSは14ビット精度BFL　REXP, RTMPは16ビット精度

３．１８　HWMCORE－ハイブリッド窓かけモジュールのコア部

本モジュールはブロック３６，４３，４９におけるハイブリッド窓かけ計算を実行するために用いられる。それぞれのブロックは固有の初期化部分を持ち、各ブロックから本モジュールへ必要な変数が受け渡される。混乱を避けるため、3つのブロック毎に名前が異なる変数については、原則的に本モジュールの擬似コード中の名前を変更してある。 下記の付表Ｇ－３－５／JT-G728は、本モジュールで用いている名前とブロック３６，４３，４９で用いている名前の対応を示している。

付表Ｇ－３－５／JT-G728
(ITU-T G.728)

変数名	ブロック36	ブロック43	ブロック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

上記の変数に加えて一時記憶用配列WSと対応するシフト量を与える変数NLSTMPが各ブロックから本モジュールへと受け渡される。

以下固定小数点擬似コードを示す。

```
サブルーチン HWMCORE(LPO,N1,N3,NLSATT,WS,NLSTMP,RREC,NLSRREC,R,ILLCOND)
NLSAA0=2＊NLSTMP
AA0=0                                  | RREC(1) の巡回部分を計算する。
N=LPO+1,...,N1について、以下の2行を実行する。
    P=WS(N)＊WS(N)                     | WSは 2ビットのヘッドルームを持つ。
    AA0=AA0+P                          | エネルギ計算のためにAA0 は 5ビットのヘッド
                                       | ルームを持つ。
                                       | ケース1: NLSRREC > NLSAA0 の場合
```

```
NLSRREC > NLSAA0ならば、以下の22行を実行する。
    AA0=AA0 >> 1
    IR=NLSRREC-NLSAA0+1
    AA1=RREC(1) << NLSATT                    | これは掛け算でも実行可能
    AA1=-AA1+(RREC(1) << 16)                 | 減衰係数でRRECをスケーリングする。
    AA1=AA1 >> IR                            | AA0とAA1の桁合わせを行う。
    AA0=AA0+AA1
    VSCALE(AA0,1,1,30,AA0,NLSRE)をコールする。  | RRECについてNLSを探す。
    RREC(1)=RND(AA0)                         | AA1の上位16ビットを保存する。
    NLSRREC=NLSAA0-1+NLSRE
    I=1,2,...,LPO について、以下の11行を実行する。
        AA0=0                                | RREC(I+1) の巡回部分を計算する。
        N=LPO+1,...,N1について、以下の2行を実行する。
            P=WS(N)＊WS(N-I)
            AA0=AA0+P
        AA0=AA0 >> 1
        AA1=RREC(I+1) << NLSATT          | RRECを3/4 または1/2 でスケーリングする。
        AA1=-AA1+(RREC(I+1) << 16)
        AA1=AA1 >> IR
        AA0=AA0+AA1
        AA0=AA0 << NLSRE
        RREC(I+1)=RND(AA0)                   | AA0の上位16ビットを保存する。
    FIN_RECUR へジャンプする。
                                         | ケース2: NLSRREC = NLSAA0 の場合
NLSRREC = NLSAA0ならば、以下の21行を実行する。
    AA1=RREC(1) << NLSATT                | RRECを3/4 または1/2 でスケーリングする。
    AA1=-AA1+(RREC(1) << 16)
    AA0=AA0 >> 1
    AA1=AA1 >> 1
    AA0=AA0+AA1
    VSCALE(AA0,1,1,30,AA0,NLSRE)をコールする。  | RRECについてNLS を探す。
    RREC(1)=RND(AA0)                         | AA1の上位16ビットを保存する。
    NLSRREC=NLSRREC-1+NLSRE
    I=1,2,...,LPO について、以下の11行を実行する。
        AA0=0                                | RREC(I+1)の巡回部分を計算する。
        N=LPO+1,...,N1について、以下の2行を実行する。
            P=WS(N)＊WS(N-I)
            AA0=AA0+P
        AA0=AA0 >> 1
        AA1=RREC(I+1) << NLSATT          | RRECを3/4 または1/2 でスケーリングする
        AA1=-AA1+(RREC(I+1) << 16)
        AA1=AA1 >> 1
        AA0=AA0+AA1
        AA0=AA0 << NLSRE
        RREC(I+1)=RND(AA0)                   | AA0の上位16ビットを保存する。
    FIN_RECUR へジャンプする。
                                         | ケース3: NLSRREC < NLSAA0 の場合
NLSRREC < NLSAA0ならば、以下の21行を実行する。
    IR=NLSAA0-NLSRREC+1
```

```
        AA0=AA0 >> IR
        AA1=RREC(1) << NLSATT              | RRECを3/4 または1/2 でスケーリングする。
        AA1=-AA1+(RREC(1) << 16)
        AA1=AA1 >> 1
        AA0=AA0+AA1
        VSCALE(AA0,1,1,30,AA0,NLSRE)をコールする。
        RREC(1)=RND(AA0)                   | AA1の上位16ビットを保存する。
        NLSRREC=NLSRREC-1+NLSRE
        I=1,2,...,LPO について、以下の11行を実行する。
            AA0=0                          | RREC(I+1) の巡回部分を計算する。
            N=LPO+1,...,N1について、以下の2行を実行する。
                P=WS(N)＊WS(N-I)
                AA0=AA0+P
            AA0=AA0 >> IR
            AA1=RREC(I+1) << NLSATT        | RRECを3/4 または1/2 でスケーリングする。
            AA1=-AA1+(RREC(I+1) << 16)
            AA1=AA1 >> 1
            AA0=AA0+AA1
            AA0=AA0 << NLSRE
            RREC(I+1)=RND(AA0)             | AA0の上位16ビットを保存する。

FIN_RECUR:                                 | ここに来た場合は巡回成分の計算が終了して
                                           | いる。
    AA0=0                                  | R(1)の非巡回部分を計算する。
    N=N1+1,...,N3 について、以下の2行を実行する。
        P=WS(N)＊WS(N)
        AA0=AA0+P
                                           | ケース1: NLSRREC > NLSAA0 の場合
    NLSRREC > NLSAA0ならば、以下の21行を実行する。
        IR=NLSRREC-NLSAA0+1
        AA1=RREC(1) << 16
        AA1=AA1 >> IR
        AA0=AA0 >> 1
        AA1=AA0+AA1
        AA0=AA1 >> 8                       | 白色雑音補正係数を適用する。
        AA1=AA1+AA0
        VSCALE(AA1,1,1,30,AA1,NLSRR)をコールする。
        R(1)=RND(AA1)                      | AA1の上位16ビットを保存する。
        I=1,2,...,LPO について、以下の10行を実行する。
            AA0=0                          | R(I+1)の非巡回部分を計算する。
            N=N1+1,...,N3 について、以下の2行を実行する。
                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)                | 上位16ビットを保存する。
```

```
        END へジャンプする。
                                                | ケース2:NLSRREC = NLSAA0 の場合
    NLSRREC = NLSAA0ならば、以下の18行を実行する。
        AA0=AA0 >> 1
        AA1=RREC(1) << 15                       | これは掛け算でも実行可能
        AA1=AA0+AA1
        AA0=AA1 >> 8                            | 白色雑音補正係数を適用する。
        AA1=AA1+AA0
        VSCALE(AA1,1,1,30,AA1,NLSRR)をコールする。
        R(1)=RND(AA1)                           | AA1の上位16ビットを保存する。
        I=1,2,...,LPO について、以下の9行を実行する。
            AA0=0                               | R(I+1)の非巡回部分を計算する。
            N=N1+1,...,N3 について、以下の2行を実行する。
                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)                     | 上位16ビットを保存する。
        END へジャンプする。
                                                | ケース 3: NLSRREC < NLSAA0の場合
    NLSRREC < NLSAA0ならば、以下の18行を実行する。
        IR=NLSAA0-NLSRREC+1
        AA0=AA0 >> IR
        AA1=RREC(1) << 15                       | これは掛け算でも実行可能
        AA1=AA0+AA1
        AA0=AA1 >> 8                            | 白色雑音補正係数を適用する。
        AA1=AA1+AA0
        VSCALE(AA1,1,1,30,AA1,NLSRR)をコールする。
        R(1)=RND(AA1)                           | AA1の上位16ビットを保存する。
        I=1,2,...,LPO について、以下の9行を実行する。
            AA0=0                               | R(I+1)の非巡回部分を計算する。
            N=N1+1,...,N3 について、以下の2行を実行する。
                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)                     | 上位16ビットを保存する。
END:                                | ここに来た場合最後の仕事として異常状態かどうか確認する。
        ILLCOND=偽
        AA1 = 0 ならば、ILLCOND=真　とする。    | AA1は32ビットでR(LPO+1)を保存して
                                                | いる。
```

(注):以下の表は、上記の擬似コードにおける全ての変数に対応するフォーマットとサイズについて示している。また、表ではそれぞれの変数が、一時的な変数なのか（すなわちモジュールがそのプログラムを終了した後に格納しておく必要の無い変数）、常置の変数なのか（現在の計算の後にも同様に使用される変数）を

それぞれ（一時）／（常置）として示している。さらに表では、以前の浮動小数点擬似コードでは含まれていなかった変数なのか否かも（新変数）／（旧変数）として示している。

付表G－３－６／JT-G728
(ITU-T G.728)

変数名	フォーマット	サイズ	一時／常置	新／旧変数
NLSRE	int	1	一時	新変数
NLSRR	int	1	一時	新変数
NLSRREC	int	1	常置	新変数
NLSTMP	int	1	一時	新変数
RREC	BFL	51	常置	旧変数
R	BFL	51	常置	旧変数
WS	BFL	105	一時	旧変数

(注) BFL = ブロック浮動小数点形式、int = 16ビット型整数
SBFL = 14 ビット精度の分割ブロック浮動小数点形式
WSは14ビット精度のBFL, RREC とR は16ビット精度
RRECはブロック49, 36, 43のどのモジュールから呼び出されたかによって、それぞれREXP, REXPW, REXPLG を意味する

３．１９　ブロック５１－ $A(z)$ 用帯域幅拡張モジュール

以下にブロック５１－ $A(z)$ 用帯域幅拡張モジュールの浮動小数点擬似コードを示す。同様のコードは、対数利得線形予測器の帯域幅拡張モジュールであるブロック４５にも用いられている。ブロック４５では、異なったテーブルが使用され、フィルタ係数の数が多くなっている。

```
I=2,3,....,LPC+1について、以下の1行を実行する。
    ATMP(I)=FACV(I)＊ATMP(I)                    | スケール係数
ICOUNT=3まで待機、それから
I=2,3,....,LPC+1について、以下の1行を実行する。
    A(I)=ATMP(I)
```

他の帯域幅拡張係数に合わせて、FACVのテーブルはQ14フォーマットである。入力配列ATMPの値はQ13, Q14またはQ15フォーマットである。先の固定小数点レビンソン・ダービン再帰法モジュールの記述にある様に、どのフォーマットがATMPで使われるかということを示すために、NLSTMPがレビンソン・ダービン再帰法モジュールによって与えられる。FACV(I)＊ATMP(I)の乗算後、相当量の左シフトが要求される。

ATMPの最終的な値は、常にQ14フォーマットである。経験的に、ATMPの値はQ14で表現できないほど（すなわち、Q13フォーマットかそれ以下を要するほど）大きくならない。しかし安全のため、帯域幅拡張モジュールの出力でQ14フォーマットでのオーバフローを扱う準備をしなければならない。以下に示す擬似コードでは、Q14フォーマットでのオーバフローする可能性をチェックする。もしそれが検出されたなら、レビンソン・ダービン再帰法モジュールと同様の措置をとる。すなわち、予測係数を更新せずに、前適応周期の古い係数を使用し続ける。他の手段としては、2つのうちのどちらのQフォーマットが使用されるかというフィルタリングモジュールへの合図となるフラグ付きの切り替え可能なQ14/Q13フォーマットを使用する事が可能であろう。しかし、これは不必要にＤＳＰコードの複雑さと実行時間を増大させてしまう。帯域幅拡張モジュールの出力では、Q14フォーマットのオーバフローは、一度も検出されなかったので、以下で実行されるような簡単な安全チェックで十分である。

以下にブロック５１の固定小数点擬似コードを示す。

```
ICOUNTが3でなければ、以下をスキップする。
そうでなければ、以下を実行する。
```

```
                                                    | まず、ILLCONDが真かどうかのチェック
    ILLCONDが真ならば、このブロックはスキップする。
    そうでなければ、以下を実行する。
        ATMP(1)=16384
        I=2,3,4,...,LPC+1 について、以下の6行を実行する。
            AA0=FACV(I)＊ATMP(I)                    | AA0 は Q27,Q28または Q29
            NLSATMP=13ならば、AA0=AA0<<3とする。    | 3ケース全てについて、適当な
            NLSATMP=14ならば、AA0=AA0<<2とする。    | シフト量でAA0をQ30にする。
            NLSATMP=15ならば、AA0=AA0<<1とする。
            AA0がオーバフローしたら、LABELへジャンプする。        | 真でなければ、
            ATMP(I)=RND(AA0)                                    | 上位ワードに丸める。
        I=2,3,...,LPC+1 について、以下の1行を実行する。
            A(I)=ATMP(I)
        モジュール正常終了。
LABEL : | ここに来た場合、AをQ14で表そうとするとオーバフローになる。
        | この場合は、合成フィルタ係数は更新しない。（つまり、前適応周期の合成
        | フィルタ係数を使用し続ける。）
```

３．２０　ブロック７１，７２ー長期ポストフィルタ、短期ポストフィルタ

ブロック７１と７２は中間変数TEMPの精度を保つために結合されている。ここで、TEMPは浮動小数点擬似コードにおいて両モジュール間で受け渡される変数である。これら2つのブロックの浮動小数点擬似コードを初めに示す。

```
    K=1,2,...,IDIMについて、以下の1行を実行する。
        TEMP(K)=GL＊(ST(K)+B＊ST(K-KP))              | 長期ポストフィルタリング
    K=-NPWSZ-KPMAX+1,...,-2,-1,0について、以下の1行を実行する。
        ST(K)=ST(K+IDIM)                             | 復号音声バッファをシフト
    K=1,2,...,IDIMについて、以下を実行する。
        TMP=TEMP(K)
        J=10,9,...,3,2について、以下の2行を実行する。
            TEMP(K)=TEMP(K)+STPFFIR(J)＊AZ(J+1)      | フィルタの全零部分
            STPFFIR(J)=STPFFIR(J-1)
        TEMP(K)=TEMP(K)+STPFFIR(1)＊AZ(2)            | 最後の乗算
        STPFFIR(1)=TMP
        J=10,9,...,3,2について、以下の2行を実行する。
            TEMP(K)=TEMP(K)-STPFIIR(J)＊AP(J+1)      | フィルタの全極部分
            STPFIIR(J)=STPFIIR(J-1)
        TEMP(K)=TEMP(K)-STPFIIR(1)＊AP(2)            | 最後の乗算
        STPFIIR(1)=TEMP(K)
        TEMP(K)=TEMP(K)+STPFIIR(2)＊TILTZ            | スペクトル傾斜補正フィルタ
    次のKに対しても、上記演算を繰り返す。
```

固定小数点擬似コードを以下に示す。変数STPFFIRとSTPFIIRはQ2に固定されている。

```
    K=1,2,...,IDIMについて、以下の字下げ行を実行する。
                                                 | 最初に長期ポストフィルタの実行
        AA0=GL＊SST(K)                           | GLはQ14、SST(1:5)はQ2
        AA0=AA0+GLB＊SST(K-KP)                   | GLBはQ16、SST(-239:0)はQ0
        AA1=AA0                                  | 短期ポストフィルタの実行
```

```
                                                    | FIRフィルタ部の実行
    J=10,9,...,3,2について、以下の2行を実行する。
        AA1=AA1+STPFFIR(J)＊AZ(J+1)                 | AZはQ14、STPFFIR(J)はQ2
        STPFFIR(J)=STPFFIR(J-1)
    AA1=AA1+STPFFIR(1)＊AZ(2)
    AA0=AA0<<2
    STPFFIR(1)=RND(AA0)                             | Q2に合わせてSTPFFIRに入れる。
                                                    | IIRフィルタ部の実行
    J=10,9,...3,2について、以下の2行を実行する。
        AA1=AA1-STPFIIR(J)＊AP(J+1)                 | APはQ14、STPFIIR(J)はQ2
        STPFIIR(J)=STPFIIR(J-1)
    AA1=AA1-STPFIIR(1)＊AP(2)
    AA0=AA1>>14
                                                    | オーバフローのチェック
    AA0>32767ならば、AA0=32767とする。
    AA0<-32768ならば、AA0=-32768とする。
    STPFIIR(1)=AA0
                                                    | スペクトル傾斜補正フィルタ
    AA1=AA1+STPFIIR(2)＊TILTZ                       | TILTZはQ14
    AA1=AA1>>14
    AA1>32767ならば、AA1=32767とする。
    AA1<-32768ならば、AA1=-32768とする。
    TEMP(K)=AA1
次のKに対しても、上記演算を繰り返す。
                                                    | 長期ポストフィルタの
                                                    | メモリバッファをシフト
K=-NPWSZ-KPMAX+1,...,-7,-6,-5について、以下の1行を実行する。
    SST(K)=SST(K+IDIM)                              | 復号音声バッファをシフト
K=-4,-3,...,0について、以下の1行を実行する。
    SST(K)=SST(K+IDIM)>>2                           | 復号音声バッファをシフトし、
                                                    | Q2からQ0に変更
```

３．２１　ブロック７３，７４－絶対値合計計算器

ブロック７３と７４は非常に似ている。両者の結果は倍精度に保たれる。ここで示す様に、これらの結果をブロック７５の処理の前に記憶する必要はない。以下にブロック７３の浮動小数点擬似コードを示す。ここで、浮動小数点コード中の変数STは名前がSSTに変えてあることに注意すること。これは、浮動小数点コードと後に示す固定小数点コードとの整合性を保つためである。SST(1:5)はQ2で表される。

```
SUMUNFIL=0
K=1,2,...,IDIMについて、以下の1行を実行する。
    SUMUNFIL=SUMUNFIL+| SST(K)|
```

以下にブロック７４の擬似コードを示す。

```
SUMFIL=0.
K=1,2,...,IDIMについて、以下の1行を実行する。
    SUMFIL=SUMFIL+［TEMP(K)の絶対値］
```

以下にブロック７３、７４の固定小数点擬似コードを示す。

```
AA1=0
AA0=0
K=1,2,...,IDIMについて、以下の2行を実行する。
    AA0=AA0+| SST(K)|                  | SST(K)の絶対値を加算
    AA1=AA1+| TEMP(K)|                 | TEMP(K)の絶対値を加算
                                       | AA0=SUMUNFIL
                                       | AA1=SUMFIL
                                       | SSTとTEMPはQ2なので、AA0,AA1はQ2
                                       | AA0,AA1はブロック７５で使用される。
```

３．２２　ブロック７５－スケーリングファクタ計算器

ブロック７５はSUMUNFIL/SUMFILの比を計算し、NLSSCALE精度でSCALEに結果を格納する。SUMUNFIL(AA0)とSUMFIL(AA1)はそれぞれブロック７３，７４の出力である。以下に浮動小数点擬似コードを示す。

```
SUMFIL>1ならば、SCALE=SUMUNFIL/SUMFIL、そうでなければSCALE=1とする。
```

以下に固定小数点擬似コードを示す。

```
AA1>4ならば、以下の字下げ行を実行する。
    VSCALE(AA1,1,1,30,AA1,NLSDEN)をコールする。
    DEN=RND(AA1)
    VSCALE(AA0,1,1,30,AA0,NLSNUM)をコールする。
    NUM=RND(AA0)                       | NLSNUMとNLSDENは16ビットのオフセット
                                       | があるがそれは相殺される。
    DIVIDE(NUM,NLSNUM,DEN,NLSDEN,SCALE,NLSSCALE)をコールする。
そうでなければ、SCALE=16384, NLSSCALE=14とする。
```

３．２３　ブロック７６－1次低域通過フィルタ、ブロック７７－出力利得調整ユニット

以下にこれら2ブロックの浮動小数点擬似コードを示す。

```
K=1,2,...,IDIMについて、以下の2行を実行する。
    SCALEFIL=AGCFAC＊SCALEFIL+(1-AGCFAC)＊SCALE    | 低域通過フィルタリング
    SPF(K)=SCALEFIL＊TEMP(K)                      | 出力の調整
```

固定小数点擬似コードでは、ループ内での減算と乗算を省くために第2項を一度計算し、それから各繰り返し計算において加算に使用する。以下に固定小数点擬似コードを示す。

```
AA1=AGCFAC1＊SCALE                     | AGCFAC1=20972 (Q21)  =0.010000228
NRS=NLSSCALE-14+(21-14)                | 右シフト量の計算
NRS≧0ならば、AA1=AA1>>NRSとする。      | AA1をQ28に合わせる。
NRS<0ならば、AA1=AA1<<-NRSとする。     | NRSが負のときは左シフト
K=1,2,...,IDIMについて、以下を実行する。
                                       | 低域通過フィルタリング
    AA0=AA1+AGCFAC＊SCALEFIL           | AGCFAC=16220 (Q14)  ,SCALEFILはQ14
    AA0=AA0<<2                         | SCALEをQ14にする。
    SCALEFIL=RND(AA0)
                                       | 出力の調整
```

```
    AA0=SCALEFIL＊TEMP(K)              | TEMP(K)はQ2
    AA0=AA0<<2
    SPF(K)=RND(AA0)                   | SPF(K)はQ2
次のKに対しても、上記演算を繰り返す。
```

３．２４　ブロック８１－10次ＬＰＣ逆フィルタ

これは、10次ＬＰＣ逆フィルタ（ブロック８１）のための浮動小数点版擬似コードである。

```
IP=NPWSZならば、IP=NPWSZ-NFRSZとする。          |  IPのチェックと更新
K=1,2,...,IDIMについて、以下の7行を実行する。
    ITMP=IP+K
    D(ITMP)=ST(K)
    J=10,9,...,3,2について、以下の2行を実行する。
        D(ITMP)=D(ITMP)+STLPCI(J)＊APF(J+1)   | FIR フィルタリング
        STLPCI(J)=STLPCI(J-1)                 | 入力シフト
    D(ITMP)=D(ITMP)+STLPCI(1)＊APF(2)         | 最後の1サンプルの処理
    STLPCI(1)=ST(K)                           | 入力シフト
IP=IP+IDIM                                    | IPの更新
```

固定小数点コードにおいて、まずSTをブロック浮動小数点から固定小数点Q2フォーマットに変換し、それから長期ポストフィルタによる後の使用のためにQ2フォーマット化されたSTを長期ポストフィルタメモリバッファSSTに書き込む必要がある。このバッファは、浮動小数点版擬似コードにおいてすでにSTと呼ばれていた事に注意しなければならない。STはブロック浮動小数点であり、メモリバッファは、Q2フォーマットである。混乱を避けるためにメモリバッファをSSTに改名する必要がある。そしてＬＰＣ逆フィルタ演算を行う。ＬＰＣフィルタ係数APFは、Q13フォーマットで表現されている事に注意しなければならない。

```
NLS=16-NLSST+2                                | Q2ﾌｫｰﾏｯﾄにするための左ｼﾌﾄ量の計算
K=1,2,...,IDIMについて、以下の2行を実行する。
    AA0=ST(K)<<NLS
    SST(K)=RND(AA0)                           | SSTは長期ﾎﾟｽﾄﾌｨﾙﾀの新しいﾊﾞｯﾌｧ
IP=NPWSZならば、IP=NPWSZ-NFRSZとする。          | IPのチェックと更新
                                              | LPC逆フィルタリングを開始
K=1,2,...,IDIMについて、以下の10行を実行する。
    AA0=SST(K)
    AA0=AA0<<13
    J=10,9,...,3,2について、以下の2行を実行する。
        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)はQ1フォーマットである
IP=IP+IDIM
```

３．２５　ブロック８２－ピッチ周期抽出モジュール

ピッチ抽出モジュール（ブロック８２）のための浮動小数点版擬似コードから始める。

```
ICOUNTが3でなければ、本ブロックをスキップする。
```

```
そうでなければ、以下を実行する。
                                        | 低域通過フィルタリングと4:1 の間引き
K=NPWSZ-NFRSZ+1,...,NPWSZについて、以下の7行を実行する。
    TMP=D(K)-STLPF(1)＊AL(1)-STLPF(2)＊AL(2)-STLPF(3)＊AL(3)  |  IIRフィルタ
    Kが 4で割り切れれば、以下の2行を実行する。
        N=K/4                           | 必要な場合のみ FIRフィルタリングを実行
        DEC(N)=TMP＊BL(1)+STLPF(1)＊BL(2)+STLPF(2)＊BL(3)+STLPF(3)＊BL(4)
    STLPF(3)=STLPF(2)
    STLPF(2)=STLPF(1)                   | 低域通過フィルタメモリのシフト
    STLPF(1)=TMP
M1=KPMIN/4                              | 間引きされたLPC残差領域での相関
M2=KPMAX/4                              | のピークの計算を開始
CORMAX=装置で表現できる負の最小値（負の絶対値最大の値）
J=M1,M1+1,...,M2について、以下の6行を実行する。
    TMP=0
    N=1,2,...,NPWSZ/4について、以下の1行を実行する。
        TMP=TMP+DEC(N)＊DEC(N-J)        | TMP = 間引きされた領域での相関
    TMP>CORMAXならば、以下の2行を実行する。
        CORMAX=TMP                      | 相関の最大値と対応する時間遅延を
        KMAX=J                          | 探索
N=-M2+1,-M2+2,...,(NPWSZ-NFRSZ)/4 について、以下の1行を実行する。
    DEC(N)=DEC(N+IDIM)                  | 間引されたLPC残差用バッファをシフト
M1=4＊KMAX-3                            | 間引きされていない領域での相関の
M2=4＊KMAX+3                            | ピークの計算を開始
M1<KPMINならば、M1=KPMINとする。        | M1が範囲外かどうかを確認
M2>KPMAXならば、M2=KPMAXとする。        | M2が範囲外かどうかを確認
CORMAX=装置で表現できる負の最小値（負の絶対値最大の値）
J=M1,M1+1,...,M2について、以下の6行を実行する。
    TMP=0
    K=1,2,...,NPWSZ について、以下の1行を実行する。
        TMP=TMP+D(K)＊D(K-J)            | 間引きされていない領域での相関
    TMP>CORMAXならば、以下の2行を実行する。
        CORMAX=TMP                      | 相関の最大値と対応する時間遅延を
        KP=J                            | 探索
M1=KP1-KPDELTA                          | 前回フレームのピッチ周期の周辺に
M2=KP1+KPDELTA                          | サーチ範囲を決定
KP<M2+1ならば、LABELへジャンプする。    | 真ならばKPはピッチの倍数になり得ない
M1<KPMINならば、M1=KPMINとする。        | M1が範囲外かどうかを確認
M2>KPMAXならば、M2=KPMAXとする。        | M2が範囲外かどうかを確認
CMAX=装置で表現できる負の最小値（負の絶対値最大の値）
J=M1,M1+1,...,M2について、以下の6行を実行する。
    TMP=0
    K=1,2,...,NPWSZ について、以下の1行を実行する。
        TMP=TMP+D(K)＊D(K-J)            | 間引きされていない領域での相関
    TMP>CMAXならば、以下の2行を実行する。
        CMAX=TMP                        | 相関の最大値と対応する時間遅延を
        KPTMP=J                         | 探索
SUM=0
TMP=0                                   | タップ係数の計算を開始
```

```
        K=1,2,...,NPWSZについて、以下の2行を実行する。
            SUM=SUM+D(K-KP)＊D(K-KP)
            TMP=TMP+D(K-KPTMP)＊D(K-KPTMP)
        SUM=0ならば、TAP=0、そうでなければ、TAP=CORMAX/SUMとする。
        TMP=0ならば、TAP1=0、そうでなければ、TAP1=CMAX/TMPとする。
        TAP>1ならば、TAP=1.とする。                    | TAPを0から1の範囲に制限
        TAP<0ならば、TAP=0.とする。
        TAP1>1ならば、TAP1=1.とする。                  | TAP1を0から1の範囲に制限
        TAP1<0ならば、TAP1=0.とする。
                                                        | TAP1が十分大きければ、KPを基本
                                                        | ピッチに置換
        TAP1>TAPTH＊TAPならば、KP=KPTMPとする。
LABEL:  KP1=KP                                          | 前回フレームのピッチ周期を更新
        K=-KPMAX+1,-KPMAX+2,...,NPWSZ-NFRSZ について、以下の1行を実行する。
            D(K)=D(K+NFRSZ)                             | LPC 残差用バッファをシフト
```

このブロックの固定小数点版では、D配列と低域通過フィルタの状態変数は、Q1フォーマットで表現されている。これは、相関とエネルギの計算においてオーバフローを避ける為である。固定小数点擬似コードは以下に示されている。

```
        ICOUNTが3でなければ、本ブロックをスキップする。
        そうでなければ、以下を実行する。
        K=NPWSZ-NFRSZ+1,...,NPWSZについて、以下の17行を実行する。
            AA0=D(K)＊BL(0)                 | FIR フィルタ部を最初に実行する。
            AA0=AA0+LPFFIR(1)＊BL(1)        | D(K)はQ1ﾌｫｰﾏｯﾄ、BL()はQ19ﾌｫｰﾏｯﾄである。
            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はQ1フォーマットである。
            AA0=AA0>>6                      | ここから IIRフィルタ部を実行する。
            AA0=AA0-LPFIIR(1)＊AL(1)        | AL()はQ13ﾌｫｰﾏｯﾄ、LPFIIR()はQ1ﾌｫｰﾏｯﾄである。
            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はQ1フォーマットである。
            N=(K>>2)
            K=(N<<2)ならば、DEC(N)=LPFIIR(1)とする。   | DEC(N)はQ1ﾌｫｰﾏｯﾄである。
        M1=KPMIN/4                          | 間引きされたLPC残差領域での相関
        M2=KPMAX/4                          | のピークの計算を開始
        AA1=-2147483648                     | =-2^31
        J=M1,M1+1,...,M2について、以下の6行を実行する。
            AA0=0
            N=1,2,...,NPWSZ/4 について、以下の1行を実行する。
                AA0=AA0+DEC(N)＊DEC(N-J)
            AA0>AA1ならば、以下の2行を実行する。
                AA1=AA0                     | 相関の最大値と対応する時間遅延を
```

```
        KMAX=J                              | 探索
N=-M2+1,-M2+2,...,(NPWSZ-NFRSZ)/4について、以下の1行を実行する。
    DEC(N)=DEC(N+IDIM)
M1=4＊KMAX-3                                | 間引きされていない領域での相関の
M2=4＊KMAX+3                                | ピークの計算を開始
M1<KPMINならば、M1=KPMINとする。            | M1が範囲外かどうかを確認
M2>KPMAXならば、M2=KPMAXとする。            | M2が範囲外かどうかを確認
AA1=-2147483648                             | =-2^31
J=M1,M1+1,...,M2について、以下の6行を実行する。
    AA0=0
    K=1,2,...,NPWSZについて、以下の1行を実行する。
        AA0=AA0+D(K)＊D(K-J)                | 間引きされていない領域での相関
    AA0>AA1ならば、以下の2行を実行する。
        AA1=AA0
        KP=J
CORMAX=AA1                                  | 倍精度の値をCORMAXに格納する。
M1=KP1-KPDELTA                              | 前回フレームのピッチ周期の周辺に
M2=KP1+KPDELTA                              | サーチ範囲を決定
KP<M2+1ならば、LABELへジャンプする。        | 真ならばKPはピッチの倍数になり得ない。
M1<KPMINならば、M1=KPMINとする。            | M1が範囲外かどうかを確認
M2>KPMAXならば、M2=KPMAXとする。            | M2が範囲外かどうかを確認
AA1=-2147483648                             | =-2^31
J=M1,M1+1,...,M2について、以下の6行を実行する。
    AA0=0
    K=1,2,...,NPWSZについて、以下の1行を実行する。
        AA0=AA0+D(K)＊D(K-J)                | 間引きされていない領域での相関
    AA0>AA1ならば、以下の2行を実行する。
        AA1=AA0                             | 相関の最大値と対応する時間遅延を
        KPTMP=J                             | 探索
CMAX=AA1                                    | 倍精度の値を CMAXに格納する。
AA0=0
AA1=0
K=1,2,...,NPWSZについて、以下の2行を実行する。
    AA0=AA0+D(K-KP)＊D(K-KP)
    AA1=AA1+D(K-KPTMP)＊D(K-KPTMP)
                                            | TAPを検出する。
                                            | 必要であればタップ係数をクリップする。
AA0=0ならば、CORMAX=0とする。
AA1=0ならば、CMAX=0とする。
CORMAX>AA0ならば、CORMAX=AA0とする。
CORMAX<0ならば、CORMAX=0とする。
CMAX>AA1ならば、CMAX=AA1とする。
CMAX<0ならば、CMAX=0とする。
AA0>AA1ならば、以下の2行を実行する。
    VSCALE(AA0,1,1,30,AA0,NLS)をコールする。
    AA1=AA1<<NLS
そうでなければ、以下の2行を実行する。
    VSCALE(AA1,1,1,30,AA1,NLS)をコールする。
    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でありQ16ﾌｫｰﾏｯﾄ である。
AA0=CMAX＊SUM
AA0>AA1ならば、KP=KPTMPとする。
LABEL: KP1=KP　　| KP1の更新とLPC残差のシフト
K=-KPMAX+1,-KPMAX+2,...,NPWSZ-NFRSZ について、以下の1行を実行する。
　D(K)=D(K+NFRSZ)　　| LPC 残差用バッファをシフト

３．２６　ブロック８３－ピッチ予測器タップ計算器

まず、浮動小数点版の擬似コードを示す。ここでは、長期ポストフィルタのメモリバッファの名前をSTではなくSSTとする。

ICOUNT=3でなければ、本ブロックをスキップする。
そうでなければ、以下を実行する。
　SUM=0.
　TMP=0.
　K=-NPWSZ+1,-NPWSZ+2,...,0について、以下の2行を実行する。
　　SUM=SUM+SST(K-KP)＊SST(K-KP)
　　TMP=TMP+SST(K)＊SST(K-KP)
　SUM=0.ならば、PTAP=0とし、そうでなければ、PTAP=TMP/SUMとする。

以下固定小数点擬似コードを示す。SST( )がQ0の13ビットであることに注意すること。自己相関の計算では、SSTどうしの乗算、あるいはSSTとその遅延サンプルとの乗算の結果はQ0の25ビットとなる。

ICOUNT=3でなければ、本ブロックをスキップする。
そうでなければ、以下を実行する。
　AA0=0
　AA1=0
　K=-NPWSZ+1,-NPWSZ+2,...,0について、以下の4行を実行する。
　　P=SST(K-KP)＊SST(K-KP)
　　AA0=AA0+P
　　P=SST(K)＊SST(K-KP)
　　AA1=AA1+P
　AA0=0ならば、PTAP=0とし、コールしたプログラムにもどる。
　AA1≦0ならば、PTAP=0とし、コールしたプログラムにもどる。
　AA1＞AA0ならば、PTAP=16384とし、　　| NLSPTAP=14
　そうでなければ、以下を実行する。
　　VSCALE(AA0,1,1,30,AA0,NLSDEN)をコールする。
　　VSCALE(AA1,1,1,30,AA1,NLSNUM)をコールする。
　　NUM=RND(AA1)
　　DEN=RND(AA0)
　　DIVIDE(NUM,NLSNUM,DEN,NLSDEN,PTAP,NLSPTAP)をコールする。

```
NRS=NLSPTAP-14
PTAP=PTAP>>NRS                          | NLSPTAP=14
```

３．２７　ブロック８４ー長期ポストフィルタ係数計算器

ブロック８４の浮動小数点の擬似コードを示す。

```
ICOUNT=3でなければ、本ブロックをスキップする。
そうでなければ、以下を実行する。
    PTAP>1ならば、PTAP=1.とする。          | PTAPを1に制限
    PTAP<PPFTHならば、PTAP=0.とする。       | PTAPがしきい値より小さければ
                                           | 長期ポストフィルタをオフ
    B=PPFZCF＊PTAP
    GL=1/(1+B)
```

以下固定小数点擬似コードを示す。GLとBの積であるGLBという変数を新たに定義する。これは、後の乗算の負荷を軽減する。BとGLBはQ16で出力され、GLはQ14で出力される。

```
                                           | ブロック83によりPTAP<16385であることに注意
ICOUNT=3でなければ、本ブロックをスキップする。
そうでなければ、以下を実行する。
    PTAP<PPFTHならば、PTAP=0とする。       | PPFTH=9830(Q14)
    AA0=PPFZCF＊PTAP                       | PPFZCF=9830(Q16)　PTAPはQ14
    B=AA0>>14                              | Q16でBを格納
    AA0=AA0>>16                            | AA0はQ14のBに等しい。
    AA0=AA0+16384
    DEN=AA0                                | DENはQ14
    DIVIDE(16384,14,DEN,14,GL,NLS)をコールする。
    AA0=GL＊B                              | NSL=14または15、BのNLSは16
    GLB=AA0>>NLS                           | GLBはGL＊Bであり、あらかじめQ16で
                                           | ブロック71のために計算される。
    NRS=NLS-14
    NRS>0ならば、GL=GL>>NRSとする。        | GLをQ14とする。
```

３．２８　ブロック８５ー短期ポストフィルタ係数計算器

このブロックの浮動小数点擬似コードを示す。

```
ICOUNT=1でなければ、本ブロックをスキップする。
そうでなければ、以下を実行する。
    I=2,3,...,11について、以下の2行を実行する。
        AP(I)=SPFPCFV(I)＊APF(I)           | 分母の係数の大きさの補正
        AZ(I)=SPFZCFV(I)＊APF(I)           | 分子の係数の大きさの補正
    TILTZ=TILTF＊RC1                       | 傾斜補正フィルタ係数
```

固定小数点擬似コードにおいては、ダービンの再帰法における異常状態あるいは予測係数が、Q13ですらオーバフローが発生する可能性について注意が必要である。（Q13で不十分な場合が観測されている訳ではないが、オーバフローの可能性はまだ考慮にいれなければならない。）変数ILLCONDPはブロック５０から与えられるフラグであり、このフラグはブロック５０の結果が有効であるか否かを示している。ＪＴ－Ｇ７２８では仮にILLCONDPが真であった場合、ダービン再帰法の結果は使用されない事を暗黙の仮定としている。これはすなわち、10次の再帰法が終了した後、ATMPはAPFにコピーされない。ここでは同様の仮定を用い、ILLCONDPが真な

らばAP, AZ, TILTZは更新しない。

つぎに、ダービン再帰法により得られる係数APF( )をQ13, Q14, Q15で扱わなければならない。NLSAPFは、APFの左シフト数を示す。出力において、後のＬＰＣ逆フィルタで使用するのに便利なように、APF( )をQ13で格納することが望ましいし、また、分母と分子の係数のAP( )とAZ( )は、結果出力としてQ14に納める必要がある。また、TILTZは、Q14で出力される。APがQ14に納まらないならば、AP, AZ, TILTZを更新しないが、APFのために新しい値を使用する。それらは、すでにQ13フォーマットであるべきである。以下固定小数点擬似コードを示す。

```
ICOUNT=1でなければ、本ブロックをスキップする。
そうでなければ、以下を実行する。
                                        | ILLCONDPが真であることを確認
ILLCONDP=真ならば、本ブロックをスキップする。
そうでなければ、以下を実行する。
                                        | 次に分母係数を確認し、Q14でオーバ
                                        | フローするならば、AP,AZ,TILTZを更新
                                        | しない。オーバフローする場合、一時
                                        | 記憶配列WSを使用する。
                                        | これによりAPの値は保護される。
    I=2,3について、以下の6行を実行する。
        AA0=SPFPCFV(I)＊APF(I)          | SPFPCFVはQ14、AA0は14+NLSAPF
        NLSAPF=13 ならば、AA0=AA0<<3とする。  | 3ケース全てについて、適当な
        NLSAPF=14 ならば、AA0=AA0<<2とする。  | シフト数でQ30にする。
        NLSAPF=15 ならば、AA0=AA0<<1とする。
        AA0 が上記でオーバフローしたならば、LABELへジャンプする。
        WS(I)=RND(AA0)                  | 上位ワードで丸めてWSに格納
                                | オーバフローは2と3の間のみに起きるので
                                | これらをAPにコピーして処理を継続
    I=2,3について、以下の1行を実行する。
        AP(I)=WS(I)
                                        | 次に残りの配列の処理を行う。
    I=4,5,...,11について、以下の5行を実行する。
        AA0=SPFPCFV(I)＊APF(I)          | SPFPCFVはQ14、AA0は14+NLSAPF
        NLSAPF=13ならば、AA0=AA0<<3とする。  | 3ケース全てについて、適当な
        NLSAPF=14ならば、AA0=AA0<<2とする。  | シフト数でAA0をQ30にする。
        NLSAPF=15ならば、AA0=AA0<<1とする。
        AP(I)=RND(AA0)                  | 上位ワードで丸めてAPに格納
                                        | 次に分子の係数の処理を行う。
                                        | 分母がオーバフローしないならば、
                                        | 分子もまたオーバフローしない。
    I=2,3,...,11について、以下の5行を実行する。
        AA0=SPFZCFV(I)＊APF(I)          | SPFZCFVはQ14、AA0は14+NLSAPF
        NLSAPF=13ならば、AA0=AA0<<3とする。  | 3ケース全てについて、適当な
        NLSAPF=14ならば、AA0=AA0<<2とする。  | シフト数でAA0をQ30にする。
        NLSAPF=15ならば、AA0=AA0<<1とする。
        AZ(I)=RND(AA0)                  | 上位ワードで丸めてAZに格納
                                        | 次にTILTZを更新する。
    AA0=TILTF＊RC1                      | TILTF=4915(Q15)
    TILTZ=RND(AA0)                      | RC1はQ15、TILTZはQ14
LABEL:                                  | 後のLPC逆フィルタのために
                                        | Q13でAPF( )を格納
```

```
                                                | ケース1:NLSAPF=13のとき
                                                | 無処理
NLSAPF=14ならば、以下の3行を実行する。          | ケース2:NLSAPF=14の
    I=2,3,4,...,11について、次の2行を実行する。 | とき15ビットシフトして
        AA0=APF(I)<<15                          | 丸める。
        APF(I)=RND(AA0)
NLSAPF=15ならば、以下の3行を実行する。          | ケース3:NLSAPF=15の
    I=2,3,4,...,11について、以下の2行を実行する。 | とき14ビットシフト
        AA0=APF(I)<<14                          | して丸める。
        APF(I)=RND(AA0)
```

上記の擬似コードにおいて、全ての擬似コードをNLSAPFの3つの値それぞれについて書き直すならば、Iに関するループに含まれる3つの“NLSAPF=...ならば... ”は省略できる。このように書きかえても正確に同じ結果を生じるし、ほとんどのプログラマブルデバイスにおいて高速に処理することができる。

４．ＬＤ－ＣＥＬＰの内部変数

この章では、表５－１／JT-G728及び表５－２／JT-G728の改訂版を示す。付表Ｇ－４－１／JT-G728は、表５－１／JT-G728の短縮版である。この表には、ほかでは定義されていないコーデックの基本的なパラメータのみを載せている。各変数の固定小数点フォーマット及びその値の欄のスペースを確保するために、表５－１／JT-G728に示されているシンボル及び初期値の欄は削除した。付表Ｇ－４－２／JT-G728は表５－２／JT-G728の固定小数点版である。ここでも、固定小数点フォーマットの欄を入れるために、表５－２／JT-G728から付表Ｇ－４－１／JT-G728と同じ欄を削除した。また固定小数点の記述のみに関係するいくつかの新しい変数を載せている。

付表Ｇ－４－１／JT-G728　ほかでは定義されていないコーデックの
(ITU-T G.728)　　基本的なパラメータ

名　称	浮動小数点の値	固定小数点の値	Ｑフォーマット	説　　明
AGCFAC	0.99	16220	Q14	ＡＧＣ適応速度補正変数
AGCFAC1	0.01	20972	Q21	(1-AGCFAC)の値
GOFF	32	16384	Q9	対数利得オフセット値
PPFTH	0.6	9830	Q14	長期ポストフィルタの有無の閾値
PPFZCF	0.15	9830	Q16	長期ポストフィルタの零点補正係数
TAPTH	0.4	26214	Q16	基本ピッチの置換に関する閾値
TILTF	0.15	4915	Q15	スペクトル傾斜補正係数

付表Ｇ－４－２／JT-G728（１／３）　ＬＤ－ＣＥＬＰの内部処理変数
(ITU-T G.728)

名　称	配列のインデックスの範囲	固定小数点フォーマット	説　　明
A	1 ~ LPC+1	Q14	合成フィルタの係数
AL	1 ~ 3	Q13	1kHz 低域通過フィルタの分母の係数
AP	1 ~ 11	Q14	短期ポストフィルタの分母の係数
APF	1 ~ 11	Q13	10次ＬＰＣフィルタの係数
ATMP	1 ~ LPC+1	Q13/Q14/Q15	合成フィルタの係数の一時記憶
AWP	1 ~ LPCW+1	Q14	聴覚重み付けフィルタの分母の係数
AWZ	1 ~ LPCW+1	Q14	聴覚重み付けフィルタの分子の係数
AWZTMP	1 ~ LPCW+1	Q13/Q14/Q15	聴覚重み付けフィルタの係数の一時記憶
AZ	1 ~ 11	Q14	短期ポストフィルタの分子の係数
B	1	Q16	長期ポストフィルタの係数
BL	1 ~ 4	Q19	1kHz 低域通過フィルタの分子の係数
D	-139 ~ 100	Q1	ＬＰＣ予測残差
DEC	-34 ~ 25	Q1	4:1に間引きしたＬＰＣ予測残差
ET	1 ~ IDIM	15b BFL	利得調整された励振ベクトル
FACV	1 ~ LPC+1	Q14	合成フィルタの帯域幅拡張ベクトル
FACGPV	1 ~ LPCLG+1	Q14	利得予測器の帯域幅拡張ベクトル
G2	1 ~ NG	Q12	利得コードブックの利得の2倍
GAIN	1	SFL	励振利得
GB	1 ~ NG-1	Q13	隣接利得値間の中間点
GL	1	Q14	長期ポストフィルタのスケーリングファクタ
GLB	1	Q16	長期ポストフィルタのGLとB の積
GP	1 ~ LPCLG+1	Q14	対数利得予測係数, 初期値 =16384, -16384, 0, .. , 0
GPTMP	1 ~ LPCLG+1	Q13/Q14/Q15	対数利得線形予測器の一時記憶配列
GQ	1 ~ NG	Q13	利得コードブックの利得値
GSQ	1 ~ NG	Q11	利得コードブックの利得値の自乗値
GSTATE	1 ~ LPCLG	Q9	対数利得予測器のメモリ、初期値 =-16384

付表Ｇ－４－２／JT-G728（２／３）　ＬＤ－ＣＥＬＰの内部処理変数
(ITU-T G.728)

名　称	配列の インデックスの範囲	固定小数点 フォーマット	説　　明
GTMP	1 ~ 4	Q9	対数利得の一時記憶、初期値＝-16384
H	1 ~ IDIM	Q13	$F(z)W(z)$ のインパルス応答ベクトル
ICHAN	1	Q0	伝送される最適なコードブックインデックス
ICOUNT	1	Q0	音声ベクトルカウンタ（1~4）
IG	1	Q0	3bit の最適な利得コードブックインデックス
ILLCOND	1	Q0	合成フィルタの異常状態フラグ
ILLCONDG	1	Q0	対数利得予測器の異常状態フラグ
ILLCONDP	1	Q0	ポストフィルタの異常状態フラグ
ILLCONDW	1	Q0	重み付けフィルタの異常状態フラグ
IP	1	Q0	ＬＰＣ予測残差のアドレスポインタ
IS	1	Q0	7bit の最適な形状コードブックインデックス
KP	1	Q0	現フレームのピッチ周期
KP1	1	Q0	前フレームのピッチ周期
LOGGAIN	1	Q9	対数励振利得
LPFFIR	3	Q1	低域通過フィルタのＦＩＲメモリ
LPFIIR	3	Q1	低域通過フィルタのＩＩＲメモリ
NLSATMP	1	Q0	ATMPに対するダービン再帰法精度フラグ
NLSAWZTMP	1	Q0	AWZTMPに対するダービン再帰法精度フラグ
NLSGPTMP	1	Q0	GPTMPに対するダービン再帰法精度フラグ
NLSET	1	Q0	ETの左シフト量
NLSGAIN	1	Q0	GAINの左シフト量
NLSREXP	1	Q0	REXPの左シフト量、初期値=31
NLSREXPLG	1	Q0	REXPLGの左シフト量、初期値=31
NLSREXPW	1	Q0	REXPWの左シフト量、初期値=31
NLSSB	21	Q0	SBの左シフト量、初期値=16
NLSST	1	Q0	復号器でのSTの左シフト量
NLSSTATE	11	Q0	STATELPCの左シフト量、初期値=16
NLSSTTMP	4	Q0	STTMPの左シフト量、初期値=16
PN	1 ~ IDIM	Q7	コードブック検索のための相関ベクトル
PTAP	1	Q14	ブロック８３で算出するピッチ予測器のタップ係数
R	1 ~ 11	BFL	自己相関係数
RC	1	Q15	反射係数
RC1	1	Q15	1次反射係数の一時記憶
REXP	1 ~ LPC+1	BFL	自己相関の巡回計算用（合成フィルタ）
REXPLG	1 ~ LPCLG+1	BFL	自己相関の巡回計算用（対数利得予測器）
REXPW	1 ~ LPCW+1	BFL	自己相関の巡回計算用（重み付けフィルタ）
RTMP	1 ~ LPC+1	BFL	自己相関係数の一時記憶
S	1 ~ IDIM	15b Q2	均一ＰＣＭ入力音声ベクトル
SB	1 ~ 105	14b BFL	過去の量子化音声のバッファ
SBLG	1 ~ 34	Q9	過去の対数利得のバッファ
SBW	1 ~ 60	Q2	過去の入力音声のバッファ
SCALE	1	SFL	低域通過フィルタを通していないポストフィルタのスケーリングファクタ

付表Ｇ－４－２／JT-G728（３／３）　ＬＤ－ＣＥＬＰの内部処理変数
(ITU-T G.728)

名　称	配列の インデックスの範囲	固定小数点 フォーマット	説　　明
SCALEFIL	1	Q14	低域通過フィルタを通ったポストフィルタの スケーリングファクタ，初期値=16384
SD	1 ~ IDIM	Q0	復号音声のバッファ
SPF	1 ~ IDIM	Q2	ポストフィルタを通った音声ベクトル
SPFPCFV	1 ~ 11	Q14	短期ポストフィルタの極補正ベクトル
SPFZCFV	1 ~ 11	Q14	短期ポストフィルタの零点補正ベクトル
SO	1	byte	入力音声サンプル
SST(過去)	-239 ~ 0	13bQ0	量子化音声バッファ
SST(現在)	1 ~ IDIM	15bQ2	量子化音声バッファ
ST	1 ~ IDIM	14bBFL	量子化音声ベクトル
STATELPC	1 ~ LPC	14bSBFL	合成フィルタのメモリ
STLPCI	1 ~ 10	Q2	ＬＰＣ逆フィルタのメモリ
STMP	1 ~ 4＊IDIM	15bQ2	聴覚重み付けフィルタのハイブリッド窓のバッファ
STTMP	1~ 4＊IDIM	14b SBFL	合成フィルタのハイブリット窓のバッファ
STPFFIR	1 ~ 10	Q2	短期ポストフィルタのメモリ（全零の部分）
STPFIIR	1 ~ 10	Q2	短期ポストフィルタのメモリ（全極の部分）
SU	1	Q2	均一ＰＣＭ入力音声サンプル
SUMFIL	1	Q2	ポストフィルタを通った音声の絶対値和
SUMUNFIL	1	Q2	復号音声の絶対値和
SW	1 ~ IDIM	Q2	聴覚重み付けした音声ベクトル
TARGET	1 ~ IDIM	BFL	正規化したＶＱターゲットベクトル
TEMP	1 ~ IDIM	a)	作業用領域
TILTZ	1	Q14	短期ポストフィルタの傾斜補正係数
WFIR	1 ~ LPCW	Q2	聴覚重み付けフィルタ４のメモリ（全零の部分）
WIIR	1 ~ LPCW	Q2	聴覚重み付けフィルタ４のメモリ（全極の部分）
WNR	1 ~ 105	Q15	合成フィルタの窓関数
WNRLG	1 ~ 34	Q15	対数利得予測器の窓関数
WNRW	1 ~ 60	Q15	聴覚重み付けフィルタの窓関数
WPCFV	1 ~ LPCW+1	Q14	聴覚重み付けフィルタの極補正ベクトル
WS	1 ~ 105	#	中間変数の作業領域
WZCFV	1 ~ LPCW+1	Q14	聴覚重み付けフィルタの零点補正ベクトル
Y	1 ~ IDIM＊NCWD	Q11	形状コードブックの配列
Y2	1 ~ NCWD	Q5	畳み込んだ形状コードブックのエネルギ
ZIR	1 ~ IDIM	15b Q2	零入力応答
ZIRWFIR	1 ~ LPCW	15b Q2	聴覚重み付けフィルタ１０のメモリ（全零の部分）
ZIRWIIR	1 ~ LPCW	15b Q2	聴覚重み付けフィルタ１０のメモリ（全極の部分）

（注）　SFLはスカラ浮動小数点、BFLはブロック浮動小数点、SBFLは分割ブロック浮動小数点、QxはQxフォーマットを示す。

14b, 15bはそれぞれ14, 15ビット精度を、その他は16ビット精度を示す。

一時記憶メモリのため、＃は使用法によってブロック浮動小数点にも固定Qフォーマットにもなり得る。

a) TEMPは、一時的な作業用配列であり、複数のブロックで使用される。従って、そのQフォーマットはブロックにより異なる。

５．利得と形状コードブックベクトルに対する対数利得表

付表Ｇ－５－１／JT-G728　利得コードブックベクトルに対する
(ITU-T G.728)　浮動小数点表現(dB)とQ11固定小数点表現

インデックス	浮動小数点(dB)	固定小数点
0　4	-5.7534180	-11783
1　5	-0.8925781	-1828
2　6	3.9682620	8127
3　7	8.8291020	18082

（注）固定小数点の値を求めるためには、浮動小数点の値に $2048 = 2^{11}$ を乗ずればよい。

付表Ｇ－５－２／JT-G728（１／３）　形状コードブックベクトルに対する利得の
(ITU-T G.728)　浮動小数点表現(dB)とQ11固定小数点表現

インデックス	浮動小数点(dB)	固定小数点	インデックス	浮動小数点(dB)	固定小数点
0	-0.1108398	-227	22	8.2045900	16803
1	5.0332030	10308	23	9.9272460	20331
2	3.1977540	6549	24	8.7983400	18019
3	3.7856450	7753	25	12.1679700	24920
4	3.7094730	7597	26	7.8901370	16159
5	8.0874020	16563	27	8.6025390	17618
6	3.1279300	6406	28	11.2656300	23072
7	5.8266600	11933	29	13.7085000	28075
8	6.6254880	13569	30	9.3598630	19169
9	5.1606450	10569	31	12.5600600	25723
10	7.9726560	16328	32	4.2333980	8670
11	3.1914060	6536	33	4.9165040	10069
12	7.7163090	15803	34	0.2456055	503
13	5.6997070	11673	35	4.2221680	8647
14	10.4091800	21318	36	5.4516600	11165
15	4.4433590	9100	37	9.0073240	18447
16	5.9790040	12245	38	2.0820310	4264
17	5.8681640	12018	39	8.4868160	17381
18	1.2221680	2503	40	1.7241210	3531
19	7.1728520	14690	41	5.1479490	10543
20	8.8818360	18190	42	-1.1679690	-2392
21	14.0629900	28801	43	1.1064450	2266

（注）固定小数点の値を求めるためには、浮動小数点の値に $2048 = 2^{11}$ を乗ずればよい。

付表Ｇ－５－２／JT-G728（２／３）　　形状コードブックベクトルに対する利得の
(ITU-T G.728)　　浮動小数点表現(dB)とQ11固定小数点表現

インデックス	浮動小数点(dB)	固定小数点	インデックス	浮動小数点(dB)	固定小数点
44	7.0932620	14527	66	1.0751950	2202
45	9.1738280	18788	67	-3.5297850	-7229
46	6.3623050	13030	68	1.5361330	3146
47	3.0458980	6238	69	-1.3759770	-2818
48	0.8911133	1825	70	-1.3056640	-2674
49	4.4384770	9090	71	-0.7651367	-1567
50	0.1030273	211	72	0.8989258	1841
51	0.9218750	1888	73	2.8334960	5803
52	8.8320310	18088	74	3.8203130	7824
53	11.0141600	22557	75	0.1557617	319
54	5.3188480	10893	76	0.8862305	1815
55	8.8652340	18156	77	0.8618164	1765
56	1.6728520	3426	78	3.3930660	6949
57	6.5429690	13400	79	1.2128910	2484
58	-2.1362300	-4375	80	1.3710940	2808
59	3.8916020	7970	81	4.7431640	9714
60	3.7861330	7754	82	-2.0581050	-4215
61	12.3388700	25270	83	3.2607420	6678
62	2.5942380	5313	84	1.2861330	2634
63	7.6245120	15615	85	1.7133790	3509
64	-3.0742190	-6296	86	0.4252930	871
65	2.2021480	4510	87	1.0693360	2190

（注）固定小数点の値を求めるためには、浮動小数点の値に $2048 = 2^{11}$ を乗ずればよい。

付表Ｇ－５－２／JT-G728（３／３）　　形状コードブックベクトルに対する利得の
(ITU-T G.728)　　浮動小数点表現(dB)とQ11固定小数点表現

インデックス	浮動小数点(dB)	固定小数点	インデックス	浮動小数点(dB)	固定小数点
88	2.7080080	5546	108	3.0083010	6161
89	7.4887700	15337	109	2.8579100	5853
90	1.8105470	3708	110	3.7104490	7599
91	1.1748050	2406	111	3.2944340	6747
92	2.8076170	5750	112	-0.9770508	-2001
93	3.6806640	7538	113	4.9892580	10218
94	1.9101560	3912	114	-0.0263672	-54
95	1.7299800	3543	115	0.9335938	1912
96	-4.9335940	-10104	116	5.6127930	11495
97	0.1479492	303	117	5.1635740	10575
98	-3.0083010	-6161	118	2.2055660	4517
99	-0.5576172	-1142	119	2.0893550	4279
100	1.8881840	3867	120	0.8852539	1813
101	2.8979490	5935	121	0.2763672	566
102	-3.5161130	-7201	122	2.2309570	4569
103	-0.3706055	-759	123	2.0278320	4153
104	-1.0219730	-2093	124	1.6445310	3368
105	-1.3979490	-2863	125	5.4584960	11179
106	1.0825200	2217	126	0.8271484	1694
107	-1.5834960	-3243	127	0.3715820	761

（注）固定小数点の値を求めるためには、浮動小数点の値に $2048 = 2^{11}$ を乗ずればよい。

６．利得コードブックに関連する配列の整数値

この章では、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｂで与えられる浮動小数点表現で記述された表と等価な、整数値で記述された表を示す。

付表Ｇ－６－１／JT-G728　利得コードブックに関連する配列の整数値
(ITU-T G.728)

配列名	1	2	3	4	5	6	7	8
GQ(Q13)	4224	7392	12936	22638	-4224	-7392	-12936	-22638
GB(Q13)	5808	10164	17787	a)	-5808	-10164	-17787	a)
G2(Q12)	4224	7392	12936	22638	-4224	-7392	-12936	-22638
GSQ(Q11)	545	1668	5107	15640	545	1668	5107	15640

a) 使用しないため、任意の値でよい。

７．符号器と復号器のメインプログラム擬似コード

この章では、符号器と復号器のメインプログラム擬似コードを示す。主目的は、種々の機能ブロックがどのような順序で実行されるかを示すことにある。したがって、プログラムの実行順序のみを示し、受け渡されるパラメータの詳細については触れない。ここで注意すべき点としては、許容される実行順序は一通りでは無いということである。ビットイグザクトな結果を生成する何通りもの実行順序が存在する。以下に示す擬似コードは2例のみであるが、もし異なるブロック実行順序を使用する場合は、ビットイグザクトな結果が得られることを確認する必要がある。

以下、符号器のメインプログラム擬似コードを示す。

符号器で使用される全ての変数を初期化する。

h=[8192, 0, 0, 0, 0]として、ブロック１４，１５を実行することによりY2()を初期化する。
ILLCOND=偽
ILLCONDW=偽
ILLCONDG=偽
ICOUNT=0

VEC_LOOP:

ICOUNT=4ならば、ICOUNT=0とする。 | ベクトルカウンタのリセット
ICOUNT=ICOUNT+1 | ベクトルカウンタの更新
入力バッファから1ベクトル分の入力音声を取り出す。
入力音声ベクトルを[-16384,+16383]の範囲に変換し、S( )とする。
| [-4096,+4095.75]のQ2表現
| フィルタ係数の更新を行うかどうかの
| チェック
ICOUNT=3ならば、以下の4行を実行する。
ILLCOND=偽ならば、ブロック５１を実行する。
ILLCONDW=偽ならば、ブロック３８を実行する。
ブロック１２を実行する。
ブロック１４と１５を実行する。
ICOUNT=2かつ ILLCONDG=偽ならば、ブロック４５を実行する。
| ベクトル単位の処理開始
ブロック４６，９８，９９，４８を実行する。 | バックワード適応利得を計算
| GSTATE(1:9)を一つずつずらす。
“blockzir”を実行する。(零入力応答計算の間、ブロック９と１０を実行する。)
ブロック４を実行する。 | 聴覚重み付けフィルタ
ブロック１１を実行する。 | ＶＱターゲットベクトルの計算
ブロック１６を実行する。 | ＶＱターゲットベクトルの正規化
ブロック１３を実行する。 | 時間反転畳込み
ブロック１７と１８を実行する。 | 励振コードブックの探索
ICHANを通信チャネルに送出する。
ブロック１９と２１を実行する。 | 選択された励振コードベクトルのスケ
| ーリング
フィルタメモリ更新の間、ブロック９と１０を実行する。 | ST( )の計算
ブロック９３，９４，９６，９７を実行する。 | 対数利得の更新
| ここで、利得適応器内の3遅延ユニットは
| ループ手順の中で自然に発生するため、
| 意図的に実行する必要がないことに
| 注意。
GSTATE(1)=ブロック９７の出力 | 利得予測器メモリの更新
I=(ICOUNT-1)＊IDIM | I=STTMP( )の開始アドレス
ST(1:5)をSTTMP(I+1:I+5)にコピーする。 | STTMP( )の更新
NLSSTTMP(ICOUNT)=NLSST
I=(ICOUNT-3)＊IDIM
ICOUNT<3ならば、I=I+20とする。 | I=STMP( )の開始アドレス
S(1:5)をSTMP(I+1:I+5)にコピーする。 | STMP( )の更新
| ベクトル単位の処理終了

| フレーム単位の処理開始
ICOUNT=4ならば、以下の2行を実行する。
ブロック４９を実行する。 | 異常状態フラグ=ILLCONDとして出力

```
        ブロック５０を実行する。                    | 予測係数=ATMP( )として出力
                                                    | 異常状態フラグ=ILLCONDとして出力
    ICOUNT=2ならば、以下の2行を実行する。
        ブロック３６を実行する。                    | 異常状態フラグ=ILLCONDWとして出力
        ブロック３７を実行する。                    | 予測係数=AWZTMP( )として出力
                                                    | 異常状態フラグ=ILLCONDWとして出力
    ICOUNT=1ならば、以下の6行を実行する。
        GTMP(1)=GSTATE(4)                           | GTMP( )の更新
        GTMP(2)=GSTATE(3)
        GTMP(3)=GSTATE(2)
        GTMP(4)=GSTATE(1)
        ブロック４３を実行する。                    | 異常状態フラグ=ILLCONDGとして出力
        ブロック４４を実行する。                    | 予測係数=GPTMP( )として出力
                                                    | 異常状態フラグ=ILLCONDGとして出力
                                                    | フレーム単位の処理終了
    VEC_LOOPへジャンプする。
```

以下、復号器のメインプログラム擬似コードを示す。ここでも、ブロックの実行順序のみを示し、受け渡されるパラメータの詳細については触れない。

```
    復号器で使用される全ての変数を初期化する。
    ILLCOND=偽
    ILLCONDG=偽
    ILLCONDP=偽
    ICOUNT=0
VEC_LOOP:
    ICOUNT=4ならば、ICOUNT=0とする。               | ベクトルカウンタのリセット
    ICOUNT=ICOUNT+1                                | ベクトルカウンタの更新
    入力バッファから現時刻のベクトルに対応したICHANを取り出す。
    ICHANから形状コードブックインデックスISと利得コードブックインデックスIG
    を求める。
                                                    | フィルタ係数の更新を行うかどうかの
                                                    | チェック
    ICOUNT=3ならば、以下の1行を実行する。
        ILLCOND=偽ならば、ブロック５１を実行する。
    ICOUNT=2かつ ILLCONDG=偽ならば、ブロック４５を実行する。
                                                    | ベクトル単位の処理開始
    ブロック４６，９８，９９，４８を実行する。     | バックワード適応利得を計算
                                                    | GSTATE(1:9)を1つずつずらす。
    ブロック１９と２１を実行する。                  | 選択された励振コードベクトルのスケ
                                                    | ーリング
    ブロック３２を実行する。
    ICOUNT=1ならば、ブロック８５を実行する。        | 短期ポストフィルタ係数の更新
    ブロック８１を実行する。
    ICOUNT=3ならば、以下の3行を実行する。
        ブロック８２を実行する。                    | ピッチ周期の抽出
        ブロック８３を実行する。                    | ピッチ予測器タップの計算
        ブロック８４を実行する。                    | 長期ポストフィルタ係数の更新
    ブロック７１を実行する。                        | 長期ポストフィルタリング
```

```
ブロック７２を実行する。                          | 短期ポストフィルタリング
ブロック７３と７４を実行する。                    | 絶対値合計計算
ブロック７５を実行する。                          | 絶対値合計値の比の計算
ブロック７６を実行する。                          | スケーリングファクタの低域通過フィ
                                                  | ルタリング
ブロック７７を実行する。                          | ポストフィルタ出力の利得制御
ブロック９３，９４，９６，９７を実行する。        | 対数利得の更新
                                                  | ここで、利得適応器内の3遅延ﾕﾆｯﾄは
                                                  | ﾙｰﾌﾟ手順の中で自然に発生するため、
                                                  | 意図的に実行する必要がないことに
                                                  | 注意。
GSTATE(1)=ブロック９７の出力                      | 利得予測器メモリの更新
I=(ICOUNT-1)＊IDIM                                | I=STTMP( )の開始アドレス
ST(1:5)をSTTMP(I+1:I+5)にコピーする。             | STTMP( )の更新
NLSSTTMP(ICOUNT)=NLSST
                                                  | ベクトル単位の処理終了

                                                  | フレーム単位の処理開始
ICOUNT=4ならば、以下の5行を実行する。
    ブロック４９を実行する。                      | 異常状態フラグ=ILLCONDとして出力
    1から10次についてブロック５０を実行する。    | 予測係数=ATMP( )
                                                  | としてNLSATMPと共に出力
                                                  | 異常状態フラグ=ILLCONDPとして出力
    NLSAPF=NLSATMP                                | 後にポストフィルタ計算で使用するた
    ATMP(2:11)をAPF(2:11)にコピーする。           | めに、10次予測器を格納する。
    11から50次についてブロック５０を引き続き実行する。  | ブロック５０
                                                  | を最後まで実行する。
                                                  | 予測係数=ATMP( )としてNLSATMPと
                                                  | 共に出力
                                                  | 異常状態フラグ=ILLCONDとして出力
ICOUNT=1ならば、以下の6行を実行する。
    GTMP(1)=GSTATE(4)                             | GTMP( )の更新
    GTMP(2)=GSTATE(3)
    GTMP(3)=GSTATE(2)
    GTMP(4)=GSTATE(1)
    ブロック４３を実行する。                      | 異常状態フラグ=ILLCONDGとして出力
    ブロック４４を実行する。                      | 予測係数=GPTMP( )として出力
                                                  | 異常状態フラグ=ILLCONDGとして出力
                                                  | フレーム単位の処理終了
VEC_LOOPへジャンプする。
```

―――――――――――――――

付属資料Ｈ
（標準ＪＴ－Ｇ７２８に対する）
主にＤＣＭＥ用16kbit/s以下のレートでの可変ビットレートＬＤ－ＣＥＬＰの動作

１．本付属資料の規定範囲

本付属資料では、符号化ビットレートを12.8kbit/sおよび9.6kbit/sに低減するためのＴＴＣ標準ＪＴ－Ｇ７２８ＬＤ－ＣＥＬＰ音声符号化アルゴリズムの変更について述べる。この変更には、形状コードブックと利得コードブックの変更も含まれている。

本資料では、読者は既にＴＴＣ標準ＪＴ－Ｇ７２８本体の仕様について精通していることを前提に、標準ＪＴ－Ｇ７２８のアルゴリズムについては触れず、標準ＪＴ－Ｇ７２８からの変更個所についてのみ記述する。

本付属資料は以下に示す４章からなる。第２章では、ビットレート低減のための動作原理について記述する。第３章では、12.8kbit/s動作のための変更について記述する。第４章では、9.6kbit/s動作のための変更について記述する。

２．動作原理

２．１　符号化ビットレート低減方法

符号化ビットレートの低減は、音声符号化アルゴリズムを全く変えずに、コードブックの大きさを低減することにより行う。ＴＴＣ標準ＪＴ－Ｇ７２８本体では、コードブックインデックスは1024個のコードブックベクトルに相当する10ビットで成り立っている。コードブックインデックスを10ビットから2ビット削減すれば、符号化ビットレートは16kbit/sから12.8kbit/sに低減し、コードブックインデックスを4ビット削減すれば、符号化ビットレートは9.6kbit/sに低減する。

10ビット（ビット9からビット0まで）のコードブックインデックスは、7ビット（ビット9からビット3まで）の形状コードブックと3ビット（ビット2からビット0まで）の利得コードブックの2つに分けられる。7ビットの形状コードブックは、128個のコードベクトルから成り、3ビットの利得コードブックは、0に関して対称な8個のスカラ値から成る。

まず、形状コードブックのベクトル数を削減する方法について述べる。通常の音声サンプルの場合、形状コードブックベクトルの生起確率は一様分布ではなく、明らかに、1番から64番のコードブックインデックス値より、65番から128番までのコードブックインデックス値の生起確率の方が高い。この非一様分布特性を利用して、音質をあまり劣化させずにコードブックベクトル数を制限することができる。例えば、形状コードブックインデックスのビット9は、削減候補の一つである。

形状コードブックを削減するもう一つの方法は、各低減ビットレート動作用にそれぞれ最適なコードブックを再設計することである。しかし、この方法ではメモリ量が増加し、また、実装上も大きな変更が必要となる。

次に、利得コードブックの利得値の数を削減する方法について述べる。利得コードブックのメモリ量は小さいので、利得コードブックのビット削減には、低減ビットレート動作用のサイズの小さい利得コードブックを再設計した方がよい。サイズの小さい利得コードブックには量子化前の利得値の確率分布に基づいて最適化したものを用いる。

12.8kbit/s動作用のコードブックインデックスのビット低減については、２．２節で記述し、9.6kbit/s動作用については、２．３節で記述する。

２．２　12.8kbit/sでの動作原理

12.8kbit/sでの動作を実現するためには、10ビットのコードブックインデックスから2ビットを削減する必要がある。そこで、形状コードブックのビット9を削り、さらに、8値の利得コードブックの値を4値に削減する。

形状コードブックインデックスのビット9を削り、コードブックインデックスを、65番から128番の範囲に制限する。利得コードブックの4値およびその関連する値は、新たに12.8kbit/s用に最適化され、３．２節のTableＨ－３－１／JT-G728（浮動小数点演算用）、および、TableＨ－３－２／JT-G728（固定小数点演算用）に

注）本付属資料は、ＴＴＣ標準ＪＴ－Ｇ７２８の追加オプションであり、コーデックに正確に実装する必要はない。本付属資料は、ディジタル回線多重化装置（ＤＣＭＥ）などの特定のアプリケーションにおいて標準ＪＴ－Ｇ７２８の性能を補強することを目的としている。本オプションを使用するかどうかは実装者の判断にまかされている。

示されるとおりである。

２．３　9.6kbit/sでの動作原理

9.6kbit/sでの動作を実現するためには、10ビットのコードブックインデックスから4ビットを削減する必要がある。そこで、形状コードブックのビット9、ビット8、ビット5を削り、さらに、8値の利得コードブックを4値に削減する。

形状コードブックインデックスのビット9、ビット8、ビット5を削り、コードブックインデックスを、97番から100番まで、105番から108番まで、113番から116番まで、そして121番から124番の範囲に制限する。利得コードブックの4値およびその関連する値は、新たに9.6kbit/s用に最適化され、４．２節のTableＨ－４－１／JT-G728（浮動小数点演算用）、および、TableＨ－４－２／JT-G728（固定小数点演算用）に示されるとおりである。

３．12.8kbit/s動作のための変更

３．１　擬似コード

本資料ではブロックの実行シーケンスのみを示し、パラメータの受け渡しといった低レベルの詳細は記述しない。

３．１．１　ブロック１７，１８ー誤差計算器、最適コードブックインデックス選択器

この節では、ブロック１７と１８の浮動小数点および固定小数点擬似コードを示す。これらのコードは12.8kbit/s動作用に変更されており、ＴＴＣ標準ＪＴ－Ｇ７２８、５．１１節に記述されている本来のブロック１７，１８と置き換えられる。以下に、まず、浮動小数点擬似コードを示す。

```
Initialize DISTM to the largest number representable in the hardware
N1=NG_128/2
For J=65,66,...,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)                    | compute inner product Pj

    If COR > 0, then do the next 3 lines
        IDXG=N1
        If COR < GB_128(1)＊Y2(J), do the next line
            IDXG=1                                | Best positive gain found

    If COR ≤ 0, then do the next 3 lines
        IDXG=NG_128
        If COR > GB_128(3)＊Y2(J), do the next line
        IDXG=3                                    | Best negative gain found

    D=-G2_128(IDXG)＊COR+GSQ_128(IDXG)＊Y2(J)      | Compute distortion D̂
    If D < DISTM, do the next 3 lines
        DISTM=D                                   | Save the lowest distortion
        IG=IDXG                                   | and the best codebook indices
        IS=J                                      | so far.

Repeat the above indented section for the next J

IS1=IS-NCWD/2
```

```
ICHAN=(IS1-1)＊NG_128+(IG-1)                    | Concatenate shape and gain
                                                | codebook indices.
```

Transmit ICHAN through communication channel. For bit-serial transmission, the most significant bit of ICHAN should be transmitted first. If ICHAN is represented by the 8-bit word b7, b6, b5, b4, b3, b2, b1, b0, then the order of the transmitted bits shall be b7, followed by b6, b5, b4, b3, b2, b1, b0 (b7 is the most significant bit).

次に、同じモジュールの固定小数点擬似コードを示す。この固定小数点擬似コードは、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ、３．９節に記述されている本来のブロック１７の固定小数点擬似コードと置き換えられる。

```
DISTM=2147483647
For J=65,66,...,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)                    | compute inner product Pj
        AA0=AA0+P                           | NLS for AA0 is 7+11=18
    If AA0 < 0,set AA0=-AA0                 | take absolute value
    IDXG=1
    P=GB_128(1)＊Y2(J)                      | NLS for P is 13+5=18
    If AA0 ≥ P, set IDXG=IDXG+1

    AA0=AA0 >> 14                           | NLS for AA0=4
    If AA0 > 32767,set AA0=32767            | clip AA0 ; AA0 in saturation mode
    AA1=GSQ_128(IDXG)＊Y2(J)                | NLSGSQ_128=11,NLSY2=5,so NLSAA1=16
    P=G2_128(IDXG)＊AA0                     | NLSG2_128=12,NLSAA0=4,so NLSP=16
    AA1=AA1-P
    If AA1 < DISTM, do the next 3 lines
        DISTM=AA1                           | double precision DISTM
        IG=IDXG
        IS=J

Repeat the above indented section for the next J

AA0=0                                       | Now find the sign bit
J1=(IS-1)＊IDIM
For K=1,2,...,IDIM, do the next 2 lines
    P=PN(K)＊Y(J1+K)                        | compute inner product
    AA0=AA0+P
If AA0 ≤ 0,set IG=IG+2

IS1=NCWD
IS1=IS1 >> 1
IS1=IS-IS1

ICHAN=(IS1-1)＊NG_128+(IG-1)
```

In the above code, we used the following four lines.

```
AA0=AA0 >> 14                          | NLS for AA0=4
If AA0 > 32767, set AA0=32767          | clip AA0
AA1=GSQ_128(IDXG) * Y2(J)              | NLSGSQ_128=11, NLSY2=5, so NLSAA1=16
P=G2_128(IDXG) * AA0                   | NLSG2_128=12, NLSAA0=4, so NLSP=16
```

In DSP chips which have a “clipping” function, these lines can be replaced with the following code to give exactly the same results.

```
AA0=AA0 << 2                           | NLS for AA0=20
AA0=CLIP(AA0)                          | AA0 is in saturation mode
AA0=AA0 >> 16                          | take high word; NLS for AA0=4
AA1=GSQ_128(IDXG) * Y2(J)              | NLSGSQ_128=11, NLSY2=5, so NLSAA1=16
P=G2_128(IDXG) * AA0                   | NLSG2_128=12, NLSAA0=4, so NLSP=16
```

The CLIP function and saturation mode refer to the concept of not allowing AA0 to overflow when the << 2 operation is performed. Instead of overflow, AA0 is set to the maximum positive or negative number, depending on its original sign. In this case, AA0 is always positive. This alternative is DSP dependent and may require more than a 32-bit accumulator. The alternative in the main pseudo-code can always be implemented.

３．１．２　ブロック１９－励振ＶＱコードブックとブロック２１－利得調整ユニット

この節では、ブロック１９と２１の浮動小数点および固定小数点擬似コードを示す。これらのコードは、12.8kbit/s動作用に変更されており、ＴＴＣ標準ＪＴ－Ｇ７２８、５．１２節に記述されている本来のブロック１９と置き換えられる。以下に、ブロック１９、励振ＶＱコードブックの浮動小数点擬似コードを示す。

```
NN=(IS-1) * IDIM
For K=1,2,...,IDIM, do the next line
    YN(K)=GQ_128(IG) * Y(NN+K)
```

以下に、ブロック２１、利得調整ユニットの浮動小数点擬似コードを示す。

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

次に、同じブロック１９と２１の固定小数点擬似コードを示す。この固定小数点擬似コードは、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ、３．１０節に記述されている本来のブロック１９と２１の固定小数点擬似コードと置き換えられる。

For the fixed-point pseudo-code, we combine Blocks 19 and 21 into a single module. Both Y and GQ_128 have fixed Q formats, Q11 and Q13, respectively. The value of GAIN has associated with it NLSGAIN. To get the maximum accuracy, the product GQ_128(IG) * GAIN is normalized to 32 bits before rounding to the upper 16 bits is performed. Let NNGQ_128(I) be (1+the number of left shifts needed to normalize the Q13 GQ_128(I)), NNGQ_128(I)=3 for I=1,3 and NNGQ_128(I)=2 for I=2,4. The pseudo-code can thus be written as follows.

```
AA0=GQ_128(IG) * GAIN                  | AA0 has NNGQ_128(IG) leading zeros
AA0=AA0 << NNGQ_128(IG)                | left shift NNGQ_128(IG) bits to
```

```
                                                    | normalize AA0
TMP=RND(AA0)                                        | round to upper 16 bits and assign to TMP

NLSAA0=13+NLSGAIN                                   | Q format of the product GQ_128(IG) * GAIN
NLSTMP=NLSAA0+NNGQ_128(IG)-16                       | Q format of TMP, because AA0 left shift
                                                    | by NNGQ_128(IG) bits then round and take
                                                    | upper 16 bits
NN=(IS-1) * IDIM                                    | normalize selected shape
Call VSCALE(Y(NN+1),IDIM,IDIM,14,TEMP,NLS)          | codevector to 16 bits;
                                                    | put in TEMP

For K=1,2,...,IDIM, do the next 2 lines             | TMP and TEMP both normalized
                                                    | to 16 bits, so the product
    AA0=TMP * TEMP(K)                               | has 1 leading zero. Directly
    ET(K) =RND(AA0)                                 | rounding to high work gives
                                                    | us a 15-bit ET array.
NLSET=NLSTMP+11+NLS-16                              | calculate the NLS for ET.
```

３．１．３　ブロック２９－復号器励振ＶＱコードブックとブロック３１－復号器利得調整ユニット

この節では、ブロック２９と３１の浮動小数点および固定小数点擬似コードを示す。これらのコードは、12.8kbit/s動作用に変更されており、ＴＴＣ標準ＪＴ－Ｇ７２８、５．１４節に記述されている本来のブロック２９と置き換えられる。以下に、ブロック２９、復号器励振ＶＱコードブックの浮動小数点擬似コードを示す。

This block first extracts the 2-bit gain codebook index IG and the 6-bit shape codebook index IS from the received 8-bit channel index. The remainder of the operation is exactly the same as for Block 19 of the encoder.

```
ITMP=integer part of (ICHAN/NG_128)
IG=ICHAN-ITMP * NG_128+1
ITMP=ITMP+NCWD/2

NN=ITMP * IDIM
For K=1,2,...,IDIM, do the next line
    YN(K)=GQ_128(IG) * Y(NN+K)
```

ブロック３１、復号器利得調整ユニットの動作は、符号器のブロック２１と全く同じである。

次に、同じブロック２９と３１の固定小数点擬似コードを示す。

For the fixed-point pseudo-code, we combine Blocks 29 and 31 into a single module. Both Y and GQ_128 have fixed Q formats, Q11 and Q13, respectively. The value of GAIN has associated with it NLSGAIN. To get the maximum accuracy, the product GQ_128(IG) * GAIN is normalized to 32 bits before rounding to the upper 16 bits is performed. Let NNGQ_128(I) be (1+the number of left shifts needed to normalize the Q13 GQ_128(I)), so NNGQ_128(I) =3 for I =1,3 and NNGQ_128(I) =2 for I=2,4. The pseudo-code can thus be written as follows.

```
IS=ICHAN >> 2
IG=ICHAN-IS * NG_128+1
IS1=NCWD
IS1=IS1 >> 1
```

```
IS=IS+IS1+1

AA0=GQ_128(IG) * GAIN                          | AA0 has NNGQ_128(IG) leading zeros
AA0=AA0 << NNGQ_128(IG)                        | left shift NNGQ_128(IG) bits to normalize AA0
TMP=RND(AA0)                                   | round to upper 16 bits and assign to TMP

NLSAA0=13+NLSGAIN                              | Q format of the product GQ_128(IG) * GAIN
NLSTMP=NLSAA0+NNGQ_128(IG)-16                  | Q format of TMP, because AA0 left
                                               | shift by NNGQ_128(IG) bits then round
                                               | and take upper 16 bits
NN=(IS-1) * IDIM                                       | normalize selected
Call VSCALE(Y(NN+1), IDIM, IDIM, 14 ,TEMP, NLS)        | shape codevector to
                                                       | 16bits; put in TEMP

For K=1,2,...,IDIM, do the next 2 lines                | TMP and TEMP both normalized
    AA0=TMP * TEMP(K)                                  | to 16 bits, so the product
    ET(K)=RND(AA0)                                     | has 1 leading zero. Directly
                                                       | rounding to high work gives
                                                       | us a 15-bit ET array.
NLSET=NLSTMP+11+NLS-16                                 | calculate the NLS for ET.
```

３．１．４　ブロック９６，９７ー対数利得補正項加算器および-32dBリミッタ

この節では、ブロック９６と９７の浮動小数点および固定小数点擬似コードを示す。これらのコードは、12.8 kbit/s動作用に変更されており、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ、３．１６節に記述されている本来のブロック９６，９７と置き換えられる。

以下に、浮動小数点擬似コードを示す。

```
GSTATE(1) = LOGGAIN + GCBLG_128(IG) + SHAPELG(IS)
If GSTATE(1) < -32.0, set GSTATE(1) = -32.0
```

以下に、固定小数点擬似コードを示す。

```
AA0=LOGGAIN << 7                   | Align decimal points at the
AA0=AA0 + (GCBLG_128(IG) << 5)     | boundary between the high and
AA0=AA0 + (SHAPELG(IS) << 5)       | low words of the accumulator.

AA0=AA0 >> 7                       | Right shift back to Q9 format

If AA0 < -16384, set AA0=-16384    | Check lower limit

GSTATE(1)=AA0                      | Lower 16 bit word saved
```

３．２　追加用新利得表

この節では、12.8kbit/s動作用にＴＴＣ標準ＪＴ－Ｇ７２８に追加された、新しい利得コードブックの値を示す。まず、TableＨ－３－１／JT-G728に利得コードブックの浮動小数点表現の値を示す。

TableＨ－３－１／JT-G728　Floating-point values of gain-codebook-related arrays
(ITU-T G.728)

Array Index	1	2	3	4
GQ_128	0.525824	1.562449	-0.525824	-1.562449
GB_128	0.869912	a)	-0.869912	a)
G2_128	1.051648	3.124898	-1.051648	-3.124898
GSQ_128	0.276491	2.441247	0.276491	2.441247
GCBLG_128	-5.5831919	3.8761170	-5.5831919	3.8761170

a) 使用しないため任意の値でよい。

次に、TableＨ－３－２／JT-G728に利得コードブックの固定小数点表現の値を示す。

TableＨ－３－２／JT-G728　Fixed-point values of gain-codebook-related arrays
(ITU-T G.728)

Array Index	1	2	3	4
GQ_128(Q13)	4308	12800	-4308	-12800
GB_128(Q13)	7126	a)	-7126	a)
G2_128(Q12)	4308	12800	-4308	-12800
GSQ_128(Q11)	566	5000	566	5000
NNGQ_128(Q0)	3	2	3	2
GCBLG_128(Q11)	-11434	7938	-11434	7938

a) 使用しないため任意の値でよい。

３．３　コーデックのパラメータ変更

この節では、NG_128という新しいパラメータについて述べる。このパラメータは、ＴＴＣ標準ＪＴ－Ｇ７２８本体のNG（値は8）を12.8kbit/s動作用に変更したものである。TableＨ－３－３／JT-G728にその値を示す。

TableＨ－３－３／JT-G728　Basic coder parameters of LD-CELP
(ITU-T G.728)

Name	Value	Description
NG_128	4	Gain codebook size (number of gain levels)

４．9.6kbit/s動作のための変更

４．１　擬似コード

本資料ではブロックの実行シーケンスのみを示し、パラメータの受け渡しといった低レベルの詳細は記述しない。

４．１．１　ブロック１７，１８－誤差計算器、最適コードブックインデックス選択器

この節では、ブロック１７と１８の浮動小数点および固定小数点擬似コードを示す。これらのコードは9.6kbit/s動作用に変更されており、ＴＴＣ標準ＪＴ－Ｇ７２８、５．１１節に記述されている本来のブロック１７，１８と置き換えられる。以下に、まず、浮動小数点擬似コードを示す。

```
Initialize DISTM to the largest number representable in the hardware
N1=NG_96/2
For K=1,2,3,4, do the following
    For K1=97,98,99,100, do the following
        J=(K-1)*8+K1
        J1=(J-1)*IDIM
        COR=0.
        For K2=1,2,...,IDIM, do the next line
        COR=COR+PN(K2)*Y(J1+K2)                    | compute inner product Pj

        If COR > 0, then do the next 3 lines
            IDXG=N1
            If COR < GB_96(1)*Y2(J) , do the next line
                IDXG=1                             | Best positive gain found

        If COR ≤ 0, then do the next 3 lines
            IDXG=NG_96
            If COR > GB_96(3)*Y2(J) , do the next line
                IDXG=3                             | Best negative gain found

        D=-G2_96(IDXG)*COR+GSQ_96(IDXG)*Y2(J)      | Compute distortion D̂
        If D < DISTM, do the next 3 lines
            DISTM=D                                | Save the lowest distortion
            IG=IDXG                                | and the best codebook
            IS=J                                   | indices so far

    Repeat the above indented section for the next K1
Repeat the above indented section for the next K
IS1=IS-(NCWD/2+NCWD/4)
IS2= integer part of (IS1/8)
IS2=IS2*4
IS3=IS1-IS2*2
IS1=IS2+IS3

ICHAN=(IS1-1)*NG_96+(IG-1)                         | Concatenate shape and
                                                   | gain codebook indices.
```

Transmit ICHAN through communication channel. For bit-serial transmission, the most significant bit of ICHAN should be transmitted first. If ICHAN is represented by the 6-bit word b5, b4, b3, b2, b1, b0, then the order of the transmitted bits shall be b5,and then b4, b3, b2, b1, b0 (b5 is the most significant bit).

次に、同じモジュールの固定小数点擬似コードを示す。この固定小数点擬似コードは、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ、３．９節に記述されている本来のブロック１７の固定小数点擬似コードと置き換えられる。

```
DISTM=2147483647
For K=1,2,3,4, do the following
    For K1=97,98,99,100, do the following
```

```
J=(K-1) * 8+K1
J1=(J-1) * IDIM
AA0=0
For K2=1,2,...,IDIM, do the next 2 lines
P=PN(K2) * Y(J1+K2)              | compute inner product Pj
    AA0=AA0+P                    | NLS for AA0 is 7+11=18
If AA0 < 0, set AA0=-AA0         | take absolute value
IDXG=1
P=GB_96(1) * Y2(J)               | NLS for P is 13+5=18
If AA0 ≥ P, set IDXG=IDXG+1

AA0=AA0 >> 14                    | NLS for AA0=4
If AA0 > 32767, set AA0=32767    | clip AA0 ; AA0 in saturation
                                 | mode
AA1=GSQ_96(IDXG) * Y2(J)         | NLSGSQ_96=11, NLSY2=5, so NLSAA1=16
P=G2_96(IDXG) * AA0              | NLSG2_96=12, NLSAA0=4, so NLSP=16
AA1=AA1-P
If AA1 < DISTM, do the next 3 lines
    DISTM=AA1                    | double precision DISTM
    IG=IDXG
    IS=J

  Repeat the above indented section for the next K1
Repeat the above indented section for the next K

AA0=0                            | Now find the sign bit
J1=(IS-1) * IDIM
For K=1,2,...,IDIM, do the next 2 lines
  P=PN(K) * Y(J1+K)              | compute inner product
  AA0=AA0+P
If AA0 ≤ 0, set IG=IG+2

IS2=NCWD
IS1=IS2 >> 1
IS2=IS2 >> 2
IS1=IS-(IS1+IS2)
IS2=IS1 >> 3
IS2=IS2 << 2
IS3=IS2 << 1
IS3=IS1-IS3
IS1=IS2+IS3

ICHAN=(IS1-1) * NG_96+(IG-1)
```

In the above code, we used the following four lines.

```
  AA0=AA0 >> 14                  | NLS for AA0=4
  If AA0 > 32767, set AA0=32767  | clip AA0
  AA1=GSQ_96(IDXG) * Y2(J)       | NLSGSQ_96=11, NLSY2=5, so NLSAA1=16
  P=G2_96(IDXG) * AA0            | NLSG2_96=12, NLSAA0=4, so NLSP=16
```

In DSP chips which have a “clipping” function, these lines can be replaced with the following code to give exactly the same results.

```
AA0=AA0 << 2                        | NLS for AA0=20
AA0=CLIP(AA0)                       | AA0 is in saturation mode
AA0=AA0 >> 16                       | take high word ; NLS for AA0=4
AA1=GSQ_96(IDXG) * Y2(J)            | NLSGSQ_96=11, NLSY2=5, so NLSAA1=16
P=G2_96(IDXG) * AA0                 | NLSG2_96=12, NLSAA0=4, so NLSP=16
```

The CLIP function and saturation mode refer to the concept of not allowing AA0 to overflow when the << 2 operation is performed. Instead of overflow, AA0 is set to the maximum positive or negative number, depending on its original sign. In this case, AA0 is always positive. This alternative is DSP dependent and may require more than a 32 bit accumulator. The alternative in the main pseudo-code can always be implemented.

４．１．２　ブロック１９－励振ＶＱコードブックとブロック２１－利得調整ユニット

この節では、ブロック１９と２１の浮動小数点および固定小数点擬似コードを示す。これらのコードは9.6kbit/s動作用に変更されており、ＴＴＣ標準ＪＴ－Ｇ７２８、５．１２節に記述されている本来のブロック１９と置き換えられる。以下に、まず、ブロック１９、励振ＶＱコードブックの浮動小数点擬似コードを示す。

```
NN=(IS-1) * IDIM
For K=1,2,...,IDIM, do the next line
    YN(K)=GQ_96(IG) * Y(NN+K)
```

以下に、ブロック２１、利得調整ユニットの浮動小数点擬似コードを示す。

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

次に、同じブロック１９と２１の固定小数点擬似コードを示す。この固定小数点擬似コードは、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ、３．１０節に記述されている本来のブロック１９と２１の固定小数点擬似コードと置き換えられる。

For the fixed point pseudo-code, we combine both Blocks 19 and 21 into a single module. Both Y and GQ_96 have fixed Q formats, Q11 and Q13, respectively. The value of GAIN has associated with it NLSGAIN. To get the maximum accuracy, the product GQ_96(IG) * GAIN is normalized to 32 bits before rounding to the upper 16 bits is performed. Let NNGQ_96(I) be (1+the number of left shifts needed to normalize the Q13 GQ_96(I)), NNGQ_96(I)=3 for I =1,3 and NNGQ_96(I) =2 for I=2,4. The pseudo-code can thus be written as follows.

```
AA0=GQ_96(IG) * GAIN                | AA0 has NNGQ_96(IG) leading zeros
AA0=AA0 << NNGQ_96(IG)              | left shift NNGQ_96(IG) bits to normalize AA0
TMP=RND(AA0)                        | round to upper 16 bits and assign to TMP

NLSAA0=13+NLSGAIN                   | Q format of the product GQ_96(IG) * GAIN
NLSTMP=NLSAA0+NNGQ_96(IG)-16        | Q format of TMP, because AA0 left shift
                                    | by NNGQ_96(IG) bits then round and take
                                                | 16 upper bits
NN=(IS-1) * IDIM                                | normalize selected shape
Call VSCALE(Y(NN+1), IDIM, IDIM, 14 ,TEMP, NLS) | codevector to 16 bits; put
```

```
                                                    | in TEMP
For K=1,2,...,IDIM, do the next 2 lines             | TMP and TEMP both normalized
                                                    | to 16 bits, so the product
    AA0=TMP * TEMP(K)                               | has 1 leading zero.
    ET(K) =RND(AA0)                                 | Directly rounding to high
                                                    | work gives us a 15-bit ET
                                                    | array.
NLSET=NLSTMP+11+NLS-16                              | calculate the NLS for ET.
```

４．１．３　ブロック２９－復号器励振ＶＱコードブックとブロック３１－復号器利得調整ユニット

この節では、ブロック２９と３１の浮動小数点および固定小数点擬似コードを示す。これらのコードは、9.6kbit/s動作用に変更されており、ＴＴＣ標準ＪＴ－Ｇ７２８、５．１４節に記述されている本来のブロック２９と置き換えられる。以下に、ブロック２９、復号器励振ＶＱコードブックの浮動小数点擬似コードを示す。

This block first extracts the 2-bit gain codebook index IG and the 4-bit shape codebook index IS from the received 6-bit channel index. The remainder of the operation is exactly the same as for Blocks 19 and 21 of encoder.

```
ITMP=integer part of (ICHAN/NG_96)
IG=ICHAN-ITMP * NG_96+1
ITMP1=integer part of (ITMP/4)
ITMP2=ITMP-ITMP1 * 4
ITMP1=ITMP1 * 8
ITMP=ITMP1+ITMP2
ITMP=ITMP+(NCWD/2+NCWD/4)

NN=ITMP * IDIM
For K=1,2,...,IDIM, do the next line
    YN(K)=GQ_96(IG) * Y(NN+K)
```

ブロック３１、復号器利得調整ユニットの動作は、符号器のブロック２１と全く同じである。

次に、同じブロック２９と３１の固定小数点擬似コードを示す。

For the fixed point pseudo-code, we combine Blocks 29 and 31 into a single module. Both Y and GQ_96 have fixed Q formats, Q11 and Q13, respectively. The value of GAIN has associated with it NLSGAIN. To get the maximum accuracy, the product GQ_96(IG) * GAIN is normalized to 32 bits before rounding to the upper 16 bits is performed. Let NNGQ_96(I) be (1+the number of left shifts needed to normalize the Q13 GQ_96(I)), NNGQ_96(I)=3 for I =1, 3 and NNGQ_96(I)=2 for I=2, 4. The pseudo-code can thus be written as follows.

```
IS=ICHAN >> 2
IG=ICHAN-IS * NG_96+1
IS1=IS >> 2
IS2=IS-IS1 * 4
IS1=IS1 << 3
IS=IS1+IS2
IS2=NCWD
IS1=IS2 >> 1
IS2=IS2 >> 2
```

```
IS=IS+IS1+IS2+1

AA0=GQ_96(IG) * GAIN                    | AA0 has NNGQ_96(IG) leading zeros
AA0=AA0 << NNGQ_96(IG)                  | left shift NNGQ_96(IG) bits to normalize AA0
TMP=RND(AA0)                            | round to upper 16 bits and assign to TMP
NLSAA0=13+NLSGAIN                       | Q format of the product GQ_96(IG) * GAIN
NLSTMP=NLSAA0+NNGQ_96(IG)-16            | Q format of TMP, because AA0 left shift
                                        | by NNGQ_96(IG) bits then round and take
                                        | upper 16 bits
NN=(IS-1) * IDIM                        | normalize selected shape

Call VSCALE(Y(NN+1), IDIM, IDIM, 14 ,TEMP, NLS)   | codevector to 16
                                                  | bits; put in TEMP
For K=1,2,...,IDIM, do the next 2 lines           | TMP and TEMP both normalized
                                                  | to 16 bits, so the product
    AA0=TMP * TEMP(K)                             | has 1 leading zero.
    ET(K)=RND(AA0)                                | Directly rounding to high
                                                  | work gives us a 15-bit ET
                                                  | array.
NLSET=NLSTMP+11+NLS-16                            | calculate the NLS for ET.
```

４．１．４　ブロック９６，９７ー対数利得補正項加算器および-32dBリミッタ

この節では、ブロック９６と９７の浮動小数点および固定小数点擬似コードを示す。これらのコードは、9.6 kbit/s動作用に変更されており、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ、３．１６節に記述されている本来のブロック９６，９７と置き換えられる。

以下に、浮動小数点擬似コードを示す。

```
GSTATE(1) = LOGGAIN + GCBLG_96(IG) + SHAPELG(IS)
If GSTATE(1) < -32.0, set GSTATE(1) = -32.0
```

以下に、固定小数点擬似コードを示す。

```
AA0=LOGGAIN << 7                    | Align decimal points at the
AA0=AA0 + (GCBLG_96(IG) << 5)       | boundary between the high and
AA0=AA0 + (SHAPELG(IS) << 5)        | low words of the accumulator.
AA0=AA0 >> 7                        | Right shift back to Q9 format
If AA0 < -16384, set AA0=-16384     | Check lower limit
GSTATE(1)=AA0                       | Lower 16 bit word saved
```

４．２　追加用新利得表

この節では、9.6kbit/s動作用にＴＴＣ標準ＪＴ－Ｇ７２８に追加された、新しい利得コードブックの値を示す。まず、TableＨ－４－１／JT-G728に利得コードブックの浮動小数点表現の値を示す。

TableＨ－４－１／JT-G728　Floating-point values of gain-codebook-related arrays
(ITU-T G.728)

Array Index	1	2	3	4
GQ_96	0.564657	1.937714	-0.564657	-1.937714
GB_96	1.007492	a)	-1.007492	a)
G2_96	1.129314	3.875428	-1.129314	-3.875428
GSQ_96	0.318838	3.754736	0.318838	3.754736
GCBLG_96	-4.9643057	5.7457935	-4.9643057	5.7457935

a) 使用しないため任意の値でよい。

次に、TableＨ－４－２／JT-G728に利得コードブックの固定小数点表現の値を示す。

TableＨ－４－２／JT-G728　Fixed-point values of the gain-codebook-related arrays
(ITU-T G.728)

Array Index	1	2	3	4
GQ_96(Q13)	4626	15874	-4626	-15874
GB_96(Q13)	8253	a)	-8253	a)
G2_96(Q12)	4626	15874	-4626	-15874
GSQ_96(Q11)	653	7690	653	7690
NNGQ_96(Q0)	3	2	3	2
GCBLG_96(Q11)	-10167	11767	-10167	11767

a) 使用しないため任意の値でよい。

４．３　コーデックのパラメータ変更

この節では、NG_96という新しいパラメータについて述べる。このパラメータは、ＴＴＣ標準ＪＴ－Ｇ７２８本体のNG（値は8）を9.6kbit/s動作用に変更したものである。TableＨ－４－３／JT-G728にその値を示す。

TableＨ－４－３／JT-G728　Basic coder parameters of LD-CELP
(ITU-T G.728)

Name	Value	Description
NG_96	4	Gain codebook size (number of gain values)

付属資料Ｉ
（標準ＪＴ－Ｇ７２８に対する）
ＬＤ－ＣＥＬＰ復号器用フレームおよびパケット消失補償

概要

ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｉは、移動体通信およびパケット環境でのフレーム消失およびパケット損失に直面した際の、9.6kbit/s，12.8kbit/sおよび16kbit/sのＴＴＣ標準ＪＴ－Ｇ７２８アルゴリズムの耐性を強める処理に関する拡張について定義したものである。

本付属資料の規定範囲

ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｉは、通信チャネルのフレームあるいはパケット損失によるビット列情報消失に対する補償方法に関するものである。通常動作中においては、復号器は、16kbit/sではＴＴＣ標準ＪＴ－Ｇ７２８（本体）もしくはＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇと全く同一の、12.8kbit/sまたは9.6kbit/sではＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｈと全く同一の方法で実行される。付属資料Ｉで提示する改良点は、受信ビット列が無効な場合に、復号器のみに変更を加えるものである。受信ビット列が無効かどうかは、何らかの外部手段によって復号器に与えられるものとする。このように、付属資料Ｉは、ＴＴＣ標準ＪＴ－Ｇ７２８の通常動作においては必須ではない。

参照とする標準

ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｉで必須の参照標準は以下のものである。

・ＴＴＣ標準ＪＴ－Ｇ７２８：　低遅延符号励振線形予測(ＬＤ－ＣＥＬＰ)を用いた16kbit/s音声符号化方式。
・ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ：　16kbit/s固定小数点の詳細記述。
・ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｈ：　主にＤＣＭＥ用16kbit/s以下のレートでの可変ビットレートＬＤ－ＣＥＬＰの動作。

１．はじめに

ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｉでは、受信したビット列におけるフレーム消失に対応できるようにするためのＴＴＣ標準ＪＴ－Ｇ７２８復号器の変更について記述する。変更箇所は、励振信号の外挿、ＬＰＣフィルタの帯域幅拡張、外挿励振信号を使用したバックワード適応処理の一部継続、外挿励振信号を使用したポストフィルタリングとポストフィルタ適応処理の継続、フレーム消失後におけるバックワード適応利得の増加率の制限、である。復号器に関するこれらの変更によってもＴＴＣ標準ＪＴ－Ｇ７２８のビット列の互換性は維持され、処理量の増加はほとんどない。

ＴＴＣ標準ＪＴ－Ｇ７２８のビット列の互換性を維持するために、全ての変更は復号器内に限られる。変更による影響が及ぶ区間は消失フレーム、及びフレーム消失から回復後の正常フレーム数フレームのみである。また、その他全ての正常フレームではＴＴＣ標準ＪＴ－Ｇ７２８と全く同一に動作する。

本資料では、読者が既にＴＴＣ標準ＪＴ－Ｇ７２８の内容に精通していることを前提に、標準ＪＴ－Ｇ７２８のアルゴリズムには触れず、ＴＴＣ標準ＪＴ－Ｇ７２８からの変更箇所についてのみ記述する。混乱を避けるため、以後「フレーム」という単語をフレーム消失（通常2.5msの倍数）に対応したブロックサイズに対してのみ使用する。また、「適応周期」はＴＴＣ標準ＪＴ－Ｇ７２８においてフィルタパラメータを更新する2.5msのブロックを表すこととする。

ＩＴＵ－Ｒ（旧ＣＣＩＲ）からの情報によれば、フレーム消失チャネルにおいては、受信フレームビットは全体として良い（ビット誤りが無い）か、全体として悪い（フレーム内ビット誤り率が50%近い）かのどちらかであると見なせる。また、受信側がある信頼度を持ってどのフレームが良くてどのフレームが悪いかを定めることができるような誤り検出符号が使用されていて、そのような情報をＴＴＣ標準ＪＴ－Ｇ７２８の復号器に対しても有効であると仮定する。

本付属資料は以下の章構成から成る。次章の「動作原理」では、節毎にフレーム消失補償区間における復号器処理の各変更点について述べる。第３章の「フレーム消失補償擬似コード」では、復号器の変更に関する全ての擬似コードを示す。完全を期すため、擬似コードとしては浮動小数点と固定小数点の両方を示す。最後の章では、

フレーム消失補償用に今回付加されたパラメータと変数をまとめた表を示す。

２．動作原理

消失フレームに対するＴＴＣ標準ＪＴ－Ｇ７２８の変更は、励振信号ならびにＬＰＣフィルタ係数の外挿、ＬＰＣと利得予測器の一部のバックワード適応動作の継続、フレーム消失後の利得増加率の制限を含んでいる。以下では、これら個々の変更点について述べる。

２．１　励振信号の外挿

まず、Figure４－１／JT-G728について述べる。本動作では、励振ＶＱコードブック（ブロック２９）の出力は外挿しない。その代わりに、利得調整ユニット（ブロック３１）の出力（利得調整された励振信号）を外挿する。この励振信号は、ＴＴＣ標準ＪＴ－Ｇ７２８の５．１４節の擬似コードにおける配列ET( )によって示されている。ＴＴＣ標準ＪＴ－Ｇ７２８本体の復号器では、ET( )の5サンプル分のみがメモリに記憶されている。ここではET( )の外挿を可能にするために、現在ならびに過去のET( )ベクトルを保持する配列ETPAST( )の追加が必要である。

ＴＴＣ標準ＪＴ－Ｇ７２８の５．１４節の擬似コードについて述べる。ＴＴＣ標準ＪＴ－Ｇ７２８の復号器では、ピッチ周期抽出モジュール（ブロック８２）はピッチ周期KPを出力する。ピッチ予測器タップ計算器（ブロック８３）は、復号音声に対するシングルタップピッチ予測器の最適タップ係数を計算する。このタップ係数PTAPは、現在の適応周期が有声音か否かのおおまかな指標として用いられる。もしPTAPがPPFTHより大きければ（ここではPPFTH=0.6）、現在の適応周期は有声音であると考えられ、長期ポストフィルタが動作する。そうでなければ、長期ポストフィルタは動作しない。ＴＴＣ標準ＪＴ－Ｇ７２８本体の復号器では、ピッチ周期と有声無声指標がすでに得られているので、励振信号の外挿のために新たにこれらの値を計算することなく直接用いることができる。

フレーム消失の開始フレームでは、直前の正常フレームにおける最後の適応周期で求められた有声無声指標PTAPの値を用いて、有声音フレームか無声音フレームかを決定する。フレーム消失中は有声フレームと無声フレームとで異なった励振信号の外挿を行う。外挿目的のために、PTAPに対する有声音閾値VTHを用いる。ここでは、VTH=PPFTH/1.4とし、PPFTHよりも低くしておく。

もし、PTAP>VTHならば、直前のフレームを有声音として扱う。この場合、配列ETPAST( )の最後のKPサンプルをスケールダウンしたものを、周期的に繰り返すことによって、ET( )の外挿を行う。あるいは、この動作は以下のように説明することができる。ETPAST( )の現在のベクトルと一致する場所に位置する5サンプルのサンプル値1の方形窓を考える。ここで、この方形窓を時間軸逆方向にKPサンプル移動させ、その場所に位置するETPAST( )の5サンプルを抜き取り、スケーリングファクタをかけることによりスケールダウンする。この結果を外挿されたET( )ベクトルとする。ETPAST( )の現在のベクトルを更新するために、この外挿されたET( )ベクトルをETPAST( )の対応する場所にコピーする。この処理は消失フレーム内のすべてのベクトルに対して行われる。消失フレーム区間でのこの動作は、周期的な形状で大きさが減衰するET( )のサンプル列を発生する。この場合の周期はKPサンプルである。このような周期的な配列ETPAST( )の外挿は、正常フレームを受信するまで続けられる。このスケーリングファクタをFESCALEとする。FESCALEは、消失区間の最初の20msの間は0.8であり、以後は10msごとに0.2ずつ減衰する。これらのスケーリングファクタは、VOICEDFEGAIN( )に保持される。消失区間が50msを越える場合は、復号音声に長く不自然で周期的な人工的雑音が入ることを避けるために、FESCALEを0にする。

もしPTAP≤VTHならば、直前のフレームを無声（厳密にいえば、無音や遷移過程等を含んだ有声音以外のすべての場合である）として扱う。この場合、周期的な外挿を行わない。なぜならば、原音声にはない人工的な周期性を入れたくないからである。ETPAST( )の直前の140サンプルから連続した5サンプルの抜き取りをランダムに繰り返すことにより、周期性を回避している。異常フレーム内の各ベクトルに対し、乱数発生器を用いて5から140までのランダムな整数を発生し、その値だけサンプル方形窓をETPAST( )の時間軸逆方向に移動させて、対応する5サンプルを抜き取り、それを外挿したET( )とする。もちろん、このような乱数系列を前もって計算し保持しておくこともできる。

無声フレーム消失間に生成された信号のレベルがその前の信号のレベルにほぼ同等になることを保証するために、外挿された各励振ベクトルの大きさの平均値を消失直前の5ms（すなわち40サンプル）の励振信号の大きさの平均値と同じになるようにする。これは以下のように行う。最初の消失フレームの開始時点で、もしその音声

が無声であるならば、ETPAST( )の最後の8ベクトルの大きさの平均値を計算する。この平均値をAVMAGとする。そして各励振ベクトルを外挿する際に、まずそのベクトルの大きさ（MAG）を計算する。（このベクトルは、ETPAST( )にある直前140サンプルからランダムに選択された連続5サンプルである。）次に、AVMAGをMAGで割りスケーリングファクタを求める。そして、このスケーリングファクタを現在の外挿された励振ベクトルの5サンプル各々にかける。この動作は、外挿され、利得調整された各励振ベクトルの大きさの平均値がAVMAGになることを保証する。非常に長い消失による人工的雑音を避けるために、消失区間が20msを越えた場合、10msごとにAVMAGを20%ずつ減衰させる。この減衰ファクタは配列UNVOICEDFEGAIN( )に保持されている。

２．２　ＬＰＣフィルタ

前述のように、消失フレーム区間では利得調整された励振信号ET( )を外挿によって求めるが、出力音声を得るためには、さらに、ＬＰＣ合成フィルタを求める必要がある。ここでは、直前の正常フレームにおける最後の正常なＬＰＣ係数のセットを、平滑化したものを使用する。この平滑化は、帯域幅拡張によって達成され、帯域幅拡張係数FAC（ＴＴＣ標準ＪＴ－Ｇ７２８、５．１節）が、$253/256 \approx 0.9883$の代わりに0.97であること以外は、ブロック５１（ＴＴＣ標準ＪＴ－Ｇ７２８、５．６節）の帯域幅拡張モジュールと同等である。正常フレームの後にくる最初の消失フレームにおいて、フレーム消失がない場合の通常のＬＰＣ係数更新時期と同じく、この帯域幅拡張操作は、最初の適応周期の第3ベクトルで行われる。

a_iを直前の正常フレームにおける最後の適応周期において使用されたi次のＬＰＣ係数とする。すると、平滑化されたＬＰＣ係数の新しいセットは、$a'_i = (0.97)^i a_i,\ i = 1, 2, \ldots, 50$として得られる。消失の最初の10ms区間は、たとえ同じフレーム中に次の適応周期があるとしても、この新しい$\{a'_i\}$のセットを使用する。消失が10msを越えるようであれば、この$\{a'_i\}$のセットは、係数0.97でさらに帯域幅拡張される。言い換えれば、消失が10ms以上継続する場合、ＬＰＣ係数は5番目の適応周期における第3ベクトルで更新され、i次のＬＰＣ係数は$a''_i = (0.97)^i a'_i = (0.97)^{2i} a_i$となる。今度はこの$\{a''_i\}$のセットを消失の次の10ms全体で使用する。同様に、10msのk倍の区間消失が連続した場合のＬＰＣ係数は、最後の正常なＬＰＣ係数のセットを係数$(0.97)^k$で帯域幅拡張したものである。直前の正常フレームから次の正常フレームが受信されるまで、さらに帯域幅拡張が適用され続ける。正常フレームが受信された後、再び異常フレームがあった時、この操作は再び帯域幅拡張係数0.97から始まる。

最初の消失フレームでは、帯域幅拡張用の最新のＬＰＣ係数を得るために、特別の操作を行う。具体的にいえば、もし直前の正常フレームで得られた配列A( )に対して帯域幅拡張を直接行うなら、その配列A( )は、直前の正常フレームより以前の合成音声に基づいて計算された自己相関関数に対応することになる。そのため、それは最初の消失フレームにとってはかなり古いものとなる。帯域幅拡張のための最新のＬＰＣ予測係数配列A( )を得るため、50次のＬＰＣ分析のためのハイブリッド窓かけとレビンソン・ダービン再帰法を、最初の消失フレームで行う。その結果得られた配列A( )を、係数0.97で帯域幅拡張する。それらは、直前の正常フレームの最後のベクトルまでの合成音声に基づいているため、結果として生じるＬＰＣ係数は最新である。10msを越えるような消失については、この様な特別の操作は必要なく、帯域幅拡張には、前の段落で記述されているように、以前に帯域幅拡張されたメモリ内における配列A( )の最後のセットを用いる。

利得予測器の帯域幅拡張については、消失フレーム区間におけるこのような処理は必要ない。利得調整された励振信号ET( )はすでに外挿されており、外挿されたET( )をＬＰＣ合成フィルタへ直接入力し、合成音声を得ている。したがって、消失フレーム区間においては、バックワード適応励振信号利得を算出したり利得予測係数を更新する必要はない。

２．３　ＬＰＣ予測器および利得予測器のバックワード適応

フレーム消失区間においては、バックワード合成フィルタ適応器およびバックワードベクトル利得適応器（図４－１／JT-G728のブロック３３および３０）の出力は必要とされない。しかしながら、次の正常フレームが来たときに内部状態の不連続性を緩和するため、これら２つのブロック内で行われるバックワード適応手順の必須動作は続ける必要がある。ここで“必須動作”とは、バックワード適応動作の中でも将来のフレームにおいてブロック３０および３３の出力に影響を及ぼす動作のことを言う。

ブロック３０（バックワードベクトル利得適応器）について説明する。これはブロック２０（ＴＴＣ標準ＪＴ－Ｇ７２８、５．７節および図３－５／JT-G728）と全く同一である。ブロック３０（バックワードベクトル利得適応器）では、ブロック６７，３９，４０，４１，４２（これらはオフセットを引いた対数利得を算出するのに用いられる）と、ブロック４３（ハイブリッド窓かけモジュール）およびブロック４６（対数利得線形予測

器）の一部が必須動作に含まれる。ブロック６７および３９～４２は、外挿されたET( )を用いてオフセットを引いた対数利得を算出する。この対数利得値はGTMP( )とGSTATE( )の更新に用いられる。GTMP( )は、ブロック４３の入力であり、GSTATE( )は、対数利得線形予測器のフィルタメモリである。ブロック４３（ハイブリッド窓かけモジュール）内部では、GTMP( )を用いてバッファSBLG( )の更新が行われ、さらに自己相関関数の巡回部分であるREXPLG( )の更新も行われる。これらの演算によって、ブロック３０の将来の出力に影響する内部状態が維持される。レビンソン・ダービン再帰法（ブロック４４）、帯域幅拡張（ブロック４５）、逆対数計算器（ブロック４８）その他の演算は、フレーム消失区間においては行う必要がない。これらの演算結果はいずれにせよ用いられないからである。

ここで注意すべきは、利得予測器の係数GP( )は、フレーム消失区間においては更新されないので、フレーム消失が終わった後の最初のベクトルはフレーム消失前の最後の正常フレームにおいて計算された古いGP( )を用いることになる、ということである。これは、ＴＴＣ標準ＪＴ－Ｇ７２８の適応周期の2番目のベクトルになるまで配列GP( )が更新されないためである。しかし、アルゴリズムのフロー制御を不必要に複雑にしてしまうため、最初のベクトルに対する配列GP( )をより新しいものにするということはしない。また、２．５節で説明する利得制限によってこの最初のベクトルに対する利得が大きく逸脱することを防いでいるため、一つのベクトルのみにやや古い利得予測器を用いても、音声品質に聴覚的な差異を生じることはない。

つぎにブロック３３（ＴＴＣ標準ＪＴ－Ｇ７２８、５．１４節の復号器バックワード合成フィルタ適応器）について説明する。これはブロック２３（ＴＴＣ標準ＪＴ－Ｇ７２８、５．６節および図３－４／JT-G728）と全く同一である。ブロック３３において、必須動作に含まれるものは、 (1)（直前の正常フレームにおけるＬＰＣフィルタを帯域幅拡張したものに、外挿したET( )を通して得られる）合成音声を用いたハイブリッド窓かけモジュール（ブロック４９）の内部で行われるバッファSB( )の更新、および、(2)SB( )を用いた通常の方法によるREXP( )の計算である。ブロック３３においては、その他のほとんどの部分はフレーム消失区間においては行う必要がない。これは、線形予測係数の更新がブロック５０，５１の出力を用いて行われるのではなく、直前の正常フレームにおける線形予測係数の帯域幅を拡張したものを用いて行われるためである。しかしながら、ブロック４９の追加部分とブロック５０の一部分は、ポストフィルタのために実行する必要がある。このことに関しては次節で述べる。

これまで説明したように、フレーム消失区間においてはブロック３０および３３に含まれている演算量の集中する多くの動作については、実行する必要がない。この演算量の節約分（あるいはＤＳＰ処理時間）は、励振信号の外挿と直前の正常フレームにおける線形予測フィルタの帯域幅拡張に当てることができるが、これらに要する演算量はかなり少ないものである。ＴＴＣ標準ＪＴ－Ｇ７２８の復号器に変更が加えられても全体の演算量が増加しないのはこのためである。

２．４　ポストフィルタ

フロー制御とプログラムの単純化のために、ポストフィルタとポストフィルタ適応器（図４－１／JT-G728のブロック３４，３５）の動作は、フレーム消失区間においても、あたかも何も起こらなかったように、正常フレームにおける場合と全く同様に実行される。これにはいくつかの理由がある。

まず第一に、フレーム消失区間においてもポストフィルタ適応器は10次のＬＰＣ予測係数と1次の反射係数をバックワード合成フィルタ適応器（図４－１／JT-G728のブロック３３）から受け取らなければならない。このため、ブロック４９（図３－４／JT-G728のハイブリット窓かけモジュール）において、RTMP(12)～RTMP(51)の計算は必要ないものの、RTMP(1)～RTMP(11)の値については計算しなければならない。同様に、ブロック５０では、11次から50次まで続ける必要はないものの、1次から10次までレビンソン・ダービン再帰法を実行しなければならない。なお、実際には、フレーム消失区間では10次のＬＰＣ係数と1次の反射係数は、外挿された励振信号ET( )を駆動して最終的に得られる合成音声から得られることに注意が必要である。

第二に、フレーム消失区間においてポストフィルタは、消失フレームで合成された音声に基づく全ての係数を更新し、この係数を用いて合成音声をフィルタにかけ、最終的な出力音声を生成する。つまり、フレーム消失区間において、ポストフィルタ係数を凍結すること、あるいは（ＬＰＣフィルタに対して行ったように）直前の正常フレームにおける最後のポストフィルタの帯域幅拡張したものを用いるということはしない。意図的に外挿された励振信号から合成された音声で、ポストフィルタを動かす。これは、ポストフィルタの周波数応答スペクトルの山と谷が、全ての時間において、合成音声のそれと釣り合うことを意味する。これにより、二者のミスマッチのために起こるポストフィルタの周波数応答スペクトルの谷による合成音声スペクトルの山の相殺、あるいは、

強調が適切になされないといったことを避けることができる。

フレーム消失区間において、ポストフィルタは、ピッチ周期KPおよびピッチ予測器タップPTAPの更新を続けるが、消失継続の間は、それらの値を用いて有声／無声判定VOICEDとピッチ周期FEDELAYを更新することはしない。VOICEDとFEDELAYの値は、消失開始時に設定され、消失中は変更しない。

２．５　フレーム消失後の利得増加の制限

100ms程度まで続く非常に長いフレーム消失後、あるいは、小振幅音声区間から大振幅音声区間へと続く途中でフレーム消失が起きた場合、フレーム消失後の復号音声は、時々大きなポップノイズや利得オーバシュートとなる。これは、ハイブリッド窓の対数利得バッファSBG( )が、フレーム消失の最後には、非常に低レベルの利得となり、フレーム消失後の最初の正常な数フレームでは、このような低い利得が急激に高い利得になるためである。このように、ＬＰＣ分析では、利得レベルの突然の変化に早く追いつこうとする利得予測器を構成する。それゆえ、予測利得（すなわちバックワード適応利得）は、非常に早く増加して、利得オーバシュートとポップノイズを引き起こす。

この問題は、フレーム消失後の最初の正常な数フレームに対して、バックワード適応利得の増加の割合を制限することによって避けることができる。具体的には、フレーム消失後、5サンプルのベクトル毎にFEGAINMAX = +2dBにしかならないように利得の増加を制限する。すなわち、予測された対数利得における増加が+2dB／ベクトル以上ならば、直前の対数利得より2dBの増加に対数利得を抑える。このような利得制限の継続時間は、フレーム消失の長さに依存する。もしフレーム消失が、AFTERFEMAX = 16（40ms）以下ならば、利得制限の継続時間はフレーム消失の長さと同じとする。もしフレーム消失が40msより長ければ、このような利得制限を消失後40msだけ行う。2つの定数FEGAINMAXとAFTERFEMAXは、調整可能な復号器のパラメータである。

３．フレーム消失補償擬似コード

最初に、復号器のメインプログラム擬似コードを以下に示す。ブロックの実行順序のみを示し、受け渡されるパラメータの詳細については触れない。続く節において、新しい各ルーチンの擬似コードを示す。

３．１　復号器の新しいメインプログラムループ

固定小数点版と浮動小数点版の復号器のメインプログラムループは、固定小数点記述用に変更されているバックワードベクトル利得適応器に関してのみ異なっている。メインループにおけるこれらの変更は比較的小さいので、ここでは一つにまとめた擬似コードを示す。どのセグメントが固定小数点記述に属し、どのセグメントが浮動小数点記述に属すかを示すために、C言語プリプロセッサ構文の#ifdef, #else, #endifを使用する。#ifdef FIXPTと次の#elseの間の擬似コードは固定小数点記述用、#elseと次の#endifの間の擬似コードは浮動小数点記述用とする。

```
Initialize all decoder variables to their initial values.
ILLCOND=.FALSE.
ILLCONDG=.FALSE.
FERROR=.FALSE.   | TRUE if current frame is erased
FESIZE=4         | Number of 2.5 msec adaptation cycles in a frame
                 | erasure(FE)
                 | e.g.: 1 for 2.5 msec FEs, 4 for 10 msec FEs
                 | FEs may be any multiple of 2.5 msec. Set by user.
ICOUNT=4         | Vector counter, 1,2,3,4,1,2,.., .625 ms each
ADCOUNT=FESIZE   | Adaptation cycle counter, 2.5 msec each
                 | Counts from 1,2,...,FESIZE,1,2,...,FESIZE,...;
FECOUNT=0        | Number of consecutively erased 2.5 msec adaptation
                 | cycles
AFTERFE=0        | Number of 2.5 msec adaptations cycles after a FE to
                 | limit the gain. Counts down.
OGAINDB=-32      | Last predicted gain in dB
```

```
VEC_LOOP:
        If ICOUNT = 4, do the next 14 lines
            ICOUNT = 0                          | reset vector counter
                        | If the last adaptation cycle was not erased, and the
                        | gain
                        | was clamped decrease the number of adaptation cycles
                        | left
                        | to clamp the gain.
            If FERROR = .FALSE. and AFTERFE > 0, do AFTERFE = AFTERFE - 1
                        | Check if the next frame is erased
            If ADCOUNT = FESIZE, do the next 7 lines
                    ADCOUNT = 0                 | reset adaptation cycle counter
                    read FERROR                 | read in new ferror
                        | At the first good frame after an erasure set AFTERFE
                        | so the gain will be clamped for the next few good frames.
                    If FERROR = .FALSE. and FECOUNT > 0, do the next 4 lines
                        AFTERFE = AFTERFE + FECOUNT     |add the length of the FE to AFTERFE
                        If AFTERFE > AFTERFEMAX, do the next line
                            AFTERFE = AFTERFEMAX        | clamp gain for 40 msec max
                        FECOUNT = 0             | reset FECOUNT
            If FERROR = .TRUE., do the next 2 lines
                FECOUNT = FECOUNT + 1           | Update FECOUNT
                        | At the start of a FE, and every 10 msec thereafter,
                        | call
                        | block31SF to initialize the FE flags and update the
                        | excitation FE gains. e.g. when FECOUNT is equal to
                        | 1, 5, 9, 13, ...
                If (FECOUNT & 3) = 1, do block31SF
            ADCOUNT = ADCOUNT + 1               | update adaptation cycle counter
        ICOUNT = ICOUNT + 1                     | update vector counter

        Get ICHAN of the current vector from the input buffer.
        Obtain the shape index IS and gain index IG from ICHAN

                                                | check whether to update LPC coeff.
        If ICOUNT = 3, do the next 2 lines
            If FERROR = .FALSE. and ILLCOND = .FALSE., do block 51.
            If FERROR = .TRUE. and (FECOUNT & 3) = 1, do block 51FE
        If ICOUNT = 2 and ILLCONDG=.FALSE. and FERROR = .FALSE., do block 45.

        do block 46                             | GSTATE(1:9) shifted down 1 position

#ifdef FIXPT
        If FERROR = .FALSE., do the next 3 lines
            do blocks 98AF, 99 and 48           | get backward-adapted gain
            OGAINDB = output of block 98AF      | for limiting gain growth after FE
            do blocks 29 and 31                 | scale selected excitation codevector
#else
```

```
        If FERROR = .FALSE., do the next 3 lines
            do blocks 47AF and 48                | get backward-adapted gain
            OGAINDB = output of block 47AF       | for limiting gain growth after FE
            do blocks 29 and 31                  | scale selected excitation codevector
#endif
        If FERROR = .TRUE., do the next line
            do block 31FE                        | get extrapolated excitation

        do block 31E                             | update ETPAST()
        do block 32                              | get ST()
        If ICOUNT = 1, do block 85               | update short-term postfilter coeff.
        do block 81                              | 10th-order LPC inverse filtering
        If ICOUNT = 3, do the next 3 lines
            do block 82                          | pitch period extraction
            do block 83                          | compute pitch predictor tap
            do block 84                          | update long-term postfilter coeff.
        do block 71                              | long-term postfilter
        do block 72                              | short-term postfilter
        do blocks 73 and 74                      | calculate sums of absolute values
        do block 75                              | ratio of sums of absolute values
        do block 76                              | low-pass filter of scaling factor
        do block 77                              | gain control of postfilter output

#ifdef FIXPT
        If FERROR = .FALSE., do the next 2 lines
            do blocks 93, 94, 96, and 97         | update log-gain
            GSTATE(1) = output of block 97       | update gain predictor memory
#else
        If FERROR = .FALSE., do the next 2 lines
            do blocks 39, 40 and 42              | update log-gain
            GSTATE(1) = output of block 42       | update gain predictor memory
#endif

        If FERROR = .TRUE., do the next 3 lines
            do block 97FE                        | compute log gain of extrapolated ET()
            OGAINDB = output of 97FE             | save for limiting gain growth after FE
            GSTATE(1) = output of block 97FE
                                                 | then update gain predictor memory

        I=(ICOUNT-1) * IDIM                      | I=starting address of STTMP()
        copy ST(1:5) to STTMP(I+1:I+5)           | update STTMP()
        NLSSTTMP(ICOUNT)=NLSST                   | update NLSSTTMP()
                                                 | end of once-per-vector processing

                                                 | start once-per-cycle processing
        If ICOUNT = 4, do the next 6 lines
            do block 49FE                        | output ill-condition flag = ILLCOND
            do block 50, order 1 to 10           | output pred coef= ATMP() with NLSATMP
                                                 | output ill-condition flag = ILLCONDP
```

```
        NLSAPF=NLSATMP                  | save the 10th-order predictor for
        copy ATMP(2:11) to APF(2:11)    | postfilter use later
        If FERROR = .FALSE., do the next line
            continue block 50, order 11 to 50   | continue to finish block 50
                                        | output pred coef= ATMP() with NLSATMP
                                        | output ill-condition flag = ILLCOND

    If ICOUNT = 1, do the next 7 lines
        GTMP(1)=GSTATE(4)               | update GTMP( ) in one shot
        GTMP(2)=GSTATE(3)
        GTMP(3)=GSTATE(2)
        GTMP(4)=GSTATE(1)
        do block 43FE                   | output ill-condition flag = ILLCONDG
        If FERROR = .FALSE., do the next line
            do block 44                 | output pred. coeff. = GPTMP( )
                                        | output ill-condition flag = ILLCONDG
                                        | end of once-per-frame processing
    Go to VEC_LOOP
```

３．２　ブロック３１ＳＦ－フレーム消失のためのフラグとスケールファクタの設定

新ブロックである。フレーム消失の最初に呼ばれ、消失中はその後も10ms毎に呼ばれる。消失の初めで、PTAPの値を基に有声／無声フラグを設定する。もし、直前の正常フレームが有声であれば、周期的な外挿のために、FEDELAYにピッチ周期KPを保存しておく。もし無声であれば、直前の正常フレームのETPAST( )から、最後の40サンプルの大きさの総和を計算し、結果を8で除算する。これらの値は消失の初めに一度計算され、消失の間ずっと使用される。10ms以上消失が継続すると、ブロック３１ＳＦは（10ms毎に）再び呼ばれ、励振外挿スケーリングファクタFESCALEが、消失の長さや有声か無声かに基づいて減衰される。有声音に対しては、もし消失が60ms以上継続すると、FESCALEは人工的なビープ音を抑えるために0に設定される。もし無声音の消失であれば、FESCALEは70msで0に設定される。先ず、浮動小数点擬似コードを示す。

```
VOICEDFEGAIN(0:4) = .8, .8, .6, .4, .2
UNVOICEDFEGAIN(0:5) = 1., 1., .8, .6, .4, .2

N10MSEC = FECOUNT >> 2
If N10MSEC == 0, do the following indented lines
    If PTAP > VTH, do the next 2 indented lines
        FEDELAY = KP
        VOICED = .TRUE.
    Otherwise, do the following indented lines
        VOICED = .FALSE.
        AVMAG = 0.
        For I = -39, -38, ..., 0, do the next line
            AVMAG = AVMAG + |ETPAST(I)|     | sum of ETPAST() magnitudes
        AVMAG = 0.125 * AVMAG               | divide by 8
If VOICED = .TRUE., do the following indented lines
    If N10MSEC < 5, do FESCALE = VOICEDFEGAIN(N10MSEC)
    Otherwise, do FESCALE = 0.
Otherwise, do the following indented lines  | not voiced
    If N10MSEC < 6, do FESCALE = AVMAG * UNVOICEDFEGAIN(N10MSEC)
    Otherwise, do FESCALE = 0.
```

次に、固定小数点擬似コードを示す。

```
                                                        | Constants below are in Q15
VOICEDFEGAIN(0:4) = 26214, 26214, 19661, 13107, 6554
UNVOICEDFEGAIN(0:5) = 32767, 32767, 26214, 19661, 13107, 6554

N10MSEC = FECOUNT >> 2
If N10MSEC == 0, do the following indented lines
    If PTAP > VTH, do the next 2 indented lines
        FEDELAY = KP
        VOICED = .TRUE.
    Otherwise, do the following indented lines
        VOICED = .FALSE.
        AA0 = 0
        For I = -39, -38, ..., 0, do the next line
            AA0 = AA0 + |ETPAST(I)|                     | sum of 40 ETPAST magnitudes
        Call VSCALE(AA0,1,1,30,AA0,NLS)                 | normalize numerator
        AVMAG = RND(AA0)
        NLSAVMAG = NLS - 16 + 3                         | "+3" is for "divided by 8"
If VOICED = .TRUE., do the following indented lines
    If N10MSEC < 5, do FESCALE = VOICEDFEGAIN(N10MSEC)
    Otherwise, do FESCALE = 0
Otherwise, do the following indented lines              | not voiced
    If N10MSEC < 6, do the following indented lines
        FESCALE = UNVOICEDFEGAIN(N10MSEC)               | scaling factor in Q15
        AA0 = FESCALE  *  AVMAG                         | scale AVMAG by FESCALE
        Call VSCALE(AA0,1,1,30,AA0,NLSFESCALE)          | normalize FESCALE
        FESCALE = RND(AA0)                              | save result in FESCALE
        NLSFESCALE = NLSFESCALE + NLSAVMAG - 1          | NLS for FESCALE
                                                        | -1 is from (Q15 - 16)
    Otherwise, do FESCALE = 0
```

３．３　ブロック３１ＦＥー励振信号の外挿

新ブロックであり、消失フレーム区間でのみ使用される。直前の正常フレームが有声か否かによって、2通りの方法で励振信号を外挿する。有声音については、周期的に一定の減衰を繰り返して外挿される。スケーリングファクタFESCALEは0.8から始まり、ブロック３１ＳＦで消失が続く間、10ms毎に減衰する。

その他（無声、無音など）については、直前の140サンプルからランダムに選んだセグメントが使用される。このランダムなセグメントは、直前の40サンプルの大きさの平均値と、同じ大きさになるように調整される。5サンプルのランダムセグメントと、直前の40サンプルの大きさの平均値を計算するには、大きさの総和を、それぞれ5と40で除算すべきであるが、必要なのは、2つの大きさの平均値の比だけである。よって実際には、5サンプルのランダムセグメントの大きさの総和としてのMAGと、直前の40サンプルの大きさの総和を8で除算したAVMAGを計算する。こうして求めた2つの値の比が、求めるべき比である。AVMAGの値は、最初の消失フレームにおける最初の適応周期において、前記のブロック３１ＳＦで計算される。AVMAGは、ブロック３１ＳＦにて、消失の継続時間を基に減衰され、FESCALEに代入される。フレーム消失区間では、ＴＴＣ標準ＪＴ－Ｇ７２８符号器との相互接続性の問題は起こらないので、乱数系列を事前に定義する必要はない。また乱数は、厳密にランダムである必要も無い。その系列が周期的でないか、または、ほぼ周期的でない限り、5から140のどんな系列の整数でも使用できる。よって乱数発生器の擬似コードは示さない。以下に、まず、浮動小数点擬似コードを示す。

```
                                                    |   Voiced case
If VOICED = .TRUE., do the next 2 indented lines
    For I = 1, 2, ..., IDIM, do the next line
        ET(I) = FESCALE  *  ETPAST(I-FEDELAY)       | KP is pitch period
                                                    | from 82

                                                    | Unvoiced case
If VOICED = .FALSE., do the following indented lines
    set FEDELAY = a random number between 5 and 140
    MAG = 0.
    For I = 1, 2,   ..., IDIM, do the next 2 lines
        ET(I) = ETPAST(I-FEDELAY)
        MAG = MAG + |ET(I)|                         | sum of ET() magnitudes
    If MAG = 0 do MAG = 1.                          | avoid divide by zero
    RATIO = FESCALE / MAG

    For I = 1, 2,   ..., IDIM, do the next line
        ET(I) = RATIO * ET(I)                       | scale to have same magnitude
```

固定小数点擬似コードでは、ETとETPASTは異なった表現を使用している。ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇで述べたように、ETは最上位が0である15ビットブロック浮動小数点で表現される。一方、このブロックの簡易化のため、長いETPAST配列の表現に固定小数点形式のQ2フォーマットを使用する。ETPASTにブロック浮動小数点を使用する理由はない。なぜなら、外挿した励振信号と符号器の真の励振信号との数値的誤差よりも、固定小数点形式のQ2フォーマット表現による誤差は十分小さいからである。ETPASTのオーバフローという非常に希なケースでは、ETPASTの値は“ラップアラウンド”を避けるために、+32767または-32768の飽和レベルでクリップされる。以下の擬似コードでは、先ずETPAST(1)からETPAST(5)の中に、外挿励振ベクトル（Q2フォーマット）を書き込み、それから外挿した5つのETPASTサンプルを、15ビットブロック浮動小数点形式のET配列に変換する。

```
If VOICED = .TRUE., do the next 4 lines             | Voiced case
    TEMP = FESCALE
    NLSTEMP = 15                                    | FESCALE is Q15
    For I = 1, 2,   ..., IDIM, do the next line
        ETPAST(I) = ETPAST(I-FEDELAY)               | FEDELAY is pitch period

If VOICED = .FALSE., do the next 13 lines           | Unvoiced case
    set FEDELAY = a random number between 5 and 140
    AA1 = 0
    For I = 1, 2,   ..., IDIM, do the next 2 lines
        ETPAST(I) = ETPAST(I-FEDELAY)
        AA1 = AA1 + |ETPAST(I)|                     | sum of ETPAST(1:5) magnitudes
    If AA1 = 0 or FESCALE = 0, do the next 2 lines
        TEMP = 0
        NLSTEMP = 15
    Otherwise, do the next 4 lines
        Call VSCALE(AA1,1,1,30,AA1,NLSDEN)          | normalize
                                                    | denominator
        DEN = RND(AA1)
        NLSDEN = NLSDEN - 16
```

```
Call DIVIDE(FESCALE,NLSFESCALE,DEN,NLSDEN,TEMP,NLSTEMP)

For I = 1, 2,  ..., IDIM, do the next 5 lines
    AA0 = TEMP  *  ETPAST(I)                  | AA0 in Q(2+NLSTEMP) format
    AA0 = AA0 >> NLSTEMP                      | shift AA0 to make Q2
    if AA0 > 32767, set AA0 = 32767           | clip if necessary
    if AA0 < -32768, set AA0 = -32768
    ETPAST(I) = AA0                           | scaled ETPAST in Q2
Call VSCALE(ETPAST,IDIM,IDIM,13,ET,NLS)       | convert to ET in 15-bit
                                              | BFL
NLSET = NLS + 2                               | update NLSET
```

３．４　ブロック３１Ｅ－励振信号の更新

このブロックでは、励振信号バッファの更新を行う。必要な場合にいつでもバッファの内容を参照できるように、フレーム消失の有無にかかわらずこのブロックは常に実行する必要がある。以下、浮動小数点擬似コードを示す。

```
For I = -KPMAX+1, -KPMAX+2, ..., -IDIM, do the next line
    ETPAST(I) = ETPAST(I+IDIM)                | shift ETPAST by IDIM samples
For I = 1, 2, ..., IDIM, do the next line
    ETPAST(I-IDIM) = ET(I)                    | copy ET to ETPAST
```

固定小数点擬似コードでは、ETPASTはQ2フォーマットで格納され、ETはNLSETで示されるQフォーマットの15ビットブロック浮動小数点形式で格納される。従って、ETからETPASTへのコピーにあたってはフォーマット変換が必要となる。以下、固定小数点擬似コードを示す。

```
For I = -KPMAX+1, -KPMAX+2, ..., -IDIM, do the next line
    ETPAST(I) = ETPAST(I+IDIM)                | shift Q2 ETPAST by 5 samples
NRS = NLSET - 2                               | # of right shifts to make Q2
For I = 1, 2, ..., IDIM, do the next 6 lines
    AA0 = ET(I)                               | get 15-bit mantissa of ET
    if NRS > 0, AA0 >> NRS                    | perform the shifting
    if NRS < 0, AA0 << -NRS
    if AA0 > 32767, set AA0 = 32767           | clip if necessary
    if AA0 < -32768, set AA0 = -32768
    ETPAST(I-IDIM) = AA0                      | ETPAST is now Q2
```

３．５　ブロック４３ＦＥ－ハイブリッド窓かけモジュール

この節では、ブロック４３ＦＥの浮動小数点擬似コードと固定小数点擬似コードを示す。この擬似コードは、フレーム消失補償時に対応するための変更がなされており、ＴＴＣ標準ＪＴ－Ｇ７２８、５．７節に記述されている本来のブロック４３と置き換えられる。浮動小数点擬似コードにおける変更は、フレーム消失時に自己相関関数の計算における最後の部分をスキップするという1行を追加するのみである。まず、浮動小数点擬似コードを示す。

```
N1=LPCLG+NUPDATE                  | compute some constants (can be
N2=LPCLG+NONRLG                   | precomputed and stored in memory)
N3=LPCLG+NUPDATE+NONRLG
```

```
For N=1,2,...,N2, do the next line
      SBLG(N)=SBLG(N+NUPDATE)          | shift the old signal buffer;
For N=1,2,...,NUPDATE, do the next line
      SBLG(N2+N)=GTMP(N)               | shift in the new signal;
                                       | SBW(N3) is the newest sample
K=1
For N=N3,N3-1,...,3,2,1, do the next 2 lines
      WS(N)=SBLG(N) * WNRLG(K)         | multiply the window function
      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   | update the recursive
                                       | component

If FERROR = .TRUE., skip all the lines below
      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)  | add the non-recursive
                                               | component

      R(1)=R(1) * WNCF                 | white noise correction
```

次に同じモジュールの固定小数点版を示す。固定小数点擬似コードに対する変更は、HWMCOREの代用モジュールであるHWMCOREFEの中だけである。ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ、３．１４節にある本来のブロック４３の固定小数点擬似コードを以下の固定小数点擬似コードに置き換える。サブルーチンHWMCOREFEは、
３．７節のブロック４９ＦＥに示す。

```
N1=LPCLG+NUPDATE (=10+4)           | compute some constants (can be
N2=LPCLG+NONRLG (=10+20)           | precomputed and stored in memory)
N3=LPCLG+NUPDATE+NONRLG (=10+4+20)

For N=1,2,...,N2, do the next line
      SBLG(N)=SBLG(N+NUPDATE)          | shift the old signal buffer
For N=1,2,..,NUPDATE, do the next line
      SBLG(N2+N)=GTMP(N)               | SBLG(N3) is the newest sample
                                       | all SBLG are Q9 and represented
                                       | in 16 bits precision
Call FINDNLS(SBLG,N3,N3,14,NLS)        | find the amount of left shifts
NLSTMP=NLS-1                           | needed in the next loop for 2
                                       | bits of headroom later

K=1
For N=34,33,...,1, do the next 5 lines
      P=SBLG(N) * WNRLG(K)             | WNRLG is Q15
```

```
    If NLSTMP = -1, set AA0=P >> 1
    If NLSTMP > -1, set AA0=P << NLSTMP
    WS(N)=RND(AA0)                          | WS(N) is 14 bits or less
    K=K+1

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

３．６　ブロック４７ＡＦ－フレーム消失後の対数利得リミッタ

ＴＴＣ標準ＪＴ－Ｇ７２８、５．７節にあるブロック４７をこの新しいブロックと置き換える。以下に浮動小数点擬似コードを示す。

```
If GAIN < 0., set GAIN = 0.             | Correspond to linear gain 1.
If GAIN > 60., set GAIN = 60.           | Correspond to linear gain 1000.
TMP = GAIN - OGAINDB
If AFTERFE > 0 and TMP > FEGAINMAX, do the next line
    GAIN = OGAINDB + FEGAINMAX
```

３．７　ブロック４９ＦＥ－合成フィルタ用ハイブリッド窓かけモジュール

このブロックは、フレーム消失時に対応して変更された、合成フィルタ用ハイブリッド窓かけモジュールである。ＴＴＣ標準ＪＴ－Ｇ７２８、５．６節に記述されている本来のブロック４９と置き換えられる。以下にハイブリッド窓かけモジュールの浮動小数点擬似コードを示す。

```
N1=LPC+NFRSZ                            | compute some constants (can be
N2=LPC+NONR                             | precomputed and stored in memory)
N3=LPC+NFRSZ+NONR

For N=1,2,...,N2, do the next line
    SB(N)=SB(N+NFRSZ)                   | shift the old signal buffer;
For N=1,2,...,NFRSZ, do the next line
    SB(N2+N)=STTMP(N)                   | shift in the new signal;
                                        | SB(N3) is the newest sample
K=1
For N=N3,N3-1,...,3,2,1, do the next 2 lines
    WS(N)=SB(N) * WNR(K)                | multiply the window function
    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         | update the recursive component

If FERROR = .TRUE., do the next 4 lines
    For I=1,2,...,11, 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)   | add non-recursive
                                                | component
```

```
If FERROR = .FALSE., do the next 4 lines
    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)        | add non-recursive
                                                    | component

RTMP(1)=RTMP(1)＊WNCF                               | white noise correction
```

ハイブリッド窓かけモジュール（ブロック４９ＦＥ）に対する固定小数点擬似コードも同様にして変更される。実際には、HWMCOREFEの中での変更である。ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ、３．１７節にある本来のブロック４９の固定小数点コードをこの固定小数点擬似コードに置き換える。

```
N1=LPC+NFRSZ(=70)                   | compute some constants (can be
N2=LPC+NONR(=85)                    | precomputed and stored in memory)
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)               | shift old part of buffer SB
For N=1,2,...,N6, do the next line
    NLSSB(N)=NLSSB(N+N5)            | shift old NLSSB
For N=1,2,...,NFRSZ, do the next line
    SB(N2+N)=STTMP(N)               | shift in new part of SB
For N=1,2,...,N5, do the next line
    NLSSB(N6+N)=NLSSTTMP(N)         | shift in new NLSSB

                    | Now find the minimum NLSSB - this determines NLSTMP
NLSTMP=Min{NLSSB(1),NLSSB(2),...,NLSSB(N4)}

K=1                                 | now multiply SB by the
N=N3                                | hybrid window function
For J=1,2,...,N4, do the next 8 lines
    NRSH=NLSSB(J)-NLSTMP-1          | -1 to compensate for Q15 mult
    For M=1,2,...,IDIM, do the next 5 lines
        P=SB(K)＊WNR(N)             | WNR is Q15 multiplication
        If NRSH = -1, set AA0=P << 1
        If NRSH > -1, set AA0=P >> NRSH
        WS(K)=RND(AA0)              | round upper word & store in WS
        N=N-1
    K=K+1

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

フレーム消失時にはREXP全体（REXP(1)～REXP(51)）の更新とRTMPの一部のみ（RTMP(1)～RTMP(11)）の計

算を行うように、HWMCOREFEの固定小数点コードは変更されている。

```
SUBROUTINE HWMCOREFE(LPO,N1,N3,NLSATT,WS,NLSTMP,RREC,NLSRREC,R,ILLCOND)
NLSAA0=2 * NLSTMP
AA0=0                                   | compute recursive part of RREC(1)
For N=LPO+1,...,N1, do the next 2 lines
    P=WS(N) * WS(N)                     | WS has 2 bits of headroom
    AA0=AA0+P                           | AA0 will have 5 bits of headroom
                                        | for energy calculation

                                        | Case 1: NLSRREC > NLSAA0
If NLSRREC > NLSAA0, do the next 22 lines
    AA0=AA0 >> 1
    IR=NLSRREC-NLSAA0+1
    AA1=RREC(1) << NLSATT               | this can be done by multiplication
    AA1=-AA1+(RREC(1) << 16)            | scale RREC by attenuation factor
    AA1=AA1 >> IR                       | align AA0 & AA1
    AA0=AA0+AA1
    Call VSCALE(AA0,1,1,30,AA0,NLSRE)   | Find NLS for RREC
    RREC(1)=RND(AA0)                    | upper 16 bits of AA1 saved
    NLSRREC=NLSAA0-1+NLSRE
    For I=1,2,...,LPO, do the next 11 lines
        AA0=0                           | compute recursive part of 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         | scale RREC by 3/4 or 1/2
        AA1=-AA1+(RREC(I+1) << 16)      |
        AA1=AA1 >> IR
        AA0=AA0+AA1
        AA0=AA0 << NLSRE
        RREC(I+1)=RND(AA0)              | upper 16 bits of AA0 saved
Go to FIN_RECUR

                                        | Case 2: NLSRREC = NLSAA0
If NLSRREC = NLSAA0, do the next 21 lines
    AA1=RREC(1) << NLSATT               | scale RREC by 3/4 or 1/2
    AA1=-AA1+(RREC(1) << 16)            |
    AA0=AA0 >> 1
    AA1=AA1 >> 1
    AA0=AA0+AA1
    Call VSCALE(AA0,1,1,30,AA0,NLSRE)   | Find NLS for RREC
    RREC(1)=RND(AA0)                    | upper 16 bits of AA1 saved
    NLSRREC=NLSRREC-1+NLSRE
    For I=1,2,...,LPO, do the next 11 lines
        AA0=0                           | compute recursive part of 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          | scale RREC by 3/4 or 1/2
            AA1=-AA1+(RREC(I+1) << 16)       |
            AA1=AA1 >> 1
            AA0=AA0+AA1
            AA0=AA0 << NLSRE
            RREC(I+1)=RND(AA0)               | upper 16 bits of AA0 saved
        Go to FIN_RECUR

                                             | Case 3: NLSRREC < NLSAA0
    If NLSRREC < NLSAA0, do the next 21 lines
        IR=NLSAA0-NLSRREC+1
        AA0=AA0 >> IR
        AA1=RREC(1) << NLSATT                | scale RREC by 3/4 or 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)                     | upper 16 bits of AA1 saved
        NLSRREC=NLSRREC-1+NLSRE
        For I=1,2,...,LPO, do the next 11 lines
            AA0=0                            | compute recursive part of 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          | scale RREC by 3/4 or 1/2
            AA1=-AA1+(RREC(I+1) << 16)       |
            AA1=AA1 >> 1
            AA0=AA0+AA1
            AA0=AA0 << NLSRE
            RREC(I+1)=RND(AA0)               | upper 16 bits of AA0 saved

FIN_RECUR:                                   | when you reach this point the
                                             | recursive component has been computed

    AA0=0                                    | compute non-recursive part of R(1)
    For N=N1+1,...,N3, do the next 2 lines
        P=WS(N) * WS(N)
        AA0=AA0+P

    LPFE = LPO
    If FERROR .=. TRUE, set LPFE = 10
                                             | Case 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                        | apply white noise correction factor
    AA1=AA1+AA0
    Call VSCALE(AA1,1,1,30,AA1,NLSRR)
    R(1)=RND(AA1)                       | upper 16 bits of AA1 saved
    For I=1,2,...,LPFE, do the next 10 lines
        AA0=0                           | compute non-recursive part of 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)                 | save upper 16 bits
    Go to END

                                        | Case 2: NLSRREC = NLSAA0
If NLSRREC = NLSAA0, do the next 18 lines
    AA0=AA0 >> 1
    AA1=RREC(1) << 15                   | this can be done by multiplication
    AA1=AA0+AA1
    AA0=AA1 >> 8                        | apply white noise correction factor
    AA1=AA1+AA0
    Call VSCALE(AA1,1,1,30,AA1,NLSRR)
    R(1)=RND(AA1)                       | upper 16 bits of AA1 saved
    For I=1,2,...,LPFE, do the next 9 lines
        AA0=0                           | compute non-recursive part of 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)                 | save upper 16 bits
    Go to END

                                        | Case 3: NLSRREC < NLSAA0
If NLSRREC < NLSAA0, do the next 18 lines
    IR=NLSAA0-NLSRREC+1
    AA0=AA0 >> IR
    AA1=RREC(1) << 15                   | this can be done by multiplication
    AA1=AA0+AA1
    AA0=AA1 >> 8                        | apply white noise correction factor
    AA1=AA1+AA0
    Call VSCALE(AA1,1,1,30,AA1,NLSRR)
```

```
        R(1)=RND(AA1)                       | upper 16 bits of AA1 saved
        For I=1,2,...,LPFE, do the next 9 lines
            AA0=0                           | compute non-recursive part of 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)                 | save upper 16 bits

END:                                        | one last job, check for ill-conditioning
    ILLCOND=.FALSE.
    If AA1 = 0, set ILLCOND=.TRUE.          | AA1 still contains 32 bit R(LPFE+1)
```

３．８　ブロック５１ＦＥ－フレーム消失時における帯域幅拡張モジュール

以下に、フレーム消失状態におけるＬＰＣ合成フィルタの帯域幅拡張モジュールであるブロック５１ＦＥの浮動小数点擬似コードを示す。このモジュールは、帯域幅拡張係数が異なるという点を除けば、ＴＴＣ標準ＪＴ－Ｇ７２８の５．６節ブロック５１と類似している。また、入力値は配列A(I)である。

```
    If FECOUNT = 1 and ILLCOND = .FALSE., do the next 2 lines
        For I=2,3,...,LPC+1, do the next line
            A(I)=FACVFE(I) ＊ ATMP(I)       | scale coeff.
    Otherwise, do the next 2 lines
        For I=2,3,...,LPC+1, do the next line
            A(I)=FACVFE(I) ＊ A(I)          | scale coeff.
```

FACVFEの表はQ14フォーマットで与えられる。入力される配列Aの値は常にQ14フォーマットである。配列Aの最終結果は常にQ14フォーマットで記述されるようにする。そのため、乗算後にシフト処理が必要である。以下にブロック５１の固定小数点擬似コードを示す。

```
    If FECOUNT = 1 and ILLCOND = .FALSE., do the next 4 lines
        For I=2,3,4,...,LPC+1, do the next 3 lines
            AA0 = FACVFE(I) ＊ ATMP(I)      | AA0 is Q28
            AA0 = AA0 << 2                  | make AA0 Q30
            A(I) = RND(AA0)                 | round to high word and get Q14
    Otherwise, do the next 4 lines
        For I=2,3,4,...,LPC+1, do the next 3 lines
            AA0 = FACVFE(I) ＊ A(I)         | AA0 is Q28
            AA0 = AA0 << 2                  | make AA0 Q30
            A(I) = RND(AA0)                 | round to high word and get Q14
```

３．９　ブロック９７ＦＥ－フレーム消失区間におけるGSTATEの更新

まず最初に、この新ブロックの浮動小数点版を示す。このブロックは、ＴＴＣ標準ＪＴ－Ｇ７２８の５．７節に記述されているブロック６７，３９，４０，及び４２と同じコードである。

```
    ETRMS = 0.
    For K = 1,...,IDIM, do the next line    | compute energy
```

```
      ETRMS = ETRMS + ET(K) * ET(K)          | of excitation
ETRMS = ETRMS  *  DIMINV
If ETRMS < 1., set ETRMS = 1.                | check lower limit
ETRMS = 10. *  log10 (ETRMS)                 | compute dB value

GSTATE(1) = ETRMS - GOFF                     | subtract dB gain offset
```

この新ブロックは固定小数点版においても同様の演算を行う。上記計算を実現するために、ＴＴＣ標準ＪＴ－Ｇ７２８本体を固定小数点版にする際に削除された関数$\log_{10}$を再び導入する必要がある。浮動小数点版の出力値と正確に整合させようとしていないため、$\log_{10}$の出力値に対して可能な限りの精度を必要とはしていない。以下に固定小数点擬似コードを示す。フレーム消失時においてはビットイグザクトな互換性を要求されないため、固定小数点で実現するためのLIN2DB関数のビットイグザクトな記述は勧告に含まないものとする。この関数LIN2DBは、スカラ浮動小数点で表現されたETRMSを、Q9フォーマットで表現されたデシベル値へ変換する。適当な精度を持っていれば、関数LIN2DBの固定小数点演算による実現方法は如何なる方法を用いても良い。このようなLIN2DBの固定小数点コードの一例も添付する。

```
AA0 = 0
For K = 1,...,IDIM, do the next line         | compute energy of excitation
      AA0 = AA0 + ET(K) * ET(K)

Call VSCALE(AA0,1,1,30,AA0,NLS)              | scale AA0
ETRMS = AA0 >> 16                            | take high word of AA0
NLSETRMS = (NLSET + NLSET + NLS) - 16        | NLS of ETRMS
AA0 = ETRMS  *  DIMINV                       | DIMINV = 0.2 in Q16
Call VSCALE(AA0,1,1,30,AA0,NLS)              | scale AA0
ETRMS = AA0 >> 16                            | take high word of AA0
NLSETRMS = NLSETRMS + NLS                    | NLS of ETRMS
If NLSETRMS > 14, do the next 2 lines        | if ETRMS is Q15 or smaller (<1)
      ETRMS = 16384                          | set ETRMS to 1 in Q16
      NLSETRMS = 14
Call LIN2DB(ETRMS,NLSETRMS,GSTATE(1)) | convert linear to dB (fixed Q9)
                                      | gain offset subtraction is done inside LIN2DB
```

関数LIN2DBの固定小数点コードで必要とする定数のリストを以下に示す。

c(0)	=	-3400	=	-0.4150374993 in Q13
c(1)	=	15758	=	1.9235933879 in Q13
c(2)	=	-10505	=	-1.2823955919 in Q13
c(3)	=	9338	=	1.1399071928 in Q13
c(4)	=	-9338	=	-1.1399071928 in Q13
c(5)	=	9961	=	1.2159010057 in Q13
FAC1	=	24660	=	3.0102999566 in Q13
GOFF	=	-16384	=	-32 in Q9
THRQTR	=	24576	=	.75 in Q15

LIN2DB固定小数点コードを以下に示す。これは、0.75近傍で関数$\log_2$をテイラ級数展開し、高次項を切り捨てたものを計算するものである。入力Xは、Qフォーマットとして与えられた変数NLSXで正規化された正の16ビット仮数部とする。デシベル値を得るための必要なスケーリングを実行し、32ビットの結果から利得オフセットGOFFの減算を行った後、固定QフォーマットQ9で表現された出力を得る。

```
LIN2DB(X, NLSX, RET)
AA0 = X                                  | normalized .5 <= X <= 1, Q15
AA0 = AA0 - THRQTR                       | subtract .75, now -.25 <= X <= .25, Q15
AA0 = AA0 << 1                           | convert to Q16
AA1 = 0
For I=5,4...,1 do the next 3 lines
    AA1 = AA1 + C(I)                     | Q13
    P = AA1  *  AA0                      | Q13  *  Q16 = Q29
    AA1 = P >> 16                        | Q29 -> Q13
AA1 = AA1 + C(0)                         | Q13
AA1 = AA1 >> 3                           | convert to Q10
AA0 = 15 - NLSX                          | exponent in Q0
AA0 = AA0 << 10                          | convert exponent to Q10
AA1 = AA0 + AA1                          | Add exp to log of mantissa, AA1 now log2(X)
AA1 = AA1  *  FAC1                       | Q10  *  Q13 = Q23, convert to 10 * log10(X)
AA1 = AA1 >> 14                          | Q23 -> Q9
AA1 = AA1 - GOFF                         | Subtract GOFF
RET = AA1                                | return Q9 value
```

３．１０　ブロック９８ＡＦ－フレーム消失後の対数利得リミッタ

この新しいブロックは、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇの３．１６節ブロック９８と置き換えれば良い。以下にブロック９８ＡＦの固定小数点擬似コードを示す。

```
If LOGGAIN > 14336, set LOGGAIN=14336
If LOGGAIN < -16384, set LOGGAIN=-16384
TMP = LOGGAIN - OGAINDB
If AFTERFE > 0 and TMP > FEGAINMAX, do the next line
    LOGGAIN = OGAINDB + FEGAINMAX
```

４．追加された復号器のパラメータと変数

この章では、フレーム消失補償対応のＴＴＣ標準ＪＴ－Ｇ７２８復号器に新しく追加された変数の一覧表を示す。「Q format」と記した欄には、各変数の固定小数点フォーマットを載せている。浮動小数点で実現する場合は、すべてのブロック浮動小数点、または固定Qフォーマットで表現された変数を、浮動小数点表現に変換して用いる。

Table Ⅰ－４－１／JT-G728　Additional Coder Parameters Needed for Frame Erasure Concealment (ITU-T G.728)

Name	Floating-Point Value	Fixed-Point Value	Q format	Description
AFTERFEMAX	16	16	Q0	Maximum 2.5ms frames for gain clipping
DIMINV	0.2	13107	Q16	Inverse of vector dimension
FEGAINMAX	2	1024	Q9	Maximum gain dB increase after frame erasures
VTH	0.4285714	7022	Q14	Voicing threshold for frame erasures
FESCALE	0.8	26214	Q15	ETPAST scaling factor for frame erasures

Table Ⅰ－４－２／JT-G728　Additional LD-CELP Internal Processing Variables Needed for Frame Erasure Concealment (ITU-T G.728)

Name	Array Index Range	Fixed-Point Format	Description
MAG	1	*	Average magnitude of current excitation vector
ADCOUNT	1	Integer	JT-G728 Adaptation cycle counter (2.5ms)
AFTERFE	1	Integer	Counter for gain clipping after FE (2.5 ms)
AVMAG	1	SFL	Average magnitude of past 8 excitation vectors
ETPAST	-139 to IDIM	Q2	Past excitation (initially zero)
ETRMS	1	SFL	Energy in dB of current vector
FECOUNT	1	Integer	Length of current frame erasure (2.5 ms)
FEDELAY	1	Integer	Pitch (KP) stored at beginning of FE
FERROR	1	Logical	Frame erasure indicator
FESCALE	1	Integer	ETPAST scaling factor for frame erasures
FESIZE	1	Integer	Number of 2.5 ms Adaptation cycles in a FE
NLSFESCALE	1	Integer	NLS for FESCALE
NLSAVMAG	1	Integer	NLS for AVGMAG
NLSETRMS	1	Integer	NLS for ETRMS
OGAINDB	1	Q9	Old prediction gain in dB
VOICED	1	Logical	Voiced/Unvoiced flag

* MAGは固定小数点擬似コードでは使用されない。

付加資料Ｉ

（ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｉに対する）

フレーム消失時の変数ETPASTのスケーリングに使用される値

以下の表は、フレーム消失区間において変数ETPASTをスケーリングするために用いられる値を、固定小数点及び浮動小数点表現で記述したものである。フレーム消失区間が有声信号である場合に用いられる値(VOICEDFEGAIN)、及び無声信号である場合に用いられる値(UNVOICEDFEGAIN)の両者について示す。固定小数点値についてはQ15フォーマットで記述してある。

付表Ｉ－１／JT-G728　Values used to scale ETPAST with voiced signals (VOICEDFEGAIN) (ITU-T G.728)

VOICEDFEGAIN index	Floating-point Value	Fixed-point Value Q15
0	.8	26214
1	.8	26214
2	.6	19661
3	.4	13107
4	.2	6554

付表Ｉ－２／JT-G728　Values used to scale ETPAST with non-voiced signals (UNVOICEDFEGAIN) (ITU-T G.728)

UNVOICEDFEGAIN index	Floating-point Value	Fixed-point Value Q15
0	1.0	32767
1	1.0	32767
2	.8	26214
3	.6	19661
4	.4	13107
5	.2	6554

フレーム消失時のＬＰＣ予測器の帯域幅拡張に使用される値

以下に示す表は、フレーム消失時のＬＰＣ予測器を平滑化するために使用される、帯域幅拡張係数列FACVFEの値を整数表現したものである。以下の値はQ14フォーマットで記述されている。浮動小数点表現された値を求めるには、整数表現された以下の値を16384で割ればよい。以下の表は、始点がFACVFE(1)、終点がFACVFE(51)である数列の形で表現されている。表は、左から右へ、右端の値から次の行の左端の値へという順に読む。

付表Ｉ－３／JT-G728
(ITU-T G.728)

16384	15892	15416	14953	14505
14069	13647	13238	12841	12456
12082	11719	11368	11027	10696
10375	10064	9762	9469	9185
8910	8642	8383	8131	7888
7651	7421	7199	6983	6773
6570	6373	6182	5996	5816
5642	5473	5309	5149	4995
4845	4700	4559	4422	4289
4161	4036	3915	3797	3683
3573				

付属資料Ｊ
（標準ＪＴ－Ｇ７２８に対する）
ＤＣＭＥにおける音声帯域データへの適用を主な用途としたＬＤ－ＣＥＬＰの可変ビットレート動作

概要

ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｊは、既存のＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇの16kbit/s固定小数点記述に関する音声帯域データ信号に最適化した40kbit/sへの拡張について定義したものである。本付属資料に記述されるコーデックと、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇに記述されるものとの主な違いは、コードブック探索の手法にトレリス符号量子化（ＴＣＱ）を適用していることにある。ＴＣＱ手法は、音声帯域データ（ＶＢＤ）モードに限り、合成による分析（Ａ－ｂ－Ｓ）法に代わってＴＴＣ標準ＪＴ－Ｇ７２８のコードブック探索に用いられる。

ＶＢＤモードにおいて行われる予測器のバックワード適応は、音声モード（ＴＴＣ標準ＪＴ－Ｇ７２８）において行われるバックワード適応とほとんど同一である。さらに、同じ適応周期が音声モード（ＴＴＣ標準ＪＴ－Ｇ７２８）とＶＢＤモードの両方に用いられる。音声モードでは、ＶＢＤモードでの40kbit/sからＴＴＣ標準ＪＴ－Ｇ７２８のＬＤ－ＣＥＬＰに戻る。

参照とする標準

[1] ＴＴＣ標準ＪＴ－Ｇ７２８：　低遅延符号励振線形予測(ＬＤ－ＣＥＬＰ)を用いた16kbit/s音声符号化方式。

[2] ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ：　16kbit/s固定小数点の詳細記述。

[3] ＩＴＵ－Ｔ勧告Ｇ．７６３：　Digital circuit multiplication equipment using ADPCM (Recommendation G.726) and digital speech interpolation

１．はじめに

本付属資料は、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｊ「（ＴＴＣ標準ＪＴ－Ｇ７２８に対する）ＤＣＭＥにおける音声帯域データへの適用を主な用途としたＬＤ－ＣＥＬＰの可変ビットレート動作」の詳細を記述する。

本付属資料では、固定小数点演算デバイスを用いて、コーデックを実現するための情報、およびモードスイッチを有効にするために付属資料Ｇにおいて必要となる変更事項を提供する。

２．本付属資料の規定範囲

付属資料Ｊで規定される符号化方式は、40kbit/sの伝送レートで動作する。アルゴリズム上の遅延は５サンプル分の長さ（0.625ms）であり、付属資料ＧおよびＴＴＣ標準ＪＴ－Ｇ７２８に記述されているすべてのＬＤ－ＣＥＬＰアルゴリズムと厳密に等しい。付属資料Ｊにおける40kbit/sのＶＢＤアルゴリズムは、ＤＣＭＥのようなアプリケーションでのＶＢＤ信号圧縮伝送が意図されている。このアルゴリズムは、ＬＤ－ＣＥＬＰ（ＴＴＣ標準ＪＴ－Ｇ７２８）アルゴリズムとの間での柔軟な遷移を許容し、なおかつ電話音声としても十分な品質を維持するように設計されている。付属資料Ｊは、ＬＤ－ＣＥＬＰ（ＴＴＣ標準ＪＴ－Ｇ７２８）を組み込んだＤＣＭＥシステムにおいて、40kbit/s ＡＤＰＣＭ（ＴＴＣ標準ＪＴ－Ｇ７２６）モードを置き換えるのがねらいである。

３．概観

本コーデックは40kbit/sの伝送レートを使用する。アルゴリズム上の遅延は５サンプル分の長さで、合計0.625msである。本コーデックは「適応周期」（2.5ms）毎にモードスイッチを作動することができる。

本付属資料に記述されるコーデックと、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇに記述されるものとの主な違いは、コードブック探索の手法にトレリス符号量子化（ＴＣＱ）を適用していることにある。ＴＣＱ手法は、ＶＢＤモードに限り、合成による分析（Ａ－ｂ－Ｓ）法に代わってＴＴＣ標準ＪＴ－Ｇ７２８のコードブック探索に用いられる。

ＶＢＤモードにおいて行われる予測器のバックワード適応は、音声モード（ＴＴＣ標準ＪＴ－Ｇ７２８）において行われるバックワード適応とほとんど同一である。さらに、同じ適応周期が音声モード（ＴＴＣ標準ＪＴ－Ｇ７２８）とＶＢＤモードの両方に用いられる。音声モードでは、ＶＢＤモードでの40kbit/sからＴＴＣ標準ＪＴ－Ｇ７２８のＬＤ－ＣＥＬＰに戻る。

本付属資料の第４章の「アルゴリズム記述」では、本コーデックの動作についての完全な記述について示す。４．３節では、符号器の実現方法の詳細記述を示し、４．４節で復号器に関する詳細を付加する。また４．５節

では、モードスイッチに関する詳細を示す。

４．アルゴリズム記述

４．１　符号器の構成

アルゴリズムは、ＲＥＬＰ構造に基づいており、以下のブロックに分割される。（Figure Ｊ－４－１／JT-G728）

４．１．１　ブロック＃１０　ＴＣＱ探索およびビタビ判定ブロック

このブロックは、トレリス生き残りパスや特定された再生値の管理およびマトリクスの計算や比較等、ＴＣＱ（トレリス符号量子化）アルゴリズムに必要なすべての演算、およびビタビ判定を行う。符号器では、５つのソースサンプルに対する最適な生き残りパスからビタビアルゴリズムによって選択されるものと同様の５つのチャネルシンボルを出力する。

Figure Ｊ－４－２／JT-G728にトレリス線図を生成する状態遷移図を、Figure Ｊ－４－３／JT-G728にトレリス線図を示す。４．６．１、４．６．２、４．６．３、４．６．４節中の表は、以下に述べるように、これらの図から得られる。

４．６．１節では、本トレリスにおける各ノードに対する前ノードへの許容されたパスを示す。例えば、第１ノード（ｓ[0]）に対して許容された前ノードは、ブランチ0（ｂ[0]）の下のノード0とブランチ1（b[1]）の下のノード2である。

４．６．２節では、本トレリスにおける各ノードに対する次ノードへの許容されたパスを示す。例えば、第１ノード（ｓ[0]）に対して許容された次ノードは、ブランチ0（b[0]）の下のノード0とブランチ1（b[1]）の下のノード2である。

４．６．３節では、各トレリスパスに対応する量子化サブセット｛D0,D1,D2,D3｝を示す。例えば、s[0]からs[0]への遷移は、サブセットD0に対応する。s[0]からs[1]への遷移は、サブセットD2に対応する。また、s[0]からs[2]とs[3]への遷移は、許容されていないため、Xと印されている。

４．６．４節では、各遷移をラベル化するインデックスビットを示し、また、各ノードから出る２つのブランチを区別する。例えば、s[0]からs[0]への遷移は、0に対応する。s[0]からs[1]への遷移は、1に対応する。（ビット５が使用されており、Ｃ言語記述においては0x10は10hとなっていることに注意されたい。）また、s[0]からs[2]とs[3]への遷移は、許容されていないため、Xと印されている。

４．１．２　ブロック＃２０　セット拡張スーパーコードブック

スーパーコードブックは、セット拡張されたスカラロイドーマックス量子化器である。64個の出力レベルは、４つのサブセットに分類されており、最負値から始まって最正値に向かい、｛D0,D1,D2,D3，…D0,D1,D2,D3｝といった値に順にラベル化している。量子化レベルは４．６．６節で与えられ、区間の境界は４．６．５節で与えられる。サブセットD0に属するレベルはs[0]の列に示される。D1のレベルはs[1]の列、…,D3のレベルはs[3]の列に示される。

４．１．３　ブロック＃３０　バックワード利得適応器

ＶＢＤモードとＪＴ－Ｇ７２８本体の音声モードとでは、バックワード利得適応器の仕組みはほとんど同じである。主な相違点は以下の通りである。

(1)　ＶＢＤモードでは、コードブック出力値のＲＭＳ値が、生き残りパスによって特定される出力レベル系列（量子化残差）に対して計算される。ＲＭＳは８つのサンプル列に対して計算される。しかしながら、ＶＢＤモードでは、あらかじめ計算した対数ＲＭＳを格納した表を持つ付属資料Ｇの場合とは異なり、ＲＭＳの対数値計算を必要とする。式(1)に対数近似式を示す。係数d_0,d_1,d_2,d_3,d_4は４．７節で与えられ、またその対数計算の詳細は４．３．１７節で与えられる。

$$2*\log_{10}(x) = d_0*(x-1) + d_1*(x-1)^2 + d_2*(x-1)^3 + d_3*(x-1)^4 + d_4*(x-1)^5 \qquad (1)$$

for $1 \le x < 2$

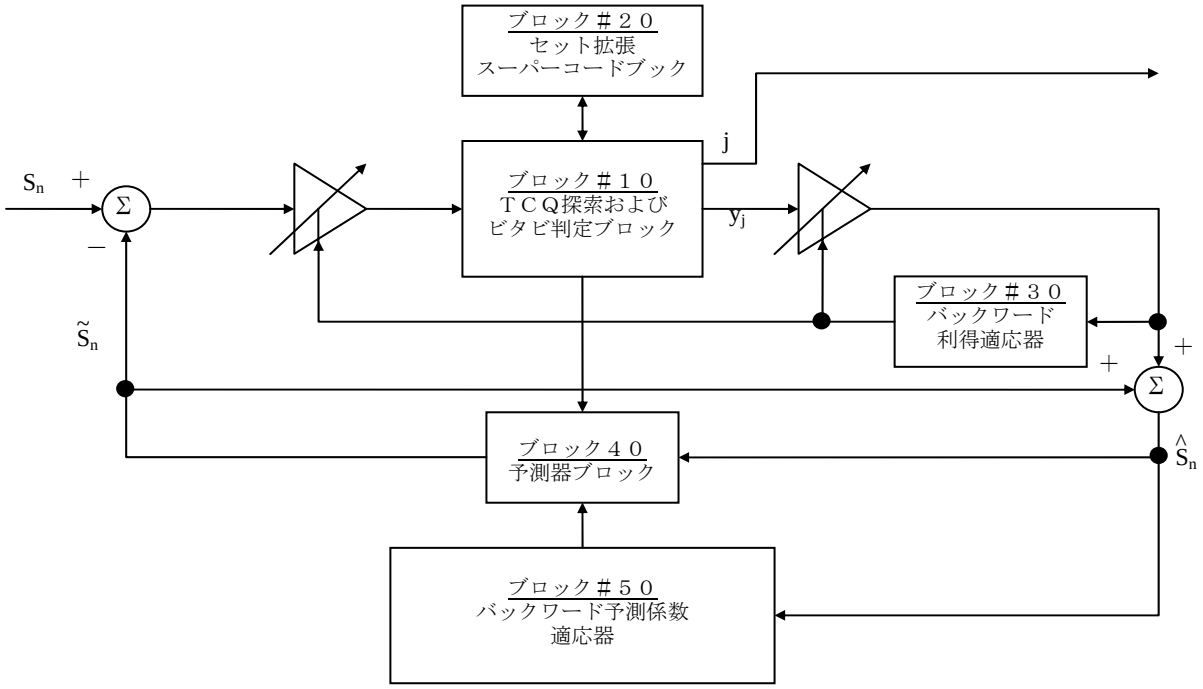


Figure Ｊ－４－１／JT-G728　符号器の構造
(ITU-T G.728)

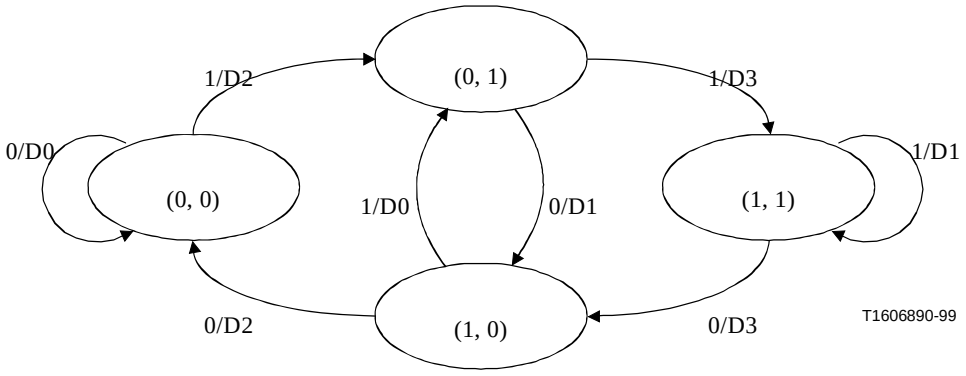


Figure Ｊ－４－２／JT-G728　ＴＣＱ状態遷移図
(ITU-T G.728)

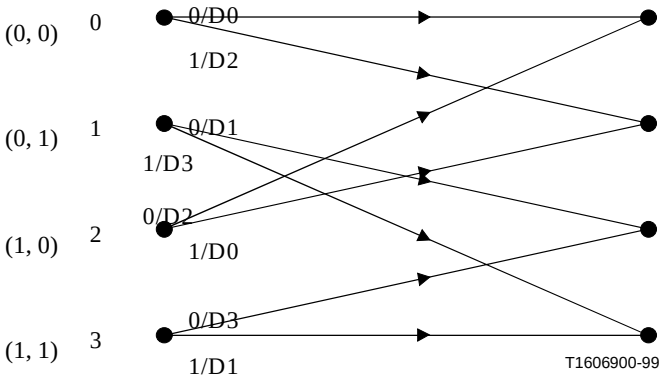


Figure Ｊ－４－３／JT-G728　ＴＣＱトレリス線図
(ITU-T G.728)

対数ＲＭＳ値は、（ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇの式(14)の最後の2つの補正項である）対数利得表ブロック＃Ｇ．９３および＃Ｇ．９４（ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇのブロック９３およびブロック９４）によって与えられる形状コードブックおよび利得コードブックの出力を置き換える。

(2) 音声帯域のデータ波形のような静的な変化をする信号に対して、定常状態の揺らぎを低減するために、対数利得ループの中に平滑化フィルタが組み込まれている。音声、データ信号の両方に対応するために、動的固定化量子化（ＤＬＱ）アルゴリズム(文献5)によって可変速適応を実現する。ＴＴＣ標準ＪＴ－Ｇ７２６にも同等の機能を持ったブロックがある。

ＤＬＱの入力は、オフセットを除いた対数利得$d(n)$である。この入力は、固定化利得G_Lを生成するために、重み付けフィルタ（４．３．１８節、ブロック＃Ｊ．１４）によって平均化される。

量子化器は、もしa_1=0であれば完全に固定化された状態であり、a_1=1であれば完全に固定化されていない状態である。a_1は量子化残差$ET(n)$（４．３．１０節、ブロック＃Ｊ．１２）の長期および短期エネルギーの比較によって算出される。この比較により、量子化残差の変化の安定度を測ることができる。

$$G = G_U * a_l + G_L * (1 - a_l) \tag{2}$$

(3) インパルス的な予測誤りによって量子化器が飽和する可能性がある。これを避けるため、追加した３つのブロック（Figure Ｊ－４－４／JT-G728参照）により量子化利得を時間的に変化させる。

これらのブロックは、以下から成る。

①ブロック＃Ｊ．２５　利得補正判定：

平滑化フィルタにより、最新のベクトル利得値$GSTATE[0]$を用いて、利得見積もりの平均値G_{ave}が計算される（４．３．１２節、ブロック＃Ｊ．２５および式(3)を参照）。$GSTATE[0]$とG_{ave}との差が算出される（G_{diff}）。

G_{diff}が定数閾値と比較される。あらかじめ定められた時間長より長い間G_{diff}が閾値より小さい場合、その信号は「安定」とみなされる。「安定」の信号が続いた後にG_{diff}が閾値を越えた場合、インパルス的な予測誤りとして検出され、このとき利得補正ブロック（４．３．１４節、ブロック＃Ｊ．２９参照）が起動される。

②ブロック＃Ｊ．２６　信号識別器：

ある種のＶＢＤ伝送においては、インパルス的な予測誤りが発生しやすくなる。したがって、インパルス的な予測誤りの検出により、利得補正が最大となる。信号識別器ブロックは、ＬＰ係数を用いて、このような伝送であることを検知する（４．３．１３節、ブロック＃Ｊ．２６参照）。

③ブロック＃Ｊ．２９　利得補正：

このブロックは、定められた時間長の間、利得ファクタを増加させる。（ただし、利得ファクタがあるピーク値に達した場合は、時間長が延長される。）（４．３．１４節、ブロック＃Ｊ．２９参照）

$$G_{ave} = G_{const} * G_{ave} + (1 - G_{const}) * GSTATE[0] \tag{3}$$

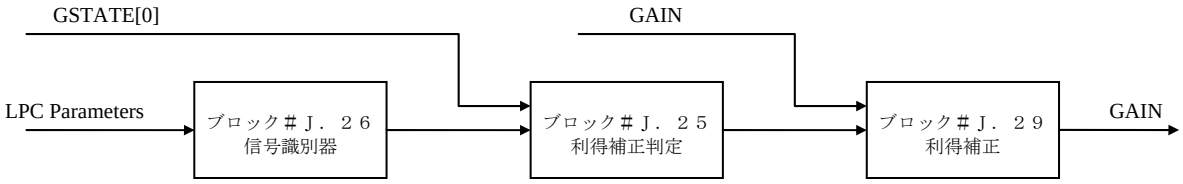


Figure Ｊ－４－４／JT-G728　利得補正
(ITU-T G.728)

４．１．４　ブロック＃４０　予測器ブロック

予測器は、ＴＴＣ標準ＪＴ－Ｇ７２８の合成フィルタ（ブロック＃Ｇ．２２）の次数を短縮した形である。合成フィルタのＬＰＣの次数は、通常の50タップに代わり10タップとなっている。以下のように、予測は、生き残りパス（４．３．４節、ブロック＃Ｊ．７参照）に基づいている。時刻*n*において、現在のサンプルの予測は、時刻*n-1*において選択された生き残りパスによって特定された再生系列を用いて、各ノードについて実行される（４．３．５節、ブロック＃Ｊ．８参照）。この手法では、１ステップのスカラ予測のみが実行され、さらに未来の方向に予測を拡張する必要はない。この手法を用いることにより、他の多くの予測ベクトル量子化手法よりも、予測処理をより「局所的」にすることができる。

４．１．５　ブロック＃５０　バックワード予測係数適応器

バックワード予測係数適応器はバックワード合成フィルタ適応器（ブロック＃Ｇ．２３）とほとんど同一である。主な相違点は以下の通りである。

- 計算されるＬＰＣパラメータは10次のみである。ただし、ハイブリッド窓かけモジュール（ブロック＃Ｇ．４９）では、「データ」から「音声」に状態遷移する際の音声品質の向上を図るため、自己相関関数の計算は、状態に関わらず51次まで行う。
- 合成フィルタの帯域幅拡張係数には、240/256を用いる。各次の帯域幅拡張係数は４．８節で与えられる。

４．２　復号器の構成

Figure Ｊ－４－５／JT-G728に復号器のブロック図を示す。また、Figure Ｊ－４－６／JT-G728には、ブロック＃６０の信号マッピングモジュールの構成を示す。復号器に受信された、サンプル当たり５ビットのビット列は、２種類のビット列が結合されたものである。ブロック＃６０の信号マッピングモジュールは、この結合されたビット列を、２種類のビット列に分離する。サンプル当たり１ビットのビット列j_{4n}は、畳込み符号器を用いて２ビットに拡張される。畳込み符号器からの出力はトレリスパスを特定し、正しいサブセットを選択する。サブセットの選択は４．６．２節及び４．６．３節に記述される表に従う。選択手順の詳細は４．４．２節に示す。残りのサンプル当たり４ビットのビット列j_{0n}～j_{3n}は、選択されたサブセットにおける出力レベルを選択するのに用いられる。出力レベルはバックワード利得適応処理で得られた利得でスケーリングされる。次に、利得でスケーリングされた出力レベルは、ブロック＃４０の予測器に入力され、再生信号を生成する（Figure Ｊ－４－５／JT-G728を参照）。

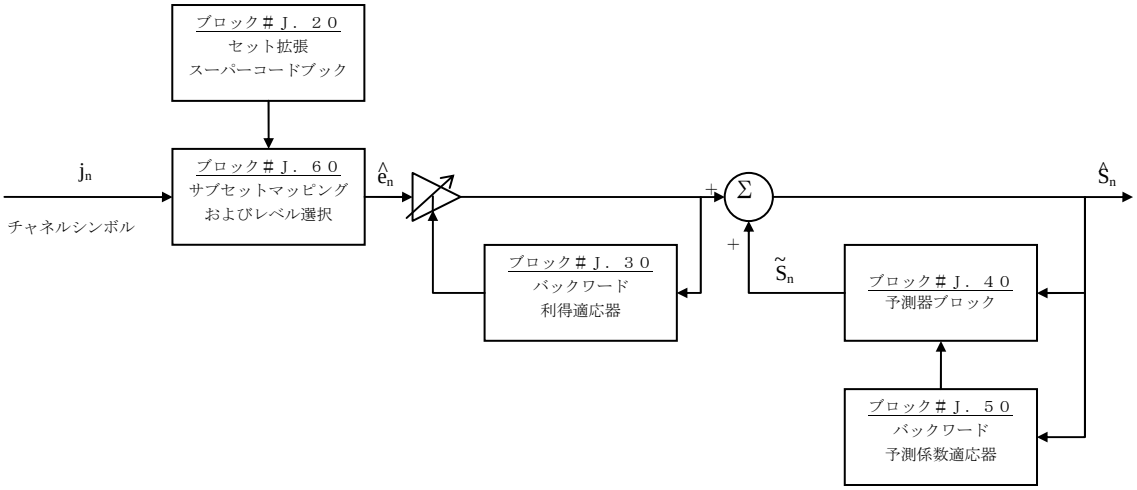


Figure Ｊ－４－５／JT-G728　復号器の構成
(ITU-T G.728)

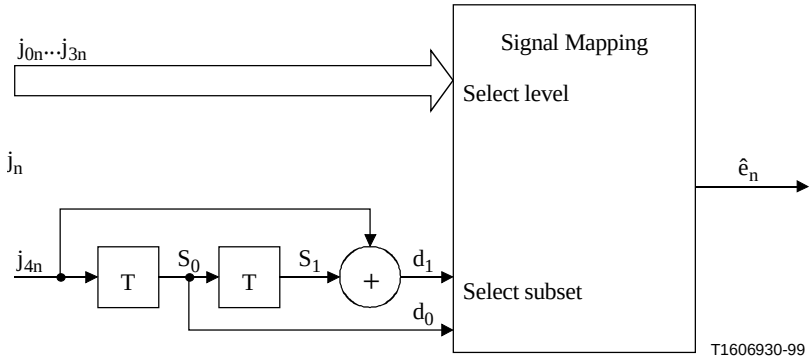


Figure Ｊ－４－６／JT-G728　マッピングモジュール（ブロック＃６０）
(ITU-T G.728)

４．３　符号器の詳細

この節では、符号器の内部ブロックの詳細を示す。この詳細は、さまざまな符号化レートのＴＴＣ標準ＪＴ－Ｇ７２８の動作の概略を示すFigure Ｊ－４－７／JT-G728から始め、上流から下流へと示す。ブロック＃Ｊ．１およびブロック＃Ｊ．２の２つのブロックは、図３－１／JT-G728から得られる。付属資料Hおよび付属資料Gのラベル付けがされている２つのブロックは、ＴＴＣ標準ＪＴ－Ｇ７２８の２つの付属資料（固定小数点記述および可変ビットレートの動作）を示す。ブロック＃Ｊ．１は、音声モードから音声帯域データモードへ、および音声帯域データモードから音声モードへ、といったモード切替スイッチを示す。ブロック＃Ｊ．２は音声帯域データモードを示す。

この詳細では、いくつかのＣ言語の命令（例えば、&，＊，>>，<<）を用いている。読者がこれらの命令の定義に精通していることを仮定している。（命令“OR”は、Ｃ言語におけるビット毎の論理和演算“|”を置き換えることに注意されたい。）

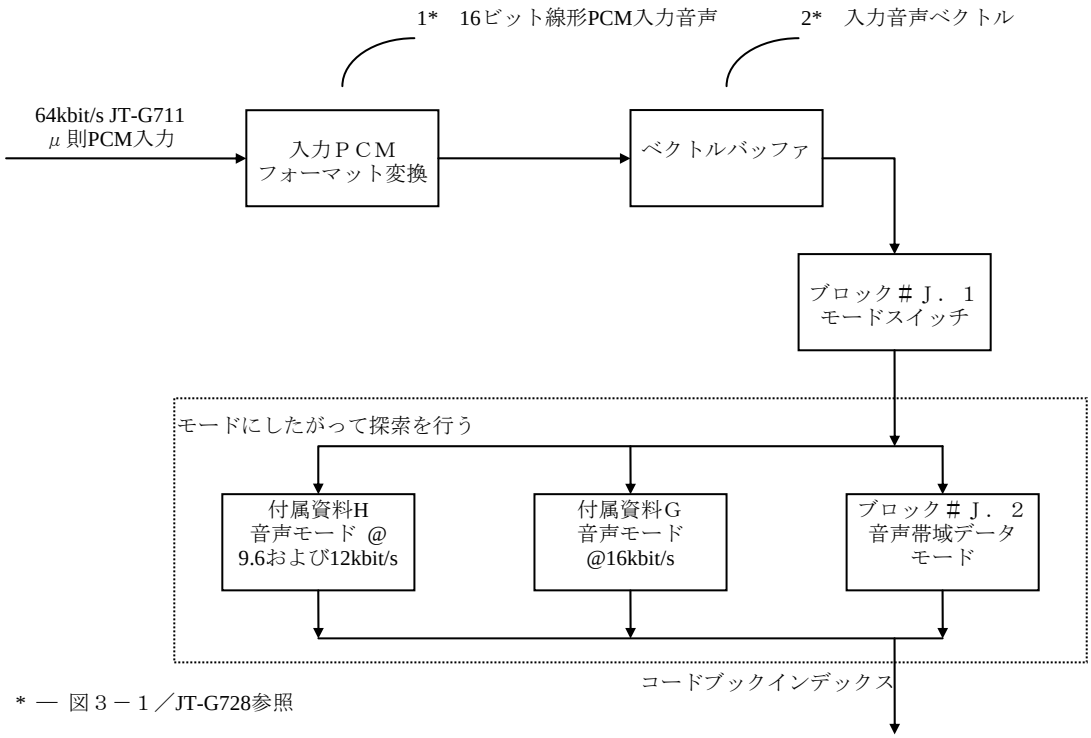


Figure Ｊ－４－７／JT-G728＊　ＴＴＣ標準ＪＴ－Ｇ７２８のさまざまなモードの概略図
(ITU-T G.728)

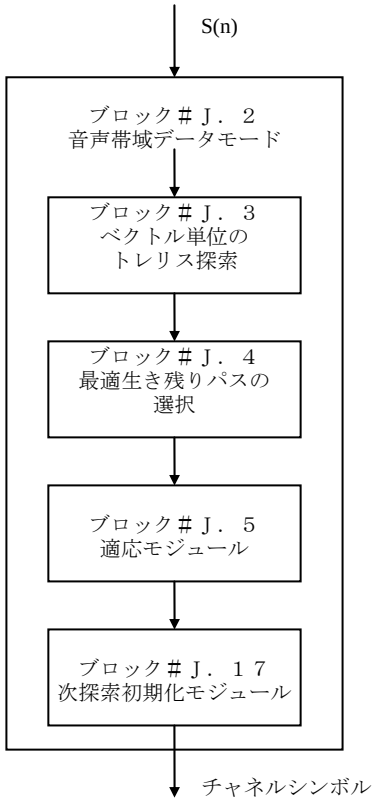


Figure Ｊ－４－８／JT-G728　音声帯域データモード（ブロック＃Ｊ．２）
(ITU-T G.728)

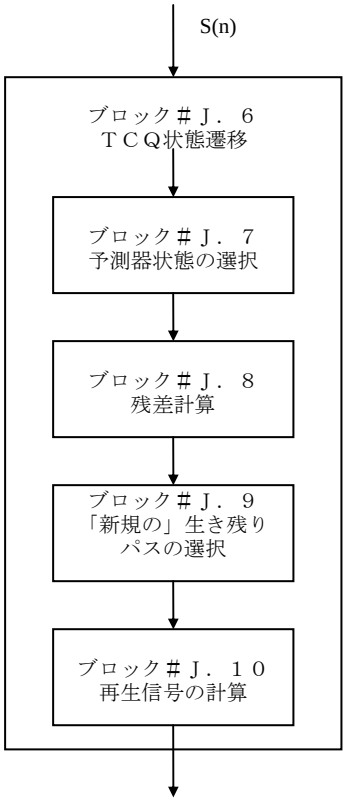


Figure Ｊ－４－９／JT-G728　ＴＣＱ状態遷移（ブロック＃Ｊ．６）
(ITU-T G.728)

４．３．１　ブロック＃Ｊ．２　音声帯域データモード

入力：S(n)

出力：TCQ_channel_symbols

動作：入力ベクトルの符号化。

```
| For every Vector (5 samples) perform the following blocks.

CALL BLOCK #J.3   | Trellis search per vector module.
CALL BLOCK #J.4   | Select Best Survivor module.
CALL BLOCK #J.5   | Adaptation module.
CALL BLOCK #J.17 | Initialize next search module.
```

４．３．２　ブロック＃Ｊ．３　ベクトル単位のトレリス探索

入力：block_depth, next_stage

出力：See output of blocks #J.8, #J.9, #J.10

動作：トレリス線図によるサンプル単位の処理。

```
FOR I = 1, 2, ..., IDIM | Do the next line
      CALL BLOCK #J.6 | TCQ transition.

block_depth = next_stage
next_stage = next_stage+1
IF (BLOCK_LEN <= next_stage)
      next_stage = 0
```

４．３．３　ブロック＃Ｊ．６　ＴＣＱ状態遷移

入力：S(n), block_depth, next_stage
出力：See output of blocks #J.8, #J.9, #J.10
動作：トレリス線図による単ステップ。

```
| For every sample perform the following blocks.

CALL BLOCK #J.7 | Select Predictor state.
CALL BLOCK #J.8 | Calculate residuals.
CALL BLOCK #J.9 | Select "new" survivor.
CALL BLOCK #J.10 | Calculate reconstructed Signal.
```

４．３．４　ブロック＃Ｊ．７　予測器状態の選択

入力：block_depth, TCQ_reconstructed_q2
出力：TCQ_predictor_state_q2
動作：（Figure Ｊ－４－３／JT-G728の）ノード0, 1, 2, 3それぞれについて、再生レベルのトレリスパスを前にたどることにより、予測器の状態を形成する。

注：予測フィルタ状態は、２つの手順によって構成される。
手順１：（block_depth+1, block_depth+2, ..., +10までのインデックスから得られる）直前に選択された再生レベルは、直前の探索サイクルにおいて選択されたものであるが、これはすべてのトレリス状態（ノード）において有効であり、サンプル毎に一回計算される。
手順２：それぞれのノードは、（1, 2, ..., block_depthまでのインデックスから得られる）現在の再生レベル手順と対応しており、再生レベルトレリスにおいて現在が形成されている。

このモジュールは、それぞれのノードの予測器状態の手順２を形成する。

```
FOR d = 0, 1, ..., (N_STATES - 1) | Do next 6 lines.
    IF (0 < block_depth)
        I=0
        src = d
        FOR k = block_depth, block_depth - 1, ..., 1 | Do the next three lines.
            TCQ_predictor_state_q2[d][i] = TCQ_reconstructed_q2 [k][src]
            src = prev_node [k] [src]
            i=i+1
```

４．３．５　ブロック＃Ｊ．８　残差計算

入力：block_depth, A, TCQ_STTMP_q2, DLQ_inv_GAIN, DLQ_inv_nls_m,sample
出力：TCQ_predict_sample_q2, TCQ_resid
動作：（Figure Ｊ－４－３／JT-G728の）ノード0, 1, 2, 3それぞれについて、対応する予測値および残差を求める。

注：現在の状態ノードは、それぞれ１つの予測に対応しており、一方、次状態ノードは、２つ（入力ブランチ毎に１つずつ）の予測レベルに対応している。よって、予測を、現在の状態ノードと対応させ（この方が処理時間がより短い）、block_depthによって参照することが望ましい。

```
FOR d = 0, 1, ..., (N_STATES - 1) | Do lines till END #J.8

| Calculate the part of the predicted value that is valid (common)
| for all trellis states (nodes). See note of section 4.3.4.

    IF (d == 0) | Do next FOR. Calculate only once, during 1st node.
```

```
            TCQ_common_part_pred_q2 = 0
            FOR i = 0, 1, ... (PREDICTOR_ORDER - block_depth-1)
            TCQ_common_part_pred_q2=
                  TCQ_common_part_pred_q2-TCQ_STTMP_q2 [(NFRSZ - 1) - i]  *  A[i+1+block_depth]
      | End of IF (d == 0).

| FOR each trellis state (node), predict the current sample.

AA0 = TCQ_common_part_pred_q2
FOR i = 0, 1, .., (block_depth - 1) | Do the next line.
      AA0 = AA0 - TCQ_predictor_state_q2[d][i]  *  A[i+1]

AA0 = rnd_int (AA0<<2) | Q2 representation.

TCQ_predict_sample_q2[d] = AA0    | save predicted value.
TCQ_resid[d] = sample - AA0

| Normalize the residuals.

AA0 = TCQ_resid[d]  *  DLQ_inv_GAIN
AA1 = -1 * (DLQ_inv_nls_m-1)

IF (0 <= AA1)
      AA0 = AA0 >> AA1
ELSE
      AA1 = -1 * AA1
      AA0 = AA0 << AA1

| Force saturation.

IF (32767L < AA0)
      AA0 = 32767L

IF (AA0 < -32768L)
      AA0 = -32768L
TCQ_resid [d] = AA0 | Save the normalized residual.

END#J.8:
```

４．３．６　ブロック＃Ｊ．９　「新規の」生き残りパスの検出

入力：block_depth, next_stage, TCQ_resid

出力：distortion_metric, Q_resid, prev_node, TCQ_channel_indices

動作：（Figure Ｊ－４－３／JT-G728の）next_stageトレリス状態（ノード）0, 1, 2, 3それぞれについて、２つの入力ブランチから１つを選択し、選択したブランチをこのトレリス状態（ノード）の「新規の」生き残りパスとして残す。

注：次状態ノードそれぞれは１つの生き残りパス（２つの入力ブランチの１つ）に対応している。よって、「新規の」生き残りパスを、次状態ノードに対応させ、next_stageポインタで参照することが望ましい。

このブロックは、多大な演算量（MIPS）を要する。

| For each trellis state, quantize the associated residuals, using the
| subset that labels the two emanating branches (of Figure J－4－3／JT-G728).

| Perform 8 Quantizations. CALL BLOCK #J.11 8 times.

```
TCQ_quantize_resid ( TCQ_resid [0], 0, &ch_index[0], &qerror[0] )
TCQ_quantize_resid ( TCQ_resid [0], 2, &ch_index[1], &qerror[1] )
TCQ_quantize_resid ( TCQ_resid [1], 1, &ch_index[2], &qerror[2] )
TCQ_quantize_resid ( TCQ_resid [1], 3, &ch_index[3], &qerror[3] )
TCQ_quantize_resid ( TCQ_resid [2], 2, &ch_index[4], &qerror[4] )
TCQ_quantize_resid ( TCQ_resid [2], 0, &ch_index[5], &qerror[5] )
TCQ_quantize_resid ( TCQ_resid [3], 3, &ch_index[6], &qerror[6] )
TCQ_quantize_resid ( TCQ_resid [3], 1, &ch_index[7], &qerror[7] )
```

| Calculate the distortion metric associated with each node.

```
AA0 = (TCQ_resid[0] - qerror[0])  *  (TCQ_resid[0] - qerror[0])
AA0 = AA0 >> 2
temp_metric[0] = distortion_metric[0] + AA0

AA0 = (TCQ_resid[0] - qerror[1])  *  (TCQ_resid [0] - qerror[1])
AA0 = AA0 >> 2
temp_metric[1] = distortion_metric[0] + AA0

AA0 = (TCQ_resid[1] - qerror[2])  *  (TCQ_resid[1] - qerror[2])
AA0 = AA0 >> 2
temp_metric[2] = distortion_metric[1] + AA0

AA0 = (TCQ_resid[1] - qerror[3])  *  (TCQ_resid[1] - qerror[3])
AA0 = AA0 >> 2
temp_metric[3] = distortion_metric[1] + AA0

AA0 = (TCQ_resid[2] - qerror[4])  *  (TCQ_resid[2] - qerror[4])
AA0 = AA0 >> 2
temp_metric[4] = distortion_metric[2] + AA0

AA0 = (TCQ_resid[2] - qerror[5])  *  (TCQ_resid[2] - qerror[5])
AA0 = AA0 >> 2
temp_metric[5] = distortion_metric[2] + AA0

AA0 = (TCQ_resid[3] - qerror[6])  *  (TCQ_resid[3] - qerror[6])
AA0 = AA0 >> 2
temp_metric[6] = distortion_metric[3] + AA0

AA0 = (TCQ_resid[3] - qerror[7])  *  (TCQ_resid[3] - qerror[7])
AA0 = AA0 >> 2
temp_metric[7] = distortion_metric[3] + AA0
```

| Select the new survivor for each node.

| node 0

```
IF (temp_metric[0] < temp_metric[4])
        distortion_metric[0] = temp_metric[0]
        Q_resid[0] = qerror[0]
        prev_node [next_stage][0] = 0
        TCQ_channel_indices [next_stage] [0] = ch_index[0]
ELSE
        distortion_metric[0] = temp_metric[4]
        Q_resid[0] = qerror[4]
        prev_node [next_stage][0] = 2
        TCQ_channel_indices [next_stage] [0] = ch_index[4]

| node 1

IF (temp_metric[1] < temp_metric[5])
        distortion_metric[1] = temp_metric[1]
        Q_resid[1] = qerror[1]
        prev_node [next_stage][1] = 0
        TCQ_channel_indices [next_stage] [1] = ch_index[1] OR 0x10
ELSE
        distortion_metric[1] = temp_metric[5]
        Q_resid[1] = qerror[5]
        prev_node [next_stage][1] = 2
        TCQ_channel_indices [next_stage] [1] = ch_index[5] OR 0x10

| node 2

IF (temp_metric[2] < temp_metric[6])
        distortion_metric[2] = temp_metric[2]
        Q_resid[2] = qerror[2]
        prev_node [next_stage][2] = 1
        TCQ_channel_indices [next_stage] [2] = ch_index[2]
ELSE
        distortion_metric[2] = temp_metric[6]
        Q_resid[2] = qerror[6]
        prev_node [next_stage][2] = 3
        TCQ_channel_indices [next_stage] [2] = ch_index[6]

| node 3

IF (temp_metric[3] < temp_metric[7])
        distortion_metric[3] = temp_metric[3]
        Q_resid[3] = qerror[3]
        prev_node [next_stage][3] = 1
        TCQ_channel_indices [next_stage] [3] = ch_index[3] OR 0x10
ELSE
        distortion_metric[3] = temp_metric[7]
        Q_resid[3] = qerror[7]
        prev_node [next_stage][3] = 3
        TCQ_channel_indices [next_stage] [3] = ch_index[7] OR 0x10
```

４．３．７　ブロック＃Ｊ．１１　ＴＣＱ残差量子化

入力：resid, state

出力：index, qerror

動作：１つのサブセットを用いた１つの残差の量子化。

注：このルーチンは、サンプル毎に8回行われ、それぞれ4ビット16レベルで量子化される。

```
index = 0

while ( (Xk[state][index] < resid) && (index < Q_CELLS) )| Do the next line.
    index = index+1

＊qerror = Yk[state][index] | return quantized residual.
```

４．３．８　ブロック＃Ｊ．１０　再生信号の計算

入力：next_stage DLQ_GAIN, DLQ_nls_p_cb_q_m_18, Q_resid, prev_node

出力：TCQ_reconstructed

動作：（Figure Ｊ－４－３／JT-G728の）ノード0, 1, 2, 3それぞれについて、再生レベルを計算する。

注：再生レベルはnext_stageによって参照される。４．３．６節を参照のこと。

内部変数（Ｃ言語定義）

- int src;
- long int AA0;
- short int nls;

```
FOR d = 0, 1, ..., (N_STATES - 1)
    | For each trellis state (node) presented by the variable d.
    | Do the next blocks till END #J.10

    src = prev_node [next_stage] [d] | find the previous node.

    TCQ_ET [ next_stage ] [ d ] = Q_resid [ d ]

    AA0 = Q_resid[d]  ＊  DLQ_GAIN

    IF (0 <= DLQ_nls_p_cb_q_m_18 )
        AA0 <<= DLQ_nls_p_cb_q_m_18
    ELSE
        AA0 >>= abs (DLQ_nls_p_cb_q_m_18)

    AA0 = AA0 + (TCQ_predict_sample_q2 [src] << 16L)
    TCQ_reconstructed_q2 [next_stage] [d] = AA0 >> 16L

END #J.10:
```

４．３．９　ブロック＃Ｊ．４　最適生き残りパスの選択

入力：distortion_metric, prev_node, TCQ_ET, TCQ_reconstructed_q2, TCQ_channel_indices

出力：TCQ_channel_symbols, ET, TCQ_STTMP_q2

動作：生き残りノードおよび対応するチャネルシンボル系列を選択する。

注：このモジュールはトレリスパスを５ステップ完了した後に行う。５チャネルシンボル（サンプルあたり５ビット）系列を（生き残りノードとして）選択すべきである。

内部変数（C言語定義）

- int src;
- int survivor_node;
- long int min_dist;

```
| shift TCQ_STTMP_q2 Kr samples backward.
FOR i = TCQ_Kr.. (NFRSZ-1)
    TCQ_STTMP_q2 [i - TCQ_Kr] = TCQ_STTMP_q2[i]

| shift ET backward.
FOR i = TCQ_Kr..(RMS_BUF_LEN-1)
    ET [i - TCQ_Kr] = ET [i]

| Select the node with a minimum quantization distortion, and the
| associated sequence of reconstructed samples.

min_dist = MAX_NUMBER
FOR d= 0..(N_STATES-1)
    IF (distortion_metric[d] <= min_dist) | Do the next 2 lines
        min_dist = distortion_metric[d]
        survivor_node= d

| Copy the last Kr samples from the
| survivor sequence into TCQ_STTMP_q2

k = 0 | TCQ_Kr == TCQ_Kd as in Kd/Kr=5/5.
src = survivor_node

FOR i = TCQ_DEPTH..(TCQ_Kd - TCQ_Kr)   | Do the next lines,
| i is decremented.
| Arrange reproductions in ascending order.

    TCQ_STTMP_q2 [i + (NFRSZ-(TCQ_DEPTH+1))] =
        TCQ_reconstructed_q2 [k] [src]
        ET[ i - (TCQ_Kd - TCQ_Kr) + (RMS_BUF_LEN - TCQ_Kr) ] = TCQ_ET [k] [src]
    TCQ_channel_symbols [ i - (TCQ_Kd - TCQ_Kr) ] =
        TCQ_channel_indices [ k ] [ src ]
    src = prev_node [k] [src]

    k = k -1
    IF (k < 0)
        k = TCQ_DEPTH
| End of FOR.

| Estimate the gain for the next 5 samples.

CALL BLOCK #J.12 | TCQ Backward Gain Adapter.
```

４．３．１０　ブロック＃Ｊ．１２　ＴＣＱバックワード利得適応器

入力：next_stage

出力：DLQ_GAIN, DLQ_NLSGAIN

動作：スケーリングされた利得s(n)のベクトル毎に一回の計算。.

注：このブロックは、データモードにおいて、（ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇの）ブロック２０を置き換えるものである。

内部変数（Ｃ言語定義）

- int src;
- int survivor_node;
- long int min_dist;

```
| Calculate the log gain RMS value for the last selected path segment.

CALL BLOCK #J.13 | vbd_log_calc_and_lim97

| Calculate the dms and dml DLQ decision parameters.

dms = (dms << 5) - dms
dml = (dml << 7) - dml
dms += RMS_Q11
dml += RMS_Q11
dms >>= 5
dml >>= 7
diff = labs(dms-dml)

IF (diff >= (dml >> 3L))
| Average Power is variable, non stationary signals such
| as speech. ap -> 2, and unlock the quantizer.

    ap_q11 = (ap_q11 ＊ 15 ) >> 4 | ap = ap ＊ 15.0 /16.0
    ap_q11 += (1 Q11 ) >> 3
ELSE
IF (DLQ_GAIN >> DLQ_NLSGAIN < 10)
| Idle Channel Conditions, ap -> 2, and unlock the quantizer.

    ap_q11 = (ap_q11 ＊ 15 ) >> 4
    ap_q11 += (1 Q11 ) >> 3
ELSE
| Average Power is constant, stationary signals, such as VBD
| ap -> 2, and lock the quantizer.

    ap_q11 = (ap_q11 ＊ 15 ) >> 4

IF (1 Q11 < ap_q11)
    al_q11 = 1 Q11
ELSE
    al_q11 = ap_q11

FOR i = 0..NUPDATE | save GSTATE as in Annex G.
```

```
        GTMP[i] = GSTATE[i]

    | Predict the gain.

    | BLOCK #46, P. 39 of Annex G of ITU-T Recommendation G.728 is the
    | LOG-GAIN LINEAR PREDICTOR.

    CALL BLOCK #G.46 | Log-gain linear prediction.
    CALL BLOCK #G.98 | log gain limiter 98, part of blocks 46 AND 47, P. 39.
    CALL BLOCK #J.14 | log_gain_weighting

    GAIN = after_limiter_98 | GAIN is the averaged log-gain.

    CALL BLOCK #G.99 | add_log_offset, part of blocks 46 AND 47, P. 39.
    CALL BLOCK #G.48 | Inverse logarithmic Calculator.

    DLQ_GAIN = GAIN
    DLQ_NLSGAIN = NLSGAIN
    CALL BLOCK #J.25 | GAIN COMPENSATION DECISION
    CALL BLOCK #J.29 | GAIN COMPENSATION
    CALL BLOCK #J.15 | GAIN_inverse
```

４．３．１１　ブロック＃Ｊ．１３　ＶＢＤモードの対数利得計算器およびリミッタ

入力：ET
出力：GSTATE[0], RMS_Q11
動作：対数利得のＲＭＳ値を算出する。

注：このブロックは、浮動小数点版におけるブロック６７，３９，４０（１ベクトル遅延、ＲＭＳ計算器、対数計算器）、およびＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇの固定小数点版におけるブロック９３，９４，９６，９７を置き換えるものである。

内部変数（Ｃ言語定義）

- long int AA0;
- long int AA1;

```
    AA0 = 0
    FOR i=(RMS_BUF_LEN - 1), (RMS_BUF_LEN - 2),..., 1
        AA0 += ET [i]  *  ET [i]
        AA0 += ET[0]  *  ET[0]

    | Divide by RMS_BUF_LEN, and scale AA0 from Q22 to Q11.
    AA0 = (AA0 + 1 Q13) >> 14

    RMS_Q11 = AA0 | 32 bit operation.

    AA0 = calc_10log10(AA0) | BLOCK #J.16 is implemented as a function.

    AA0 = AA0 + (after_limiter_98 << 2) | 32 bit operation.

    AA0 = AA0 >> 2 | scale to Q9 format.
```

```
| block 97 limit GSTATE to -32dB.

IF (AA0 < -16384)
    AA0 = -16384

| If immediately after transition, than use the GSTATE of the speech mode.
IF (speech_to_vbd_transition == 1)
    speech_to_vbd_transition = 0
ELSE
    GSTATE[0]=AA0
```

４．３．１２　ブロック＃Ｊ．２５　利得補正判定

入力：: GSTATE[0], GC_ATMP_SUM, GC_ATMP1
出力：: GC_FLAG,G_CNT, GC_CNT, G_AVE
動作：：利得の準平均値の算出。狭帯域信号の検出。あらかじめ定めた安定利得区間後の急激な増加を調べる。

内部変数（Ｃ言語定義）

- long int GDIFF;

```
GDIFF=GSTATE[0]-G_AVE;
If GDIFF < G_TRS                        | Do the next 2 lines
        G_AVE=((G_AVE<<G_CONST)-G_AVE+GSTATE[0])>>G_CONST
        G_CNT++
ELSE                                    | Do the next 6 lines
    IF G_CNT>G_LEN                      | Do the next 3 lines
            IF ((GC_ATMP_SUM*ATMP_CONST)>>3)<ABS(GC_ATMP1) | Do the next 2 lines
            GC_FLAG=1                   | GAIN COMPENSATION FLAG
            GC_CNT=GC_LEN
    G_AVE=GSTATE[0]
    G_CNT=0
```

４．３．１３　ブロック＃Ｊ．２６　信号識別器

入力：ATMP
出力：GC_ATMP_SUM, GC_ATMP1
動作：狭帯域信号の検出。

```
GC_ATMP_SUM=0
GC_ATMP1=ATMP[1]
FOR I=2,3,...LPC,           | do the next line
    GC_ATMP_SUM=GC_ATMP_SUM+ABS(ATMP[I])
```

４．３．１４　ブロック＃Ｊ．２９　利得補正

入力：GC_FLAG, DLQ_NLSGAIN,GC_CNT, GC_COMPENSATION, GC_NLS_LIMIT
出力：GC_FLAG
動作：あらかじめ定めた時間区間、固定値によってDLQ_NLSGAINを減衰させる。

```
IF GC_FLAG = 1                          |Do the next 8 lines
    IF DLQ_NLSGAIN>GC_NLS_LIMIT-1       | Do the next 7 lines
        If DLQ_NLSGAIN>GC_NLS_LIMIT     | Do the next line
            GC_CNT=GC_CNT-1
        DLQ_NLSGAIN=DLQ_NLSGAIN-GC_COMPENSATION
```

```
IF DLQ_NLSGAIN < GC_NLS_LIMIT | Do the next line
    DLQ_NLSGAIN=GC_NLS_LIMIT
IF GC_CNT=0   | Do the next line
    GC_FLAG=0
```

４．３．１５　ブロック＃Ｊ．１６　対数計算器

入力：AA0. Q11 format. (Ｃ言語定義：long int AA0)
出力：logarithmic value
動作：対数関数の近似。

注：この近似は、１≦ｘ＜２の範囲で有効である。

内部変数（Ｃ言語定義）

- long int T;
- short int exp;

```
IF (AA0 < 1)
    AA0 = 0 | illegal input number clipping to 0dB.
ELSE
    | Scale the input number to the range 1Q14..2Q14
    exp = 3 | Q3 is the difference Q14 - Q11
WHILE ( AA0 < 1 Q14 ) | Do the next two lines.
    AA0 = (AA0 << 1)
    exp -= 1

WHILE (2 Q14 <= AA0) | Do the next two lines.
    AA0 = (AA0 >> 1L)
    exp += 1

T = AA0 - 1 Q14
T <<= 1 | Translate T from Q14 to Q15
AA0 = log_pol[LOG_POL_ORDER-1]

FOR i = (LOG_POL_ORDER-1), (LOG_POL_ORDER-2).., 1
    AA0 = ((T  *  AA0)  *  2 + (log_pol[i-1] << 16) + 1 Q15) >> 16

AA0 = ( (AA0  *  T)  *  2 + 1 Q15) >> 16

AA0 = AA0 >> 4 | Translate AA0 to Q11.

AA0 = AA0  *  5 | Divide AA0 by 2 and multiply by 10 to get dB units.

| Add the mantisa, and calculate the 10 log dB value of the input

AA0 = AA0 + (log_2  *  exp) >> 2
return(AA0)
```

４．３．１６　ブロック＃Ｊ．１４　対数利得重みづけ

入力：after_limiter_98, GAIN_state, al_q11
出力：after_limiter_98
動作：対数利得の平均を求め、可変適応速度を生成する。

内部変数（Ｃ言語定義）

- long int AA0, AA1; | 32 bit accumulators
- short int locked, unlocked;

AA0 = (long)GAIN_state
AA1 = AA0 << 6L | 63/64 iir.
AA1 = AA1 - AA0

| Add AA1 (Q9 format) to after_limiter_98 and round the result to 16 bit Q9.

AA0 = after_limiter_98
AA1 = AA1 + AA0

AA0 = AA1 >> 6L | 63/64 iir.

locked = AA0
unlocked = after_limiter_98

AA0 = locked ＊ (1 Q11 - al_q11)
AA0 = AA0 + unlocked ＊ al_q11
AA0 >>= 11

after_limiter_98 = AA0
GAIN_state = AA0

４．３．１７　ブロック＃Ｊ．１５　利得逆数計算

入力：: DLQ_GAIN, DLQ_NLSGAIN
出力：: DLQ_inv_GAIN, DLQ_nls_p_cb_q_m_18, DLQ_inv_nls_m
動作：利得の逆数を求める。

内部変数（Ｃ言語定義）

- long int NUM; | constant 16384: 1 in the format Q14
- long int NUMNLS; | constant 14

DLQ_nls_p_cb_q_m_18 = 18 - CODEBOOK_Q - DLQ_NLSGAIN |

| Initialize the numerator for inversion.

NUM　= 16384
NUMNLS = 14

divide (NUM, NUMNLS, DLQ_GAIN, DLQ_NLSGAIN, &DLQ_inv_GAIN, &inv_nls)
| The divide is function Annex G.
DLQ_inv_nls_m = - 2 - inv_nls + CODEBOOK_Q + 1

４．３．１８　ブロック＃Ｊ．５　適応モジュール

動作：適応サイクルを実行する。
注：同一の適応サイクルがデータと音声の双方に用いられ、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇの入力適応フォーマットへの変換が、いくつかの変数についてのみ、それぞれの適応フェーズの前に必要となる。

内部変数（C言語定義）

- long int nls;
- short int min_nls;

```
ICOUNT = ICOUNT + 1
IF (ICOUNT > NUPDATE)
    ICOUNT = 1

IF (ICOUNT == 4)
    FOR k = 0, 1, .., (NUPDATE - 1)
        | Prepare STTMP & NLSSTTMP for autocore routine,-HW_s.

        VSCALE( &TCQ_STTMP_q2[k ＊ IDIM], IDIM,
            IDIM, 12, &STTMP[k ＊ IDIM], &NLSSTTMP[k])
        NLSSTTMP[k] = NLSSTTMP[k] + 2 | Q2 correction.

    CALL BLOCK #G.49 | HW_s

IF (ICOUNT == 2)
    IF (ILLCOND == 0)
        durbin (RTMP, ATMP, 10) | BLOCK #G.50
        IF (DurbinFaultFlag == 0)
            CALL BLOCK #J.26 | signal classifier
            CALL BLOCK #J.51 | bandexpand51_vbd()
        ELSE
            DurbinFaultFlag = 0
            FOR I=1, 2,..LPC
                ATMP[i]=A[i]

IF (ICOUNT == 1)
    CALL BLOCK #G.43 | HW_gain
    IF ( ILLCOND == 0 )
        durbin(R, GPTMP, LPCLG) | BLOCK #G.44
    IF (DurbinFaultFlag == 0)
        CALL BLOCK #G.45 | bandexpand45
    ELSE
        DurbinFaultFlag = 0

IF (ICOUNT == 3)
    FOR i = 1, 2.., PREDICTOR_ORDER
        A[i] = ATMP[i]
```

４．３．１９　ブロック＃Ｊ．５１　帯域幅拡張モジュール

入力：ブロック＃Ｇ．５１を参照のこと。

出力：ブロック＃Ｇ．５１を参照のこと。

動作：ブロック＃Ｇ．５１を参照のこと。

注：このブロックは、配列FACVを10個の非ゼロ要素を持つFACV_vbdに置き換えることを除けば、ブロック＃Ｇ．５１と同一である。

４．３．２０　ブロック＃Ｊ．１７　次探索初期化モジュール

入力：survivor_node
出力：distortion_metric
動作：新しい探索のためのメトリックを設定する。

```
FOR i = 0, 1, ..,(N_STATES-1)
    distortion_metric[i] = (MAX_NUMBER >> 2)

| Force the next selected path to pass through the survivor node.
distortion_metric[survivor_node] = 0L

block_depth = 0
next_stage = 1
```

４．４　復号器の詳細

この節では、復号器の詳細を示す。

４．４．１　ブロック＃Ｊ．２０　復号器モジュール

入力：channel_symbol
出力：reconstructed_sample_q2
動作：ベクトル単位の復号処理を実行する。

```
For I = 1, 2, ..., IDIM | Do the next line
    CALL BLOCK #J.21 | Decoder transition module.

CALL BLOCK #J.12 | TCQ Backward Gain Adapter
CALL BLOCK #J.5 | Adaptation Module.
```

４．４．２　ブロック＃Ｊ．２１　復号器状態遷移モジュール

入力：channel_symbol
出力：reconstructed_sample_q2
動作：サンプル単位の復号処理を実行する。

注：予測は、ノード0の符号器の予測値に対して行われる。

内部変数（Ｃ言語定義）

- short int next_state_label;
- short int level_selector;
- short int subset;
- short int next_state;
- short int codebook_level;
- long int AA0;

```
TCQ_predict_sample_q2[0] = 0 | Use the Encoder's node 0 space.

FOR i = 0, 1,..,(PREDICTOR_ORDER-1)
    TCQ_predict_sample_q2[0] -=
        TCQ_STTMP_q2 [(NFRSZ - 1) - i]  *  A[i+1]


FOR i = 0, 1,..,(NFRSZ-1)
```

```
        TCQ_STTMP_q2[i] = TCQ_STTMP_q2[i + 1]

    TCQ_predict_sample_q2[0] =
        rnd_int ( TCQ_predict_sample_q2[0] << 2)

    | Separate the index into 2 sets the next state label and
    | the level selector.

    next_state_label = (channel_symbol >> BITS_PER_LEVEL)
    level_selector = (channel_symbol & LEVEL_MASK)
    next_state = TCQ_next_state [TCQ_decoder_state] [next_state_label]
    subset = TCQ_trans_from_src_to_dst [TCQ_decoder_state] [next_state]
    TCQ_decoder_state = next_state

    codebook_level = Yk[subset][level_selector] | Get the codebook level

    FOR i = 1, 2, ..,(RMS_BUF_LEN - 1)
        ET[i-1] = ET[i]

    ET[RMS_BUF_LEN - 1] = codebook_level

    AA0 = codebook_level  *  DLQ_GAIN

    IF (0 <= DLQ_nls_p_cb_q_m_18)
        AA0 <<= DLQ_nls_p_cb_q_m_18
    ELSE
        AA0 >>= abs (DLQ_nls_p_cb_q_m_18)

    AA0 += (TCQ_predict_sample_q2[0] << 16L)
    TCQ_STTMP_q2 [(NFRSZ-1)] = AA0 >> 16L
    reconstructed_sample_q2 = AA0 >> 16L
```

４．５　モードスイッチモジュールの詳細

この節では、モードスイッチモジュールの詳細を示す。

４．５．１　ブロック＃Ｊ．１８　音声ーデータ遷移モジュール

入力：NLSSTATE, STATELPC, GAIN, NLSGAIN, IAQ_for_VBD, ATMP_for_VBD
出力：TCQ_STTMP_q2, DLQ_GAIN, DLQ_NLSGAIN, IAQ_for_VBD, ATMP_for_VBD
動作：音声モードからデータモードへの遷移を行う。

```
    | Translate 20 elements in SBFL format of STATELPC and NLSSTATE
    | to the Q2 format of TCQ_STTMP_q2.
    | Note that the opposite ordering of STATELPC, NLSSTATE and
    | TCQ_STTMP_q2.

    I = 0
    FOR J = 0, 1, 2, 3
        FOR L = 0, 1,..,4
            k = NLSSTATE [9] - 2
        IF (k<0)
            k = -k
```

```
            TCQ_STTMP_q2[NFRSZ - 1 - I] = STATELPC [I] << k
        ELSE
            TCQ_STTMP_q2[NFRSZ - 1 - I] = STATELPC [I] >> k
        I = I + 1

    | GAIN VARIABLES.

    speech_to_vbd_transition = 1

    GAIN_state = after_limiter_98

    dms = 0
    dml = 0
    ap_q11 = 0

    DLQ_GAIN = GAIN
    DLQ_NLSGAIN = NLSGAIN

    CALL BLOCK #J.15 | GAIN_inverse()

    | Use the previously saved 10 LPC parameters (saved during last
    | adaptation cycle of speech-mode).

    FOR I = 1,2, .., PREDICTOR_ORDER
        ATMP[i] = ATMP_for_VBD[i]

    IAQ = IAQ_for_VBD
    CALL BLOCK #J.51 | bandexpand51_vbd

    FOR I = 1, 2,.., PREDICTOR_ORDER
        A[i] = ATMP[i]

    GC_FLAG=0;   | Gain Compensation
    G_CNT=0;     | Gain Compensation
```

４．５．２　ブロック＃Ｊ．１９　データー音声遷移モジュール

入力：下を参照のこと。

出力：下を参照のこと。

動作：データモードから音声モードへの遷移を行う。

内部変数（Ｃ言語定義）

- long int temp[NFRSZ];

```
    | Reinitialize the perceptual weighting internal variables.
    | (Only in the Encoder).

    AWP[0]=16384
    FOR I=1, 2, ..,LPCW
        AWP[i]=0
    AWZ[0]=16384
    FOR I = 1, 2, ..,LPCW
```

```
        AWZ[i]=0
FOR I = 0, 1, 2, ..(NFRSZ-1)
        STMP[i]=0
FOR I = 0, 1,..,(N3weight-1)
        SBW[i]=0

FOR I = 0, 1, ..,LPCW
        REXPW[i]=0
NLSREXPW= 31

| Reinitialize the post filter internal variables.
| (Only in the decoder).

ILLCONDP = 1
AP[0] = 16384
FOR I =1, 2, ..,10
        AP[i]=0
AZ[0] = 16384
FOR I =1, 2,..,10
        AZ[i]=0
FOR I = 0,1,..,59
        DEC[i]=0
FOR I = 0, 1,..,239
        D[i]=0
FOR I = 0, 1,..,9
        STLPCI[i]=0
FOR I = 0, 1, ..,9
        STPFFIR[i]=STPFIIR[i]=0
FOR I =0, 1, 2, 3
        LPFFIR[i]=0
        LPFIIR[i]=0
FOR I = 0, 1, .., 244
        SST[i]=0
SCALEFIL= 16384
B=0
GL=16384
GLB = 0
TILTZ=0
APF[0] = 16384
FOR I = 1, 2, ..,10
        APF[i]=0
PF_delay = 100
KP=KP1=50

FOR I =11, 12, .., LPC
        A[i]=0
        ATMP[i]=0

FOR I = 0, 1, ..,(NFRSZ-1)
        temp[i] = TCQ_STTMP_q2[(NFRSZ-1) - i]
```

```
FOR I = 0, 1, 2, 3
    VSCALE( &temp[i ＊ 5], 5, 5, 12,
        &STATELPC[i ＊ 5], &NLSSTATE[9-i])
    NLSSTATE[9-i] += 2 | Q2 Correction.

FOR I = NFRSZ, NFRSZ+1, .., (LPC - 1)
    STATELPC [i] = 0
FOR I = 0, 1, .., 5
    NLSSTATE [i] = 16

| Update several LD-CELP internal blocks.
| These blocks are usually updated during the third adaptation phase
| ( ICOUNT == 3), but they are needed from phase 1.

CALL BLOCK #G.12 | Impulse response vector calculation.
CALL BLOCK #G.14 | Shape codevector convolution and
                 | energy calculation.

| GAIN VARIABLES.

CALL BLOCK #J.13 | vbd_log_calc_and_lim97()
vbd_to_speech_transition = 1
```

４．５．３　ブロック＃Ｊ．２２　付属資料Ｇのバックワードベクトル利得適応器に対する変更点

動作：以下のコードは、＃Ｇ．９３，＃Ｇ．９４，＃Ｇ．９６，＃Ｇ．９７を置き換えるものである。

注：データモードから音声モードへの遷移の間、利得および形状コードブックインデックスの１つのインデックス遅延値が利用できない。そのため、GSTATE[0] は遷移中に計算される。

```
IF ( vbd_to_speech_transition == 1)
    vbd_to_speech_transition = 0
ELSE
    | default operation of Annex G.
    | Perform blocks #G.93, #G.94, #G.96 and #G.97.
```

４．５．４　ブロック＃Ｊ．２３　付属資料Ｇのポストフィルタの適用に対する変更点

動作：遷移後の最初の500サンプル（62.5ms）はポストフィルタを用いず、復号信号をその区間の復号器出力とする。62.5ms後ポストフィルタが利用可能となったところで、その出力を復号器の出力として用いる。

```
IF (PF_delay == 0)
    FOR I =0, 1, .., (IDIM-1)
        ST[i]=SPF[i]<<1
ELSE
    PF_delay--
    FOR I = 0, 1, ..,(IDIM-1)
        ST[i]=ST[i]<<(16-NLSST+3)
        ST[i]=rnd_int(ST[i])
```

４．５．５　ブロック＃Ｊ．２４　ＬＰＣパラメータの保存のために必要な付属資料Ｇに対する変更点

動作：途中10次までのＬＰＣパラメータを保存する。このＬＰＣパラメータは音声モードからデータモードへの状態遷移に必要とされる。

注：途中10次までのＬＰＣパラメータはブロック＃Ｇ．２３の実行途中に保存すべきである。（ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇにおけるポストフィルタ係数APFの記述にあるように）ダービンブロックの実行は中断すべきである。係数を保存し、それからダービンブロックを再開すべきである。

```
IF (DurbinFaultFlag == 0)
    FOR I =1, 2,.., 10
        ATMP_for_VBD[I]=ATMP[I];
    IAQ_for_VBD = IAQ;
```

４．６　トレリス状態遷移表

この節では、トレリス状態遷移表を示す。これらの表は、最小帰還フリー畳み込み符号器の実際を示す。

４．６．１　ＴＣＱ前状態

この表は、Figure Ｊ－４－２／JT-G728およびFigure Ｊ－４－３／JT-G728に示すものと同様に、トレリスブランチ単位の、各トレリス状態（ノード）に対する前の（元の）トレリス状態（ノード）を示す。

Ｃ言語定義: int TCQ_prev_state[N_STATES][N_BRANCHES];

TCQ_prev_state		
Trellis State (Node)	**prev_state**	
	b[0]	b[1]
s[0]	0	2
s[1]	2	0
s[2]	1	3
s[3]	3	1

４．６．２　ＴＣＱ次状態

この表は、Figure Ｊ－４－２／JT-G728およびFigure Ｊ－４－３／JT-G728に示すものと同様に、トレリスブランチ単位の、各トレリス状態（ノード）に対する次の（目標となる）トレリス状態（ノード）を示す。.

Ｃ言語定義: int TCQ_next_state[N_STATES][N_BRANCHES];

TCQ_next_state		
Trellis State (Node)	**next_state**	
	b[0]	b[1]
s[0]	0	1
s[1]	2	3
s[2]	0	1
s[3]	2	3

４．６．３　ＴＣＱ状態遷移とコードブックサブセットとの対応

この表は、Figure Ｊ－４－２／JT-G728およびFigure Ｊ－４－３／JT-G728に示すものと同様に、状態遷移に対応するコードブックのサブセットを示す。

Ｃ言語定義: int TCQ_trans_from_src_to_dst [N_STATES][N_STATES];

TCQ_trans_from_src_to_dst				
Trellis State (Node)	**transition label**			
	s[0]	s[1]	s[2]	s[3]
s[0]	0	2	X	X
s[1]	X	X	1	3
s[2]	2	0	X	X
s[3]	X	X	3	1

４．６．４　ＴＣＱ状態遷移とインデックスビットとの対応

この表は、Figure Ｊ－４－２／JT-G728およびFigure Ｊ－４－３／JT-G728に示すものと同様に、復号器のための、状態遷移を識別する伝送インデックスビットを示す。

Ｃ言語定義: int TCQ_index_from_src_to_dst [N_STATES][N_STATES];

TCQ_index_from_src_to_dst				
Trellis State (Node)	**channel index bit**			
	s[0]	s[1]	s[2]	s[3]
s[0]	0	0x10	X	X
s[1]	X	X	0	0x10
s[2]	0	0x10	X	X
s[3]	X	X	0	0x10

４．６．５　量子化器の区間境界 Xk

この表は、スーパーコードブックの区間の境界をＱ１１フォーマットで示す。第１列はセルのインデックスを示す。第２列以降は、それぞれのサブセットにおけるレベルを示す。

Ｃ言語定義: int Xk [N_STATES][Q_CELLS];

Xk				
index	**Codebook Limits**			
	s[0]	s[1]	s[2]	s[3]
0	-9 547	-8 509	-7 779	-7 191
1	-6 690	-6 246	-5 845	-5 477
2	-5 134	-4 812	-4 507	-4 217
3	-3 939	-3 672	-3 414	-3 164
4	-2 921	-2 684	-2 453	-2 226
5	-2 003	-1 783	-1 567	-1 353
6	-1 141	-931	-723	-515
7	-309	-103	103	309
8	515	723	931	1 141
9	1 353	1 567	1 783	2 003
10	2 226	2 453	2 684	2 921
11	3 164	3 414	3 672	3 939
12	4 217	4 507	4 812	5 134
13	5 477	5 845	6 246	6 690
14	7 191	7 779	8 509	9 547
15	32767	32767	32767	32767

４．６．６　量子化レベル Yk

この表は、スーパーコードブックの量子化レベルをＱ１１フォーマットで示す。第１列はセルインデックスを示し、第２列以降はそれぞれのサブセットのレベルを示す。

C言語定義: int Yk [N_STATES][Q_CELLS];

Yk				
index	**Codebook Levels**			
	s[0]	s[1]	s[2]	s[3]
0	-11 502	-9 955	-8 962	-8 209
1	-7 592	-7 062	-6 595	-6 174
2	-5 788	-5 430	-5 095	-4 780
3	-4 480	-4 194	-3 919	-3 655
4	-3 399	-3 151	-2 910	-2 674
5	-2 444	-2 218	-1 996	-1 777
6	-1 562	-1 348	-1 138	-928
7	-721	-514	-308	-103
8	103	308	514	721
9	928	1 138	1 348	1 562
10	1 777	1 996	2 218	2 444
11	2 674	2 910	3 151	3 399
12	3 655	3 919	4 194	4 480
13	4 780	5 095	5 430	5 788
14	6 174	6 595	7 062	7 592
15	8 209	8 962	9 955	11 502

４．７　対数計算多項式の係数

この表は、対数計算器において用いられる多項式の係数を示す。

C言語定義: int log_poly[LOG_POL_ORDER];

log_poly		
Index	**Floating Point presentation**	**Fix Point presentation Q15**
0	0.8678284	28 437
1	-0.4255677	-13 945
2	0.2481384	8 131
3	-0.1155701	-3 787
4	0.0272522	893

４．８　データモードにおける帯域幅拡張係数

この表は、データモードにおける帯域幅拡張係数を示す。（注：ゼロでない係数は１０個のみである。）

C言語定義: int FACV_vbd[LPC];

Table J－4－1／JT-G728　帯域幅拡張係数

FACV_vbd	
index	**Coefficient**
1	15 360
2	14 400
3	13 500
4	12 656
5	11 865
6	11 124
7	10 428
8	9 777
9	9 166
10	8 593
11-50	0

４．９　内部ブロック

Table Ｊ－４－２／JT-G728は、内部ブロックのリストを示す。

Table Ｊ－４－２／JT-G728　内部ブロック

Block ID	Block Name
ブロック＃Ｊ．１	モードスイッチ
ブロック＃Ｊ．２	音声帯域データモード
ブロック＃Ｊ．３	ベクトル単位のトレリス探索
ブロック＃Ｊ．４	最適生き残りパスの選択
ブロック＃Ｊ．５	適応モジュール
ブロック＃Ｊ．６	ＴＣＱ状態遷移
ブロック＃Ｊ．７	予測器状態の選択
ブロック＃Ｊ．８	残差計算
ブロック＃Ｊ．９	「新規の」生き残りパスの検出
ブロック＃Ｊ．１	再生信号の計算
ブロック＃Ｊ．１	ＴＣＱ残差量子化
ブロック＃Ｊ．１	ＴＣＱバックワード利得適応器
ブロック＃Ｊ．１	ＶＢＤモードの対数利得計算器およびリミッタ
ブロック＃Ｊ．１	対数利得重みづけ
ブロック＃Ｊ．１	利得逆数計算
ブロック＃Ｊ．１	$10\log_{10}$の計算（対数計算）
ブロック＃Ｊ．１	次探索初期化モジュール
ブロック＃Ｊ．１	音声－データ遷移モジュール
ブロック＃Ｊ．１	データ－音声遷移モジュール
ブロック＃Ｊ．２	復号器モジュール
ブロック＃Ｊ．２	復号器遷移モジュール
ブロック＃Ｊ．２	付属資料Ｇのバックワードベクトル利得適応器に対する変更点
ブロック＃Ｊ．２	付属資料Ｇのポストフィルタの適用に対する変更点
ブロック＃Ｊ．２	ＬＰＣパラメータの保存のために必要な付属資料Ｇに対する変更点
ブロック＃Ｊ．２	利得補正モジュールの利得平均計算器
ブロック＃Ｊ．２	利得補正モジュールの信号識別器
ブロック＃Ｊ．２	利得補正モジュールのインパルス検出
ブロック＃Ｊ．２	利得補正モジュールの判定ブロック
ブロック＃Ｊ．２	利得補正モジュールの利得補正
ブロック＃Ｊ．５	帯域幅拡張モジュール

４．１０　内部処理における変数および定数

Table Ｊ－４－３／JT-G728およびTable Ｊ－４－４／JT-G728は、内部変数および内部定数のリストおよび詳細を示す。

Table J －４－３／JT-G728　内部処理変数

Name	Array Index Range	Fixed Point Format	Description
after_limiter_98			Q9 input of adder block #G.99
al_q11	1	Q11	Quantizer locking factor
ap_q11	1	Q11	Quantizer locking factor
ATMP_for_VBD	1..11	Q13/Q14/Q15	Temporary buffer for LPC parameters
block_depth	1	Q0	Points at current Trellis step (time n)
ch_index	0..7	Q0	Temporary memory of the selected index
distortion_metric	[0..(N_STATES-1)]	32 bit Q20	Accumulated distortion metric, per trellis state (node)
DLQ_GAIN	1	SFL	Linear Excitation gain, equivalent to GAIN in Annex G
DLQ_inv_GAIN	1	SBFL	Inverted GAIN for VBD
DLQ_inv_nls_m	1	SBFL	NLS for the inverted VBD GAIN
DLQ_nls_p_cb_q_m_18	1	Q0	NLS for linear gain (+ Q format correction offset), equivalent to GAIN in Annex G
dml	1	Q11	Quantized residuals' long term energy
dms	1	Q11	Quantized residuals' short term energy
ET	[0..(RMS_BUF_LEN-1)]		Memory of quantized residuals
FACV_vbd	---	---	See FACV of Annex G
GAIN_state	1	Q9	Weighting filter memory
IAQ_for_VBD	1	Q0	Durbin's recursion precision flag for ATMP
next_stage	1	Q0	Points at the next Trellis step (time n+1)
prev_node	[0..(BLOCK_LEN-1)]＊[0..(N_STATES-1)]	Q0	Trellis memory of previous states (nodes)
Q_resid	[0..(N_STATES-1)]	Q11	Temporary memory of the quantized residuals
qerror	1	Q11	Temporary memory of quantized residuals
RMS_Q11	1	Q11	RMS of ET

sample	1	Q2	Input sample, single element of S
speech_to_vbd_transition	1	Q0	Speech-to-VBD transition flag
survivor_node	1	Q0	Survivor node index
TCQ_channel_indices	[0..(BLOCK_LEN-1)] * [0..(N_STATES-1)]	Q0	Trellis memory of channel indices
TCQ_channel_symbols	[0..(IDIM-1)]	Q0	Memory of compressed signal
TCQ_common_part_pred_q2	1	Q2	Common part of the predicted value
TCQ_ET	[0..(BLOCK_LEN-1)] [0..(N_STATES-1)]		Trellis memory of residuals
TCQ_predict_sample_q2	[0..(N_STATES-1)	Q2	Memory of predicted value for each trellis state (node)
TCQ_predictor_state_q2	[0..(N_STATES-1)][0..(PREDICTOR_ORDER-1)]	Q2	Trellis memory of the predictor state
TCQ_reconstructed_q2	[0..(BLOCK_LEN-1)] [0..(N_STATES-1)]	Q2	Trellis memory of reconstructed levels
TCQ_reconstructed_q2	[0..(BLOCK_LEN-1)] * [0..(N_STATES-1)]	Q2	Trellis memory of reconstructed samples
TCQ_resid	[0..(N_STATES-1)]	Q11	Memory of nodes' (trellis state) residuals
TCQ_STTMP_q2	[0..NFRSZ]	Q2	Buffer for synthesis filter hybrid window
temp_metric	[0..7]	32 bit Q20	Temporary memory of distortion metrics
vbd_to_speech_transition	1	Q0	VBD-to-Speech transition flag
Xk	[0..(N_STATES-1)] [0..(Q_CELLS-1)]	Q11	Quantization intervals limits
Yk	[0..(N_STATES-1)] [0..(Q_CELLS-1)]	Q11	Quantization levels
GDIFF	1	32bitQ9	The difference between the current gain and its average
G_AVE	1	32bitQ9	Gain average
GC_ATMP_SUM	1		Sum of absolute LPC Parameters of block #G.50
GC_ATMP1	1		Second LPC Parameter of block #G.50
GC_NLS_LIMIT	1	Q0	Gain NLS threshold for use in the Gain Compensation block
GC_COMPENSATION	1	Q0	The compensation factor

Table J－4－4／JT-G728　内部処理定数

Name	Value	Symbol	Description
LOG_POL_ORDER	5		Order of logarithmic approximation polynomial
BITS_PER_LEVEL	4		Number of bits, that identify quantization levels
LEVEL_MASK	0x0F		Mask for level bits
PREDICTOR_ORDER	10		Order of data-mode predictor
CODEBOOK_Q	1 Q11		The Q presentation of the codebook
Log_2	24 660		10 * log10(2) in Q13
RMS_BUF_LEN	8		Length of RMS calculation
BLOCK_LEN	5	K_d	Trellis block length
MAX_NUMBER	0x7fffffff		Maximum positive number
N_BRANCHES	2		Number of branches emanating from or incoming to each trellis state
N_STATES	4		Number of Trellis states
Q_CELLS	16		Number of Quantization levels in each subset
TCQ_Kd	5	K_d	Trellis delay length
TCQ_Kr	5	K_r	Trellis release role
TCQ_DEPTH	BLOCK_LEN-1		Trellis block length - 1
G_CONST	5		Used for the calculation of G_{ave} in the Gain Compensation module
ATMP_CONST	3		A threshold for detection of narrow bandwidth signal in the Signal Classifier
G_TRS	1800		Gain Compensation G_{diff} threshold
GC_NLS_LIMIT	7		Gain Compensation limiter
GC_COMPENSATION	3		The value subtracted from the gain NLS when a Gain Compensation occurs
G_LEN	80		The period of time in which the Gain was steady prior to activation of Gain Compensation
GC_LEN	11		The period of time in which the Gain Compensation is active

４．１１　初期値

Table Ｊ－４－５／JT-G728　初期値

NAME	Initial Value
ATMP_for_VBD	16 384, 0, ..., 0
IAQ_for_VBD	14
vbd_to_speech_transition	0
block_depth	0
next_stage	1
TCQ_decoder_state	0
ET	0..0
GAIN	512
NLSGAIN	0
DLQ_GAIN	16 384
DLQ_NLSGAIN	14
DLQ_nls_p_cb_q_m_18	-7
DLQ_inv_GAIN	16 384
DLQ_inv_nls_m	-4
GAIN_state_fx	0L
after_limiter_98	-16 384
TCQ_STTMP_q2	0..0
TCQ_reconstructed_q2	0, ..., 0
distortion_metric	0, (MAX_NUMBER>>2), …, (MAX_NUMBER>>2)
dms	0
dml	0
RMS_Q11	0
ap_q11	0
al_q11	0
GAVE	0
G_CNT	0
GC_CNT	0
GC_FLAG	0

５　参考文献

(1) *Vector quantization and signal compression* - Allen Gersho and Robert M. Gray.

(2) "Trellis-Coded Quantization of Memoryless and Gauss-Markov Sources" *IEEE transactions on communications Vol. 38, No. 1, January 1990* - Michael W. Marcellin and Thomas R. Fischer.

(3) "On the design of an optimal Quantizer" - A.V. Trushkin - *IEEE transactions on information theory Vol. 39, No. 4, July 1993*.

(4) *Digital coding of waveforms* - N.S. Jayant, P. Noll.

(5) "*32 Kbps ADPCM-DLQ coding for network applications*" - David W. Petr - IEEE 1982.

(6) MOC/Israel "Test results for non-voiced performance assessment of LD-CELP algorithm operating at 40 kbit/s mainly for DCME applications", SG 16 COM-D.278, Santiago, Chile, 18-27 May 1999.

付録
（標準ＪＴ－Ｇ７２８に対する）
実現装置の検証

本標準で規定されている浮動小数点アルゴリズムに基づく様々な実現装置の適合性を容易に検証するために、1組の検証ツールが設計され、用意されている。これらの検証ツールは、1組の1.44Mbyte ＤＯＳフォーマットの3.5インチディスケットと関連資料からなり、新日本ＩＴＵ協会から入手できる。

付録１

（標準JT－G728に対する）

ＴＴＣ標準ＪＴ－Ｇ７２８ 16kbit/s ＬＤ－ＣＥＬＰ音声符号化アルゴリズム実現装置
検証用プログラムおよびテストシーケンス

１．１　概要

本付録では、ＴＴＣ標準ＪＴ－Ｇ７２８の実現装置を検証するために用いるディジタルテストシーケンス、および評価用ソフトウエアについて述べる。本規定は、ＴＴＣ標準ＪＴ－Ｇ７２８本体に基づく浮動小数点演算による実現装置、およびＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇに基づく固定小数点演算による実現装置の両方を含む。

１．１．１　浮動小数点演算実現装置の検証原理

ＬＤ－ＣＥＬＰアルゴリズム標準本体は、異なる種類のハードウエア上で容易に実現することを考慮し、ビットイグザクトではない表現で記述されている。つまり、この検証手順が、試験対象となる実現装置が他の参照用実現装置と正確に等しくなるとは仮定できないことになる。したがって、試験対象と参照用実現装置との間の誤差の度合いを規定するために、客観評価法が必要である。評価された誤差が十分に小さいならば、試験対象は、試験に合格した他の実現装置と相互接続性があると見なされる。有限長の試験では、実現装置の各々の局面をすべて試験する事ができないため、実現装置が100％正確であることを保証することはできない。しかしながら、記述された試験手順は、ＬＤ－ＣＥＬＰアルゴリズムの主要な部分をすべて網羅しており、インプリメンタにとって有益なツールとなるであろう。

本付録に記述されている浮動小数点演算用の検証手順は、32ビット浮動小数点演算で実行することを想定して設計されている。この試験手順は、あらゆるＬＤ－ＣＥＬＰ実現装置にも適用できるが、試験の要求条件を満足するためには、おそらく32ビット浮動小数点フォーマットが必須であろう。

１．１．２　固定小数点演算実現装置の検証原理

ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇには、固定小数点ＬＤ－ＣＥＬＰアルゴリズムが、ビットイグザクトな表現で記述されている。このことは、２つの符号化器、または復号器実現装置を、初期化状態から同じ入力信号で動作させた場合、これらの入力信号の処理が継続する限り、同じ瞬間においては、すべての状態変数が正確に一致することを意味している。従って、浮動小数点入力テストシーケンスを用いることは可能であり、その結果としてビットイグザクトな出力シーケンスが得られなくてはならない。

補足の試験用入力信号として、短区間の入力音声が提供されている。また、その入力信号を処理している間に生成される、全ての関連する内部状態変数も提供されている。これは、実現装置における全ての処理が、本標準に記述されている処理に正確に一致することを、インプリメンタが検証するための手段となる。ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇに完全に準拠していると見なすためには、実現装置は、浮動小数点演算試験用入力信号に対して規定された出力信号とも、短区間音声に対する全ての内部状態変数とも、正確に一致する必要がある。

浮動小数点実現装置の検証試験と同様、有限長の試験では、実現装置の各々の局面を全て試験する事ができないため、実現装置が100％正確であることを保証することはできない。しかしながら、記述された試験手順は、ＬＤ－ＣＥＬＰアルゴリズムの主要な部分をすべて網羅しており、インプリメンタにとって有益なツールとなるであろう。

１．２　試験構成

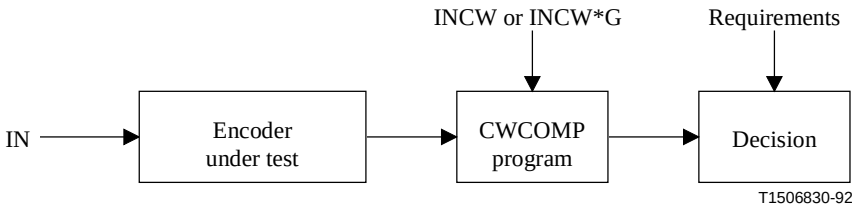
本章では、検証試験を実施する上で用いるさまざまなテストシーケンス、および評価プログラムの組み合わせ使用方法について述べる。試験手順はブラックボックス試験法に基づき、試験対象の符号化器のインタフェースにはSUおよびICAHNを、試験対象の復号器のインタフェースにはICHANおよびSPFを、それぞれ適用する。信号SUおよびSPFは、１．４．２節で詳述するように16ビット固定小数点精度で表現されている。また、試験対象の復号器実現装置には、適応ポストフィルタを無効にできる機能が実装されている必要がある。ＴＴＣ標準ＪＴ－Ｇ７２８に記述されているように、全てのテストシーケンスの処理にあたって、試験対象の実現装置は、初期化リセット状態からスタートさせる必要がある。試験対象の出力シーケンスの評価を行うために、CWCOMP、SNR、およびWSNRの３つの評価プログラムが必要である。これらのプログラムについては、後に第１．３章で詳述す

る。適用する各種試験の構成については、以下の節（１．２．１～１．２．６節）で述べる。

１．２．１　符号化器の試験

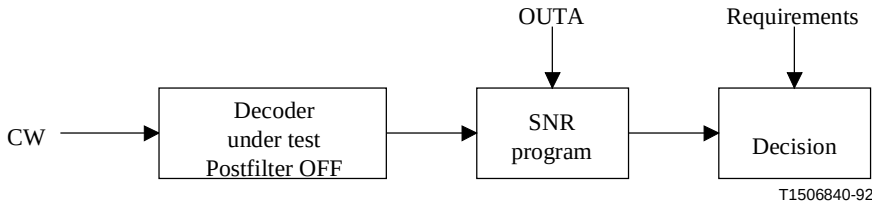
符号化器の基本動作は、付図１－１に示す構成によって試験を行う。入力テストシーケンスINが評価対象の符号化器に適用される。出力される符号語は、CWCOMPプログラムを用いて、参照用符号語であるINCWないしINCW*Gと直接比較される。

付図１－１　試験構成１　符号化器の試験



１．２．２　復号器の試験

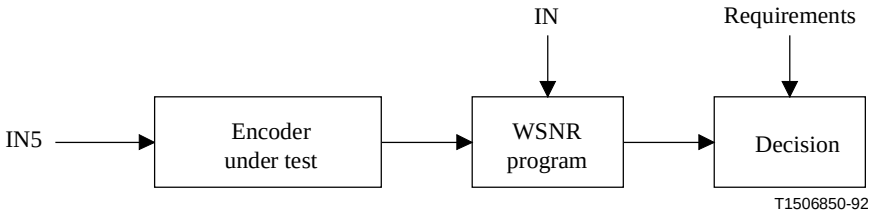
復号器の基本動作は、付図１－２に示す構成によって試験を行う。符号語テストシーケンスCWが、適応ポストフィルタを無効にした試験対象の復号器に適用される。そして、出力信号がSNRプログラムを用いて参照用出力信号OUTAと比較される。



付図１－２　試験構成２　復号器の試験

１．２．３　聴覚重み付けフィルタの試験

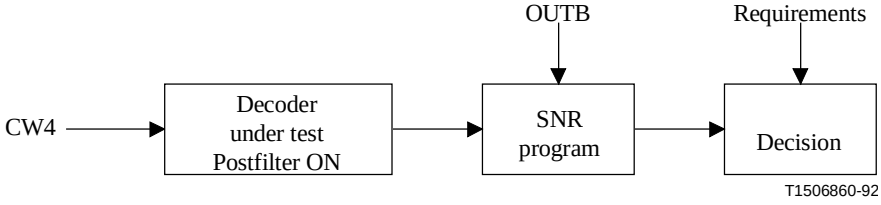
符号化器の聴覚重み付けフィルタは、付図１－３に示す構成によって試験を行う。入力テストシーケンスIN5が試験対象の符号化器に通され、出力される符号語の品質がWSNRプログラムを用いて評価される。また、正確な誤差の評価を行うために、WSNRプログラムも入力シーケンスを必要とする。



付図１－３　試験構成３　符号化器の試験

１．２．４　ポストフィルタの試験

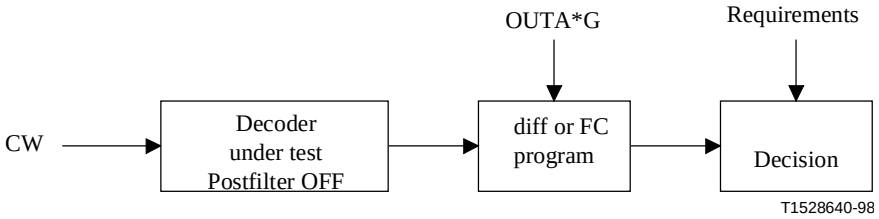
復号器の適応ポストフィルタは、付図１－４の構成によって試験を行う。テストシーケンス符号語CWが、適応ポストフィルタを有効にした試験対象の復号器に適用される。そして出力信号がSNRプログラムを用いて参照用出力信号OUTBと比較される。



付図１－４　試験構成４　復号器の試験

１．２．５　固定小数点復号器の試験

ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇ復号器の基本動作は、付図１－５の構成によって試験を行う。テストシーケンス符号語CWが、適応ポストフィルタを無効にした試験対象の復号器に適用される。そして出力信号が、UNIX[1]のdiffプログラムないしはMS-DOS[1]のFCプログラムを用いて、参照出力信号OUTA*Gと比較される。いかなる違いも検出されてはならない。

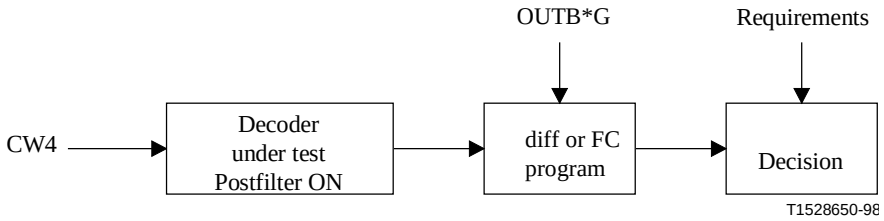


付図１－５　試験構成５　固定小数点復号器の試験

１．２．６　固定小数点ポストフィルタの試験

復号器の固定小数点適応ポストフィルタは、付図１－６の構成にて試験を行う。テストシーケンス符号語CWが、適応ポストフィルタを有効にした試験対象の復号器に適用される。そして出力信号が、UNIXのdiffプログラムないしはMS-DOSのFCプログラムを用いて参照出力信号OUTB*Gと比較される。いかなる違いも検出されてはならない。

付図１－６　試験構成６　固定小数点ポストフィルタの試験



１．２．７　固定小数点内部状態変数

固定小数点の実現装置に対して、補足の試験用入力信号として短区間の入力音声が提供されている。この（符号化器および復号器の両方に対する）入力の処理に対して要求される全ての関連する内部状態変数も、同様に提供されている。この信号は、実現する全ての処理が、仕様に記載されている処理と正確に一致することを検証するために、インプリメンタによって使用される。ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇに完全に準拠していると見なすために、実現装置は、１．２．１、１．２．５、および１．２．６の各節にて記述した浮動小数点テスト入力信号に対して規定された出力と、この短区間音声に対する全ての内部状態変数と、の両方と正確に一致しなければならない。

１．３　検証プログラム

本章では、試験構成の節で言及したCWCOMP、SNR、WSNRの各プログラムについて、また同様に、インプリメンタ用デバッグツールとして提供されるLDCDECプログラムについて記述する。

この検証用ソフトウェアは、Fortranで記述されており、可能な限りANSI Fortran77標準に近い形式にて書かれている。参照用ＬＤ－ＣＥＬＰモジュールとの数値的誤差を最小限にするために、倍精度浮動小数点演算が広く使われている。このプログラムは、386/87ベースのPCに対する実行形式を生成するために、市販のFortranコンパイラを用いてコンパイルされている。配布物に含まれるREAD.MEファイルは、その他のコンピュータにおいて実行形式プログラムを生成する方法を記述している。

[1] Unix は、Unix Systems Laboratories の登録商標である。MS-DOS は、Microsoft Corporation の登録商標である。

１．３．１　ＣＷＣＯＭＰ

CWCOMPプログラムは、二つの符号語ファイルの内容を比較する単純なツールである。ユーザは、参照用符号化器出力（付表１－１の最後列記載のファイル名）および試験対象符号化器出力の、二つの符号語ファイル名入力を指示される。このプログラムは、それらのファイルにおける、それぞれの符号語を比較し、比較結果を端末に出力する。試験構成１に対する要求条件は、いかなる符号語の違いも存在してはならない、である。

１．３．２　ＳＮＲ

SNRプログラムは、二つの信号ファイル間の信号対雑音比の測定を実装している。ひとつ目の信号ファイルは参照用復号器プログラムによって与えられる参照用ファイルであり、ふたつ目の信号ファイルは、試験対象の復号器の出力ファイルである。グローバルSNR（GLOB）は、ファイル全体の信号対雑音比として計算される。セグメンタルSNR（SEG256）は、所定の閾値より大きなリファレンス電力を持つ256サンプルのセグメントすべての平均の信号対雑音比として計算される。最小セグメントSNR群は、同じ閾値より大きな電力を持つ256, 128, 64, 32, 16, 8サンプルの長さのセグメントに対して与えられる。

SNRプログラムを実行するためには、ユーザは二つの入力ファイルの名前を入力する必要がある。ひとつ目のファイルは、付表１－４の最後列に記されている参照用復号器出力ファイルである。ふたつ目のファイルは、試験対象の復号器によって生成された復号出力ファイルである。このプログラムは、この二つのファイルを処理し、上に述べたさまざまなＳＮＲを端末に出力する。試験構成２および４において要求される値は、このSNRの値で与えられる。

１．３．３　ＷＳＮＲ

WSNRアルゴリズムは、符号語列の平均聴覚重み付け歪みを計算するため、参照用復号器、および距離測定器に基づく。対数の信号対雑音比が５サンプルの信号ベクトルごとに計算され、所定の閾値より大きなエネルギを持つ信号ベクトルすべてについての信号対雑音比が平均される。

WSNRプログラムを実行するためには、ユーザは二つの入力ファイルの名前を入力する必要がある。ひとつ目のファイルは、符号化器の入力信号ファイル（付表１－１の一列目）であり、ふたつ目のファイルは、符号化器の出力する符号語ファイルである。WSNRプログラムは、このシーケンスを処理し、出力のWSNRを端末に出力する。試験構成３に要求される値は、このWSNRの値で与えられる。

１．３．４　ＬＤＣＤＥＣ

上の三つの評価プログラムに加え、この配布物は、リファレンスの復号器デモプログラム（LDCDEC）を含んでいる。このプログラムは、WSNRと同一の復号サブルーチンに基づいており、デバッグ用途で復号器内の各値を監視するために変更することができる。ユーザは、入力の符号語ファイル、出力の信号ファイル、および適応ポストフィルタを含めるか否かを入力する。

１．３．５　ｄｉｆｆおよびＦＣ

インプリメンタが、ディスクで配布されるソフトウェアのほかに、UnixあるいはMS-DOSといったオペレーティングシステムが利用可能であると仮定する。diffおよびFCといったコマンドは、二つのファイルを比較し、その二つのファイルが等しいか異なっているかを示すものである。DOSにおいてバイナリのファイル同士を比較する場合の適切なコマンドは「FC /B FILE1 FILE2」である。Unixで二つのファイルを比較する場合のコマンドは「diff FILE1 FILE2」である。

１．４　テストシーケンス

以下は、適用すべきテストシーケンスの詳細である。この詳細は、それぞれのシーケンス特有に要求されるものを含む。

１．４．１　名前付けのルール

テストシーケンスは、信号の種類を識別する接頭語を持ち、順番に番号付けられている。

IN	符号化器入力信号
INCW	浮動小数点符号化器出力符号語

INCW*G	固定小数点符号化器出力符号語
CW	復号器入力符号語
OUTA	復号器出力信号（ポストフィルタなし）
OUTA*G	固定小数点復号器出力信号（ポストフィルタなし）
OUTB	復号器出力信号（ポストフィルタあり）
OUTB*G	固定小数点復号器出力（ポストフィルタあり）

すべてのテストシーケンスは「*.BIN」という拡張子を持つ。

１．４．２　ファイルフォーマット

信号ファイル（接頭辞IN, OUTA, およびOUTBを持つファイル）は、ＬＤ－ＣＥＬＰのインタフェースSUおよびSPFにしたがい、すべて２の補数表現の16ビットバイナリフォーマットであり、付図１－５に示すように、ビット２とビット３の間に固定的にバイナリポイントがあるように解釈しなくてはならない。テストの測定において最大の精度を得るためには16の有効ビットすべてを用いなくてはならないことに注意されたい。

符号語ファイル（ＬＤ－ＣＥＬＰ信号ICHAN。CWおよびINCWの接頭辞を持つファイル）は、信号ファイルと同様に16ビットのバイナリフォーマットで格納される。付図１－７に示すように、16ビット符号語それぞれの下位10ビットが10ビットの符号語を表す。他のビット（ビット12からビット15まで）はゼロにセットされる。

信号ファイルおよび符号語ファイルは、IBM／DOSおよびVAX／VMSのコンピュータで一般に用いられている下位バイトを先に格納するフォーマットで格納される。ほとんどのUNIXマシンのような他のプラットフォームで利用するためには、バイトスワップ命令によってこの順番を変更する必要があるかもしれない。

１．４．３　テストシーケンスおよび要求条件

この副節の付表は、ＬＤ－ＣＥＬＰの浮動小数点実現装置が、仕様にしたがっているか、および他の正しい実現装置と相互接続性があるかどうか、を評価するために実行すべき試験の完全なセットについて示している。付表１－１は、符号化器のテストシーケンスの概要である。関連する要求条件が付表１－２および１－３に示されている。付表１－４、１－５、および１－６は、復号器のテストシーケンスの概略および要求条件を含んでいる。

付表１－１　符号化器試験

Input signal	Length, vectors	Description of test	Test config.	Output signal
IN1	1 536	Test that all 1024 possible codewords are properly implemented	1	INCW1, INCW1G
IN2	1 536	Exercise dynamic range of log-gain autocorrelation function	1	INCW2, INCW2G
IN3	1 024	Exercise dynamic range of decoded signals autocorrelation function	1	INCW3, INCW3G
IN4	10 240	Frequency sweep through typical speech pitch range	1	INCW4, INCW4G
IN5	84 480	Real speech signal with different input levels and microphones	3	– INCW5G
IN6	256	Test encoder limiters	1	INCW6, INCW6G

付表１－２　浮動小数点符号化器試験の要求条件

Input signal	Output signal	Requirement
IN1	INCW1	0 different codewords detected by CWCOMP
IN2	INCW2	0 different codewords detected by CWCOMP
IN3	INCW3	0 different codewords detected by CWCOMP
IN4	INCW4	0 different codewords detected by CWCOMP
IN5	–	WSNR > 20.55 dB
IN6	INCW6	0 different codewords detected by CWCOMP

付表１－３　固定小数点符号化器試験の要求条件

Input signal	Output signal	Requirement
IN1	INCW1G	0 different codewords detected by CWCOMP
IN2	INCW2G	0 different codewords detected by CWCOMP
IN3	INCW3G	0 different codewords detected by CWCOMP
IN4	INCW4G	0 different codewords detected by CWCOMP
IN5	INCW5G	0 different codewords detected by CWCOMP
IN6	INCW6G	0 different codewords detected by CWCOMP

付表１－４　復号器試験

Input signal	Length, vectors	Description of test	Test config.	Output signal
CW1	1 536	Test that all 1024 possible codewords are properly implemented	2, 5	OUTA1, OUTA1G
CW2	1 792	Exercise dynamic range of log-gain autocorrelation function	2, 5	OUTA2, OUTA2G
CW3	1 280	Exercise dynamic range of decoded signals autocorrelation function	2, 5	OUTA3, OUTA3G
CW4	10 240	Test decoder with frequency sweep through typical speech pitch range	2, 5	OUTA4, OUTA4G
CW4	10 240	Test postfilter with frequency sweep through typical speech pitch range	4, 6	OUTB4, OUTB4G
CW5	84 480	Real speech signal with different input levels and microphones	2, 5	OUTA5, OUTA5G
CW6	256	Test decoder limiters	2, 5	OUTA6, OUTA6G

付表１－５　浮動小数点復号器試験の要求条件

Output	Requirements (minimum values for SNR, in dB)								
file name	SEG256	GLOB	MIN256	MIN128	MIN64	MIN32	MIN16	MIN8	MIN4
OUTA1	75.00	74.00	68.00	68.00	67.00	64.00	55.00	50.00	41.00
OUTA2	94.00	85.00	67.00	58.00	55.00	50.00	48.00	44.00	41.00
OUTA3	79.00	76.00	70.00	28.00	29.00	31.00	37.00	29.00	26.00
OUTA4	60.00	58.00	51.00	51.00	49.00	46.00	40.00	35.00	28.00
OUTB4	59.00	57.00	50.00	50.00	49.00	46.00	40.00	34.00	26.00
OUTA5	59.00	61.00	41.00	39.00	39.00	34.00	35.00	30.00	26.00
OUTA6	69.00	67.00	66.00	64.00	63.00	63.00	62.00	61.00	60.00

固定小数点復号器試験の要求条件

diffあるいはFCあるいは同等のファイルを比較するプログラムを実行した場合に、出力テストベクトル（OUTA*GあるいはOUTB4G）と任意の入力テストベクトル（CW*）との間にまったく違いがない。さらに、ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇに完全に準拠すると見なすためには、実現装置は、短区間音声についてすべての内部状態変数を正確に倣わなくてはならない。

１．５　検証ツールの配布物

READ.MEファイルは、ディスク１に含まれており、プログラムをコンパイルおよびリンクするために必要なファイルの内容および手順について述べている。異なるファイルの種類を区別するために拡張子を用いている。*.FORというファイルはFortranプログラムのソースコード、*.EXEは386/87の実行可能ファイル、そして*.BINはバイナリのテストシーケンスファイルである。それぞれのディスクの内容は付表１－６、１－７、１－８、および１－９に記載されている。

付表１－６　配布ディスク＃１

Diskette	Filename	Number of bytes
Diskette #1	READ.ME	10 430
	CWCOMP.FOR	2 642
Total size:	CWCOMP.EXE	25 153
1 289 859 bytes	SNR.FOR	5 536
	SNR.EXE	36 524
	WSNR.FOR	3 554
	WSNR.EXE	103 892
	LDCDEC.FOR	3 016
	LDCDEC.EXE	101 080
	LDCSUB.FOR	37 932
	FILSUB.FOR	1 740
	DSTRUCT.FOR	2 968
	IN1.BIN	15 360
	IN2.BIN	15 360
	IN3.BIN	10 240
	IN5.BIN	844 800
	IN6.BIN	2 560
	INCW1.BIN	3 072
	INCW2.BIN	3 072
	INCW3.BIN	2 048
	INCW6.BIN	512
	CW1.BIN	3 072
	CW2.BIN	3 584
	CW3.BIN	2 560
	CW6.BIN	512
	OUTA1.BIN	15 360
	OUTA2.BIN	17 920
	OUTA3.BIN	12 800
	OUTA6.BIN	2 560

付表１－７　配布ディスク＃２

Diskette	Filename	Number of bytes
Diskette #2	IN4.BIN	102 400
	INCW4.BIN	20 480
Total size:	CW4.BIN	20 480
1 361 920 bytes	CW5.BIN	168 960
	OUTA4.BIN	102 400
	OUTB4.BIN	102 400
	OUTA5.BIN	844 800

付表１－８　配布ディスク＃３

Diskette	Filename	Number of bytes
Diskette #3	INCW1G.BIN	3 072
	INCW2G.BIN	3 072
Total size:	INCW3G.BIN	2 048
1 297 280 bytes	INCW4G.BIN	20 480
	INCW5G.BIN	168 960
	INCW6G.BIN	512
	OUTA1G.BIN	15 360
	OUTA2G.BIN	17 920
	OUTA3G.BIN	12 800
	OUTA4G.BIN	102 400
	OUTB4G.BIN	102 400
	OUTA5G.BIN	844 800
	OUTA6G.BIN	2 560
	READ.ME	896

ディスク４　内部状態変数

ＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｇにおける固定小数点ＪＴ－Ｇ７２８符号化器の仕様はビットイグザクトである。二つの異なる実装間のすべての状態変数の内部表現を比較するためのテスト入力として、短区間音声が用いられる。すべての出力ベクトルが一致しなくてはならない。混乱を避けるために、この出力すべてはディスク４にASCIIフォーマットで格納されている。

付表１－９　配布ディスク＃４

Size	Filename	Remarks
36 100	a.q14	a(2) to a(51) in Q14. A "-" means no update for that vector.
7 700	ap.q14	ap() in Q14
8 400	apf.bf	apf() as the intermediate output of block 50, then Q format
7 700	apf.q13	the final apf() in Q13 (converted from apf.bf)
36 900	atmp.bf	Atmp(2) to atmp(51), then IAQ Q format. (block 50 output)
7 700	awp.q14	awp(2) to awp(11) in Q14. A "-" means no update for that vector.
7 700	awz.q14	awz(2) to awz(11) in Q14. A "-" means no update for that vector.
8 400	awztmp.bf	awztmp(2) to awztmp(11) in Q13, Q14, or Q15, followed by the Q format in the last column. (block 37 output)
7 700	az.q14	az() in Q14
1 200	b.q16	b in Q16 (long-term postfilter coeff. computed in block 84)
14 400	d.q1	the newest vector of d() array in Q1
4 200	dec.q1	dec(21:25) in Q1 (new decimated LPC residual for current frame)
15 600	et.bf	et() in block floating-pt; 5 mantissas and nlset in each line.

付表１－９　配布ディスク＃４（続き）

Size	Filename	Remarks
3 600	gain.sf	linear gain used to scale codevector (mantissa, then nlsgain)
4 000	gaininv.sf	1/GAIN used to normalize target vector (mantissa, then NLS)
1 400	gl.q14	gl in Q14
1 400	glb.q14	glb in Q14
7 700	gp.q14	gp(2) to gp(11) in Q14. A "-" means no update for that vector.
8 400	gptmp.bf	gptmp(2) to gptmp(11), then gptmp Q format. (block 44 output)
2 800	gstate.q9	gstate(1) in Q9. (The other 9 gstate() are in previous lines.)
3 500	gtmp.q9	gtmp() in Q9. Note the first gtmp() vector has three -16384.
4 200	h.q13	h() vector in Q13. A "-" means no update for that vector.
2 000	ichan.q0	encoder output channel index "ichan" (one per line)
14 400	input.q3	16-bit linear PCM input vector (fixed Q3, one vector a line)
2 400	isig.q0	shape index "is" followed by gain index "ig" in each line
1 400	kp.q0	the pitch period kp in Q0
2 800	loggain.q9	log-gain before converting to linear gain (block 48 input)
14 800	lpfiir.q1	the 20 elements of lpfiir() corresponding to the current frame
14 400	output.q3	decoder (with postfilter) output vector in 16-bit linear PCM
14 400	pn.q7	pn() in Q7 (block 13 output)
1 200	ptap.q14	Ptap in Q14 (output of block 83)
8 400	r_b36.bf	r(1) to r(11) at block 36 output
8 400	r_b43.bf	r(1) to r(11) at block 43 output
1 400	rc1.q15	rc1 of block 50 in Q15 (the one used to derive tiltz)
5 821	readme	describes contents of disk #4
37 600	rexp.bf	Rexp(1) to rexp(51), then nlsrexp. (block 49 output)
9 200	rexplg.bf	rexplg(1) to rexplg(11), then nlsrexplg. (block 43 output)
9 200	rexpw.bf	rexpw(1) to rexpw(11), then nlsrexpw. (block 36 output)
36 900	rtmp.bf	Rtmp(1) to rtmp(51) at block 49 output
14 400	s.q2	input s() vector after converting input.q3 to Q2 with rounding
3 600	scale.sf	Scale in scalar floating-point (output of block 75)
14 400	scalefil.q14	scalefil in Q14 (output of block 76)
14 400	sst.q0	Q0 sst(-4:0) after SST() buffer shift (i.e. sst(1:5) >> 2)
14 400	sst.q2	sst(1:5) in Q2
14 400	st.bf	st() in block floating-point format
15 600	statelpc.sbf	The newest 5 elements of statelpc() and nlsstate(10) for the current vector (The other 45 elements are in previous lines.)
15 600	stmp.q2	Stmp() in Q2. Its content is up to vector 2 of current frame.
14 800	stpffir.q2	stpffir(1:5) after postfiltering the current vector

付表１－９　配布ディスク＃４（続き）

Size	Filename	Remarks
14 400	stpfiir.q2	stpfiir(1:5) after postfiltering the current vector
14 400	sttmp.sbf	sttmp(), 20 mantissas followed by 4 exponents (nlssttmp()).
17 700	sw.q2	sw() in Q2 (block 4 output)
3 200	sumfil.q2	sumfil in Q2 at the output of block 74 (AA1 in pseudo-code)
3 200	sumunfil.q2	sumunfil in Q2 at the output of block 73 (AA0 in pseudo-code)
14 400	target.q2	unnormalized target vector in Q2 (block 11 output)
14 400	targetn.bf	gain-normalized target vector in block floating-point
15 600	temp_b72.q2	temp() at the output of block 72, in Q2
14 400	tiltz.q14	tiltz in Q14
1 400	wiir.q2	newest 5 elements of wiir() in Q2 after weighting filtering
14 400	y2.q5	y2() array in Q5. A "-" means no update for that vector.
78 700	zir.q2	zir() vector in Q2
14 400	zirwfir.q2	newest 5 elements of zirwfir() after memory update of block 10
14 400	zirwiir.q2	newest 5 elements of zirwiir() after memory update of block 10

付録２
（標準ＪＴ－Ｇ７２８に対する）
ＬＤ－ＣＥＬＰアルゴリズムの音声性能ガイドライン

２．１　本付録について

本付録の目的は、網の他の部分と接続したときの、16kbit/s ＬＤ－ＣＥＬＰアルゴリズムの音声性能のガイドラインを示すことである。音声に加えて非音声信号についても、一般的な方向づけを示す。

本付録では、16kbit/s ＬＤ－ＣＥＬＰとは、ＴＴＣ標準ＪＴ－Ｇ７２８に示されたアルゴリズム、32kbit/s ＡＤＰＣＭとは、ＴＴＣ標準ＪＴ－Ｇ７２６に示されたアルゴリズム、64kbit/s ＰＣＭとは、ＴＴＣ標準ＪＴ－Ｇ７１１に示されたアルゴリズムをさす。

２．２　音声性能

２．２．１　単一符号化

伝送誤りのない場合、16kbit/s ＬＤ－ＣＥＬＰコーデックの聴感上の品質は、64kbit/s ＰＣＭコーデックの品質より低いが、32kbit/s ＡＤＰＣＭコーデックの品質と同等である。

16kbit/s ＬＤ－ＣＥＬＰコーデックは実質上、ビット誤り率で10^{-3}までのノイズ（ガウス分布）に影響されないことが知られており、またビット誤り率が10^{-2}の場合、その性能は、32kbit/s ＡＤＰＣＭコーデックと同等である。

16kbit/s ＬＤ－ＣＥＬＰコーデックに割り当てられた計画値はＩＴＵ－Ｔ勧告Ｇ．１１３に示されている。

２．２．２　アナログベースで符号化システムを相互接続した場合の音声性能

２．２．２．１　16kbit/s ＬＤ－ＣＥＬＰの多重タンデム接続

16kbit/s ＬＤ－ＣＥＬＰを多数の音声符号化装置を使用してタンデム接続する場合、3段のタンデム接続までは、32kbit/s ＡＤＰＣＭの場合と同等の性質を示す。16kbit/s ＬＤ－ＣＥＬＰコーデックのタンデム接続に間する厳密な規定は、ＩＴＵ－Ｔ勧告Ｇ．１１３に示されている。

２．２．２．２　32kbit/s ＡＤＰＣＭと接続した場合の性能

16kbit/s ＬＤ－ＣＥＬＰを、32kbit/s ＡＤＰＣＭとタンデム接続したときの音声性能については、以下の2つの構成において同等である。

ＪＴ－Ｇ７２８ ＋ ＪＴ－Ｇ７２６ ＋ ＪＴ－Ｇ７２８

ＪＴ－Ｇ７２６ ＋ ＪＴ－Ｇ７２８ ＋ ＪＴ－Ｇ７２６

ここで、”＋”は相互接続を表す。

２．２．３　16kbit/s ＬＤ－ＣＥＬＰを同期タンデム接続した場合の音声性能

主観的な実験によれば、同期タンデム接続（即ち、64kbit/s ＰＣＭを介して2つまたはそれ以上のＬＤ－ＣＥＬＰの相互接続）での音声性能は、非同期タンデム接続の場合と同等である（２．２．２．１節参照）。つまり、16kbit/s ＬＤ－ＣＥＬＰには、 32kbit/s ＡＤＰＣＭ（標準ＪＴ－Ｇ７２６）のような同期タンデムのための特別な機能はない。

２．２．４　16kbit/s ＬＤ－ＣＥＬＰおよび32kbit/s ＡＤＰＣＭ以外のコーデックと接続した場合の性能

一般に、16kbit/s ＬＤ－ＣＥＬＰと、それ以外の2つまでの装置と相互接続した場合の性能は、32kbit/s ＡＤＰＣＭを同様に相互接続した場合と同等である。しかしこの付録の作成時においては、この問題に関しては限られたデータしか得られなかった。そのため、16kbit/s ＬＤ－ＣＥＬＰや32kbit/s ＡＤＰＣＭ以外のコーデックとの相互接続が行なわれる際には、細心の注意を払う必要がある。

２．３　非音声信号に対する性能

16kbit/s ＬＤ－ＣＥＬＰコーデックは、音声に対して最適化された適応システムである。非音声を用いた測定を行なう場合には、時不変性と線形性の仮定は成り立たないので、十分注意すべきである。（例えば、テストトーンを用いてネットワークの保守を実施する場合。）

２．３．１　情報トーンに対する性能

ネットワークから発するＩＴＵ－Ｔ勧告Ｑ．３５に従った情報トーンは、16kbit/s ＬＤ－ＣＥＬＰコーデック（1段の符号化）を通過した場合、容易に認識されるということが実験によって明らかになっている。

２．３．２　音楽に対する性能

多種多様な音楽に対して、特に不快となるような歪みはないということが実験によって明らかになっている。（２．４節参照）

２．３．３　ＤＴＭＦ（Dual-Tone Multi-Frequency）信号に対する性能

一般的に、16kbit/s ＬＤ－ＣＥＬＰコーデックの1段の符号化における性能は32kbit/s ＡＤＰＣＭおよび64kbit/s ＰＣＭと同等であることがわかっている。

２．３．４　シグナリングシステム５：レジスタ間シグナリングに対する性能

一般的に、16kbit/s ＬＤ－ＣＥＬＰコーデックの1段の符号化における性能は32kbit/s ＡＤＰＣＭおよび64kbit/s ＰＣＭと同等であることがわかっている。

２．３．５　音声帯域データに対する性能

一般的に、16kbit/s ＬＤ－ＣＥＬＰコーデックにおける性能は（1段の符号化でさえも）32kbit/s ＡＤＰＣＭおよび64kbit/s ＰＣＭよりかなり劣ることがわかっている。しかしながら16kbit/s ＬＤ－ＣＥＬＰは聴覚重み付けフィルタおよびポストフィルタをともに無効にすれば、実際の伝送路状態において2400bit/s以下で動作するほとんどの（ただしすべてではない）音声帯域データモデムに対応することができる。

２．４　擬似音声信号

16kbit/s ＬＤ－ＣＥＬＰコーデックの検証を行う中で、ある種の試験信号は符号器および復号器を発散させ、その結果大きく歪んだ出力信号を発生する可能性があることがわかった。この動作は最初にある種の音楽信号および擬似音声信号（ＩＴＵ－Ｔ勧告Ｐ．５０）に対して観測された。ある話者が連続的に発声した母音に対して続けて観測された。実際の話し言葉に対しては観測されてはいない。

試験信号の入力開始時点で、符号器および復号器の状態がわずかに異なっているとこの現象が発生することがある。このような差異は、符号器と復号器が異なったメーカのものである場合のわずかな処理の違いによって引き起こされる可能性がある。まったく同じ処理を行なう符号器と復号器においてさえも、伝送路誤りや符号器と復号器の初期化の時間ずれによって差異が発生してしまう可能性がある。出力信号の歪みは、符号器および復号器の実現における精度の劣化とともに増大することが確かめられている。

発散が起こるための別の条件は、入力信号がある特性を持っていることである。まず第1に、信号がいくつもの鋭いスペクトルピークを持っていることである。（今まで観測されたもののうち、最小のピーク数は14である。）第2にこれらのピーク周波数は数100msの間相対的に変化がないことであり、この期間中に歪みが増大する可能性がある。

発散している期間において復号器の出力信号は、無限大になるといったような不安定な状態にはならない。それは有限な信号であるが、スペクトルが元の信号に対して大きく変わっている。入力信号が変化すれば、符号器および復号器は再び収束し、高品質な信号を出力する。

参考文献

Speech Communication, Vol.21, No.2, June 1993, (Special Issue on CCITT Standard on 16kbit/s Speech Coding).

付録

（標準ＪＴ－Ｇ７２８に対する）

用語対照表

英　　語	ＴＴＣ標準用語
accumulator	アキュムレータ
adaptation cycle	適応周期
adapter	適応器
all-pole filter	全極フィルタ
analysis-by-synthesis	合成による分析（Ａ－ｂ－Ｓ）法
autocorrelation coefficient	自己相関係数
backward adaptation	バックワード適応
backward vector gain adapter	バックワードベクトル利得適応器
bandwidth expansion	帯域幅拡張
best codebook index	最適コードブックインデックス
best codevector	最適コードベクトル
best survivor	最適生き残りパス
bit-exact	ビットイグザクト
block floating point	ブロック浮動小数点
branch	ブランチ
call	コールする
codebook	コードブック
codebook index	コードブックインデックス
codebook search	コードブック探索
codebook vector	コードブックベクトル
codevector	コードベクトル
communication channel	通信チャネル
compensation	補正
convolution	畳込み
decoder	復号器
denominator	分母
determination of Viterbi decision	ビタビ判定
digital signal processing (DSP)	ディジタル信号処理(ＤＳＰ)
dynamic locking quantizer algorithm	動的固定化量子化アルゴリズム
encoder	符号器
excitation	励振
excitation codebook	励振コードブック
excitation gain	励振利得
excitation vector	励振ベクトル
excitation VQ codebook	励振ＶＱコードブック
floating point pseudo-code	浮動小数点擬似コード
fixed point pseudo-code	固定小数点擬似コード
format	フォーマット
gain adaptation	利得の適応
gain codebook	利得コードブック
gain-scaled excitation vector	利得調整された励振ベクトル
gain scaling unit	利得調整ユニット
guard bit	ガードビット
headroom	ヘッドルーム

英　　語	ＴＴＣ標準用語
impulse response vector	インパルス応答ベクトル
indentation	字下げ
inner product	内積
interoperability	相互接続性
label	ラベル化する
Levinson-Durbin recursion	レビンソン・ダービン再帰法
low-delay code excited linear prediction (LD-CELP)	低遅延符号励振線形予測（ＬＤ－ＣＥＬＰ）
linear prediction	線形予測
linear prediction analysis	線形予測分析
logarithmic gain	対数利得
log-gain	対数利得
log-gain offset value	対数利得オフセット値
log-gain predictor	対数利得予測器
LPC analysis	ＬＰＣ分析
mantissa	仮数（部）
mean-squared error (MSE)	自乗平均誤差（ＭＳＥ）
multiply-add	積和演算
node	ノード
numerator	分子
perceptual weighting filter	聴覚重み付けフィルタ
pitch period	ピッチ周期
pole controlling vector	極補正ベクトル
pole-zero filter	極零フィルタ
postfilter	ポストフィルタ
predictor	予測器
product register	乗算レジスタ
pseudo-code	擬似コード
quantized signal vector	量子化信号ベクトル
quantized speech vector	量子化音声ベクトル
residual excited linear prediction (RELP)	残差励振線形予測
robustness	耐性
root-mean-square (RMS)	実効値（ＲＭＳ）
round	丸め
round-off error	丸め誤差
scalar floating point	スカラ浮動小数点
scaling	スケーリング
scaling factor	スケーリングファクタ
segmented block floating point	分割ブロック浮動小数点
shape codebook	形状コードブック
signal classifier	信号識別器
simulated decoder	局部復号器
speech vector	音声ベクトル
subroutine	サブルーチン
super codebook	スーパーコードブック
survivor path	生き残りパス
synchronization	同期

英　　語	ＴＴＣ標準用語
synthesis filter	合成フィルタ
synthesis filter coefficient	合成フィルタ係数
target vector	ターゲットベクトル
time-reversed convolution	時間反転畳込み
transition	遷移
Trellis-Coded Quantization (TCQ)	トレリス符号量子化
Trellis survivor	トレリス生き残りパス
unquantized speech	非量子化音声
vector quantization (VQ)	ベクトル量子化（ＶＱ）
Viterbi decision block	ビタビ判定ブロック
voice band data (VBD)	音声帯域データ
white noise correction	白色雑音補正
wrap around	ラップアラウンド
zero controlling vector	零点補正ベクトル
zero-input response	零入力応答
zero-state response	零状態応答

付録
（標準ＪＴ－Ｇ７２８に対する）
用語解説

合成による分析（Ａ－ｂ－Ｓ）法 (analysis-by-synthesis)

信号生成モデルを用いて合成した信号と実際の信号との誤差を最小とするように、パラメータを変化させて求める分析方法。

バックワード適応 (backward adaptation)

直前の符号化結果をもとに、係数の更新を適応的に行う方法。適応動作を逐次的に行うため処理遅れがなく、係数を伝送する必要がない。

ブロック浮動小数点（block floating point)

スカラ浮動小数点をN要素の配列に拡張しN+1ワードで表現したもの。まず、配列の絶対値最大要素をスカラ浮動小数点で表現する。この際の左シフト数をNLSとして1ワードで表現し、配列のその他全ての要素もこのNLSだけシフトし表現したもの。

コードブック／コードブックベクトル (codebook／codebook vector)

ベクトル量子化を用いた符号化方式において、送受信側で共通に持っている代表ベクトル（コードブックベクトル）の配列。本標準では、励振ベクトルを表現するためコードブックとして励振ＶＱコードブックを用いており、これは励振ベクトルの形状を表現する形状コードブック、および利得の時間的変化を表現する利得コードブックからなる。

符号励振線形予測（ＣＥＬＰ）(code excited linear prediction)

音声信号の短期および長期相関を取り除いた残差信号をベクトル量子化により生成されたコードブックで表現し、このコードブックを用いて音声を符号化する方式。コードブックの探索は、合成による分析（Ａ－ｂ－Ｓ）法を用いる。

ガードビット (guard bit)

ガードビットとは積和演算等の演算過程で生じるアキュムレータのオーバフローが、最終結果に影響を及ぼさないように、アキュムレータの上位に拡張された余剰ビットである。

ガードビットを備えても最終結果がオーバフローすることがあるが、この場合は適切なクリップ処理を併用する。

ヘッドルーム (headroom)

ビット列の数値表現における上位側の空きビット。例えば、ヘッドルームの2ビットとは、上位2ビット分の空きスペースである。

低遅延符号励振線形予測（ＬＤ－ＣＥＬＰ）(low-delay code excited linear prediction)

ＣＥＬＰ方式を基本にした符号化方式で、短いベクトル長のコードブックを用いると共に、バックワード適応方式を用いて符号化遅延を抑えた方式。

レビンソン・ダービン再帰法 (Levinson-Durbin recursion)

対象とする信号の相関係数から再帰的に線形予測係数と反射係数を求める能率的な方法で、本標準では、聴覚重み付けフィルタ適応器、バックワード合成フィルタ適応器、バックワードベクトル利得適応器で用いている。

ＬＰＣ分析 (LPC analysis)

現時刻におけるサンプル値を、過去のサンプル値の線形結合として予測する線形予測モデルを用いて、現サンプルの観測値と予測値の誤差が最小となるように線形式の係数を求め、それを分析値とする音声信号の分析方法。

聴覚重み付けフィルタ (perceptual weighting filter)

音声信号のホルマント領域での量子化雑音は、マスキング効果により小さく感じられる。この効果を利用して、聴感上の雑音感を低減するためのフィルタ。

ピッチ周期 (pitch period)

人間の声の有声音（一様な周期の繰り返し部分よりなる母音系の音）における声帯振動の周期。聴覚上は音の高さに対応する。

極補正ベクトル (pole controlling vector) および零点補正ベクトル (zero controlling vector)

スペクトラムを補正するためにフィルタ係数の補正を行うベクトル。聴覚上の品質を向上させるために、聴覚重み付けフィルタ、短期ポストフィルタにおいて、分母および分子のフィルタ係数をそれぞれ極補正ベクトル、零点補正ベクトルで補正する。

ポストフィルタ (postfilter)

復号された音声信号の聴感上の品質を向上させるために、復号器の出力側に設けられたフィルタ。本標準では、復号音声のピッチ（声帯の振動周波数）成分を強調し、かつ周波数スペクトルの包絡線の谷間の成分を減衰させる事によりこれを行っている。

Qフォーマット（Q format)

小数部分を持つような数値を固定小数点で扱う場合に、小数点以下を何ビットで表すかを示す。例えばQ14フォーマットとは、小数点以下が14ビットであることを示している。

残差励振線形予測（ＲＥＬＰ）(residual excited linear prediction)

音声の予測残差信号を量子化することによって音声を符号化する方式。

スカラ浮動小数点（scalar floating point)

単一の値を浮動小数点により表現したフォーマット。単精度の場合は2ワードで表現され、第1の16ビットワードは値の絶対値が16384と32767の間に揃えられた数により値の仮数部を表し（大きさの範囲が決まっているため正規化フォーマットと呼ぶ）、第2の16ビットワードは値を正規化フォーマットにするために行った左シフトの数を表す。

分割ブロック浮動小数点 (segmented block floating point)

大きさMNの配列を、大きさNの副配列がM個になるように分割し、分割された副配列のそれぞれに対して、ブロック浮動小数点を適用して各副配列をN+1ワードで表現したもの。これにより、配列全体としては、M個のブロック浮動小数点を用いてM(N+1)ワードで表現出来る。

ベクトル量子化（ＶＱ）(vector quantization)

波形符号化や分析合成系において、複数のサンプル値をグループ（ベクトル）にまとめて1つの符号で表現し、量子化する方法。本標準では、連続する5つの励振サンプルを1符号として量子化している。

ラップアラウンド（wrap around)

2の補数表現の演算において、オーバフローが発生した際に、正の絶対値の大きな数が負の絶対値の大きな数に変換されたり、あるいは逆に負の数が正の数に変換される現象。

2の補数表現（2's complement arithmetic)

2進数の代表的な1つの表現方法。大きさMの正の数は+M、大きさMの負の数は $2^n - M$ で表される。ここで、nは語長を表す。最上位ビット（ＭＳＢ）が0ならば正の数を示し、1ならば負の数を示す。