TTC標準
Standard

JT-G711

音声周波数帯域信号の PCM 符号化方式

Pulse Code Modulation (PCM) of Voice Frequencies

第 6 版

2011 年 5 月 31 日制定

一般社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

TTC Telecommunication Technology Committee

目　　次

＜参考＞

１．国際勧告等との関連

本標準はＩＴＵ－Ｔ勧告 1988 年版Ｇ．７１１に準拠したものである。

本標準の付録１は、1999 年９月に承認されたＩＴＵ－Ｔ勧告Ｇ．７１１　ＡＰＰＥＮＤＩＸ　Ｉに準拠したものである。

本標準の付録２は、2000 年２月に承認されたＩＴＵ－Ｔ勧告Ｇ．７１１　ＡＰＰＥＮＤＩＸ　IIに準拠したものである。

また、本標準の付属資料Ａは、2009 年８月に承認されたＩＴＵ－Ｔ勧告Ｇ．７１１に対するＡｍｅｎｄｍｅｎｔ１に準拠して改定されたものである。

本標準の付録３は、2009 年 11 月に承認されたＩＴＵ－Ｔ勧告Ｇ．７１１に対するＡｍｅｎｄｍｅｎｔ２に準拠して改定されたものである。

２．上記国際勧告等に対する追加項目等

(1)　本標準は上記ＩＴＵ－Ｔ勧告に対し、下記の項目についての記述を削除している。

(a)　Ａ則符号化方式に関する事項

(b)　Ａ則←→μ則の相互変換に関する事項

上記項目(a)につき削除した理由は、ＰＣＭ符号化方式としてμ則を採用している我が国の現状による。
次に項目(b)につき削除した理由は、μ則を国内網間インタフェースでのＰＣＭ符号化方式と定めるため、ＴＴＣに於いて標準化する必要性が認められないことによる。

３．改版の履歴

版　数	制　　定　　日	改　版　内　容
第１版	昭和 62 年４月 28 日	制定
第２版	平成 元年４月 28 日	ＩＴＵ－Ｔ勧告準拠年号の変更
第３版	平成 12 年 11 月 30 日	付録１の追加
第４版	平成 13 年４月 19 日	付録２の追加
第５版	平成 23 年２月 23 日	付属資料Ａの追加
第６版	平成 23 年 5 月 31 日	付録３の追加

４．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。

１．本標準の規定範囲

本標準は、音声周波数帯域信号のＰＣＭ符号化方式に関するものである。

２．標本化周波数

標本化周波数の公称値は 8000Hz とする。周波数の偏差は±50ppm とする。

３．符号化則

(1)　８ビット／標本の符号化法を標準とする。

(2)　符号化則として μ 則を規定する。符号化則の定義を表３－１／JT-G711 と表３－２／JT-G711 に示す。

(3)　８ビット全て“０”の信号を防ぐ必要のある回線では負の入力レベルにおける識別値番号 127 と 128 の間の信号の符号化は “00000010” とし、その値の復号器出力値番号は 125 とする。この時の復号器出力値は-7519 とする。

４．符号化則と音声レベルの関係

符号化則と音声レベルとの関係を以下のように定義する。

μ 則に関する表４－１／JT-G711 の符号語の繰り返し信号を、復号器入力に加えた時、μ 則に使用する装置の音声出力は公称 0dBm0 の 1000Hz 正弦波信号である。

この場合、理論上の最大負荷レベルは＋3.17dBm0 となる。

表４－１／JT-G711*　1000Hz　正弦波信号の符号語

(ITU-T G.711)

ビット / 標本化番号	1	2	3	4	5	6	7	8
1	0	0	0	1	1	1	1	0
2	0	0	0	0	1	0	1	1
3	0	0	0	0	1	0	1	1
4	0	0	0	1	1	1	1	0
5	1	0	0	1	1	1	1	0
6	1	0	0	0	1	0	1	1
7	1	0	0	0	1	0	1	1
8	1	0	0	1	1	1	1	0

表３－１／JT-G711　μ則　正の入力
(ITU-T G.711)

1	2	3	4	5	6	7	8
折線番号	ステップ数 × ステップ幅	折線端の値	識別値番号 n	識別値　Xn 注１	符号語 ビット番号 １２３４５６７８	復号器 出力値　Yn 注３	復号器 出力値 番　号
		8159	(128)	(8159)			
8	16×256		127 113	7903 4319	１０００００００ 注２ １０００１１１１	8031 4191	127 112
7	16×128	4063	112 97	4063 2143	注２ １００１１１１１	2079	96
6	16×64	2015	96 81	2015 1055	注２ １０１０１１１１	1023	80
5	16×32	991	80 65	991 511	注２ １０１１１１１１	495	64
4	16×16	479	64 49	479 239	注２ １１００１１１１	231	48
3	16×8	223	48 33	223 103	注２ １１０１１１１１	99	32
2	16×4	95	32 17	95 35	注２ １１１０１１１１	33	16
1	15×2	31	16 2	31 3	注２ １１１１１１１０	2	1
↓	1×1		1 0	1 0	１１１１１１１１	0	0

注１　最大負荷レベル 3.17dBm0 を 8159 とした値。
注２　２つの連続する識別値番号ｎと（ｎ＋１）間の正入力レベルに対応する符号語は（255－n）の２進数で表される。
注３　復号器出力の値は

$$Y_0 = X_0 = 0 \quad (n = 0)$$

$$Y_n = \frac{X_n + X_{n+1}}{2} \quad (n = 1,2,\cdots,127)$$

である。

注４　X_{128}は仮想的な識別値である。

表３－２／JT-G711　μ則　負の入力
(ITU-T G.711)

1	2	3	4	5	6	7	8
折線番号	ステップ数 × ステップ幅	折線端の値	識別値番号 n	識別値　Xn 注１	符号語 ビット番号 １２３４５６７８	復号器 出力値　Yn 注３	復号器 出力値 番　号
↑ 1	1×1		0	0	０１１１１１１１	0	0
			1	-1	０１１１１１１０	-2	1
	15×2		2 ⋮	-3 ⋮	注２ ⋮	⋮	⋮
		-31	16	-31	０１１０１１１１	-33	16
2	16×4		17 ⋮	-35 ⋮	注２ ⋮	⋮	⋮
		-95	32	-95	０１０１１１１１	-99	32
3	16×8		33 ⋮	-103 ⋮	注２ ⋮	⋮	⋮
		-223	48	-223	０１００１１１１	-231	48
4	16×16		49 ⋮	-239 ⋮	注２ ⋮	⋮	⋮
		-479	64	-479	００１１１１１１	-495	64
5	16×32		65 ⋮	-511 ⋮	注２ ⋮	⋮	⋮
		-991	80	-991	００１０１１１１	-1023	80
6	16×64		81 ⋮	-1055 ⋮	注２ ⋮	⋮	⋮
		-2015	96	-2015	０００１１１１１	-2079	96
7	16×128		97 ⋮	-2143 ⋮	注２ ⋮	⋮	⋮
		-4063	112	-4063	０００００１１１	-4191	112
			113 ⋮	-4319 ⋮	注２ ⋮	⋮	⋮
8	16×256		126	-7647	０００００００１	-7775	126
			127	-7903	００００００００	-8031	127
		-8159	(128)	(-8159)	- - - - - - - - - - - - - - - -		

注１　最大負荷レベル 3.17dBm0 を 8159 とした値。
注２　２つの連続する識別値番号ｎと（ｎ＋１）間の負入力レベルに対応する符号語は（127－n）の２進数で表される。
注３　復号器出力の値は

$$Y_0 = X_0 = 0 \quad (n = 0)$$

$$Y_n = \frac{X_n + X_{n+1}}{2} \quad (n = 1,2,\cdots,127)$$

である。

注４　X_{128}は仮想的な識別値である。

付属資料Ａ

（標準ＪＴ－Ｇ７１１に対する）

パルス符号変調向けロスレス符号化

標準ＪＴ－Ｇ７１１に適合する対数 PCM の 40、80、160、240、320 サンプルから構成されるフレームのロスレス符号化を要求するアプリケーションに対して、標準ＪＴ－Ｇ７１１．０の使用を推奨する。

参考文献

[1] ＴＴＣ標準ＪＴ－Ｇ７１１．０
- ＪＴ－Ｇ７１１パルス符号変調向けロスレス符号化

付録１

（標準ＪＴ－Ｇ７１１に対する）

標準ＪＴ－Ｇ７１１向けパケット損失補償のための高品質低演算量アルゴリズム

１．１　はじめに

パケット損失補償（ＰＬＣ）アルゴリズムは、フレーム消失補償アルゴリズムとしても知られており、入力信号が送信器で符号化され、パケット化され、ネットワークに送出され、パケットの復号化を行い、出力を再生する受信器において受信されるようなオーディオシステムにおける伝送損失を補償するものである。ＴＴＣ標準ＪＴ－Ｇ７２３．１[1]、ＪＴ－Ｇ７２８[2]、ＪＴ－Ｇ７２９[3]の様なＣＥＬＰを基にした音声コーデック標準の多くは、それらの標準のなかにＰＬＣアルゴリズムが組み込まれている。ここで述べられているアルゴリズムは、ＴＴＣ標準ＪＴ－Ｇ７１１に対する一つの方法を提供する。

ＰＬＣの目的は、受信されたビットストリーム中で失われたデータ（消失区間）を補償するための合成音声信号を生成することである。理想的には、合成信号は消失した信号と同じ音色と周波数特性をもつことであり、不自然な人工的雑音を作らないことである。音声信号は、しばしば局所的には定常なので、消失した区間の妥当な近似を作るのに過去の履歴の信号を使うことが可能である。もし、消失区間がそれほど長くなく、かつ信号が急激に変化している領域でなければ、補償後には消失区間は知覚されないかもしれない。

１．２　アルゴリズムの説明

損失補償を有しないＴＴＣ標準ＪＴ－Ｇ７１１システムにＰＬＣを追加するには、受信器のみの変更でよい。ＴＴＣ標準ＪＴ－Ｇ７１１の符号化オーディオデータは８kHz で標本化されている。この付録では、オーディオデータが 10ms（80 サンプル）に分割されていると仮定する。いくつかのパラメータを調整することにより、他のパケットサイズや標本化周波数にも適応させることが可能である。

１．２．１　正常フレーム

正常動作（正常パケットあるいはフレーム）の間は、受信器は受信されたパケットを復号し、オーディオポートに出力を送る。ＰＬＣをサポートするために、正常フレームを処理する際に受信器に２つの部分的変更を行う。

(1)復号出力のコピーが 48.75ms（390 サンプル）長の巡回型の履歴バッファに保存される。履歴バッファは現在のピッチ周期を計算するため、および消失区間の波形を抽出するために使われる。このバッファリングは出力信号に何らの遅延を生じさせたりはしない。

(2)出力はオーディオポートに送られる前に、3.75ms（30 サンプル）遅延される。消失区間の始まりでのオーバラップ加算（ＯＬＡ）に使われるこのアルゴリズム遅延により、ＰＬＣコードが実信号と合成信号との間でなめらかに遷移することができる。

１．２．２　最初の異常フレーム

消失区間の最初に、巡回型の履歴バッファから、ピッチバッファと呼ぶ非巡回型のバッファへのコピーを行う。これは、作業を容易にするためである。ピッチバッファの内容は消失が継続する間使用される。消失が 10ms 以上継続する場合は、lastq バッファと呼ぶ、最近の 1/4 ピッチ周期部分のコピーを追加して行う。

１．２．３　ピッチ検出

ピッチ周期は、5ms（40 サンプル）から 15ms（120 サンプル）までの範囲で、履歴バッファにある最近の 20ms の音声と過去の音声との正規化相互相関のピークを見つけることで推定される。これは 200Hz から 66Hz の周波数に相当する。このピッチ範囲はＴＴＣ標準ＪＴ－Ｇ７２８のポストフィルタで使用されている範囲を基に

して定めた。ＴＴＣ標準ＪＴ－Ｇ７２８では2.5ms（20サンプル）が下限であるが、10msの消失フレームひとつのなかに２回以上同じピッチ周期が繰り返されないようにするため、ここでは下限を40サンプルに増やしている。低演算量化のため、ピッチ推定は次の２段階に分けて計算される。まず、2:1に間引いた信号で粗い探索を行い、その後、粗い探索のピーク付近で詳細な探索を行う。品質はわずかに劣化するが、詳細な探索を省くことで演算量を減らすことが可能である。消失した信号は、有声音の場合も無声音の場合もあるので、以下では、この計算の出力値を参照するためにwavelengthという用語を使用することもある。

波形シフトオーバラップ加算（ＷＳＯＬＡ）では、正規化相互相関関数は、非正規化相互相関や相互平均振幅差関数（ＡＭＤＦ）で置き換えることが可能であり、同様な総合特性結果が得られることが知られている。

１．２．４　最初の10msのための合成信号の生成

消失区間の最初の10msのためには､減衰を行わずに最後のピッチ周期から合成信号を生成することにより最良の結果が得られる。最初の10msの間は、ピッチバッファの最近の1.25ピッチ周期だけが使用される。実信号と合成信号の間のなめらかな遷移とピッチ周期が繰り返される場合のなめらかな遷移を行うために、最後とその１つ前のピッチ周期の間の1/4ピッチ周期に三角窓を用いたオーバラップ加算（ＯＬＡ）が行われる。1/4 wavelengthのために、ピッチバッファの終わりから1.25ピッチ周期のところから始まる信号に右上がりの傾斜重みを乗じ、右下がりの傾斜重みを乗じたlastqバッファの最後の0.25ピッチ周期に加算される。もし演算量が問題でなければ、すべてのＯＬＡ処理において三角窓はハニング窓で置き換えてもよい。

ＯＬＡの結果はピッチバッファと履歴バッファの両方の末尾を書き換える。また、最後の正常フレームの末尾の間、受信器によって出力され、元の信号を置き換える。これによりアルゴリズム遅延が生じる。すなわち、次フレームが消失かどうかわかるまで、最後のフレームの末尾を出力することができない。もし消失が起これば､最後の正常フレームの末尾の信号は合成信号へのなめらかな遷移を行うためにＯＬＡによって修正される。

消失の間の10msに対する合成信号は、ピッチバッファの終わりからポインタを１ピッチ周期分戻し、サンプルを出力にコピーすることにより生成される。ピッチ周期が10msより短い場合、ポインタがピッチバッファの最後からはみ出すときは、処理を続ける前にちょうど１ピッチ周期だけ前にポインタを戻す。ピッチ周期が短い（周波数が高い）場合は、ピッチバッファの中の最後のピッチ周期は10msの消失の間、繰り返される。

消失が続く間、履歴バッファは合成された出力信号で更新される。このようにして、履歴バッファにはいつもなめらかで連続的な信号がある。この連続性は、「異常フレーム、正常フレーム、異常フレーム」という系列が発生したときに重要である。

１．２．５　10ms以後の合成信号生成

次のフレームも消失した場合、消失区間は少なくとも20ms間継続するので、更なる処理が必要になる。短い消失区間（例えば10ms）に対しては１つのピッチ周期を繰り返しても問題はないが、長い消失区間に対して行うと不自然で調波的な人工的雑音（ビープ音）が発生する。これは消失区間が音声の無声領域にかかる場合、あるいは閉鎖音のような急激な変化領域にかかる場合に特に顕著になる。これらの人工音は、消失区間の長さに応じて信号合成に用いるピッチ周期を増やしていけば十分に少なくできることが実験によってわかっている。多くのピッチ周期を合成に用いることにより信号のバリエーションが増加する。このとき、原信号で現れた順序通りにピッチ周期は再生されないが、出力音の自然性は維持される。10ms消失した時点において、音声を合成するために用いるピッチ周期の数は２つに増え、20ms 消失した時点において３つ目のピッチ周期が加わる。20msより長い消失区間についてはピッチバッファを修正しない。

ピッチバッファに用いるピッチ周期の数を増やすにあたっては、合成信号がなめらかにつながることが重要になる。これは次のようにして達成される。すなわち、第２および第３消失フレームの開始点直後の 1/4 ピッチ周期は現在のピッチバッファの出力を継続し、ピッチバッファを更新し、正しい位相とバッファポインタの同期を保ち、そして、新しいピッチバッファの出力とＯＬＡを行うことである。

ピッチバッファは、ピッチ周期の数を増やすことを除いて、最初の消失フレームの場合と全く同じように更新される。たとえば、第２消失フレームの最初において 1/4 wavelength については、ピッチバッファの終点から過去に 2.25 ピッチ周期の位置から始まる信号に右上がりの傾斜重みを掛け、それに lastq バッファ内の 1/4 wavelength に右下がりの傾斜重みを掛けた波形が加算される。ＯＬＡの結果はピッチバッファの最後の 1/4 wavelength を置き換える。また、現在の出力ポインタの位相を保つために、先頭ピッチ周期の区間内になるまでポインタからピッチ周期を差し引いていく。

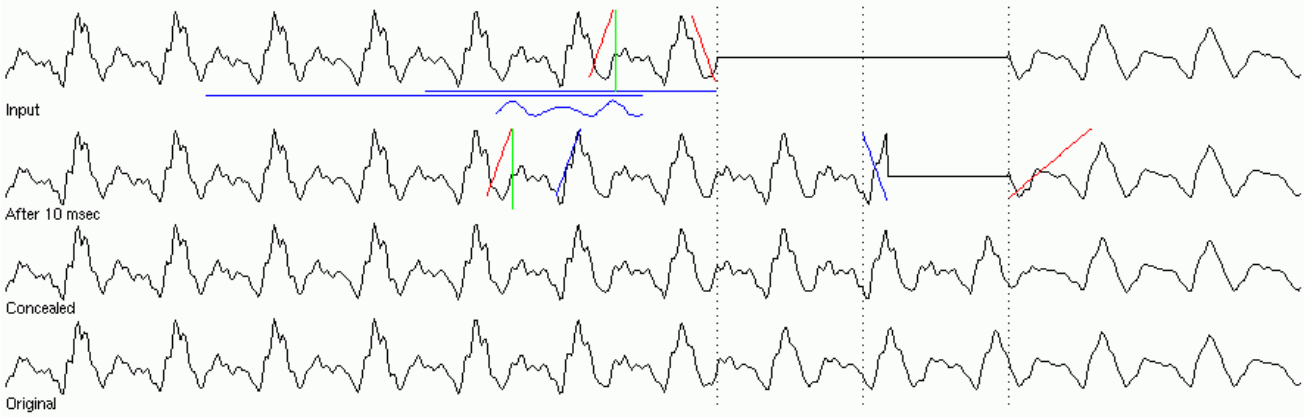
１．２．６　減衰

ＴＴＣ標準ＪＴ－Ｇ７２９やＴＴＣ標準ＪＴ－Ｇ７２８付属資料Ｉなどに含まれる他のＰＬＣアルゴリズムと同様に、長期間の消失が生じた場合は消失時間長に応じて信号の減衰が必要になる。消失時間が長くなるにつれ、合成信号は実信号と異なってくるはずである。減衰を加えないと、たとえ個々の合成信号が自然性をもっていたとしても、ある特定の波形を継続した場合には不自然な人工的雑音が形成される。消失区間の最初の 10ms 間、信号は減衰させない。２番目の 10ms 区間の開始点からは合成信号に対し 10ms あたり 20%の割合で傾斜する線形的な減衰を与える。60ms 後、合成信号はゼロになる。

１．２．７　消失区間後の最初の正常フレーム

消失区間後の最初の正常フレームでは、合成した消失区間の音声と実信号の間になめらかなつながりが求められる。このために、ピッチバッファからの合成音声は消失区間の終点を越えても継続され、実信号とＯＬＡされる。ＯＬＡの窓長はピッチ周期と消失継続期間により変える。最初の 10ms で終わる短期の場合はそれを 1/4 wavelength 窓とし、10ms 以上長く続く場合は 10ms あたり 4ms の延長を行う。ただしフレーム長である 10ms を上限とする。

１．２．８　適用例



付図１－２－１／JT-G711　Frame erasure concealment algorithm for JT-G711 (ITU-T G.711)

付図１－２－１／JT-G711 は男性話者音声の有声区間で 20ms（すなわち２フレーム）の消失が発生した場合に本アルゴリズムを適用した例を示す。最上段の波形図は入力を示す。消失発生区間はすべての波形図を交差し、10ms 間隔で引いてある３本の縦点線で示される。波形図"Input"で、消失開始点より前方の波形は、消失発生時の履歴バッファの記憶内容である。消失区間の後の音声は、消失区間が終わった後に到着する 20ms の誤り無しの音声である。

波形図"Input"の直下には２本の横線が引かれているが、これらは正規化相互相関を用いてピッチ検出を行なう際の時間窓を表す。上側の短い横線は消失区間直前の 20ms を示し、参照信号とされる。下側の横線は、40 サンプルから 120 サンプルの範囲で後方にスライドする 20ms 窓を表す。その下の小さな波形は正規化相互相関の結果である。縦線で示しているその波形のピークはピッチ推定値である。この縦線から消失開始点までをピッチ周期とする。

消失開始点の直前の 1/4 wavelength が、後の処理に使用するために、lastq バッファに記憶される。ピッチバッファをつくるために、消失開始点直前の 1/4 wavelength は三角窓で重み付けをし、直前のピッチ周期の 1/4 wavelength とＯＬＡされる。窓の重みと位置は２つの斜線として波形図"Input"に示されている。ＯＬＡの結果は消失開始点直前の 1/4 wavelength と置き換える。消失している最初の 10ms 間では、単周期のピッチバッファを繰り返すことで合成信号がつくられる。例えば、始点を相互相関のピーク位置を示す縦線、終点を消失開始点とする領域を必要回数分繰り返す。

この結果は２番目の波形図"After 10ms"に示されている。この波形図では、合成信号は 1/4 wavelength だけ最初の 10ms 消失区間を越えて広がっており、これは次のＯＬＡのために必要となる。10ms 終了時点における単周期のピッチバッファのオフセット分は poffset という変数に記憶される。もし消失区間が 10ms であるなら、この 1/4 wavelength は入力信号とＯＬＡされ、その後、補償アルゴリズムは終了する。ここでは消失区間が 10ms より長いので信号のバリエーションを増やすためピッチバッファが延長される。

信号のバリエーションはもう一つのピッチ周期をピッチバッファに追加することで増やされる。縦線が消失開始点から２ピッチ周期過去に印されている。この縦線の前 1/4 wavelength、すなわち図中の右上がり斜線で示した波形は lastq バッファ（波形図"Input"の右下がり斜線の範囲）とＯＬＡされ、ピッチバッファに置かれる。このように、２番目の 10msec に対するピッチバッファは波形図 c After 10ms"において最初の右上がり斜線の終端から消失開始点までの範囲となる。消失開始点直前の 1/4 wavelength は波形図に示したピッチバッファの内容とは異なり、ＯＬＡの結果を示している。

単周期ピッチバッファと倍周期ピッチバッファとの間のなめらかな遷移を保証するために、２つ目の消失フレームにおける最初の 1/4 wavelength 部分について、ＯＬＡが行われる。消失区間が始まってから 10ms 時点の右下がり斜線領域が、倍周期ピッチバッファにおける最初の信号ピーク近傍の右上がり斜線領域とＯＬＡされる。その結果は、右下がり斜線の領域を置き換える。右上がり斜線の位置は、ピッチバッファの現在用いている部分における最初の wavelength 区間内にピッチポインタが来るように、保存した poffset ポインタからピッチ周期を引くことによって計算される。これにより、ピッチバッファにおけるピッチ周期の数が増加しても、正確な波形位相を維持することが保証される。２つ目の消失フレームの残りの部分については、合成信号はピッチバッファの信号を単純にコピーすることによって生成される。

最初の 10ms の場合と同様に、合成波形は次の区切りで行うＯＬＡのため次の 10ms 境界を越えて延長される。消失が続くと、1/4 wavelength で示されるピッチバッファ内の区間の数が再度増加する。消失が終了すると、ＯＬＡ窓長は、合成波形と実波形の位相の不適合がより少なくなるよう、消失区間が 10ms 増える毎に 4ms 増加する（最大値 10ms）。20ms 消失（1/4 wavelength＋4ms）の最後のＯＬＡ窓は波形図"After 10ms"の消失の最後の右上がり斜線領域として示される。２番目の 10ms 消失波形区間は線形の傾斜で減衰する。長い区間の消失が起こると、60ms 間で合成信号をゼロとする減衰がはたらく。

波形図"Concealed"はアルゴリズムの最終的な出力を示す。比較のために、消失無しの本来の波形をその下に示す。合成音声は消失前の音声と極めて似ており、原波形を良く近似している。合成波形を細かく見ると、消失区間の最初のピッチ周期は消失前の最後の周期から来ており、２番目の周期は消失の２周期前から来ており、３番目の周期は消失前の最初の周期を繰り返している。また、消失区間のピッチが変化するために、合成信号の最終区間のピークは原信号の最後のピッチと正確には一致していない。消失がより長くなるにつれて、消失の最後のＯＬＡ窓を広げなければならないのはこのためである。

１．３　注釈付きＣ＋＋コードによるアルゴリズムの詳細

ＰＬＣアルゴリズムは浮動小数点のＣ＋＋クラスで実現される。動作説明とともにコードがこの章で述べられる。コメントを含む約 360 行のＣ＋＋コードで完全に実現される。本付録には説明の目的でのみＣ＋＋コードが示されている。

１．３．１　型定義と定数

倍精度と単精度の浮動小数点演算の切り替を可能とするために、以下の型を定義する。

```
Typedef     float   Float;
```

倍精度モードに切り替えるためには、"float"を"double"に変えればよい。コードは以下のプリプロセッサ定数を定義する。

```
#define     PITCH_MIN       40                  /* minimum allowed pitch, 200 Hz */
#define     PITCH_MAX       120                 /* maximum allowed pitch, 66 Hz */
#define     PITCHDIFF       (PITCH_MAX - PITCH_MIN)
#define     POVERLAPMAX     (PITCH_MAX >> 2)    /* maximum pitch OLA window */
#define     HISTORYLEN      (PITCH_MAX * 3 + POVERLAPMAX) /* history buffer length*/
#define     NDEC            2                   /* 2:1 decimation */
#define     CORRLEN         160                 /* 20 ms correlation length */
#define     CORRBUFLEN      (CORRLEN + PITCH_MAX) /* correlation buffer length */
#define     CORRMINPOWER    ((Float)250.)       /* minimum power */
#define     EOVERLAPINCR    32                  /* end OLA increment per frame, 4 ms */
#define     FRAMESZ         80                  /* 10 ms at 8 KHz */
#define     ATTENFAC        ((Float).2)         /* attenuation factor per 10 ms frame */
#define     ATTENINCR       (ATTENFAC/FRAMESZ)/* attenuation per sample */
```

１．３．２　クラス宣言

```
 1 class LowcFE {
 2 public:
 3                  LowcFE();
 4          void    dofe(short *s);             /* synthesize speech for erasure */
 5          void    addtohistory(short *s);     /* add a good frame to history buffer */
 6 protected:
 7          int     erasecnt;                   /* consecutive erased frames */
 8          int     poverlap;                   /* overlap based on pitch */
 9          int     poffset;                    /* offset into pitch period */
10          int     pitch;                      /* pitch estimate */
11          int     pitchblen;                  /* current pitch buffer length */
12          Float   *pitchbufend;               /* end of pitch buffer */
```

```
13          Float     *pitchbufstart;                 /* start of pitch buffer */
14          Float     pitchbuf[HISTORYLEN];           /* buffer for cycles of speech */
15          Float     lastq[POVERLAPMAX];             /* saved last quarter wavelength */
16          short     history[HISTORYLEN];            /* history buffer */
17
18          void      scalespeech(short *out);
19          void      getfespeech(short *out, int sz);
20          void      savespeech(short *s);
21          int       findpitch();
22          void      overlapadd(Float *l, Float *r, Float *o, int cnt);
23          void      overlapadd(short *l, short *r, short *o, int cnt);
24          void      overlapaddatend(short *s, short *f, int cnt);
25          void      convertsf(short *f, Float *t, int cnt);
26          void      convertfs(Float *f, short *t, int cnt);
27          void      copyf(Float *f, Float *t, int cnt);
28          void      copys(short *f, short *t, int cnt);
29          void      zeros(short *s, int cnt);
30 };
```

クラスは低演算量フレーム消失補償の意味で LowcFE と呼ぶ。そのインタフェースは３つの public 関数で定義される。３行目のコンストラクタは内部変数を初期化する。５行目の addtohistory()関数は正常フレームを引き渡す。その引数、長さ FRAMESZ の short 配列のポインタは復号されたデータフレーム一つを含まなければならない。コードは short が 16bit 符号付きデータを含み、復号器出力が 16bit リニアＰＣＭデータであることを仮定している。dofe()関数は消失区間中の合成音声を生成する。

クラスの残りのメンバや関数は protected であり、クラスの使用者が、アクセス出来ない事を意味する。変数 erasecnt は消失区間中連続的な消失フレームの数を記録する。その値は０からスタートし、消失区間中 10ms 毎に１づつ加算されていき、消失後の最初の正常フレームで０にリセットされる。

変数 poverlap はＯＬＡ窓長で現ピッチの 1/4 に相当する。９行目の変数 poffset は現在のピッチバッファ内のオフセットで合成波形生成に使用される。10 行目の変数 pitch は現在のピッチ推定値である。変数 pitchblen はピッチバッファ全体のうち現在使用している長さである。この長さは消失区間が１フレーム以上になるとき変更される。変数 pitchbufend はピッチバッファの最後を指す。変数 pitchbufstart はピッチバッファ内で現在使用されている部分の開始位置を示すポインタである。14 行目の pitchbuf は消失開始点での履歴バッファの内容が記憶されている。このバッファは消失区間中、後尾 1/4 wavelength 以外については記憶内容を保持する。lastq バッファは消失開始直前の 1/4 wavelength を一時的に記憶し、ＯＬＡ演算に使用される。16 行目の履歴バッファは直近の信号履歴を記憶する。これは正常および消失フレーム区間で更新される。

18～29 行目の protected 関数はアルゴリズムの中核をなしており、これらについては後述する。

１．３．３　メインループ

ＴＴＣ標準ＪＴ－Ｇ７１１で LowcFE クラスを使用したプログラムのメイン処理ループが以下に示される。10ms のＴＴＣ標準ＪＴ－Ｇ７１１ビットストリームフレームがエラー無しで受信されれば、receiveframe()関数は真を返し、消失が発生すれば偽を返す。関数 g711dec()はＴＴＣ標準ＪＴ－Ｇ７１１ビットストリームを 16 ビットリニアＰＣＭに変換し、output()はフレーム消失を補償したオーディオ信号を出力する。これらの関数の

実現は本付録の範囲を越えており、ここでは述べない。

```
1 void process()
2 {
3           char        bitstream[FRAMESZ];
4           short       speech[FRAMESZ];
5           LowcFE      fec;
6           bool        frameisgood;
7
8           for(;;) {
9                       frameisgood = receiveframe(bitstream);
10                      if (frameisgood) {
11                              g711dec(bitstream, speech);
12                              fec.addtohistory(speech);
13                      } else
14                              fec.dofe(speech);
15                      output(speech);
16          }
17 }
```

10 行目では入力フレームが消失フレームであるかどうかを調べる。もし、正常フレームであれば、11 行目で復号器に送られる。復号器出力はメンバ関数 addtohistory()を用いるＰＬＣアルゴリズムに入力される。フレームが消失しているとメンバ関数 dofe()が呼ばれ合成音声を生成する。

12 行目の addtohistory 関数は履歴バッファに音声信号以上の範囲をコピーしている点に注意すべきである。また、addtohistory 関数は音声信号の内容を変更し、出力信号を POVERLAPMAX サンプル分遅延させ、さらに消失直後の正常フレームの場合には入力信号と合成信号をＯＬＡした後に戻る関数である。

１．３．４　ユーティリティメンバ関数

クラス宣言の 25～29 行目のユーティリティ関数は、単に short から Float への変換（convertsf()）とその逆変換(convertfs())、Float 配列のコピー（copyf()）と short 配列のコピー（copys()）そして short 配列のゼロクリア（zeros()）である。他のルーチンに使用されるので、これらを次に示す。

```
1 void LowcFE::convertsf(short *f, Float *t, int cnt)
2 {
3           for (int i = 0; i < cnt; i++)
4                       t[i] = (Float)f[i];
5 }
6
7 void LowcFE::convertfs(Float *f, short *t, int cnt)
8 {
9           for (int i = 0; i < cnt; i++)
```

```
10                      t[i] = (short)f[i];
11 }
12
13 void LowcFE::copyf(Float *f, Float *t, int cnt)
14 {
15          for (int i = 0; i < cnt; i++)
16                      t[i] = f[i];
17 }
18
19 void LowcFE::copys(short *f, short *t, int cnt)
20 {
21          for (int i = 0; i < cnt; i++)
22                      t[i] = f[i];
23 }
24
25 void LowcFE::zeros(short *s, int cnt)
26 {
27          for (int i = 0; i < cnt; i++)
28                      s[i] = 0;
29 }
```

convertfs ルーチンで飽和あるいは丸めは存在しない。

１．３．５　コンストラクタ

コンストラクタは、クラスの内部メンバを初期化する。

```
1 LowcFE::LowcFE()
2 {
3          erasecnt = 0;
4          pitchbufend = &pitchbuf[HISTORYLEN];
5          zeros(history, HISTORYLEN);
6 }
```

３行目において、現在消失が起きてないことをコードが知ることができるように erasent を 0 に設定する。次に、pitchbufend をピッチバッファの終点を指すように設定する。さらに、信号の最初において消失が起きた場合に人工的雑音が生じないように、履歴バッファをクリアする。

１．３．６　関数addtohistory および savespeech

次に、public インタフェース関数 addtohistory() について、addtohistory() 内で呼び出される protected 関数 savespeech とともに説明する。addtohistory() は、消失のないフレームを復号化した後で、信号を送出する前にアプリケーションによって呼び出される。

```
 1 /*
 2   * Save a frames worth of new speech in the history buffer.
 3   * Return the output speech delayed by POVERLAPMAX.
 4   */
 5 void LowcFE::savespeech(short *s)
 6 {
 7         /* make room for new signal */
 8         copys(&history[FRAMESZ], history, HISTORYLEN - FRAMESZ);
 9         /* copy in the new frame */
10         copys(s, &history[HISTORYLEN - FRAMESZ], FRAMESZ);
11         /* copy out the delayed frame */
12         copys(&history[HISTORYLEN - FRAMESZ - POVERLAPMAX], s, FRAMESZ);
13 }
14
15 /*
16   * A good frame was received and decoded.
17   * If right after an erasure, do an overlap add with the synthetic signal.
18   * Add the frame to history buffer.
19   */
20 void LowcFE::addtohistory(short *s)
21 {
22         if (erasecnt) {
23                 short overlapbuf[FRAMESZ];
24                 /*
25                   * longer erasures require longer overlaps
26                   * to smooth the transition between the synthetic
27                   * and real signal.
28                   */
29                 int olen = poverlap + (erasecnt - 1) * EOVERLAPINCR;
30                 if (olen > FRAMESZ)
31                         olen = FRAMESZ;
32                 getfespeech(overlapbuf, olen);
33                 overlapaddatend(s, overlapbuf, olen);
34                 erasecnt = 0;
35         }
36         savespeech(s);
37 }
```

22 行目において、addtohistory()は、このフレームが消失後の最初の正常フレームであるかどうかを調べる。直前のフレームで消失が起きた場合、erasecnt には消失した 10ms 長のフレームの数が入る。直前のフレームが消失しなかった場合、erasecnt は 0 である。

直前のフレームが消失した場合、23～34 行目において、合成信号の生成を続け、合成信号と入力信号とのO

ＬＡを行い、erasecnt をクリアする。29 行目はＯＬＡ窓長を決定する。変数 poverlap には現在の推定されたピッチ周期の 1/4 のサンプル数が入る。消失区間が 10ms（すなわち erasecnt=1）のみの場合、ＯＬＡ窓長は poverlap に設定される。複数のフレームが消失した場合は、ＯＬＡ窓長は消失 10ms ごとに 4ms 増加する。30,31 行目は、ＯＬＡ窓が長期の消失に対して 10ms を越えないように保証する。32 行目は、一時的なバッファである overlapbuf に合成音声を生成する。33 行目において、合成信号を入力信号にＯＬＡする。出力は s に置き、元の入力信号を置き換える。

36 行目は信号を履歴バッファに保存し、アルゴリズム遅延を加えるために savespeech()を呼ぶ。消失フレーム直後の最初のフレームでない場合は、原音が保存される。それ以外の場合は、33 行目におけるＯＬＡの結果が保存される。

５～13 行目の関数 savespeech()は、音声を履歴バッファに保存する。ＤＳＰにおいては巡回バッファで実現することができるが、シミュレーション目的の場合は内容をシフトする方が簡単である。８行目は新たなフレームのためのスペースを空けるためにバッファのシフトを行う。新たなフレームは 10 行目においてバッファの後部にコピーされ、12 行目は入力列の内容を POVERLAPMAX (30) サンプル数遅延した信号で置き換える。これにより、アルゴリズム遅延が 3.75ms になる。

１．３．７　関数dofe

public メンバ関数 dofe は、消失中の合成信号を生成し、ＰＬＣアルゴリズムの大部分を含む。

```
 1 /*
 2  * Generate the synthetic signal.
 3  * At the beginning of an erasure determine the pitch, and extract
 4  * one pitch period from the tail of the signal. Do an OLA for 1/4
 5  * of the pitch to smooth the signal. Then repeat the extracted signal
 6  * for the length of the erasure. If the erasure continues for more than
 7  * 10 ms, increase the number of periods in the pitchbuffer. At the end
 8  * of an erasure, do an OLA with the start of the first good frame.
 9  * The gain decays as the erasure gets longer.
10  */
11 void LowcFE::dofe(short *out)
12 {
13         if (erasecnt == 0) {
14                 convertsf(history, pitchbuf, HISTORYLEN);   /* get history */
15                 pitch = findpitch();                        /* find pitch */
16                 poverlap = pitch >> 2;                      /* OLA 1/4 wavelength */
17                 /* save original last poverlap samples */
18                 copyf(pitchbufend - poverlap, lastq, poverlap);
19                 poffset = 0;                                /* create pitch buffer with 1 period */
20                 pitchblen = pitch;
21                 pitchbufstart = pitchbufend - pitchblen;
22                 overlapadd(lastq, pitchbufstart - poverlap,
23                         pitchbufend - poverlap, poverlap);
24                 /* update last 1/4 wavelength in history buffer */
```

```
25                      convertfs(pitchbufend - poverlap, &history[HISTORYLEN-poverlap],
26                              poverlap);
27                      getfespeech(out, FRAMESZ);              /* get synthesized speech */
28              } else if (erasecnt == 1 || erasecnt == 2) {
29                      /* tail of previous pitch estimate */
30                      short tmp[POVERLAPMAX];
31                      int saveoffset = poffset;                       /* save offset for OLA */
32                      getfespeech(tmp, poverlap);             /* continue with old pitchbuf */
33                      /* add periods to the pitch buffer */
34                      poffset = saveoffset;
35                      while (poffset > pitch)
36                              poffset -= pitch;
37                      pitchblen += pitch;                     /* add a period */
38                      pitchbufstart = pitchbufend - pitchblen;
39                      overlapadd(lastq, pitchbufstart - poverlap,
40                              pitchbufend - poverlap, poverlap);
41                      /* overlap add old pitchbuffer with new */
42                      getfespeech(out, FRAMESZ);
43                      overlapadd(tmp, out, out, poverlap);
44                      scalespeech(out);
45              } else if (erasecnt > 5) {
46                      zeros(out, FRAMESZ);
47              } else {
48                      getfespeech(out, FRAMESZ);
49                      scalespeech(out);
50              }
51              erasecnt++;
52              savespeech(out);
53 }
```

13 行目において、erasecnt が０かどうかを調べる。もし０であれば、消失した最初のフレームであり、14～27 行目のコードが実行される。14 行目は履歴バッファの内容をピッチバッファにコピーし、その過程で浮動小数点に変換する。最後の 1/4 wavelength を除き、消失中はピッチバッファの内容は不変である。15 行目は findpitch を呼び、ピッチを推定するための正規化相互相関を計算する。関数 findpitch()は、MIN_PITCH(40)から MAX_PITCH(120)の間の値を返す。findpitch()は本アルゴリズムの演算量の大部分を占め、最初の消失フレームにおいてのみ呼ばれる。16 行目はＯＬＡ窓長をピッチ周期の 1/4 に設定する。18 行目においては、消失が 1 フレーム以上続いた場合に、ピッチバッファの最後の 1/4 wavelength を lastq に保存する。19～21 行目は、バッファの最終区間のみが合成音声の生成に用いられるようにピッチバッファを設定する。22 行目においては、lastq の内容と、ピッチバッファを 1.25 ピッチ周期戻った点を開始点としたピッチ周期の 1/4 とをＯＬＡする。これにより、例えばピッチ周期が 80 サンプル（10ms）以下など、一つの周期のピッチバッファが最初のフレームで繰り返される場合になめらかな遷移が保証されるのと同様に、消失の最初のところで元の音声と合成音声とのなめらかな遷移が保証される。

このＯＬＡの結果はピッチバッファの後部に置かれ、25 行目において履歴バッファの最後の 1/4 ピッチ周期

を置き換える。履歴バッファを更新することにより、52 行目において savespeech が呼ばれた時にＯＬＡされた音声が出力され、履歴バッファが不連続性を持たないように保証する。さらに、27 行目は getfespeech()を呼ぶことによってそのフレームの合成音声を生成する。getfespeech()は、ピッチバッファにおける最後のピッチ周期を、80 サンプルを満たすまで必要な回数繰り返すことによって出力列に単純にコピーする。27 行目の後は、コードは erasecnt をインクリメントする 51 行目に飛ぶ。次に 52 行目は合成音声で履歴バッファを更新する。関数 savespeech()もまた信号を遅延させ、そのため savespeech()から戻った時は、22 行目のＯＬＡの結果は出力の最初の 3.75ms に現れる。

消失の２番目および３番目のフレームにおいては、29～44 行目への分岐が行われる。この分岐は、ピッチを合成するために用いられるピッチバッファ内のピッチ周期の数を増加させる。ピッチ周期の数を増加させることにより、信号のバリエーションが増加し、これにより、単一のピッチ周期のみを用いた場合に生じる不自然で調波的な人工的雑音（ビープ音）が順々に低減する。30～32 行目においては、前の消失フレームで用いられたピッチバッファの出力を 1/4 ピッチ周期継続させ、一時的なバッファに置く。この信号は、ピッチバッファ内のピッチ周期の数が増加した場合に出力信号におけるなめらかな遷移を保証するために、新たに拡張されたピッチ周期バッファとＯＬＡする。

31 行目においては、ピッチ周期バッファへのオフセットを saveoffset に保存する。32 行目において古いピッチバッファが波形を生成した後、34 行目において poffset を saveoffset に保存する。これにより、合成信号の位相を維持することが保証される。35,36 行目は、ピッチ周期が最初のピッチ周期を示すまで poffset からピッチ周期を差し引く。

37 行目は周期をピッチバッファに加える。38 行目はピッチバッファの使用されている部分の開始点を指すポインタである pitchbufstart を更新する。次に 39 行目は、lastq に保存されたデータと pitchbufstart の前の 1/4 ピッチ周期とＯＬＡを行い、結果をピッチバッファの最後 1/4 ピッチ周期に置く。最初のフレームにおける場合と同様に、これにより、出力信号においてピッチ周期が複数回繰り返される場合にピッチバッファがなめらかになることが保証される。

42 行目において、新たなピッチバッファからの合成信号をフレーム間隔分抽出し、このデータの最初の 1/4 ピッチ周期を 32 行目で生成された一時的バッファとＯＬＡさせる。44 行目において、scalespeech を呼ぶことにより、合成信号を線形傾斜で減衰させる。この減衰は、10ms ごとに 20%の比率で減衰させるものであり、２番目の消失フレームの先頭で開始する。合成信号は最初の消失フレームにおいては減衰させないことに注意されたい。

４,５,６番目の消失フレームにおいては、48,49 行目に示すように、ピッチバッファが静的になるため、処理はより簡単になる。getfespeech()を呼ぶことにより合成信号を生成し、scalespeech()によって減衰する。46 行目において、60ms を越えた場合は、合成信号を 0 ゼロに設定する。

１．３．８　ピッチ検出

ピッチ検出器は、本アルゴリズムの中で演算量が多い唯一の部分である。その演算量を低く抑えるために、初めに間引き信号によって粗い探索が行われる。次に、その粗い探索のピークの近傍で、詳細な探索が行われる。最新の 20ms の信号について、PITCH_MIN から PITCH_MAX のラグについて、以前の信号との相互相関が求められる。

```
1 /*
2   * Estimate the pitch.
3   * l - pointer to first sample in last 20 ms of speech.
4   * r - points to the sample PITCH_MAX before l
```

```
 5  */
 6 int LowcFE::findpitch()
 7 {
 8          int         i, j, k;
 9          int         bestmatch;
10          Float       bestcorr;
11          Float       corr;               /* correlation */
12          Float       energy;             /* running energy */
13          Float       scale;              /* scale correlation by average power */
14          Float       *rp;                /* segment to match */
15          Float       *l = pitchbufend - CORRLEN;
16          Float       *r = pitchbufend - CORRBUFLEN;
17
18          /* coarse search */
19          rp = r;
20          energy = 0.f;
21          corr = 0.f;
22          for (i = 0; i < CORRLEN; i += NDEC) {
23                  energy += rp[i] * rp[i];
24                  corr += rp[i] * l[i];
25          }
26          scale = energy;
27          if (scale < CORRMINPOWER)
28                  scale = CORRMINPOWER;
29          corr = corr / (Float)sqrt(scale);
30          bestcorr = corr;
31          bestmatch = 0;
32          for (j = NDEC; j <= PITCHDIFF; j += NDEC) {
33                  energy -= rp[0] * rp[0];
34                  energy += rp[CORRLEN] * rp[CORRLEN];
35                  rp += NDEC;
36                  corr = 0.f;
37                  for (i = 0; i < CORRLEN; i += NDEC)
38                          corr += rp[i] * l[i];
39                  scale = energy;
40                  if (scale < CORRMINPOWER)
41                          scale = CORRMINPOWER;
42                  corr /= (Float)sqrt(scale);
43                  if (corr >= bestcorr) {
44                          bestcorr = corr;
45                          bestmatch = j;
46                  }
47          }
```

```
48          /* fine search */
49          j = bestmatch - (NDEC - 1);
50          if (j < 0)
51                  j = 0;
52          k = bestmatch + (NDEC - 1);
53          if (k > PITCHDIFF)
54                  k = PITCHDIFF;
55          rp = &r[j];
56          energy = 0.f;
57          corr = 0.f;
58          for (i = 0; i < CORRLEN; i++) {
59                  energy += rp[i] * rp[i];
60                  corr += rp[i] * l[i];
61          }
62          scale = energy;
63          if (scale < CORRMINPOWER)
64                  scale = CORRMINPOWER;
65          corr = corr / (Float)sqrt(scale);
66          bestcorr = corr;
67          bestmatch = j;
68          for (j++; j <= k; j++) {
69                  energy -= rp[0] * rp[0];
70                  energy += rp[CORRLEN] * rp[CORRLEN];
71                  rp++;
72                  corr = 0.f;
73                  for (i = 0; i < CORRLEN; i++)
74                          corr += rp[i] * l[i];
75                  scale = energy;
76                  if (scale < CORRMINPOWER)
77                          scale = CORRMINPOWER;
78                  corr = corr / (Float)sqrt(scale);
79                  if (corr > bestcorr) {
80                          bestcorr = corr;
81                          bestmatch = j;
82                  }
83          }
84          return PITCH_MAX - bestmatch;
85  }
```

findpitch()が呼び出される時、ピッチバッファの中身は、履歴バッファの中身と一致している。15 行目で、参照信号のポインタ l を消失区間の起点から 20ms 前のサンプルに設定する。16 行目で、ラグ信号のポインタ r を l のさらに MAX_PITCH サンプル前に設定する。19～29 行目で、2:1 に間引いた信号によって、ラグが MAX_PITCH の正規化相互相関を計算する。26～28 行目で、ゼロまたは非常に低いエネルギに弱められた区間

による割り算を避けるために、エネルギを最小レベル以下にならないようにする。

32～47 行目で、他の全てのラグについて、正規化相互相関計算を繰り返す。これは偶数のラグについてのみ調べられる。連続するエネルギの合計は保持されているので、繰り返し毎の更新には、２回の積和演算（ＭＡＣ）のみが必要とされる。相関のピークは bestcorr に保持され、対応するラグは bestmatch に保持される。

48～54 行目で、詳細な探索で使われる区間を計算する。もし粗い探索のピークが最小または最大ラグでなければ、３つのラグについて詳細な探索が行われる。55～83 行目で、粗い探索の計算の実行を、間引き無しで繰り返す。84 行目で、詳細な探索によるピークが、ピッチ推定値として返される。

１．３．９　合成信号生成と減衰

関数 getfespeech()はピッチバッファから合成波形を抽出し、一方、scalespeech は合成音声に減衰傾斜を適用する。

```
 1 /*
 2  * Get samples from the circular pitch buffer. Update poffset so
 3  * when subsequent frames are erased the signal continues.
 4  */
 5 void LowcFE::getfespeech(short *out, int sz)
 6 {
 7         while (sz) {
 8                 int cnt = pitchblen - poffset;
 9                 if (cnt > sz)
10                         cnt = sz;
11                 convertfs(&pitchbufstart[poffset], out, cnt);
12                 poffset += cnt;
13                 if (poffset == pitchblen)
14                         poffset = 0;
15                 out += cnt;
16                 sz -= cnt;
17         }
18 }
19
20 void LowcFE::scalespeech(short *out)
21 {
22         Float g = (Float)1. - (erasecnt - 1) * ATTENFAC;
23         for (int i = 0; i < FRAMESZ; i++) {
24                 out[i] = (short)(out[i] * g);
25                 g -= ATTENINCR;
26         }
27 }
```

5～18 行目の getfespeech()はピッチバッファから出力配列へ波形をコピーする以外にいくつかの処理を行っている。もし要求したコピーの大きさがピッチバッファよりも大きければ、出力ポインタ poffset はバッファの初めに戻り、そこからコピーを続ける。ピッチバッファは pitchbufstart から始まり、pitchblen サンプルの長さであ

る。変数 poffset はバッファ内の現在の位置である。poffset はそのループを繰り返す毎に更新され、戻り値として、出力すべき次のサンプルを指す。この方法により、getfespeech()が再び呼ばれた場合に、合成信号の生成を続行することができる。

20～27 行目の関数 scalespeech()は、１フレーム分の合成音声に減衰傾斜を適用する。これは、最初の消失フレームでは呼ばれない。２番目のフレームで、erasecnt はまだインクリメントされていないので１のままであり、よって利得は 1.0 で始まり、0.8 に向って減少していく。消失フレームが続く間、フレーム当たり 20%で減衰は続く。

１．３．１０　オーバラップ加算操作

オーバラップ加算操作で、ソースコードは完成する。３つのルーチンが提供される。そのうちの２つは、同じメンバ関数名 overlapadd であり、それらの引数の型を除き、まったく同じである。ピッチバッファに対しては、浮動小数点版のＯＬＡが呼ばれ、出力信号に対しては、固定小数点版のＯＬＡが呼ばれる。

```
 1 /*
 2   * Overlap add left and right sides
 3   */
 4 void LowcFE::overlapadd(Float *l, Float *r, Float *o, int cnt)
 5 {
 6          Float incr = (Float)1. / cnt;
 7          Float lw = (Float)1. - incr;
 8          Float rw = incr;
 9          for (int i = 0; i < cnt; i++) {
10                    Float t = lw * l[i] + rw * r[i];
11                    if (t > 32767.)
12                              t = 32767.;
13                    else if (t < -32768.)
14                              t = -32768.;
15                    o[i] = t;
16                    lw -= incr;
17                    rw += incr;
18          }
19 }
20
21 void LowcFE::overlapadd(short *l, short *r, short *o, int cnt)
22 {
23          Float incr = (Float)1. / cnt;
24          Float lw = (Float)1. - incr;
25          Float rw = incr;
26          for (int i = 0; i < cnt; i++) {
27                    Float t = lw * l[i] + rw * r[i];
28                    if (t > 32767.)
29                              t = 32767.;
30                    else if (t < -32768.)
```

```
31                                      t = -32768.;
32                              o[i] = (short)t;
33                              lw -= incr;
34                              rw += incr;
35              }
36 }
```

ＯＬＡは引数 cnt の長さを基にルーチン内で計算される三角窓を使う。６行目で、サンプル毎の窓の増加量を計算し、７,８行目で、左上がり（右下がり）窓の重み lw および右上がり窓の重み rw を初期化する。10～17行目で、各サンプルに重みを適用し、16 ビットの整数値に制限し、結果を出力し、次の繰り返しのために重みを更新する。

もう一つのＯＬＡルーチン overlapaddatend()は、消失区間後の最初の正常フレームで呼ばれる。このルーチンは、合成音声を入力信号と組み合わせる前に減衰ファクタでスケーリングする点において、上の２つのルーチンと異なる。

```
 1 /*
 2   * Overlap add the end of the erasure with the start of the first good frame
 3   * Scale the synthetic speech by the gain factor before the OLA.
 4   */
 5 void LowcFE::overlapaddatend(short *s, short *f, int cnt)
 6 {
 7              Float incr = (Float)1. / cnt;
 8              Float gain = (Float)1. - (erasecnt - 1) * ATTENFAC;
 9              if (gain < 0.)
10                              gain = (Float)0.;
11              Float incrg = incr * gain;
12              Float lw = ((Float)1. - incr) * gain;
13              Float rw = incr;
14              for (int i = 0; i < cnt; i++) {
15                              Float t = lw * f[i] + rw * s[i];
16                              if (t > 32767.)
17                                              t = 32767.;
18                              else if (t < -32768.)
19                                              t = -32768.;
20                              s[i] = (short)t;
21                              lw -= incrg;
22                              rw += incr;
23              }
24 }
```

１．４　演算量と遅延

本アルゴリズムの演算量は、ＤＳＰで約 0.5MIPS のピークレートを持つと見積もられる。その平均値ははるかに低い。演算量は、ピッチ検出ルーチンの相互相関区間の計算によって占められる。この計算は、消失区間

の最初のフレームの間にのみ発生する。他の全てのフレームでは、１サンプル当たり数回の積和演算（ＭＡＣ）命令程度で、演算量は非常に少ない。

演算量は次のように見積もられる。消失区間の最初の消失フレームで、本アルゴリズムはピッチを推定し、ピッチ周期の 1/4 の区間でＯＬＡを実行しなければならない。ピッチの最大値は 120 サンプルであり、よってこの 1/4 は 30 である。findptitch()と overplapadd()ルーチンは、以下の実行カウントを持つ。

Routine	**MAC**	**Compare**	**div**	**sqrt**
findpitch()	3764	86	44	44
overlapadd()	121	0	1	0
Total	3885	86	45	44

比較を２サイクル、除算や平方根を 10 サイクルと仮定すると、この結果として 3885 + 86 ＊ 2 + (45 + 44) ＊ 10 = 4947 サイクルとなる。これが 10ms フレームの中で生じるので、100 を掛けて 0.5MIPS となる。

前述のように、本アルゴリズムには 3.75ms のアルゴリズム遅延がある。演算遅延を最小にするために、ＰＬＣアルゴリズムは、次のフレームが消失されたことを知る前に、わずかなメモリコストで、各正常フレームの後で実行することができる。もし次のフレームが消失すれば、合成信号が直ちに利用される。もし次のフレームが消失しなければ、合成信号はただ単に捨てられる。

参考文献

[1]ＴＴＣ標準ＪＴ－Ｇ７２３．１
- マルチメディア通信伝送のための 5.3 および 6.3kbit/s デュアルレート音声符号化方式

[2]ＴＴＣ標準ＪＴ－Ｇ７２８
- 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

[3]ＴＴＣ標準ＪＴ－Ｇ７２９
- 8kbit/s ＣＳ－ＡＣＥＬＰを用いた音声符号化方式

付録２

（標準ＪＴ－Ｇ７１１に対する）

パケットベースマルチメディア通信システムにおける標準ＪＴ－Ｇ７１１向け

擬似背景雑音ペイロード定義

２．１　本付録の規定範囲

この付録はパケットベースマルチメディア通信システム用にＴＴＣ標準ＪＴ－Ｇ７１１を使用するための擬似背景雑音ペイロードフォーマット（またはビットストリーム）を定義している。これには汎用性があり、ＴＴＣ標準ＪＴ－Ｇ７２６[1]、ＪＴ－Ｇ７２７[2]、ＪＴ－Ｇ７２８[3]、ＪＴ－Ｇ７２２[4]のような不連続伝送（ＤＴＸ）の能力が方式に組み込まれていない他の音声コーデックにも使用できる。このペイロードフォーマットは擬似背景雑音パラメータの通信に必要な最低限の相互接続仕様を与えている。有音検出（ＶＡＤ）やＤＴＸアルゴリズムと同様に、擬似背景雑音の分析と合成については指定せずに実現に任せる。しかしながら、一つの実装例について評価試験を行っており、ここで述べるものとする。その方法はＴＴＣ標準ＪＴ－Ｇ７２９付属資料Ｂ[5]のＶＡＤとＤＴＸ、そして、参考として示した擬似背景雑音発生アルゴリズム（ＣＮＧ）を使う。

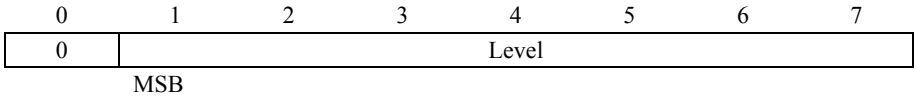
このペイロードフォーマットの使用は、ヘッダのオーバヘッドが大きく、パケット伝送レートが全体システムのビットレートに重要な役割を果たすようなパケットベースのシステムを想定している。このときには、ＶＡＤ／ＤＴＸ／ＣＮＧを使うことでパケット伝送レートを大幅に低減することができ、帯域幅の利用効率が改善される。

２．２　擬似背景雑音のペイロード定義

擬似背景雑音のペイロードは雑音レベルの記述と反射係数で表されたスペクトル情報から構成される。スペクトル情報の使用はオプションであり、全極モデルの次数は未指定としておく。符号器は品質、演算量、想定される環境雑音、信号の帯域幅などを考慮して適切なモデルの次数を決めることができる。モデルの次数は受信側でペイロードの長さから導出できるので、陽には伝送されない。演算量や他の理由のために、復号器は高いほうの次数の反射係数をゼロに設定することにより、モデルの次数を減少させることが可能である。

２．２．１　雑音レベル

雑音レベルは-dBov で表され、０から 127 の値で 0 dBov から-127dBov を表す。単位の dBov はシステムのオーバロードに対する相対レベルである。雑音レベルは付図２－１に従って、非使用ビットを常に０に設定し、ＭＳＢ（Most Significant Bit）から順に詰めていく。



付図２－１／JT-G711　Noise level bit packing

(ITU-T G.711)

２．２．２　反射係数

スペクトル情報は反射係数[6]を使って伝送される。線形予測分析により得られる多項式：

$$A(z) = 1 - \sum_{j-1}^{M} \alpha_j z^{-j}$$

から、以下のバックワード再帰式を使って一組のＬＰＣ係数から一組の反射係数を得ることができる。

$$k_i = -a_i^{(i)}$$

$$a_j^{(i-1)} = \frac{a_j^{(i)} + a_i^{(i)} a_{i-j}^{(i)}}{1 - k_1^2} \qquad 1 \le j \le i-1$$

ここで、i は以下の初期条件の下に、M、M-1 と減少し、１までの値をとる。

$$a_j^{(M)} = \alpha_j \qquad 1 \le j \le M$$

上式から以下のように k_1 が求まる。

$$k_1 = -\frac{r_i}{r_0}$$

ここで、r_i は入力信号の第 i 番目の自己相関係数である。

それぞれの反射係数は－１と１の間の値を持ち、８ビットで一様量子化される。量子化値は８ビットのインデックスＮによって表される。ここで N= 0 , ... , 254 であり、N=255 は将来の使用のために残しておく。インデックスＮはそれぞれＭＳＢから順に別々のバイトに詰められる。反射係数の量子化値は対応するインデックスから次式を使って求めることができる。

$$\hat{k}_i(N_i) = \frac{258}{32768} \cdot (N_i - 127) \quad \text{for } N_i = 0, \ldots, 254; -1 < \hat{k}_i(N_i) < 1$$

２．２．３　ペイロードパッキング

ペイロードの１バイト目は、付図２－１に示すように雑音レベルを含んでいなければならない。２バイト目以降には付図２－２のように量子化された反射係数が次数の低いものから順に詰められる。Mはモデルの次数である。

Byte	1	2	3	...	M+1
	Level	N_1	N_2	···	N_M

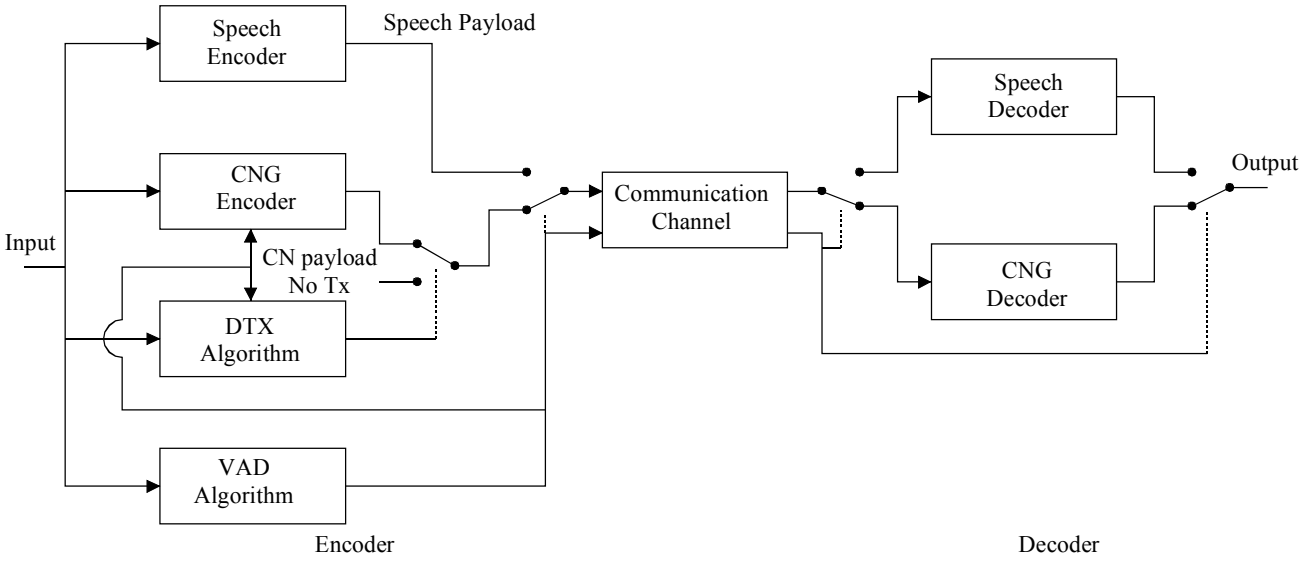
付図２－２／JT-G711　CN payload packing format

(ITU-T G.711)

ペイロードのトータルの長さはM＋１バイトである。０次のモデル（すなわち、スペクトル包絡の情報なし）の場合は、エネルギのレベルを伝送するだけになることに注意されたい。

２．３　使用のためのガイドライン

ＶＡＤ／ＤＴＸ／ＣＮＧ機能を持つ音声通信システムのブロック図を付図２－３に示す。ＶＡＤの機能は入力信号における音声区間の有音と無音を識別することである。無音区間中において、ＣＮＧの役割は伝送レートを最小にしつつ背景雑音を十分に記述することである。雑音の記述を含む無音挿入記述子（ＳＩＤ）フレームはＣＮペイロードに格納され、受信側に送られる。ＤＴＸアルゴリズムはＳＩＤフレームがいつ伝送されるかを決定する。ＳＩＤフレームは周期的または背景雑音の特徴が大きく変化した場合のみ送られる。受信側におけるＣＮＧアルゴリズムはＳＩＤの情報を用いて雑音発生モデルを更新し、適切な量の擬似背景雑音を生成する。



付図２－３／JT-G711　Speech communication system with DTX
(ITU-T G.711)

２．３．１　システムの性能に影響を与える要因

ＶＡＤ／ＤＴＸ／ＣＮＧ構成の目的は出力品質を許容できるレベルに維持しつつ、無音区間の伝送レートを低減することである。各構成の性能が品質と効率の両方に影響を与える。ＶＡＤ、ＤＴＸ、ＣＮＧアルゴリズムの特徴を合わせて考慮するように注意する必要がある。さもなければ、そのシステムは十分な性能を発揮することができない可能性がある。

２．３．１．１　ＶＡＤ

ＶＡＤアルゴリズムの役割は入力信号を有音と、無音または背景雑音とに分類することである。無音を有音と誤って分類すると、不必要に伝送レートを増やすことになり、システム効率に逆効果を与える。この場合、音声品質は影響されることはない。しかしながら、有音を無音として誤った場合は、音声信号が欠落してしまい、音声品質が劣化する。多くのＤＴＸアルゴリズムでは有音から無音に移行するときに音声の語尾が欠落しないようにハングオーバ期間を用いる。ハングオーバ期間中には無音フレームは有音として再分類される。ハングオーバ期間はＣＮＧ符号器が背景雑音を正確に推定できるためにも重要である。

２．３．１．２　ＤＴＸ

ＤＴＸアルゴリズムは無音区間中におけるＳＩＤフレーム伝送の周波数を決定する。単純なＤＴＸは周期的（例えば５Hz～30Hz）に更新する。複雑なＤＴＸアルゴリズムは入力信号を分析し、背景雑音特性の大きな変化が検出されたときのみ伝送する[5]。

２．３．１．３　ＣＮＧ

ＣＮＧの役割は背景雑音を記述し再生することである。雑音はそのエネルギとスペクトル特性を用いて十分に記述することができる。擬似背景雑音特性が急激に変化することを避けるために、ある期間にわたってパラメータの推定値を平均することが重要である。ＤＴＸの更新レートと同様に、平均をとる期間の適切な長さは背景雑音やＶＡＤの性能とハングオーバに依存する。

使用するモデルの次数はスペクトル推定の精度の一要因である。最適な次数は現在の背景雑音や信号帯域幅に依存する。ＣＮＧによって生成された雑音のスペクトル特性と音声コーデックのスペクトル特性を整合させ

ることも重要である。よって、音声符号器内で分析前に行われる入力信号へのあらゆる前処理は、ＣＮＧ符号器の内部でも行うべきである。

２．３．２　パケット網に適用した場合の帯域削減効果

付表２－１／JT-G711 は、パケット伝送システムにおける不連続伝送の使用が、どの程度伝送レートを削減することができるのか、また、それによってどの程度帯域幅の利用効率が改善されるのか、を説明している。本例は１パケット当たり 40 バイトのヘッダがあり、有音率が 60%、10Hz のＤＴＸ更新レートを仮定している。

付表２－１／JT-G711　Bandwidth Savings

(ITU-T G.711)

Codec	Bit rate (bit/s)	Packet size (ms)	IP bit rate (bit/s)	1 byte CN payload		11 byte CN payload	
				IP bit rate (Ave. bit/s)	Savings (%)	IP bit rate (Ave. bit/s)	Savings (%)
G.711	64 000	5 ms	128 000	78 112	39.0	78 432	38.7
G.711	64 000	10 ms	96 000	58 912	38.6	59 232	38.3
G.711	64 000	20 ms	80 000	49 312	38.4	49 632	38.0
G.726	32 000	5 ms	96 000	58 912	38.6	59 232	38.3
G.726	32 000	10 ms	64 000	39 712	38.0	40 032	37.5
G.726	32 000	20 ms	48 000	30 112	37.3	30 432	36.6
G.728	16 000	5 ms	80 000	49 312	38.4	49 632	38.0
G.728	16 000	10 ms	48 000	30 112	37.3	30 432	36.6
G.728	16 000	20 ms	32 000	20 512	35.9	20 832	34.9

例：40 バイトの RTP/UDP/IP ヘッダ、有音率 60%、10Hz のＤＴＸ更新レートを仮定すると、ＪＴ－Ｇ７１１で 11-byte ＣＮペイロードでの平均ＩＰビットレートは次式で与えられる。((64 000 bit/s) + (40 bytes × 8 bit/byte × (1.0/0.005 s))) × (0.6) + ((40+11) bytes × 8 bit/byte × 10/s) × (0.4) = 78 432 bit/s.

２．４ 性能評価結果

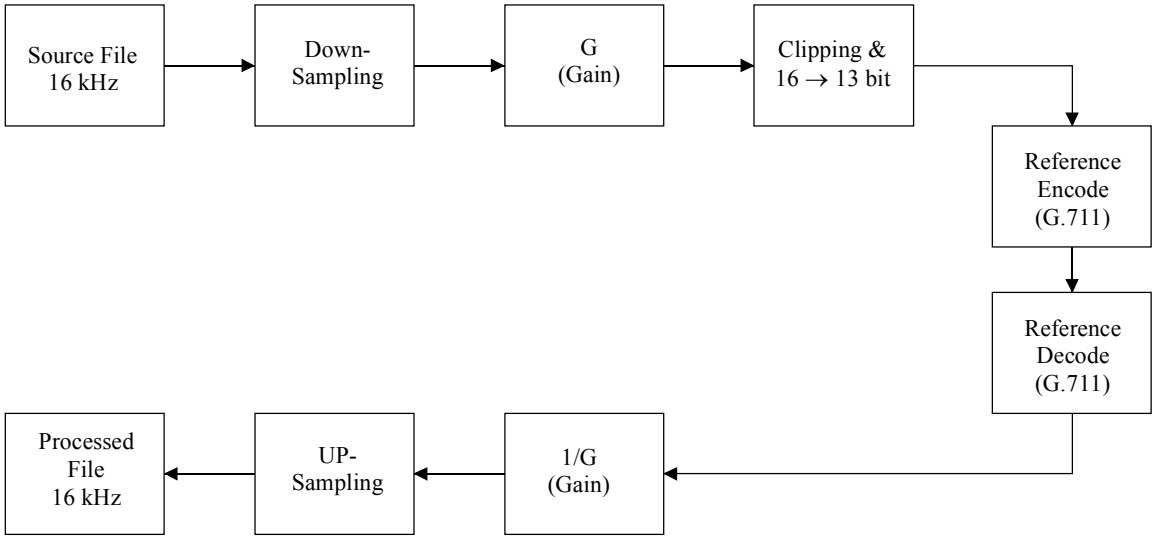
ＣＮペイロードを用いたＣＮＧ実装例の主観評価を行った。評価方法はＩＴＵ－Ｔ勧告Ｐ．８００で定義されるＡＣＲ（Absolute Category Rating）法を使用した。評価には、簡単で、意味の有る、短い、北米語による音声サンプルを使用した。音声サンプルは、Modified IRS フィルタ(ＩＴＵ－Ｔ勧告Ｐ．８３０ＡＮＮＥＸ－Ｄ)で処理され、二つの文章をペアにして構成した。それぞれの文章ペアは、文章間の時間間隔約１秒で、約７～８秒の長さであった。評価には雑音のない入力条件と、雑音のある入力条件（babble、street、office、car）の双方を使用した。

実験に用いた音声コーデックは、付図２－４の手順に従って処理されたＴＴＣ標準ＪＴ－Ｇ７１１である。この実験で実装したのはＣＮＧアルゴリズムのみである。ＶＡＤとＤＴＸアルゴリズムはＴＴＣ標準ＪＴ－Ｇ７２９付属資料Ｂ[5]のものを用いた。ＶＡＤとＤＴＸの判定を含むトレースファイルは、出力ファイルを入力ファイルに同期させるために SYNC フラグを立てて、付図２－５の手順に従って得た。

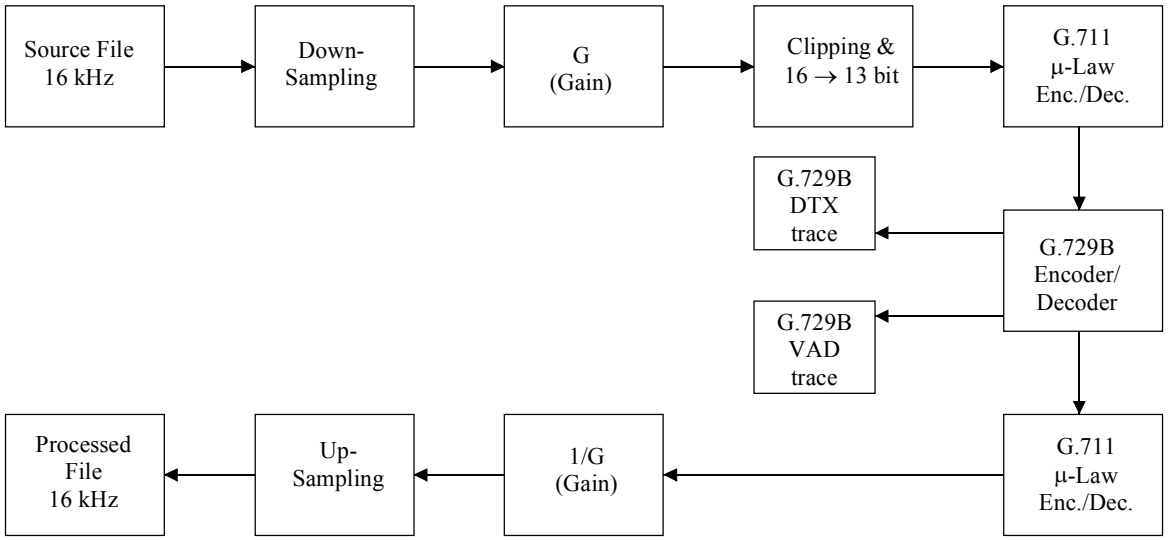
擬似背景雑音を用いるＴＴＣ標準ＪＴ－Ｇ７１１は付図２－６の手順から得た。ソースファイルをダウンサンプリングした後、利得Ｇでレベル調整し、ＴＴＣ標準ＪＴ－Ｇ７１１とＣＮＧの組み合わせによって符号化した。入力データは 10ms フレームにバッファリングした。ＣＮＧアルゴリズムの符号化フレームは、ＶＡＤとＤＴＸのトレースファイルに対応したフレームと同期化するために、音声ファイルの先頭に合わせた。10ms フレームのもとに、ＶＡＤとＤＴＸのトレースファイルを、ＣＮＧアルゴリズムの動作を制御するために使用した。有音フレームについては、入力データフレームを処理するためにＴＴＣ標準ＪＴ－Ｇ７１１を使用した。

無音フレームに関しては、ＣＮＧアルゴリズムを使用した。ＤＴＸフラグはＣＮＧパラメータの更新を制御する。復号器においては、ＶＡＤフラグを現在のフレームが有音か無音かを示すために使用した。その後、受聴レベルを固定にするための利得 1/G を適用し、結果をアップサンプリングし、これを“processed ファイル”として蓄積した。

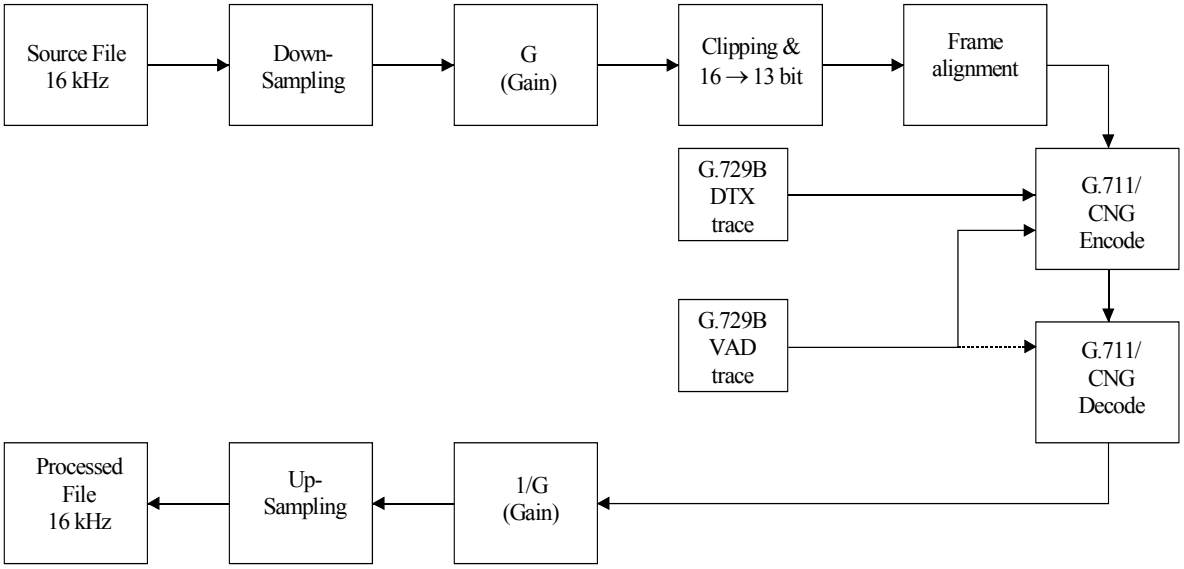
この雑音を含んだＡＣＲの実験結果は、全ての場合において、試験用に実装したＣＮＧアルゴリズムを用いたＴＴＣ標準ＪＴ－Ｇ７１１が、ＶＡＤ／ＣＮＧ無しのＴＴＣ標準ＪＴ－Ｇ７１１と同等の性能であることを示した。このことは雑音のある場合（babble、car、office、street）と同様に背景雑音の無い場合を含んでいる。



付図２－４／JG-711　Processing G.711 without CNG
(ITU-T G.711)



付図２－５／JT-G711　G.729B processing to obtain VAD/DTX trace files
(ITU-T G.711)



付図２－６／JT-G711　Processing G.711 with CNG
(ITU-T G.711)

２．５　ＣＮＧの実装例

本節では、本付録で述べている擬似背景雑音ペイロードフォーマットを用いた擬似背景雑音の一生成手法について述べる。この手法は、２．４節で述べた評価において用いられたものである。

２．５．１　アルゴリズムの詳細

２．５．１．１　符号器

符号器は、プログラムからフレームごとに呼び出す必要がある。有音フレームの場合、入力信号を前処理し、プログラムへリターンする前に内部バッファを更新する。無音フレームの場合、背景雑音のエネルギおよびスペクトル特性の推定値を更新する。ＳＩＤフレームの場合、推定したパラメータを量子化し、復号器に送信するためのチャネルバッファに格納する。ＳＩＤ更新のレートは、ＴＴＣ標準ＪＴ－Ｇ７２９付属資料Ｂ[5]のＤＴＸによって決定した。以下にＣＮＧ符号器の詳細を述べる。

２．５．１．１．１　前処理

すべての不要な低周波成分を除去するために１次の高域通過型のＩＩＲフィルタによって入力信号を前処理する。高域通過型フィルタは以下の式で与えられる。

$$H(z) = \frac{1 - z^{-1}}{1 - (127/128)z^{-1}}$$

２．５．１．１．２　自己相関解析

25ms の非対称窓を適用した前処理信号をもとに、正規化自己相関係数 r_m およびフレームのエネルギ E を計算する。サンプリング周波数が 8.0kHz の場合、窓は以下の式で与えられる。

$$w(n) = \begin{cases} 0.54 - 0.46\cos\left(\dfrac{2\pi \mathrm{n}}{339}\right) & \mathrm{n} = 0, 1, \ldots, 169 \\ \cos\left(\dfrac{2\pi(\mathrm{n} - 170)}{119}\right) & \mathrm{n} = 170, 171, \ldots, 199 \end{cases}$$

次に、i 番目の正規化自己相関係数およびフレームのエネルギの移動平均を以下の式によって計算する。

$$\bar{r}_m(i) = \bar{r}_m(i-1)\cdot\beta_1 + \bar{r}_m(i)\cdot(1.0 - \beta_1) \qquad m = 1, 2, \ldots, \mathrm{M}$$
$$\overline{LE}(i) = \overline{LE}(i-1)\cdot\beta_2 + \overline{LE}(i)\cdot(1.0 - \beta_2)$$

ただし、$\overline{LE}$ はフレームのエネルギの底が２の対数であり、M はモデルの次数である。β_1 および β_2 はフレームの大きさに依存する定数である。フレームの大きさが 7.5ms 以下の場合、β_1 および β_2 は 0.8 にセットし、その他の場合は 0.6 にセットする。直前のフレームが有音の場合、これらの平均は現在のフレームの値にリセットする。

２．５．１．１．３　反射係数の計算

フレームの正規化自己相関係数と、平均の正規化自己相関係数との自乗平均誤差を、以下の式によって計算する。

$$d = \frac{1}{M}\Sigma_{m=1}^{M}\left(\bar{r}_m(i) - r_m(i)\right)^2$$

d が適応的に決定される閾値 Th よりも小さくかつ前のフレームが無音の場合は平均の係数 $\overline{r_m}(i)$ を反射係数の計算に用い、その他の場合はその時点の係数 $r_m(i)$ を用いる。閾値 Th は以下のアルゴリズムによってフレームごとに決められる。

```
if (PrevVad == 1)
        Th = 0.0
else
        Th += 0.2857＊(FRAME_SIZE/SAMPLING_RATE)
        if (Th > 0.06)
              Th = 0.06
        End
End
```

反射係数 $k_m(i)$は、選ばれた自己相関係数から、Levinson-Durbin のアルゴリズムによって計算する。

２．５．１．１．４　量子化

無音挿入記述子（ＳＩＤ）フレームの場合、エネルギ $LE(i)$および反射係数 $k_m(i)$を量子化し、指定されたペイロードフォーマットに基づいて格納する。

２．５．１．２　復号器

復号器は、スケーリングされた白色雑音励振信号を線形予測合成フィルタに通すことによって、擬似背景雑音を生成する。以下に詳細を述べる。

２．５．１．２．１　パラメータ更新

現在のフレームでは、最後に受信したＳＩＤフレームの反射係数を用いる。擬似背景雑音のパラメータをエネルギが dBov から底が２の対数に変換された LE_{SID} で示すものとする。現在のフレームで用いるエネルギは以下の式で与えられる。

$$LE(i) = LE(i-1)\cdot\alpha + LE_{SID}\cdot(1.0-\alpha)$$

ただし、α＝0.9 である。擬似背景雑音の信号エネルギの急激な変化を避けるために、平滑化処理を行う。

２．５．１．２．２　励振信号の生成

ガウス分布の乱数発生器を用いて系列 *Rn* を生成する。系列 *Rn* は正しいエネルギとなるように、次式で与えられる係数 η でスケーリングされる。

$$\eta = \sqrt{\frac{E(i)\cdot\prod_{m=1}^{M}\left(1.0-\hat{k}(N_m)^2\right)}{\frac{1}{L}\cdot\Sigma_{j=0}^{L-1}Rn(j)^2}}$$

ここで、*L*は励振信号の長さであり、*E(i)*はフレームのエネルギである。

上式の分母を定数に近似することにより、内積演算を回避して演算量を削減することができる。

２．５．１．２．３　ＬＰ合成

反射係数は、次の再帰式[6]によって、線形予測（ＬＰ）合成フィルタで用いるために線形予測係数に変換される。

$$a_i^{(i)} = -\hat{k}_i(N_i)$$
$$a_j^{(i)} = a_j^{(i-1)} + \hat{k}_i(N_i)a_{i-j}^{(i-1)} \qquad 1\le j\le i-1$$

i=1,2,...,M について求め、線形予測係数が得られる。

$$\alpha_j = a_j^{(p)} \qquad 1\le j\le M$$

線形予測合成フィルタは次のように定義される。

$$\frac{1}{A(z)} = \frac{1}{1-\sum_{j=1}^{M}\alpha_j z^{-j}}$$

スケーリングされた励振信号は、最終的な擬似背景雑音を生成するためにフィルタへ通過させる。一般に、励振信号の長さＬはフレーム長と等しい。しかし、有音フレームに続く最初の無音フレームでは、Ｌはフレーム長にモデルの次数オーダ（Ｍ）を加えたものと等しくなる。この場合、合成フィルタからの最初のＭ個の出力サンプルは無視される。

２．５．１．３　遅延

擬似背景雑音アルゴリズムに固有の遅延はない。

２．５．１．４　メモリ量・演算量

本アルゴリズムは、ＩＴＵ－Ｔソフトウェアツールライブラリを用いた 16 ビット固定小数点演算によって実現された。標本化周波数 8.0kHz、10 次の全極モデルとして動作する時の、いろいろなフレームサイズにおけるメモリ量と演算量を付表２－２にまとめる。WMOPS はライブラリ内の動作カウンタを用いて得られた最悪値を表す。ＲＯＭは典型的な固定小数点型ＤＳＰ上で見積もられたサイズである。

付表２－２／JT-G711　CNG Resource Requirements for a 10th order model
(ITU-T G.711)

Frame Size	RAM (words)	ROM (words)	WMOPS
5 ms	650	1300	1.1
10 ms	690	1300	0.66
20 ms	760	1300	0.47

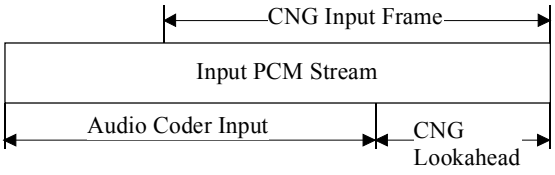
２．５．２　ＣＮＧの設定条件

付表２－３に試験に用いたＣＮＧの設定条件を示す。

付表２－３／JT-G711　CNG Tested Configuration
(ITU-T G.711)

Parameter	As Tested
Sampling Rate	8.0 kHz
Frame Size	10 ms
Model Order	10
Look-Ahead Delay	5 ms

付図２－７に示すように、試験においては音声コーデック（ＴＴＣ標準ＪＴ－Ｇ７１１）への入力を遅延させることによって、５ms の先読みを加えた。先読みは、ＣＮＧの実装例に、ＴＴＣ標準ＪＴ－Ｇ７２９付属資料ＢのＶＡＤの使用法を適切に合わせるために導入した。先読みの遅延は、ＴＴＣ標準ＪＴ－Ｇ７２９付属資料ＢのＶＡＤに、余分にハングオーバフレームを加えることで、実際には無効にできる。



付図２－７／JT-G711　CNG Look-ahead during Testing
(ITU-T G.711)

参考文献

[1]ＴＴＣ標準ＪＴ－Ｇ７２６

- 40,32,24,16kbit/s 適応差分パルス符号変調方式

[2]ＴＴＣ標準ＪＴ－Ｇ７２７

- 5 ビット、4 ビット､3 ビット及び 2 ビット/サンプルエンベデッド適応差分パルス符号変調

[3]ＴＴＣ標準ＪＴ－Ｇ７２８

- 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

[4]ＴＴＣ標準ＪＴ－Ｇ７２２

- 64kbit/s 以下の 7kHz オーディオ符号化方式

[5]ＴＴＣ標準ＪＴ－Ｇ７２９付属資料Ｂ

- （標準 JT-G729 に対する）ITU－T 勧告Ｖ．70 端末に適した標準 JT-G729 に対する無音圧縮手法

[6] RABINER (L.R.), SCHAFER (R.W.): Digital processing of speech signals, Prentice-Hall, 1978.

[7]ＩＴＵ－Ｔ勧告Ｇ．１９１

- Software tools for speech and audio coding standardization.

付録３

（標準ＪＴ－Ｇ７１１に対する）

オーディオ品質向上ツールボックス

３．１　本付録の規定範囲

この付録は、標準ＪＴ－Ｇ７１１符号化器に対するオーディオ品質向上を提供するツールボックスの説明を含んでいる。

この付録は以下の構成となっている。参照文献、定義、略語と頭字語、この付録を通して使われる表記法がそれぞれ３．２節、 ３．３節、 ３．４節、 および３．５節に定義されている。３．６節では４つのアルゴリズムの一般的な概要が述べられる。ノイズシェーピング（ＮＳ）は３．７．１節に、フレーム消失補償（FERC）は３．８．１節に、ノイズゲート（NG）とポストフィルタ（PF）はそれぞれ３．８．２節、および３．８．３節に記載されている。３．９節はソフトウェアについて記載しており、16-32 ビット固定小数点でこのツールボックスを定義している。

３．２　参照文献

(1) ＩＴＵ－Ｔ勧告Ｇ．１９１

Software tools for speech and audio coding standardization.

(2) ＩＴＵ－Ｔ勧告Ｇ．１９２

A common digital parallel interface for speech standardization activities.

(3) ＴＴＣ標準ＪＴ－Ｇ７１１．１

Ｇ．７１１パルス符号変調に対する広帯域エンベデッド拡張

３．３　定義

本節は意図的にブランクとしている。

３．４　略語と頭字語

この付録は付表３－１／JT-G.711 に載せた略語と頭字語を使う。

付表３－１／JT-G.711 － Glossary of abbreviations and acronyms

(ITU-T G.711)

Acronym	Description
FERC	Frame Erasure Concealment
NB	Narrow-Band
NG	Noise Gate
NS	Noise Shaping
PCM	Pulse Code Modulation
PF	PostFilter
WMOPS	Weighted Millions of Operations Per Second

３．５　表記法

時間領域の信号は、例えば s(n)のように、そのシンボルと丸括弧にはさまれたサンプルインデックスによって表現される。n の値はサンプルインデックスとして使われる。

付表３－２／JT-G.711 には本付録を通して最も関連のあるシンボルを記載している。

付表３－２／JT-G.711 － Glossary of most relevant symbols

(ITU-T G.711)

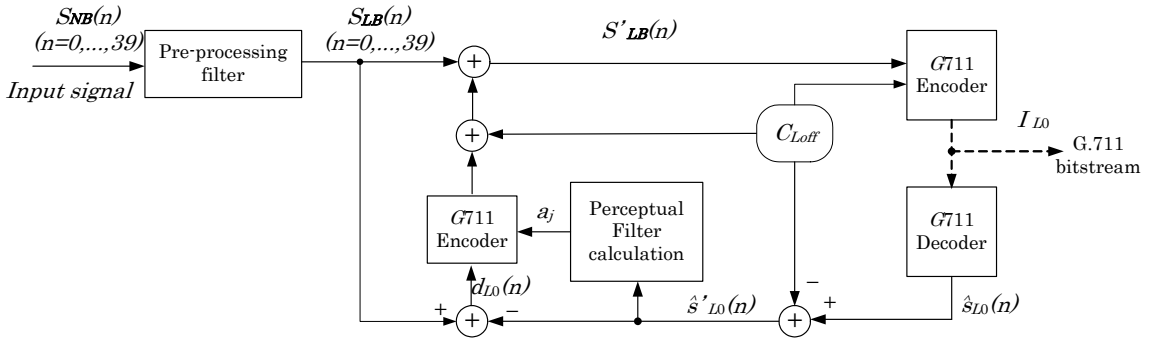
Type	Name	Description
Filters	$F(z)$	Perceptual weighting filter
Signals	$s_{NB}(n)$	Input signal
	$s_{LB}(n)$	Pre-processed input signal
	$s'_{LB}(n)$	Perceptually weighted target signal
	$d_{L0}(n)$	Difference signal of $s_{LB}(n)$ and $s'_{LB}(n)$
	$s'_{L0}(n)$	Decoded signal of ITU-T G.711 bit stream, without offset c_{Loff}
	$\hat{s}_{L0}(n)$	Decoded signal of ITU-T G.711
	$\hat{s}_{LB1}(n)$	Signal after decoding and FERC
	$\hat{s}_{LB}(n)$	Signal after postfilter
	$\hat{s}_{NB}(n)$	Signal after noise gate
Parameters	c_{Loff}	Encoder offset value
	a_i	LP coefficient of the perceptual filter
	I_{L0}	ITU-T G.711 compatible bit stream

３．６　ツールボックス概略

このツールボックスは、標準ＪＴ－Ｇ７１１のオーディオ品質向上に対する４つのアルゴリズムを含んでいる。ノイズシェーピング（ＮＳ）は、符号化器のみに適用され、フレーム消失補償（ＦＥＲＣ）、ノイズゲート（ＮＧ）、そしてポストフィルタ（ＰＦ）は復号器のみに適用される。これらのアルゴリズムは、標準ＪＴ－Ｇ７１１．１の階層的な符号化器／復号器から引用されており、標準ＪＴ－Ｇ７１１の符号化器／復号器と共に使うことが出来る。ツールは単独でも組み合わせでも使うことが出来る。このツールボックスは、標準ＪＴ－Ｇ１９１のソフトウェアツールライブラリ（ｖ２．２）の中で定義された基本演算子を使って固定小数点で実装することができる。本付録は、４つのアルゴリズムの詳細について記載する。

３．６．１　標準ＪＴ－Ｇ７１１符号化器に対するツール

一つのツール、つまりノイズシェーピング（ＮＳ）だけが符号化器に適用される。付図３－１／JT-G711 に、ノイズシェーピング（ＮＳ）ツールを用いた標準ＪＴ－Ｇ７１１符号化器のハイレベルブロック図を示す。付図３－１／JT-G711 の詳細は３．７節に記載されている。

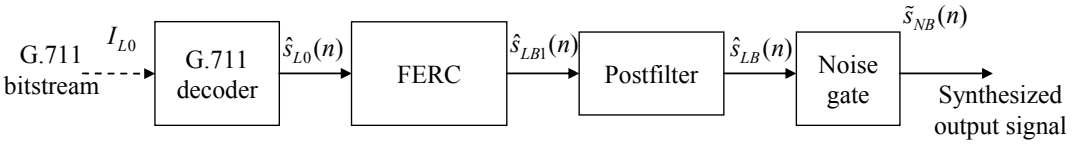


付図３－１／JT-G.711 - High-level block diagram of the noise shaping tool

(ITU-T G.711)

３．６．２　標準ＪＴ－Ｇ７１１復号器に対するツール

付図３－２／JT-G711 に、３つのツール、つまりフレーム消失補償（FERC）、ノイズゲート（NG）、そしてポストフィルタ（PF）を用いた標準ＪＴ－Ｇ７１１復号器のハイレベルブロック図を示す。この図は、各ツールが結合されるときの推奨される実行順序を示している。



付図３－２／JT-G.711 - High-level block diagram of decoder toolbox

(ITU-T G.711)

３．６．３　アルゴリズム遅延

付表３－３／JT-G711 に、各ツールのアルゴリズム遅延、及び３つのツールを結合した場合の復号器側でのアルゴリズム遅延を示す。これらのアルゴリズム遅延は 5ms フレームに対して与えられることに注意されたい。

付表３－３／JT-G.711 - Algorithmic delay of the toolbox (ms)

(ITU-T G.711)

NS	NG	PF	FERC	FERC+PF+NG
0	0	2	5	5

３．６．４　計算量、所要記憶容量

ツールボックスの測定されたワーストケースの計算量は、標準ＪＴ－Ｇ１９１のＳＴＬ２００５ｖ２．２のＩＴＵソフトウェアライブラリの基本演算子を使った実装に基づいている。ワーストケースの計算量は付表３－４／JT-G711 に詳細に記述されており、図はμ則、及び A 則を通じて最大の計算量を示している。４つのツールの 16 ビットワードでの所要記憶容量は付表３－５／JT-G711 に示されている。RAM 量は主要部を形成する配列に基づいており、一つの変数に基づいているものではない。そのような変数の数は、配列に必要なサイズと比較すると重要ではない。

付表３－４／JT-G.711 － Worst computational complexity of the toolbox [WMOPS]

(ITU-T G.711)

NS	NG	PF	FERC	FERC+PF+NG
0.87	0.23	2.02	2.05	3.31

付表３－５／JT-G.711 － Storage requirements of the toolbox

(ITU-T G.711)

Memory type	NS	NG	PF	FERC
Static RAM (kWords)	0.093	0.003	0.353	0.984
Scratch RAM (kWords)	0.107	0.012	0.529	0.314
Data ROM (kWords)	0.088	0	0.191	0.121
Program ROM (number of basic ops)	191	37	593	728

３．６．５　ツールボックス概略

ツールボックスアルゴリズムはビットイグザクト固定小数点の算術演算の観点から記述される。３．９節に記載されているＡＮＳＩ Ｃコードは、本付録の主要な部分を構成するものであり、このビットイグザクトな、固定小数点記述での記述を反映している。符号化器、および復号器の算術的な記述は、他の方法でも実装し得るが、本標準に準拠しないコーデックを実装することになってしまう可能性がある。したがって、不一致が生じた場合には、算術的な記述よりも、３．９節のＡＮＳＩ Ｃコードによるアルゴリズム記述の方が優先される。

３．７　符号化器に対するツールボックスの機能記述

３．７．１　ノイズシェーピング（ＮＳ）ツール

入力信号 $s_{NB}(n)$は、ＰＣＭ符号化器の符号化ノイズを聴感的に整形するノイズフィードバックを用いた μ 則、またはＡ則パルス符号変調（ＰＣＭ）を使って符号化される。重み付きフィードバックループを持つ符号化器を、付図３－１／JT-G711 に示す。最初に、入力信号 $s_{NB}(n)$には、50Hz のカットオフ周波数を持つハイパスフィルタによって前処理が施される。それから、前処理された信号 $s_{LB}(n)$はノイズフィードバック信号とオフセット値 c_{Loff} に加えられ、最終的な信号である $s'_{LB}(n)$は標準ＪＴ－Ｇ７１１本体の符号化器に入力される。得られたビット列 I_{L0}(n)に基づいて、標準ＪＴ－Ｇ７１１本体のデコーダは局所的に $\hat{s}_{L0}(n)$を復号し、$\hat{s}'_{L0}(n)$を得るためにオフセット値 c_{Loff} は除去される。続いて、ＬＰ解析は係数 a_i を得るために $\hat{s}'_{L0}(n)$に対して実行され、聴覚フィルタ $F(z)$が算出される。続いて、$F(z)$でフィルタ処理された量子化ノイズ $d_{L0}(n)$は、入力信号 $s_{LB}(n)$に加えられるためにフィードバックされる。ここで、非常に小さいエネルギの信号に対しては、標準ＪＴ－Ｇ７１１本体の符号化器は、対数ＰＣＭに基づいて、デッドゾーン量子化器と呼ばれる異なる符号化法によって置き換えられる。これは３．７．１．４で後述される。

３．７．１．１　前処理のハイパスフィルタ

標準ＪＴ－Ｇ７１１．１の７．１節に同じ。

３．７．１．２　標準ＪＴ－Ｇ７１１に基づくＰＣＭ符号化器

標準ＪＴ－Ｇ７１１．１の７．３．１節に同じ。

３．７．１．３　聴覚フィルタ

標準ＪＴ－Ｇ７１１．１の７．３．２節に同じ。

３．７．１．４　デッドゾーン量子化

標準ＪＴ－Ｇ７１１．１の７．３．３節に同じ。

３．８　復号器に対するツールボックスの機能記述

ツールボックスは復号器側においては３つのツールを含んでいる。これら全てのツールは、組み合わせ、あるいは単独のどちらでも使うことができる。付図３－２／JT-G711 は一連の処理におけるツールの位置を記述している。ツールのアルゴリズムについては後述する。

３．８．１　狭帯域フレーム消失補償（ＦＥＲＣ）

標準ＪＴ－Ｇ７１１．１の８．４節に同じ。

３．８．２　ノイズゲート（NG）

標準ＪＴ－Ｇ７１１．１の８．７節に同じ。

３．８．３　ポストフィルタ（ＰＦ）

標準ＪＴ－Ｇ７１１．１の付録Ｉに同じ。

３．９　標準ＪＴ－Ｇ７１１に対するオーディオ品質向上ツールボックスのビットイグザクトな記述

16ビット固定小数点においてオーディオ品質向上ツールボックスをシミュレートしたＡＮＳＩ　Ｃコードは、ＩＴＵ－ＴのＷｅｂサイトから入手可能である。

３．９．１　シミュレーションソフトウェアの使用

Ｃコードは encoder.c と decoder.c の２つのメインプログラムから構成されており、それぞれ符号化器、及び復号器に対するツールボックスをシミュレートしている。

符号化器に対するコマンドラインは以下の通りである。

encoder [-options] <law> <infile> <codefile>

ここで、

law	is the desired ITU-T G.711 law (A or μ)
infile	is the name of the input file to be encoded
codefile	is the name of the output bit stream file
options:	
-ns	indicates that noise shaping tool is activated
-hardbit	output bit stream file is in multiplexed hardbit format
-quiet	quiet processing

復号器に対するコマンドラインは以下の通りである。

decoder [-options] <law> <codefile> <outfile>

ここで、

law	is the desired ITU-T G.711 law (A or μ)
codefile	is the name of the input bit stream file
outfile	is the name of the decoded output file

options:

-ng	indicates that noise gate tool is activated
-pf	indicates that postfilter tool is activated
-ferc	indicates that frame erasure concealment tool is activated
-hardbit	input bit stream file is in multiplexed hardbit format
-quiet	quiet processing

符号化器への入力ファイルと復号器からの出力ファイルは、16 ビットＰＣＭ信号を含むサンプリングされたデータファイルである。符号化器の出力ファイルと復号器の入力ファイルは、デフォルトで標準ＪＴ－Ｇ１９２ビット列フォーマットに従う。フレーム消失は、標準ＪＴ－Ｇ１９２ビット列フォーマットに対してのみシミュレート可能である。

３．９．２　シミュレーションソフトウェアの構成

付表３－６／JT-G711 から付表３－９／JT-G711 はシミュレーションソフトウェアの構成を記述している。

付表３－６／JT-G.711 － Tables in C-code

(ITU-T G.711)

Table name	Symbol	Size	Format	Description
NS_window	$w_{LP1}(i)$	80	Q15	LPC analysis window for noise shaping
NS_lag_h	$w_{lag}(i)$	4	Q15	Lag windowing for noise shaping (MSB of double precision format)
NS_lag_l		4	Q16	Lag windowing for noise shaping (LSB of double precision format)
LBFEC_lag_h	$w_{lag}^{FERC}(i)$	16	Q15	Lag windowing for FERC (MSB and LSB of double precision format)
LBFEC_lag_l		16	Q16	Lag windowing for FERC (LSB of double precision format)
LBFEC_lpc_win_80	$w_{LP2}(i)$	80	Q15	LPC analysis window for FERC
LBFEC_fir_lp	$H_{dec}(z)$	9	Q16	FIR decimation filter coefficients in FERC
max_err_quant	$d'_{\max}(j)$	16	Q0	A-law quantization step size
Hann_sh16	$w_a(n)$	64	Q15	Asymmetric Hanning window (for 64-point FFT processing)
Hann_sh16_p6	$\gamma_w(n)$	7	Q15	Half of 16-point Hanning window for filter interpolation
Hann_sh16_p6m1	$1-\gamma_w(n)$	7	Q15	Complementary half of Hanning window for filter interpolation
WinFilt	$Wt_p(n)$	17	Q15	Truncating window for impulse response of noise reduction filter

付表３－７／JT-G.711 － Summary of specific routines for encoder toolbox

(ITU-T G.711)

Filename	Description
encoder.c	ITU-T G.711 toolbox encoder interface
G711Appenc.c	ITU-T G.711 toolbox main encoder
prehpf.c	High-pass pre-filter

Filename	Description
lowband_enc.c	Encoder toolbox

付表３－８／JT-G.711 － Summary of specific routines for decoder toolbox

(ITU-T G.711)

Filename	Description
decoder.c	ITU-T G.711 toolbox decoder interface
g711Appdec.c	ITU-T G.711 toolbox main decoder
fec_lowband.c	Frame erasure concealment (FERC)
lowband_dec.c	Decoder toolbox
post.c	Main routine that calls all post-processing subroutines
post_anasyn.c	Analysis/synthesis subroutines for post-processing
post_gainfct.c	Subroutines for estimating of the post-processing filter
post_rfft.c	64-point real FFT and ifft
table_post.c	Tables for postfilter

付表３－９／JT-G.711 － Summary of common routines

(ITU-T G.711)

Filename	Description
softbit.c	Conversion between hardbit and softbit
autocorr_ns.c	Autocorrelation of signal for noise shaping
g711a.c	PCM coder and decoder (A-law)
g711mu.c	PCM coder and decoder (μ-law)
lpctools.c	Linear prediction tools
table_lowband.c	Tables for lower-band modules
dsputil.c	Fixed-point utility routines
errexit.c	Exit routine
mathtool.c	Square-root routines
oper_32b.c	Basic operators in double precision (32 bits)
table_mathtool.c	Tables for square-root routines

付録Ⅰ（標準ＪＴ－Ｇ７１１に対する）用語対照表

英　　語	ＴＴＣ標準用語
ambient noise	背景雑音
argument	引数
artifacts	人工的雑音
attenuation	減衰
attenuation ramp	減衰傾斜
audio port	オーディオポート
audio system	オーディオシステム
bad frame	異常フレーム
bandwidth	帯域幅
buffer pointer	バッファポインタ
circular history buffer	巡回型のヒストリバッファ
coarse search	粗い探索
comfort noise	擬似背景雑音
complexity	演算量
down-sloping ramp	右下がりの傾斜重み、右下がり斜線
erasure	消失、消失区間
fine search	詳細な探索
frame erasure	フレーム消失
frame erasure concealment	フレーム消失補償
good frame	正常フレーム
Hanning window	ハニング窓
history buffer	履歴バッファ
interoperability specification	相互接続仕様
lag	ラグ
lag signal	ラグ信号
lastq buffer	lastq バッファ
non-circular buffer	非巡回型のバッファ
normalized cross-correlation	正規化相互相関
OLA window	ＯＬＡ窓
operator	操作
original signal	原信号
Overlap Add, overlap add	オーバラップ加算
packet loss concealment	パケット損失補償
packet size	パケットサイズ
payload	ペイロード
peak rate	ピークレート
pitch buffer	ピッチバッファ
pitch multiples	倍数ピッチ

英　　語	ＴＴＣ標準用語
pitch period	ピッチ周期
protected function	protected 関数
public functions	public 関数
receiver	受信器
reference signal	参照信号
sampling rate	標本化周波数
search	探索
sequence	系列
signal variation	信号のバリエーション
synthesized signal	合成信号
synthesized speech	合成音声
synthetic signal	合成信号
tail	末尾
the float version	浮動小数点版
the short version	固定小数点版
transition	遷移
transmitter	送信器
triangular window	三角窓
unnatural harmonic artifacts	不自然で調波的な人工的雑音
unvoiced region of speech	音声の無声領域
up-sloping ramp	右上がりの傾斜重み、右上がり斜線
variation in the signal	信号のバリエーション
voiced segment of speech	音声の有声領域
Waveform Shift Overlap Add	波形シフトオーバラップ加算

付録Ⅱ（標準ＪＴ－Ｇ７１１に対する）用語解説

ハニング窓 (Hanning window)

波形を切り出すために用いる時間窓の一種。周波数特性の点で、ハミング窓(Hamming window)はサイドローブが緩やかに減少するのに対し、ハニング窓は急速にサイドローブが減少する特徴がある。

ＭＩＰＳ

Million Instructions Per Second の略。百万回を単位とした１秒当たりの命令実行回数。

ペイロード

パケットなどで制御用のヘッダー部分を除いた実データを伝送する情報フィールド