お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

RENESAS

アプリケーション・ノート

78K0/Kx2

サンプル・プログラム

コーデック接続編

対象デバイス

78K0/KD2マイクロコントローラ
78K0/KE2マイクロコントローラ
78K0/KF2マイクロコントローラ

資料番号 U19723JJ1V0AN00（第1版）
発行年月 February 2010 N

〔メ　モ〕

目次要約

CMOSデバイスの一般的注意事項

(1) 入力端子の印加波形：入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOSデバイスの入力がノイズなどに起因して，VIL（MAX.）からVIH（MIN.）までの領域にとどまるような場合は，誤動作を引き起こす恐れがあります。入力レベルが固定な場合はもちろん，VIL（MAX.）からVIH（MIN.）までの領域を通過する遷移期間中にチャタリングノイズ等が入らないようご使用ください。

(2) 未使用入力の処理：CMOSデバイスの未使用端子の入力レベルは固定してください。未使用端子入力については，CMOSデバイスの入力に何も接続しない状態で動作させるのではなく，プルアップかプルダウンによって入力レベルを固定してください。また，未使用の入出力端子が出力となる可能性（タイミングは規定しません）を考慮すると，個別に抵抗を介してVDDまたはGNDに接続することが有効です。資料中に「未使用端子の処理」について記載のある製品については，その内容を守ってください。

(3) 静電気対策：MOSデバイス取り扱いの際は静電気防止を心がけてください。MOSデバイスは強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には，当社が出荷梱包に使用している導電性のトレーやマガジン・ケース，または導電性の緩衝材，金属ケースなどを利用し，組み立て工程にはアースを施してください。プラスチック板上に放置したり，端子を触ったりしないでください。また，MOSデバイスを実装したボードについても同様の扱いをしてください。

(4) 初期化以前の状態　電源投入時，MOSデバイスの初期状態は不定です。電源投入時の端子の出力状態や入出力設定，レジスタ内容などは保証しておりません。ただし，リセット動作やモード設定で定義している項目については，これらの動作ののちに保証の対象となります。リセット機能を持つデバイスの電源投入後は，まずリセット動作を実行してください。

(5) 電源投入切断順序　内部動作および外部インタフェースで異なる電源を使用するデバイスの場合，原則として内部電源を投入した後に外部電源を投入してください。切断の際には，原則として外部電源を切断した後に内部電源を切断してください。逆の電源投入切断順により，内部素子に過電圧が印加され，誤動作を引き起こしたり，異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源投入切断シーケンス」についての記載のある製品については，その内容を守ってください。

(6) 電源OFF時における入力信号　当該デバイスの電源がOFF状態の時に，入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により，誤動作を引き起こしたり，異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源OFF時における入力信号」についての記載のある製品については，その内容を守ってください。

・本資料に記載されている内容は2010年2月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
・文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
・当社は，本資料に記載された当社製品の使用に関連し発生した第三者の特許権，著作権その他の知的財産権の侵害等に関し，一切その責を負いません。当社は，本資料に基づき当社または第三者の特許権，著作権その他の知的財産権を何ら許諾するものではありません。
・本資料に記載された回路，ソフトウエアおよびこれらに関連する情報は，半導体製品の動作例，応用例を説明するものです。お客様の機器の設計において，回路，ソフトウエアおよびこれらに関連する情報を使用する場合には，お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し，当社は，一切その責を負いません。
・当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。また，当社製品は耐放射線設計については行っておりません。当社製品をお客様の機器にご使用の際には，当社製品の不具合の結果として，生命，身体および財産に対する損害や社会的損害を生じさせないよう，お客様の責任において冗長設計，延焼対策設計，誤動作防止設計等の安全設計を行ってください。
・当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「特定水準」に分類しております。また，各品質水準は，以下に示す用途に製品が使われることを意図しておりますので，当社製品の品質水準をご確認ください。

「標準水準」：コンピュータ，OA機器，通信機器，計測機器，AV機器，家電，工作機械，パーソナル機器，産業用ロボット

「特別水準」：輸送機器（自動車，電車，船舶等），交通用信号機器，防災・防犯装置，各種安全装置，生命維持を目的として設計されていない医療機器

「特定水準」：航空機器，航空宇宙機器，海底中継機器，原子力制御システム，生命維持のための医療機器，生命維持のための装置またはシステム等

当社製品のデータ・シート，データ・ブック等の資料で特に品質水準の表示がない場合は，標準水準製品であることを表します。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

注1. 本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNECエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいう。

注2. 本事項において使用されている「当社製品」とは，注1において定義された当社の開発，製造製品をいう。

（M8E0909J）

は じ め に

対 象 者　このマニュアルは,78K0マイクロコントローラの応用システムを設計,開発するユーザを対象とします。

目　　的　78K0マイクロコントローラでのオーディオ・コーデック接続方法についてユーザに理解していただくことを目的とします。

構　成　このマニュアルは，大きく分けて次の内容で構成しています。

- 概　　説
- オーディオ・コーデック接続方式
- アプリケーション実装例
- サンプル・アプリケーションのビルド方法
- サンプル・アプリケーションの実行方法

読 み 方　このマニュアルの読者には，電気，論理回路，マイクロコンピュータおよびC言語に関する一般知識を必要とします。

本アプリケーションの実行評価環境を理解しようとするとき

→ヒューマン・マシンI/Fデモ用ボードまたは漢字表示デモ用ボードの**ユーザーズ・マニュアル**を参照してください。

マイクロコントローラのハードウエア機能の詳細を理解しようとするとき

→各製品の**ユーザーズ・マニュアル　ハードウエア編**を参照してください。

凡　　例　データ表記の重み：左が上位桁，右が下位桁

アクティブ・ローの表記：$\overline{xxx}$（端子，信号名称に上線）

メモリ・マップのアドレス：上部－上位，下部－下位

注　：本文中に付けた注の説明

注意：気を付けて読んでいただきたい内容

備考：本文の補足説明

数の表記：2進数　… xxxxまたはxxxxB

10進数 … xxxx

16進数 … xxxxH

2のべき数を示す接頭語（アドレス空間，メモリ容量）：

K（キロ）… $2^{10} = 1024$

M（メガ）… $2^{20} = 1024^2$

G（ギガ）… $2^{30} = 1024^3$

関連資料　関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

ヒューマン・マシンI/F**デモ・ボート，漢字表示デモ・ボードの関連資料**

資　料　名	資料番号	
	和　　文	英　　文
ヒューマン・マシンI/Fデモ用ベース・ボード　ユーザーズ・マニュアル	U20117J	未定
ヒューマン・マシンI/Fデモ用78K0ボード　ユーザーズ・マニュアル	U20118J	未定
ヒューマン・マシンI/Fデモ用78K0Rボード　ユーザーズ・マニュアル	U20119J	未定
ヒューマン・マシンI/Fデモ用V850ESボード　ユーザーズ・マニュアル	U20120J	未定
漢字表示デモンストレーション用ベース・ボード　ユーザーズ・マニュアル	U19207J	未定
漢字表示デモンストレーション用78K0/KF2ボード　ユーザーズ・マニュアル	U19208J	未定
漢字表示デモンストレーション用78K0R/KG3ボード　ユーザーズ・マニュアル	U19209J	未定
漢字表示デモンストレーション用V850ES/JG3ボード　ユーザーズ・マニュアル	U19210J	未定
78K0/Kx2サンプル・プログラム（簡易OS編）アプリケーション・ノート	U19214J	未定
78K0R/Kx3サンプル・プログラム（簡易OS編）アプリケーション・ノート	U19215J	未定
V850ES/Jx3サンプル・プログラム（簡易OS編）アプリケーション・ノート	U19216J	未定
フォント・ユーティリティ　ユーザーズ・マニュアル	U19527J	未定
78K0/Kx2 サンプル・プログラム（フォント選択編）アプリケーション・ノート	U19528J	未定
78K0R/Kx3 サンプル・プログラム（フォント選択編）アプリケーション・ノート	U19529J	未定
V850ES/Jx3 サンプル・プログラム（フォント選択編）アプリケーション・ノート	U19530J	未定
漢字表示デモンストレーション用拡張ボード　ユーザーズ・マニュアル	U19526J	未定
78K0/Kx2 サンプル・プログラム（ドットＬＣＤ制御編）アプリケーション・ノート	U19531J	未定
78K0R/Kx3 サンプル・プログラム（ドットＬＣＤ制御編）アプリケーション・ノート	U19532J	未定
V850ES/Jx3 サンプル・プログラム（ドットＬＣＤ制御編）アプリケーション・ノート	U19533J	未定
78K0/Kx2 サンプル・プログラム（タッチスクリーン編）アプリケーション・ノート	U19720J	未定
78K0R/Kx3 サンプル・プログラム（タッチクリーン編）アプリケーション・ノート	U19721J	未定
V850ES/Jx3 サンプル・プログラム（タッチクリーン編）アプリケーション・ノート	U19722J	未定
78K0/Kx2 サンプル・プログラム（コーデック接続編）アプリケーション・ノート	このノート	未定
78K0R/Kx3 サンプル・プログラム（コーデック接続編）アプリケーション・ノート	U19724J	未定
V850ES サンプル・プログラム（コーデック接続編）アプリケーション・ノート	U19725J	未定

78K0**マイクロコントローラ・デバイスの関連資料**

資　料　名	資料番号	
	和　　文	英　　文
78K0/Kx2　ユーザーズ・マニュアル	U18598J	U18598E
78K0マイクロコントローラ ユーザーズ・マニュアル 命令編	U12326J	U12326E

注意　上記関連資料は予告なしに内容を変更することがあります。設計などには，必ず最新の資料をご使用ください。

目　　次

第1章　概　　説

この資料では，78K0用オーディオ・コーデック接続サンプル・プログラムについて説明します。

1. 1　プログラム概要

本サンプル・プログラムは，音声データやBEEP波形データをオーディオ・コーデックへ送る処理を行います。オーディオ・データ・インタフェースとしてはDSPモードを使用し，クロックや同期信号はコーデックからもらう方法で接続しています。オーディオ・データ・インタフェース方式がI^2Sモードで，MCU内部でクロックや同期信号を生成する場合は，次の資料を参考にしてください。

「78K0R/Kx3サンプル・プログラム　I^2Sバス・インタフェース（U19514）」

1. 2　開発環境

サンプル・プログラムからオブジェクト・コードを生成するソフトウエア・ツールと，生成したコードを実行する評価ボードについて説明します。

1. 2. 1　ソフトウエア・ツール

本サンプル・プログラムのビルドに推奨するソフトウエア・ツールは，NECエレクトロニクス製の有償ソフトウエア・パッケージ（SP78K0）です。

（1）推奨バージョン

ソフトウエア・ツールの推奨バージョンです。

（a）CC78K0　　：4.00以上

（b）RA78K0　　：4.01以上

（2）無償ダウンロード版の制約

無償ダウンロード版のコンパイラおよびアセンブラ（CC78K0, RA78K0）には生成できるオブジェクト・サイズに制限があり，音声データ・サイズが限られます。

1.2.2 評価ボード

実際に音を出して評価するために，下記（1）または（2）のボードを使用できます。

ボードの入手に関する情報については，下記のURLを参照してください。

http://www.necel.com/micro/ja/designsupports/board/index.html

（1）ヒューマン・マシンI/Fデモ用ボード

- ヒューマン・マシンI/Fデモ用ベース・ボード（SM07A）
- ヒューマン・マシンI/Fデモ用78K0ボード（SM06E）

（2）漢字表示デモンストレーション用ボード

- 漢字表示デモンストレーション用ベース・ボード（SM05A2）
- 漢字表示デモンストレーション用78K0ボード（SM05F3）
- 漢字表示デモンストレーション用拡張ボード（SM06B2）

（3）プログラム書き込みツール

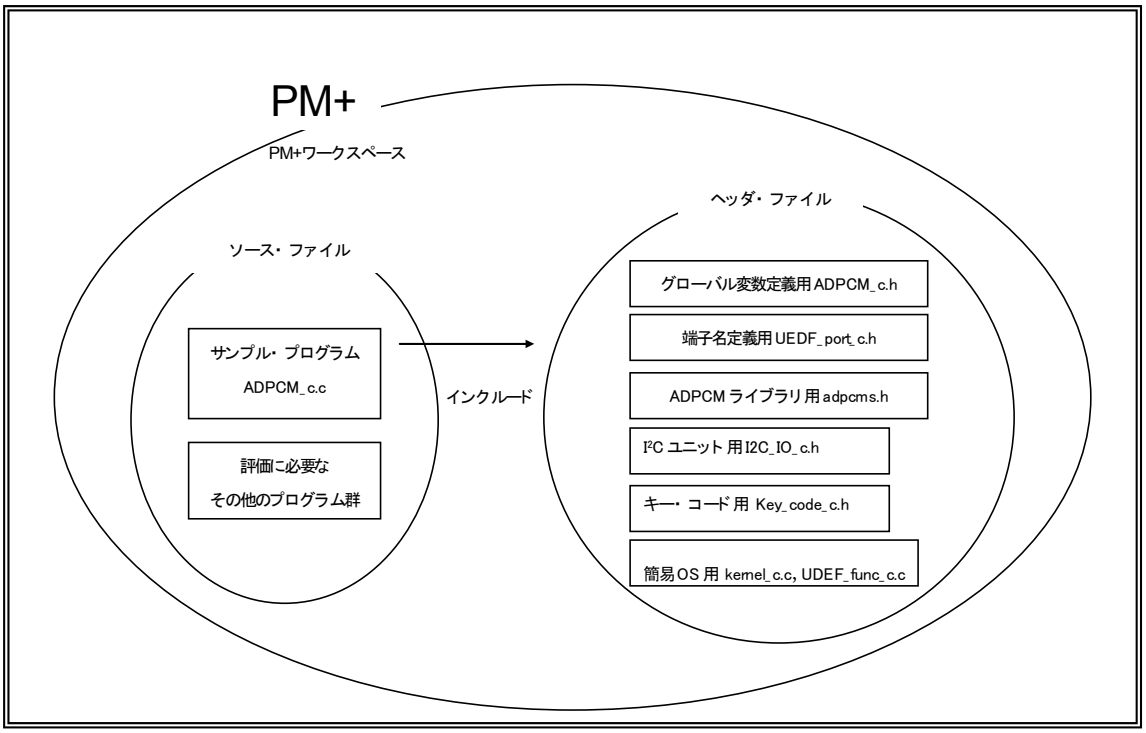
推奨するプログラム書き込みツールは，オンチップ・デバッグ・エミュレータ（QB-MINI2）です。

PG-FP5などのフラッシュ・メモリ・プログラマ（以降，プログラマ）でも可能です。

1. 3　アプリケーションの依存情報

プロジェクト・マネージャ（以降，PM+）環境下での依存情報を次の図に示します。

図1－1　PM+環境下での依存情報の関係



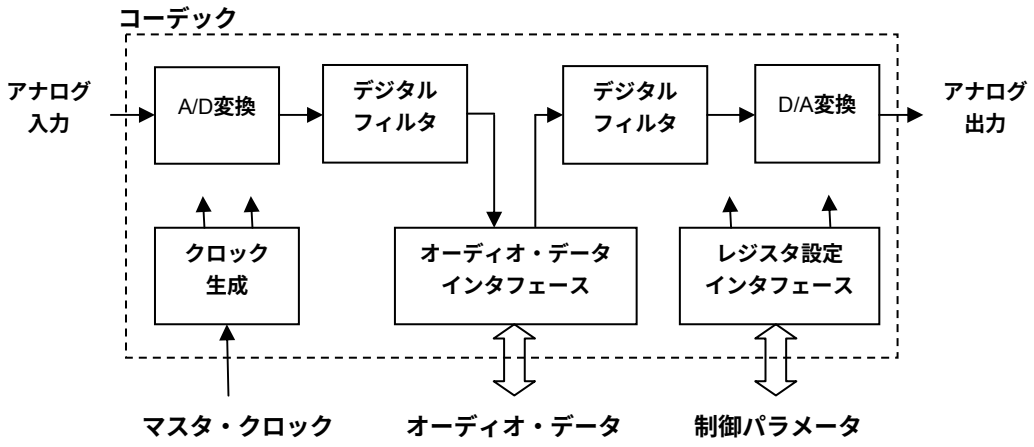
第2章　オーディオ・コーデック接続方式

この章では，コーデックのインタフェース全般を概説し，サンプル・プログラムで対応しているインタフェースについて詳細に説明します。

2. 1　インタフェースの概要

マイクロコントローラにオーディオ・コーデックを接続する場合，大きく分けて次の3点を考慮します。

・コーデックへのマスタ・クロック供給
・コーデックの各種制御パラメータ（レジスタ）の設定インタフェース
・オーディオ・データ・インタフェース



2. 1. 1　マスタ・クロック供給

コーデック内部のA/D変換，D/A変換，デジタル・フィルタはサンプリング周波数あるいはその整数倍で動作します。例えば，8kHzサンプリングの256倍が必要であれば2.048MHzが必要になります。このクロックをマイクロコントローラで直接生成できる場合は良いのですが，一般的には異なることが多いため，最近はコーデックにPLLを搭載して異なる周波数からクロック生成できるものが増えてきました。

PLL搭載タイプのコーデックでも，何種類かの周波数から選択するものと，ある範囲の任意の周波数を入力できるものに分かれます。後者は若干パラメータ計算の手間はかかりますが，1回計算するだけなので，一般的には後者のタイプの方が便利でしょう。

2. 1. 2　制御インタフェース

コーデックの動作開始／停止やフィルタ特性，ボリューム調整などは，コーデック内のレジスタに制御パラメータを書き込むことにより行います。

インタフェースの種類としては，3線シリアルやI^2Cバスなどが用いられます。マイクロコントローラから送るデータは，レジスタ・アドレスとその設定データになります。レジスタの種類とその内容はコーデックにより異なります。

2. 1. 3　オーディオ・データ・インタフェース

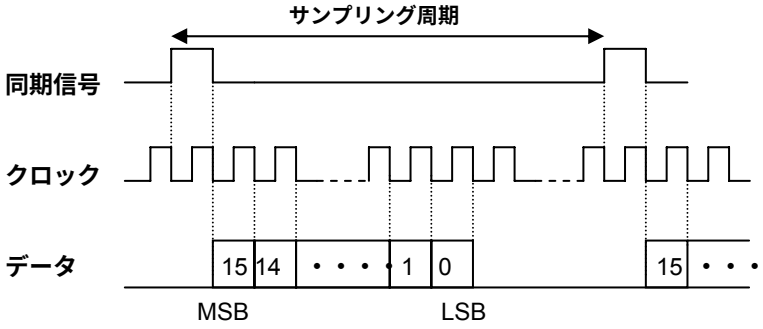
コーデックにより特定のインタフェースのみ対応したものと，複数のインタフェースに対応したものがあります。主な違いは，同期信号の波形形状と，同期信号のエッジからオーディオ・データまでの位置（クロック数）です。

- DSP（PCM）モード
- I^2Sフォーマット
- 右詰め，左詰めフォーマット
- AC'97

以下，最初の二つについてタイミングを示します。

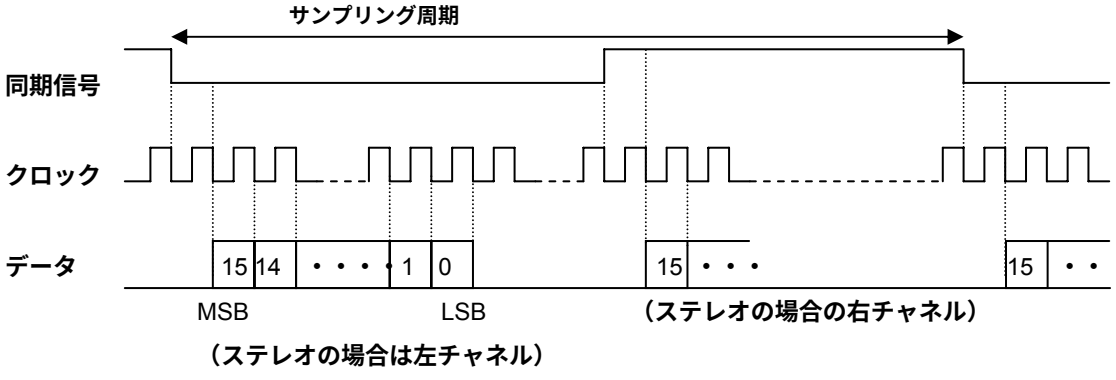
（1）DSP（PCM）モード

- フレーム同期信号（サンプリング・レート同期）として，ビット・クロックの1クロック分だけの細いパルス波形になります。
- オーディオ・データの位置は，フレーム同期信号の次のビット・クロックからになります。



（2）I^2Sフォーマット

- フレーム同期信号は，サンプリング周期を1周期とする方形波になります。
- オーディオ・データの位置は，エッジから2クロック目からになります。



2.2 接 続 例

ここでは，スピーカ・アンプを内蔵しているWolfson製モノラル・コーデックWM8974と接続する場合を例として説明します。

2.2.1 クロック供給とPLL設定

WM8974はマスタ・クロックの周波数を比較的自由に選ぶことが出来ます。このため，78K0のクロック出力（PCL端子からの出力）をマスタ・クロックとして入力できます。ただし78K0/Kx2のPCL端子は，10MHzを超える出力は禁止されているため，78K0/Kx2の発振周波数は7.562～10 MHzまたは15.124～20 MHz（2分周出力）に限られます。

WM8974のPLL関連レジスタには，マスタ・クロックの周波数に対応する値を設定する必要があります。以下，簡易な計算方法を示します。

① 98.304／PCL出力周波数（MHz単位） = 除算結果（整数部.小数部）

② 除算結果の小数部 x 16,777,216 = PLLK設定値（小数部を切り捨てて，24ビットの2進数で表す）

③ コーデックのレジスタ番号36～39設定値は，次のとおり（注：一つの番号で9ビットを設定）。

36：上位5ビットは00000，下位4ビットに除算結果の整数部を格納

37：上位3ビットは000，下位6ビットにPLLKの上位6ビットを格納

38：PLLKの中9ビットを格納

39：PLLKの下位9ビットを格納

これにより，コーデックのPLL出力周波数（f_{PLLOUT}）は24.576 MHzとなり，後段の分周器の設定により48 kHzおよびその分周周波数でのサンプリング・レートに対応できるようになります。

代表的なPCL周波数に対する計算値を下表に示します。

f_{PRS}周波数［MHz］	PCL出力設定［MHz］	設定内容
8または16	8	整数部 = 1100，PLLK = 010010　011011101　001011110
10または20	10	整数部 = 1001，PLLK = 110101　001001010　100011000

2.2.2 制御インタフェースの方式選択

WM8974は，I^2Cバス・インタフェースあるいは3線シリアル・インタフェースで使用できます。どちらのインタフェースで使用するかは，コーデックのMODEピンで選択できます。後述のサンプル・プログラムではI^2Cバスを使用します。

・MODEピン = ‘L’：I^2Cバス

・MODEピン = ‘H’：3線シリアル（データ，クロック，ストローブ）

I^2Cバスの場合は，1バイト目にI^2Cアドレス0011010＋リード／ライト指定，2バイト目にレジスタ番号7ビット＋設定データMSB，3バイト目に設定データ下位8ビットを転送します。

3線シリアルの場合は，レジスタ番号7ビット＋設定データ9ビットの合計16ビット単位で転送します。

2. 2. 3　オーディオ・データの入出力制御

データやインタフェースの条件を次のとおりとします。

- 音声データ：8 kHzサンプリング，16ビット・データ（リニアPCMデータ）。
- インタフェース形式：DSPモード
- フレーム同期信号，ビット・クロック信号はコーデック側から供給。78K0側は，割り込み検出端子（フレーム同期信号用），3線シリアル・インタフェース（スレーブ・モード）を使用。

(1) コーデックの設定

レジスタ番号4：000011000　（16ビット・データ，DSPモード）

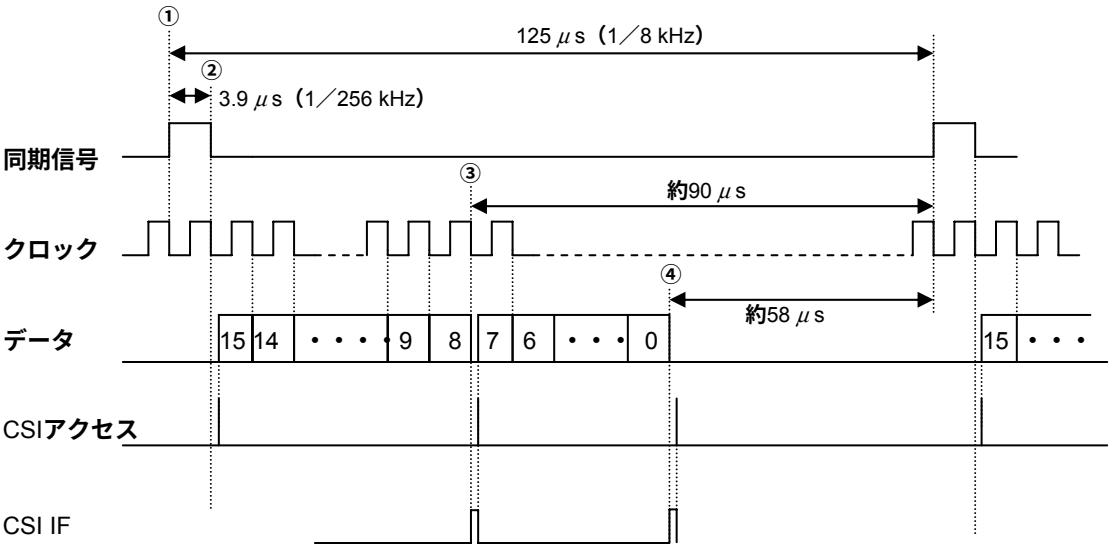
レジスタ番号6：111101101　（PLL出力を12分周，ビット・クロックはさらに8分周，
フレーム同期信号とビット・クロック信号をコーデックから出力）

ここで，PLL出力は2. 2. 1項の設定により24.576 MHz，必要クロックはサンプリング周波数の256倍 = 8 kHz x 256 = 2.048 MHzであるため，12分周を行います。
また，1フレームのビック・クロック数を32ビットとすると，256／32 = 8から，ビット・クロック生成用分周器を8に設定します。

レジスタ番号7：000001010　（デジタル・フィルタの係数を8 kHzサンプリング用に設定）

(2) タイミング

78K0の3線シリアル・インタフェース（CSI1n）は，位相タイプ4，スレーブ・モードに設定します。
フレーム周波数8 kHz，ビット・クロック周波数256 kHzより，タイミングは次のようになります。



① 同期信号の立ち上がりで割り込み処理に入ります。

② 同期信号の立ち下がり検出で，すかさずCSIをアクセス（データ送信時はSOTB1nへ書き込み，データ受信時はSIO1nから読み出し）します。

③ CSIIF1nフラグが立ったら，すかさずCSIをアクセスします。データ送信時はこれで1フレームの処理が終わります。

④ データ受信の場合は，再度SIO1nの読み出しを行うと1フレームの処理が終わります。

(3) プログラミング上の考慮点

78K0/Kx2にコーデックを接続するのは実は負担が大きいため，他のプログラムと同時に動作させるにはいくつか考慮すべき点があります。

① フレーム同期検出

同期信号の立ち下がりでCSIを確実にスタートさせるには，同期信号の立ち上がりで割り込みに入り，立ち下がりを待つのが良いでしょう。それでも‘H’レベル幅が3.9 μsと短いため，他の割り込みはいったん禁止するなどの処理が必要になる場合もあります。

後述のサンプル・プログラムでは，コーデックへのデータ送信開始前に全割り込みマスク・レジスタをいったん退避し，同期信号の立ち上がり検出割り込み以外を全てマスクしています。

② CSIのスタート

同期信号の立ち下がり検出でCSIをスタートさせる場合，送信データをその時に準備したのでは間に合わない可能性があります。専用のレジスタ・バンクを割り当てて，あらかじめ準備しておくのが確実でしょう。

③ 2バイト目のスタート

78K0／Kx2のCSI1nは連続転送モードが無く，かつ転送終了から次のビット・クロック立ち上がりまで1.95 μsしかないため， 1バイト目の転送終了を割り込みフラグの監視で判定し，2バイト目の転送を開始するようにします。

④ 他の割り込み処理

1フレームの処理が終了すると，次のフレーム処理までに約90 μs（送信時）または約58 μs（受信時）の余裕があります。後述のサンプル・プログラムでは，この時間に他の優先度の高い割り込みのフラグを判定して，直接その割り込み処理へ飛ぶようにしています。

第3章　アプリケーション実装例

この章では，ADPCM再生サンプル・プログラムADPCM_c.cを例にコーデック接続について説明します。

3. 1　アプリケーションの概要

3. 1. 1　機能概要

サンプル・プログラムADPCM_c.c は，ROMに格納された圧縮音声データを伸長し，コーデックへ送信するプログラムです。BEEP用のPCM波形データを直接送る機能もあります。

機能概要を図3－1に示します。アプリケーション全体としては簡易OSやI^{2}Cバス制御プログラムが含まれていますが，ADPCM_c.c以外の説明は省略します。

図3－1　機能概要図

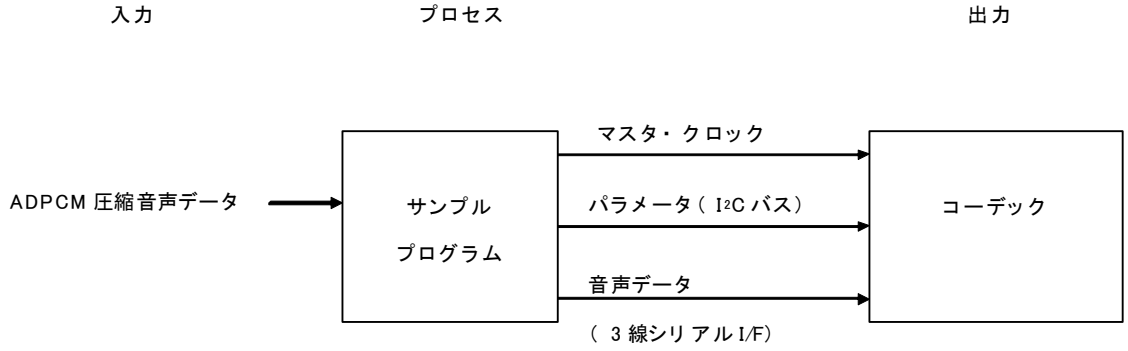
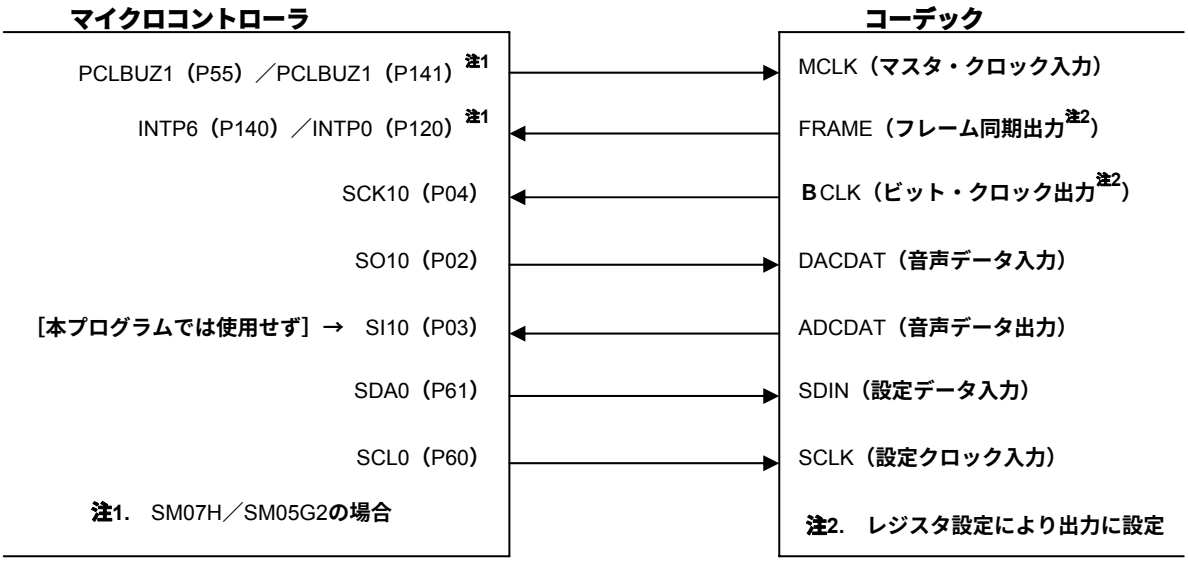


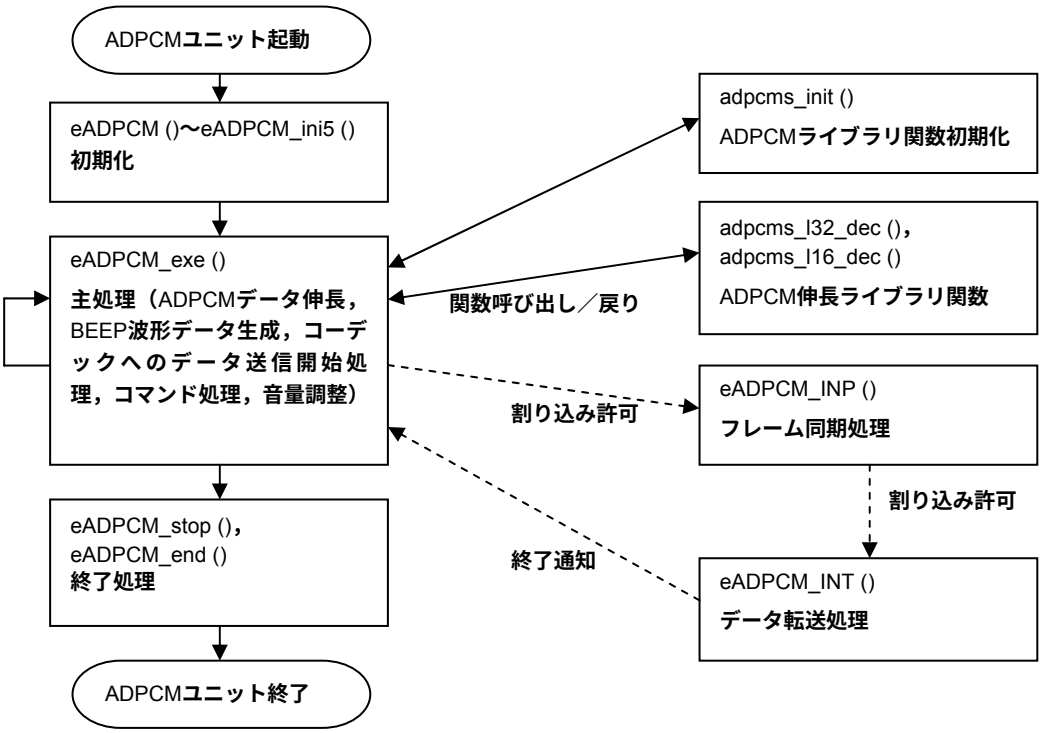
図3－2　接続図



3. 1. 2　制御手順の概要

ADPCM_c.c中の主要な関数の流れを図3－3に示します。本ソース・プログラムには，簡易OS依存部が多少含まれますが，簡易OSを使わない場合の修正方法がコメントに記載されています。

図3－3　全体フローチャート



3. 1. 3　主要な変数・定数・定義値

(1) 音声データ，BEEP波形データ

const unsigned long tlADPCM_data[ ]	音声データの開始アドレスへのポインタを格納
const unsigned char　tcADPCM_mode[ ]	各音声データの圧縮形式
const short tiADPCM_table[]	BEEP波形データ（正弦波1周期分）

(2) コーデック制御パラメータ，データ送信バッファ制御

const unsigned char tcADPCM_init[ ]	コーデックの初期化データ
const unsigned char tcADPCM_close[ ]	コーデック使用終了時に送付するデータ（ミュート用）
unsigned char　　tcADPCM_fifo[ ]	コーデックへ送る音声データのバッファ
extern unsigned char tcADPCM_rc	音声データ・バッファのリード・カウンタ
extern unsigned char tcADPCM_wc	音声データ・バッファのライト・カウンタ

(3) ホスト・マシンとのコマンド・インタフェース，プログラム動作パラメータ

struct ADPCM_REG gADPCM	コマンド受け取り変数およびパラメータ変数の構造体

(4) ポート定義，SFR定義

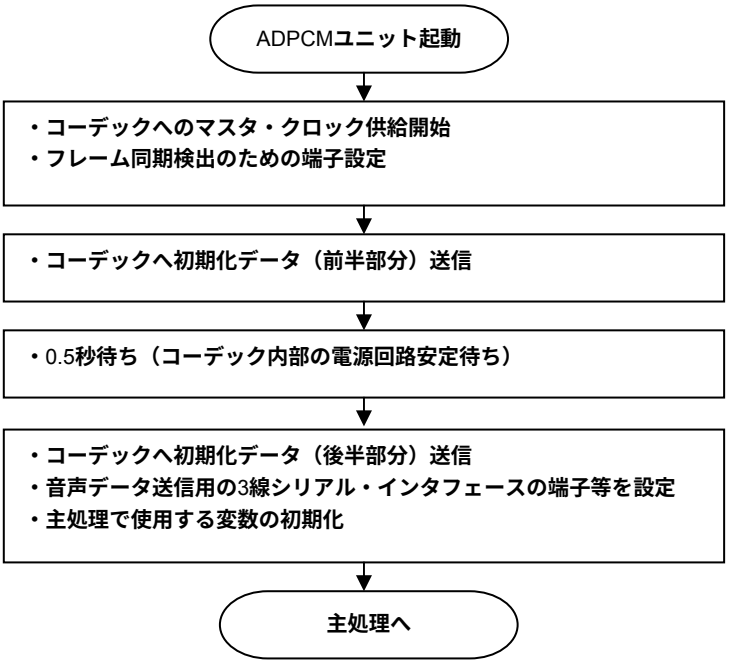
UDEF_port_c.hを参照してください。

3. 2　初 期 化

コーデックの初期化および主処理で使用する変数の初期化をeADPCM～eADPCM_ini5までの5つの関数で行います。

(1) 概略フロー

図3－4　初期化の概略フローチャート



(2) 初期化のプログラム・リスト

```
void eADPCM(void) {
    /* 資源フラグ初期化 */
#if  ADPCM_SP > 0
    tbADPCM_CSI = 0;
#endif

    /* I2C資源確認 */                                                    ①
    if (!(vRefer_task(hI2C_IO) & TASK_ACTIVE)) {
            if (!(vRefer_task(hI2C_IO) & TASK_START)) {
                    vStart_task(hI2C_IO, eI2C_IO);
                    }
            return;
    }
    /* I2C_IO ロック */
    gI2C_IO.adpcm = 1;

    /* 端子設定 */
#if  ADPCM_SP > 0
    /* クロック出力 */                                                   ②
    zPCL = 0;
    zmPCL = 0;
    yPCL_CKS = kCKS1_1;
    yPCL_OE = 1;

    /* フレーム端子設定 */
    yFRAME_EGP = 1;                 /* 立ち上がりエッジ割り込み */
#endif

    /* コーデック初期設定（前半）の準備 */
    tcADPCM_wk = 0;
    vTrans_task(eADPCM_ini2);               /* 簡易OSを使用しない場合は    */   ③
}                                           /* ここの部分を削除して        */
void eADPCM_ini2(void) {                    /* ループ文等で初期化します    */
    /* コーデック初期設定（前半） */
    uI2C_IO_write(CODEC_ADDR, tcADPCM_init[tcADPCM_wk], tcADPCM_init[tcADPCM_wk+1]);
    tcADPCM_wk += 2;
    if (tcADPCM_wk < nCODEC_OPEN1) return;

    /* コーデック内部電源安定待ちの準備 */
    tcADPCM_wk = (unsigned char) vGet_time();
    vTrans_task_delay(eADPCM_ini3);         /* 簡易OSを使用しない場合は    */   ④
}                                           /* ここの部分を削除して        */
void eADPCM_ini3(void) {                    /* ループ文等で待ちます        */
    unsigned char i;

    /* コーデック内部電源安定待ち */
    i = (unsigned char) vGet_time();
    i -= tcADPCM_wk;
    if (i < kCODEC_POW_wait) { vDelay_task(); }

    /* コーデック初期設定（後半）の準備 */
    tcADPCM_wk = nCODEC_OPEN1;
    vTrans_task(eADPCM_ini4);               /* 簡易OSを使用しない場合は    */   ⑤
}                                           /* ここの部分を削除して        */
void eADPCM_ini4(void) {                    /* ループ文等で初期化します    */
    /* コーデック初期設定（後半） */
    uI2C_IO_write(CODEC_ADDR, tcADPCM_init[tcADPCM_wk], tcADPCM_init[tcADPCM_wk+1]);
    tcADPCM_wk += 2;
    if (tcADPCM_wk < nCODEC_OPEN2) return;
    uI2C_IO_stop();

    tcADPCM_vol = CODEC_VOLINI;                                              ⑥
    tcADPCM_aux = CODEC_AUXINI;
```

```
#if   ADPCM_SP > 0
      vTrans_task(eADPCM_ini5);                 /* 簡易OSを使用しない場合は      */
}                                               /* ここの部分を削除して          */   ⑦
void eADPCM_ini5(void) {                        /* 資源確保します                */
      /* CSI資源確保 */
      if (!vPolling_sema(yCSI_sem)) return;

      /* 資源フラグ・セット */
      tbADPCM_CSI = 1;

      /* レジスタ設定 */
      zCSI_SO   = 0;                                                          ⑧
      zmCSI_SO = 0;
      yCSI_C1 = kCSIC11;
      yCSI_M1 = kCSIM11;
      yCSI_MK = 1;
      yCSI_IF = 0;

      /* 変数初期化 */
      #if         ADPCM_SP > 0
                  tbADPCM_start = 0;            /* 開始フラグ */                  ⑨
                  tbADPCM_playb = 0;            /* beep動作 */
                  tbADPCM_playbe = 0;           /* beep終了 */
                  tpADPCM_fifo = tcADPCM_fifo;
                  tbADPCM_endm = 0;             /* 再生終了通知フラグ */
      #endif

      #if         ADPCM_SP >= 10
                  tbADPCM_play32 = 0;           /* 32kbps */
                  tbADPCM_play16 = 0;           /* 16kbps */
      #endif

#endif
      vActive_task(hADPCM);
      vTrans_task(eADPCM_exe);/* 簡易OSを使用しない場合はreturn文にします */
}
```

（3）詳細説明

① I^2Cバス制御を行うI2C_IOユニットが起動していなければ起動します。

② 78K0のPCL端子からマスタ・クロックを出力します。使用するデモ・ボードに応じて8 MHzまたは10 MHz出力になります。

③ tcADPCM_init[ ]に格納されたコーデック初期化データを送信します。送信にはI2C_IOユニットで定義している関数を使用しています。初期化データは，前半部分をまず送信します。

④ 0.5秒のウエイトを行います。ここでは簡易OSでサポートされているシステム・タイマの読み取り関数を使用して経過時間を判定しています。簡易OSを使用しない場合は，システム・タイマを直接読むなどして経過時間を判定します。

⑤ tcADPCM_init[ ]に格納された残りのコーデック初期化データを送信します。

⑥ コーデックに初期値として送ったボリューム値を音量管理変数にも格納します。

⑦ 音声データを送るための3線シリアル・インタフェースの共有制御をしています。共有制御が不要であれば削除可能です。

⑧ 3線シリアル・インタフェースの端子設定，レジスタ設定を行います。
ポート名やレジスタ名は，UDEF_port_c.hで定義しています。

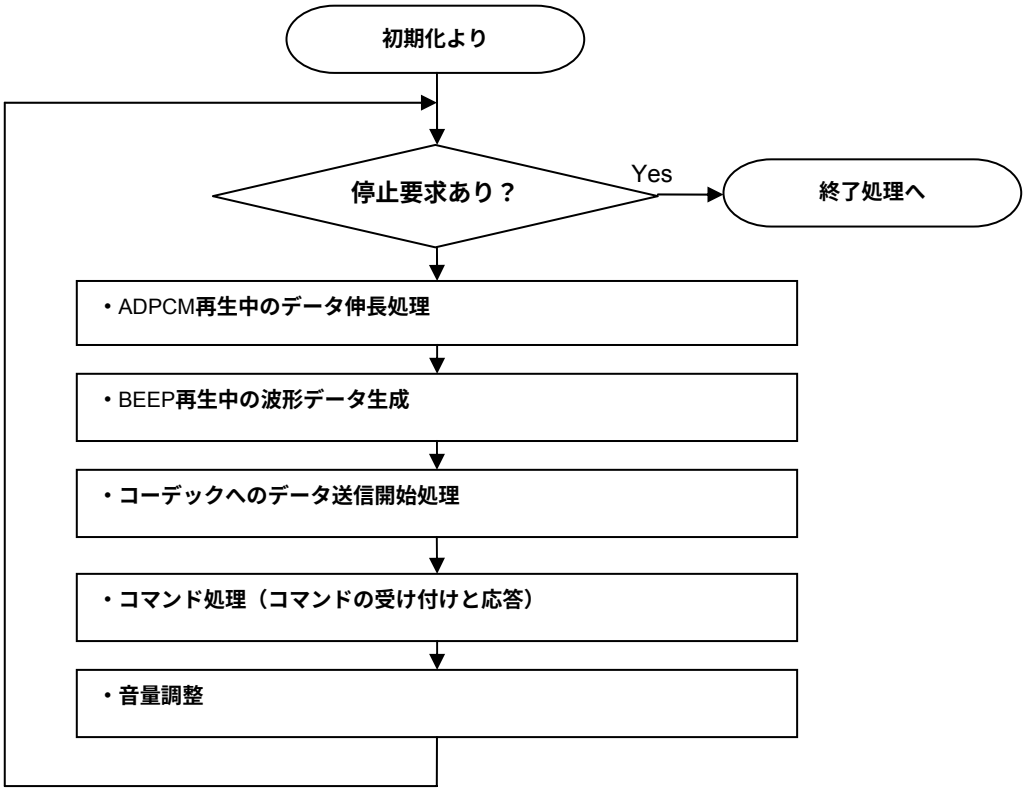
⑨ 主処理で使用する管理変数の初期化を行います。

3. 3　主 処 理

主処理では，ADPCM音声再生またはBEEP再生の開始／停止制御，ADPCMデータ伸長，BEEP波形データ生成，コマンド処理，キー操作による音量制御を行います。

（1）概略フロー

図3－5　主処理の概略フローチャート



（2）主処理のプログラム・リスト

```
void eADPCM_exe(void) {
    short m;
    unsigned char *p, i,j,k;

    /* 停止処理判定 */
    if (!(vRefer_task(hADPCM) & TASK_ACTIVE) ) {          /* 簡易OSを使用しない場合は,    */   ①
            vTrans_task(eADPCM_stop);                     /* 削除します                   */
    }
/****************************************
 ADPCMデータ伸長
 ****************************************/
#if ADPCM_SP >= 10
    if (tbADPCM_play32 || tbADPCM_play16) {
            if ((m = (unsigned short)tpADPCM_lim - (unsigned short)tpADPCM_data) > 0) {
                    /* ADPCMデータ伸長 */
                    BANK = tcADPCM_BANK;          /* ADPCMデータが格納されたバンクに切り替え */   ②
                    j = (tcADPCM_rc - tcADPCM_wc - 1);               /* バッファ空き容量計算 */

                    if (tbADPCM_play32) {
                            /* 32kbps ADPCM */
                            if (m > 63) m = 63;
                            k = (unsigned char)m << 2;
                            if (j > k) j = k;
                            j &= 0xfc;            /* 2データ(4バイト)単位のデコード */   ③
                            while (j) {
                                    k = *tpADPCM_data++;
                                    m = adpcms_l32_dec(k & 0x0f);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)(m >> 8);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)m;
                                    m = adpcms_l32_dec(k >> 4);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)(m >> 8);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)m;
                                    j -= 4;
                            }
                    }
                    else {
                            /* 16kbps ADPCM */
                            if (m > 31) m = 31;
                            k = (unsigned char)m << 3;
                            if (j > k) j = k;
                            j &= 0xf8;            /* 4データ(8バイト)単位のデコード */   ④
                            while (j) {
                                    k = *tpADPCM_data++;
                                    m = adpcms_l16_dec(k & 0x03);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)(m >> 8);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)m;
                                    k >>= 2;
                                    m = adpcms_l16_dec(k & 0x03);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)(m >> 8);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)m;
                                    k >>= 2;
                                    m = adpcms_l16_dec(k & 0x03);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)(m >> 8);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)m;
                                    k >>= 2;
                                    m = adpcms_l16_dec(k & 0x03);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)(m >> 8);
                                    tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)m;
                                    j -= 8;
                            }
                    }
                    BANK = 0;             /* バンクをデフォルトに戻す */   ⑤
            }
```

```
                else {
                        /* 1回の再生終了 */
                        if (tcADPCM_rcnt != 0)  {                                          ⑥
                                /* 繰り返し再生処理 */
                                tpADPCM_data = (unsigned char*) *(unsigned short*)tiADPCM_t;
                                if (tcADPCM_rcnt != 0xff) tcADPCM_rcnt--;
                        }
                        else if (!gADPCM.status) {
                                /* 再生完了 */
                                tbADPCM_playbe = 1;  /* 最終データのバッファ格納フラグ・セット */
                                tbADPCM_endf = 0;    /* 最終データの送信終了フラグ・クリア */
                                tbADPCM_play32 = 0;
                                tbADPCM_play16 = 0;
                        }
                }
        }
#endif
/***************************************
 BEEP波形生成
 ***************************************/
#if ADPCM_SP > 0
        /* Beep */                                                                     ⑦
        if (tbADPCM_playb) {
                if ((m = (unsigned short)tpADPCM_lim - (unsigned short)tpADPCM_data) > 0) {
                        j = (tcADPCM_rc - tcADPCM_wc - 1);                    /* バッファ空き容量計算 */
                        if (m > 127) m = 127;
                        k = (unsigned char)m << 1;
                        if (j > k) j = k;
                        j &= 0xfe;
                        while (j) {
                                        tpADPCM_data++;
                                        tiADPCM_bt += gADPCM.delta;
                                        k=(unsigned char)(tiADPCM_bt >> 9);/**/
                                        m = tiADPCM_table[k];
                                        tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)(m >> 8);
                                        tcADPCM_fifo[tcADPCM_wc++] = (unsigned char)m;
                                        j -= 2;
                        }
                }
                else {
                        tbADPCM_playbe = 1;             /* 最終データのバッファ格納フラグ・セット */
                        tbADPCM_endf = 0;               /* 最終データの送信終了フラグ・クリア */
                        tbADPCM_playb = 0;
                }
        }
/***************************************
 コーデックへのデータ送信開始
 ***************************************/
        if (tbADPCM_start) {
                        /* 割り込みマスクの退避と全割り込みのマスク */                    ⑧
                        DI();
                        tcADPCM_MK0L = MK0L;
                        tcADPCM_MK0H = MK0H;
                        tcADPCM_MK1L = MK1L;
                        tcADPCM_MK1H = MK1H;
                        MK0L = 0xff;
                        MK0H = 0xff;
                        MK1L = 0xff;
                        MK1H = 0xff;
                        gADPCM.int_adpcm = 1;
                        EI();

                        /* フレーム同期検出の開始とシリアル・インタフェースの転送準備*/   ⑨
                        uADPCM_INP_prep();              /* 1バイト目の準備 */
                        tbADPCM_playbe = 0;             /* 最終データのバッファ格納フラグ・クリア */
                        yCSI_E = 1;                     /* シリアル・インタフェースの動作許可 */
                        yFRAME_IF = 0;                  /* フレーム同期割り込みフラグのクリア*/
                        yFRAME_MK = 0;                  /* フレーム同期割り込みマスクの解除 */
                        tbADPCM_start = 0;              /* 開始要求フラグをリセット */
        }
#endif
```

```
/***************************************
 再生終了判定
 ***************************************/                                    ⑩
    if (tbADPCM_playbe && tbADPCM_endf || gADPCM.cmd == 'e') {
            if (gADPCM.cmd == 'e') gADPCM.status = 'o';
            else                   tbADPCM_endm = 1;

            yFRAME_MK = 1;                  /* フレーム同期割り込みをマスク */
            yCSI_E = 0;                     /* シリアル・インタフェースを停止 */
            tbADPCM_playbe = 0;
            tbADPCM_playb = 0;
            #if ADPCM_SP >= 10
                    tbADPCM_play32 = 0;
                    tbADPCM_play16 = 0;
            #endif
            if (gADPCM.int_adpcm) {
                    /* 全割り込みマスクの復旧 */
                    DI();
                    MK0L = tcADPCM_MK0L;
                    MK0H = tcADPCM_MK0H;
                    MK1L = tcADPCM_MK1L;
                    MK1H = tcADPCM_MK1H;
                    gADPCM.int_adpcm = 0;
                    EI();
            }
    }

    /* 終了通知 */
    if (tbADPCM_endm && !gADPCM.status) {
            tbADPCM_endm = 0;
            gADPCM.status = '!';
    }
#endif

/***************************************
 コマンドの処理
 ***************************************/
    /*** コマンド応答 ***/
    if (gADPCM.status) {
            /* 再生終了および応答メッセージの送信処理 */                        ⑪
            if (p = uUDEF_get_REG(gADPCM.req)) {        /* メッセージ送付先バッファのアドレスを取得します */
                if (p != UDEF_REQ_ERROR) {
                    *p++ = (unsigned char) hADPCM;   /* ADPCMユニットからの返答であることを示します */
                    if (gADPCM.status == '!') *p++ = 'F';           /* 再生終了を示します */
                    else            *p++ = gADPCM.cmd; /* ホスト・コマンドに対する応答の場合を示します */
                    *p++ = 1;                       /* 1バイトのデータが続くことを示します */
                    *p++ = gADPCM.status;           /* ステータス記号 */
                }
                gADPCM.cmd = 0;
                gADPCM.status = 0;
            }
            else return;                /* 返信先ビジーの場合，本関数を再実行(1ラウンドロビン後) */
    }
```

```
    /*** コマンド受付 ***/
    else if (gADPCM.cmd) {
            gADPCM.status = 'o';                    /* 成功ステータスを仮に入力。*/
#if ADPCM_SP > 0
        if (gADPCM.cmd == 'b') {
            /* BEEP出力コマンド処理 */
            if (gADPCM.num < 1)   gADPCM.status = 'p';
            else {
                tpADPCM_lim = (unsigned char *)((unsigned short)gADPCM.para1 * 80);  ⑫
                if (!tbADPCM_playb && !tbADPCM_play16 && !tbADPCM_play32) {
                    tbADPCM_start = 1;
                    tcADPCM_rc = 0;
                    tcADPCM_wc = 0;
                }
                else {
                    tbADPCM_play16 = 0;
                    tbADPCM_play32 = 0;
                }
                if (gADPCM.EQ5) {
                    uI2C_IO_write(CODEC_ADDR, 22*2+0, 0x38);
                    uI2C_IO_stop();
                }
                tbADPCM_playb = 1;
                tbADPCM_playbe = 0;
                tpADPCM_data = 0;
                tiADPCM_bt = 0;
            }
        }
#endif
#if ADPCM_SP >= 10
        else if (gADPCM.cmd == 'p') {
            /* ADPCM再生コマンド処理 */
            if (gADPCM.num < 2 || gADPCM.para1 >= nADPCM_DATA) gADPCM.status = 'p';
            else {
                tcADPCM_num = gADPCM.para1;                                          ⑬
                tcADPCM_rcnt = gADPCM.para2;
                tiADPCM_t = tlADPCM_data[tcADPCM_num];      /* POINTER TMP */
                tcADPCM_BANK = (unsigned char) *(unsigned short*)tiADPCM_t;
                tiADPCM_t += 2;
                tpADPCM_data = (unsigned char*) *(unsigned short*)tiADPCM_t;
                tiADPCM_t += 2;
                tpADPCM_lim = tpADPCM_data + *(unsigned short*)tiADPCM_t - 1;
                tiADPCM_t -= 2;
                if (!tbADPCM_playb && !tbADPCM_play16 && !tbADPCM_play32) {          ⑭
                    tbADPCM_start = 1;
                    tcADPCM_rc = 0;
                    tcADPCM_wc = 0;
                }
                else {
                    tbADPCM_playb    = 0;
                    tbADPCM_play16 = 0;
                    tbADPCM_play32 = 0;
                }
                adpcms_init();                          /* ADPCM伸長ライブラリ関数初期化 */  ⑮

                switch      (tcADPCM_mode[tcADPCM_num]) {
                    case 1: tbADPCM_play16 = 1;
                            if (gADPCM.EQ5) {
                                uI2C_IO_write(CODEC_ADDR, 22*2+0, 0x18);
                                uI2C_IO_stop();
                            }
                            break;
                    case 3: tbADPCM_play32 = 1;
                            if (gADPCM.EQ5) {
                                uI2C_IO_write(CODEC_ADDR, 22*2+0, 0x38);
                                uI2C_IO_stop();
                            }
                            break;
                    default: tbADPCM_start = 0; gADPCM.status = 'p';
                }
            }
        }
```

```
#endif
                else                    gADPCM.status = 'c';
                return;
    }
/*************************************
 キー操作によるボリューム調整
 *************************************/
#if Task_Key_code > 0
    if (gADPCM.vol_key) {
        if (gKey_code.code == 0x1c) {
                if (tcADPCM_vol <= CODEC_VOLLIM - CODEC_VOLSTP) {
                        tcADPCM_vol += CODEC_VOLSTP;
                        uI2C_IO_write(CODEC_ADDR, 54*2+0, tcADPCM_vol);   ⑯
                }
                gKey_code.code = 0;
        }
        else if (gKey_code.code == 0x1d) {
                if (tcADPCM_vol >= CODEC_VOLSTP) {
                        tcADPCM_vol -= CODEC_VOLSTP;
                        uI2C_IO_write(CODEC_ADDR, 54*2+0, tcADPCM_vol);
                }
                gKey_code.code = 0;
        }

        else if (gKey_code.code == '<' || gKey_code.code == ',') {
                if (tcADPCM_aux > 0) {
                        tcADPCM_aux--;
                        uI2C_IO_write(CODEC_ADDR, 47*2+0, tcADPCM_aux);   ⑰
                }
                gKey_code.code = 0;
        }
        else if (gKey_code.code == '>' || gKey_code.code == '.') {
                if (tcADPCM_aux < CODEC_VOLLIM) {
                        tcADPCM_aux++;
                        uI2C_IO_write(CODEC_ADDR, 47*2+0, tcADPCM_aux);
                }
                gKey_code.code = 0;
        }
    }
#endif

/*************************************
 処理の継続判定
 *************************************/
#if   ADPCM_SP > 0
    if (tbADPCM_playb || tbADPCM_play32 || tbADPCM_play16)
                return;         /*  簡易OSを使用しない場合は，なるべく短い間隔で本関数を繰り返し呼び出します。*/   ⑱
#endif
    vDelay_task();              /*  簡易OSを使用しない場合は，10ms間隔で本関数を呼び出します。*/
}
```

(3) 詳細説明

① ADPCMユニットの停止を判定します。簡易OSを使用しない場合は，本関数の呼び出しを停止して，終了関数の呼び出しを行います。

② ADPCMデータが存在するバンクに切り替えます。また音声データ・バッファのリード・カウンタ／ライト・カウンタの差分から空き容量を計算します。
なお，バッファはリング・バッファとして管理します。リード・カウンタ，ライト・カウンタは，0～255の範囲で巡回インクリメントします。

③ 32kbps ADPCM（4ビットADPCM）で圧縮されているデータを，ライブラリ関数を使用して伸長します。圧縮コード1バイトにつき16ビットPCMデータ2つが復号できます。

④ 16kbps ADPCM（2ビットADPCM）で圧縮されているデータを，ライブラリ関数を使用して伸長します。圧縮コード1バイトにつき16ビットPCMデータ4つが復号できます。

⑤ 本アプリケーションでは，バンク0に共用データを置いているため，バンク0に戻します。

⑥ 1つの圧縮データを複数回再生する場合の判定を行います。最終的に終了した場合は，終了フラグを立てます。

⑦ BEEP再生の場合，コマンド・パラメータで指定された経過時間を超えてなければ，波形データを音声バッファに格納します。

⑧ コーデックへのデータ送信を開始する前に，全マスク・レジスタを保存した上で，すべての割り込みをマスクします。78K0においてフレーム同期を安定してとるために必要な処置です。

⑨ 送信する1バイト目の準備をした上で，フレーム同期割り込みを許可します。

⑩ コーデックへのデータ送信終了（音声再生終了）した場合に，フレーム同期やシリアル・インタフェースを停止し，割り込みマスクを元に戻します。（注意：音声再生期間中に他のユニットでマスク状態を変えないようにします）

⑪ 再生終了のメッセージやコマンドに対する返信メッセージを送る処理です。共用関数を使用してコマンドの返信先バッファ・アドレスを取得し，所定の順序でメッセージを格納します。

⑫ BEEP再生コマンドがきたら，開始要求フラグを立てます。ただし，すでにコーデック動作中であれば，開始要求はせずに，BEEP波形データ生成の指示だけ行います。また，コーデックのイコライザの初期化を行います（音声再生のために変更されている場合があるため）。

⑬ 音声再生コマンドが来た場合，再生すべきデータの位置や音声バッファのポインタを初期化します。

⑭ 音声再生を開始するための要求フラグを立てます。ただしコーデック動作中は，要求フラグを立てずに，BEEP波形生成やADPCM伸長の指示をいったん取り消します。

⑮ ADPCMライブラリ関数を使用して初期化を行います。また，音声データの圧縮タイプに応じたフラグを立てて，コーデックのイコライザ（高域カット・フィルタとして使用）を設定します。

⑯ キー・コードに応じて，音量アップまたは音量ダウンしたボリューム値をコーデックに設定します。

⑰ キー・コードに応じて，アナログ入力（方形波メロディのミキシング用）ボリュームの値をアップまたはダウンします。

⑱ 再生中かどうかで，継続して本関数を呼び出すか，ある程度の間隔で呼び出すか（コマンドの判定用）を決めます。

3. 4　ADPCMデータ伸長関数

本サンプル・プログラムに添付してあるADPCM伸長ライブラリは簡易版です。正式版の入手方法や差し替え方法，簡易版との違いについては，adpcms.hを参照してください。

パソコン上で圧縮データを作るためのツールも正規版と同時に申込みいただけます。

なお，圧縮データは，簡易版でもそのまま使用できます。

3. 5　データ転送準備関数

本サンプル・プログラムでは，音声データ・バッファの管理用ポインタをバンク2に専用に割り当てて処理の高速化を計っています。本関数は，データ送信に先だって送信1バイト目のデータを準備します。

(1) バンク2レジスタと変数定義の関係

バンク2レジスタ	Data_a.asm内での定義	ADPCM_c.c内での宣言
Cレジスタ	_tcADPCM_rc equ 0FEEAh	extern unsigned char tcADPCM_rc;　/* リード・カウンタ */
Eレジスタ	_tcADPCM_wc equ 0FEECh	extern unsigned char tcADPCM_wc;　/* ライト・カウンタ */
HLレジスタ	_tpADPCM_fifo equ 0FEEEh	extern unsigned char *tpADPCM_fifo;　/* バッファ・ポインタ */ （初期化の中でバッファ先頭アドレス=&tcADPCM_fifo[0]を設定）

(2) プログラム・リスト

```
void uADPCM_INP_prep(void) {

#asm
        sel     RB2                     ①

        ;/* 上位バイト準備 */
        mov     a,[hl+c]                ②
        inc     c

        sel     RB0                     ③
#endasm

    return;
}
```

(3) 詳細説明

① レジスタ・バンクを2に切り替えます。

② レジスタAにリード位置のバッファ内容（= tcADPCM_fifo[tcADPCM_rc]）を格納します。
また，リード・カウンタをインクリメント（tcADPCM_rc++）します。

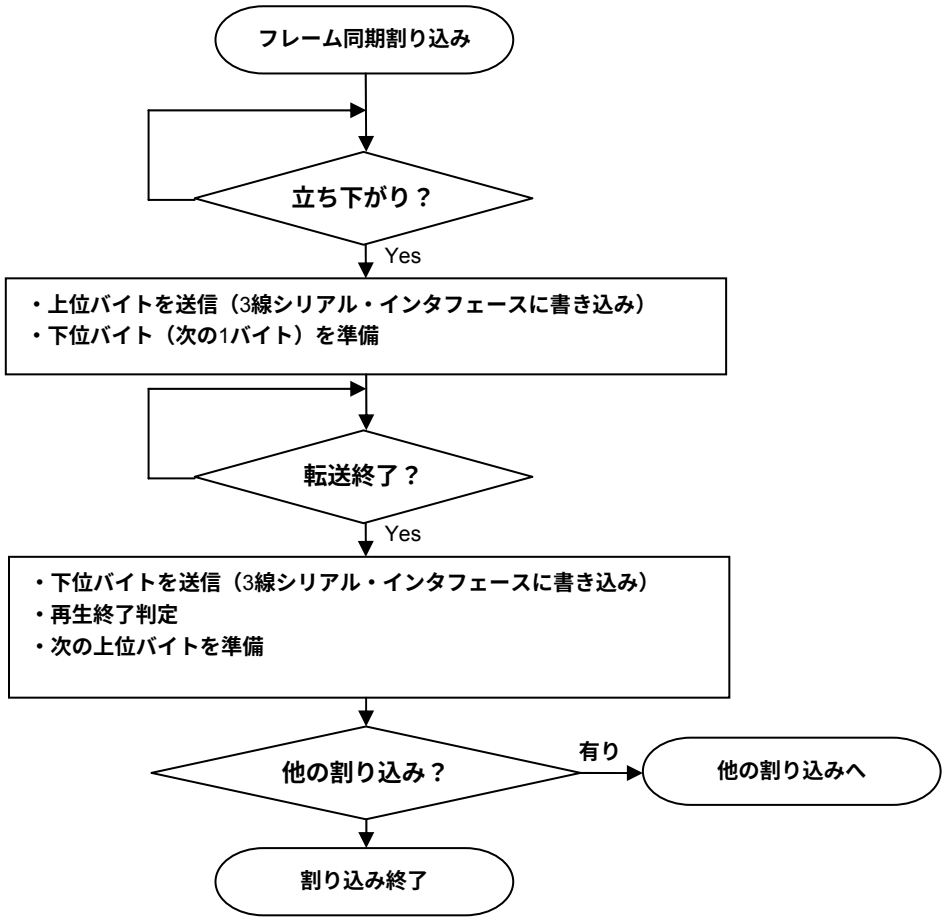
③ レジスタ・バンクを元に戻します。

3. 6　フレーム同期処理

フレーム同期割り込みがあるごとに，音声データをコーデックへ転送します。フレーム同期割り込みが許可されている間は他の割り込みはマスクされていますが，転送後に他の割り込みフラグを判定することにより他の割り込みも受付可能にしています。

(1) 概略フロー

図3－6　フレーム同期処理の概略フローチャート



(2) プログラム・リスト

```
void eADPCM_INP(void) {                                                    ①

    /* 同期信号立ち下がり待ち */
    while(zFRAME);                                                         ②

    /* 上位バイト転送と下位バイトの準備 */
    yCSI_SIO = __geta();                    /* レジスタaの内容をCSIへ書き込み */ ③
    yCSI_IF = 0;
    __asm("mov a, [hl+c]");
    __asm("inc c");

    /* 上位バイト転送終了待ち */
    while(!yCSI_IF);                                                       ④

    /* 下位バイト転送 */
    yCSI_SIO = __geta();                    /* レジスタaの内容をCSIへ書き込み */ ⑤

    /* 再生終了判定 */
    __asm("mov a, e");
    __asm("cmp a, c");                                                     ⑥
    __asm("bnz $$+4");
    tbADPCM_endf = 1;

    /* 次の上位バイトの準備 */
    __asm("mov a, [hl+c]");                                                ⑦
    __asm("inc c");

    /*********************************************/
    /* 他の割り込み処理                          */
    /*********************************************/

#if Task_COM1_rx > 0
    /* ホスト・コマンド受信 */                                             ⑧
    if (yTRxD_SRIF) {
            yTRxD_SRIF = 0;
            __asm("extrn _eCOM1_rx_INT");
            __asm("br !_eCOM1_rx_INT");
    }
#endif

    /* システム・タイマのカウント */
    if (yBTIMER_IF) {
            yBTIMER_IF = 0;
            __asm("extrn _eBTimer_INT");
            __asm("br !_eBTimer_INT");
    }

#if Task_COM1_tx > 0
    /* ホストへのメッセージ送信 */
    if (yTRxD_STIF) {
            yTRxD_STIF = 0;
            __asm("extrn _eCOM1_tx_INT");
            __asm("br !_eCOM1_tx_INT");
    }
#endif

#if Task_Calendar > 0
    /* 時計(RTC)カウント */
    if (WTIF) {
            WTIF = 0;
            __asm("extrn _eCalendar_INT");
            __asm("br !_eCalendar_INT");
    }
#endif

}
```

(3) 詳細説明

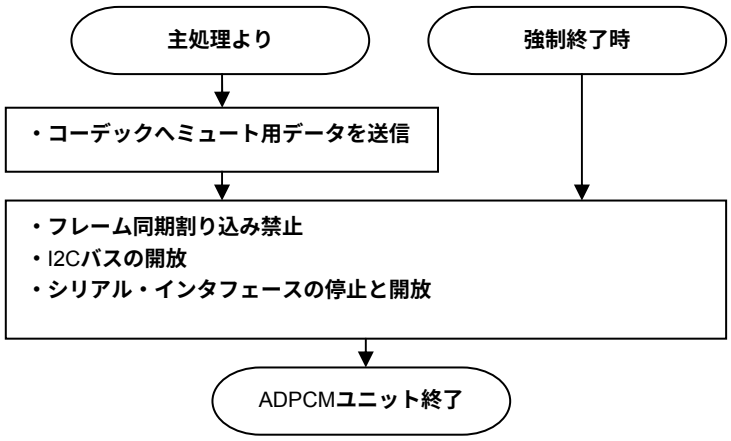
① 関数eADPCM_INPは，ADPCM_c.c先頭付近でレジスタ・バンク2を使用する割り込みとして定義してあります。
② フレーム同期信号の立ち下がりを待ちます。
③ あらかじめ準備したデータをシリアル・インタフェースに書き込み，次のデータの準備も行います。なお，__geta関数は，コンパイラ（CC78K0）の拡張機能で，レジスタAの値を取得する関数です。
④ シリアル・インタフェースのデータ転送終了を待ちます。
⑤ 次のデータをシリアル・インタフェースに書き込みます。
⑥ 音声データ・バッファのリード・カウンタとライト・カウンタが等しくなったら転送終了と判定します。
⑦ 次の上位バイトを準備しておきます。
⑧ 他の割り込み要求フラグを優先順に調べて，フラグが立っていれば当該割り込みへ飛びます。

3.7 終了処理

コーデックをミュートし，シリアル・インタフェース等の資源を開放して処理を終了します。

(1) 概略フロー

図3－7　終了処理の概略フローチャート



(2) プログラム・リスト

```
void eADPCM_stop(void) {
        /* コーデックの終了設定 */
        uI2C_IO_write(CODEC_ADDR, tcADPCM_close[0], tcADPCM_close[1]);      ①
        uI2C_IO_write(CODEC_ADDR, tcADPCM_close[2], tcADPCM_close[3]);
        vTrans_task(eADPCM_end);
}

void eADPCM_end(void) {

#if ADPCM_SP > 0
        /* FRAME割り込み禁止 */                                              ②
        yFRAME_MK = 1;
        yPCL_OE = 0;
#endif

        /* I2C_IO ロック解除 */                                              ③
        gI2C_IO.adpcm = 0;

        /* CSI開放 */
#if ADPCM_SP > 0
        if (tbADPCM_CSI) {
                /* 資源開放 */                                               ④
                yCSI_MK = 1;
                yCSI_E = 0;
                vSignal_sema(yCSI_sem);

                /* 資源フラグ・リセット */
                tbADPCM_CSI = 0;
        }
#endif

        vExit_task();                   /* 簡易OSを使用しない場合は,return文にします。 */
}
```

(3) 詳細説明

① コーデックへ終了時のデータを送信します。

② フレーム同期割り込みのマスクと，マスタ・クロック出力の停止を行います。

③ I^2Cバスの使用終了をI2C_IOユニットへ通知します。

④ シリアル・インタフェースを停止し，共有フラグ（yCSI_sem）を開放します。共有制御が不要であれば，単にシリアル・インタフェースの停止のみ行います。

第4章　サンプル・アプリケーションのビルド方法

4.1　フォルダ構成

本サンプル・アプリケーションは下記からダウンロードできます。

http://www.necel.com/micro/ja/designsupports/sampleprogram/78k0/index.html

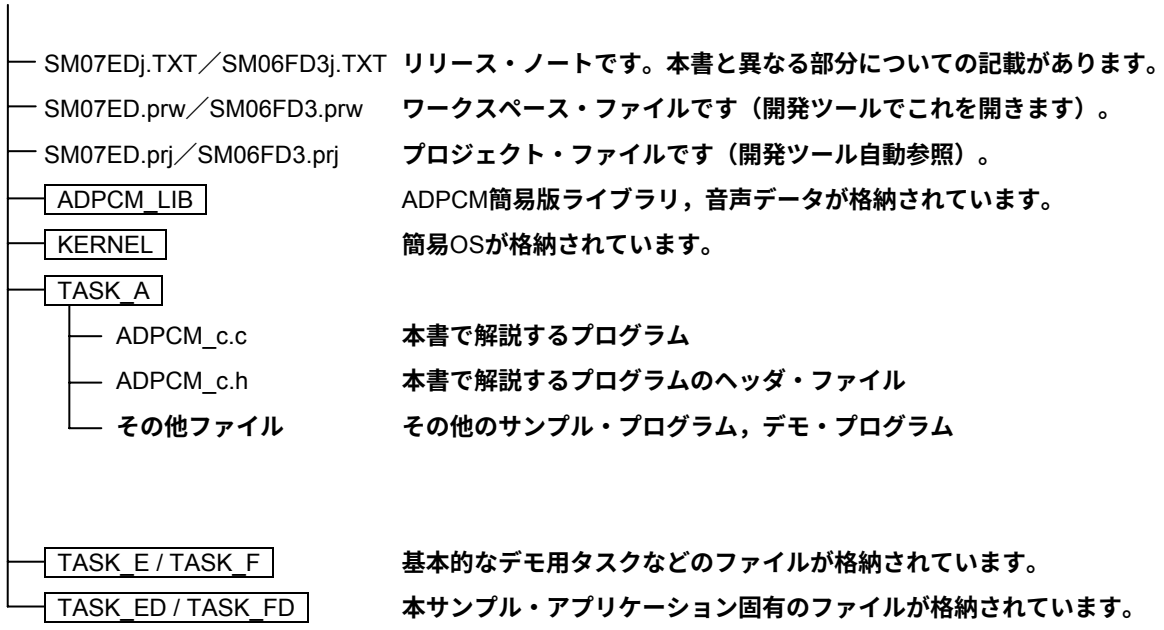
ダウンロードしたファイルの名称は，次のようになっています。

SM07ED_Rxxx（xxx**はリビジョン）・・・**SM07A**ボード**＋SM06E**ボード用**

SM06FD3_Rxxx（xxx**はリビジョン）・・・**SM05A2**ボード**＋SM06B2**ボード**＋SM05F3**ボード用**

フォルダの構成は次のようになっています。

SM07ED_Rxxx.zip／SM06FD3_Rxxx.zip（xxx**はリリース番号です）**



4.2　実行モジュールの作成

本サンプル・プログラムの実行モジュール作成について説明します。
あらかじめ，推奨バージョン以上のソフトウエア・ツールをインストールして下さい。
また，使用するマイクロコントローラのデバイス・ファイルもインストールして下さい。

4.2.1　プロジェクト・ファイルによるビルド

(1) ワークスペース・ファイルを開く

次のワークスペース・ファイルを開いてPM＋を起動します。

・SM07ED.prw／SM06FD3.prw

拡張子.prwで別のアプリケーションが立ち上がるようになっている場合は，PM＋起動後に
「ファイル」→「ワークスペースを開く」→上記ファイル指定
を行ってください。

(2) ツール選択し直し

① 初めてプロジェクト・ファイルを開くと，バージョンに関する警告が出ます。
② ＯＫを押すと，選択画面が出ますが，いったんキャンセルします。
③ 「プロジェクト」→「プロジェクトの設定」→「ツールバージョン設定」→「詳細設定」で，画面を開き，使用するツールにチェックをつけます。

(3) ビルド

① 「ビルド」→「ビルド」（または「リビルド」）によりビルドを開始します。
② ビルド・エラーが出た場合で，最新バージョンのツールを使用してない場合は，念のため最新バージョンのツールをインストールしてビルドし直してみてください。
③ 正規版ツールの場合，次のソース・ファイル入れ替えてビルドを行うと，後述のサンプル音声番号2および3の音声がすべて聞けます。
・プロジェクト・ウインドウのソース・ファイル一覧に表示されているSound_sample_a.asmを右クリックして，「ソース・ファイルの削除」を行います。
・プロジェクト・ウインドウのソース・ファイルを右クリックし，「ソース・ファイルの選択」を行います。ADPCM_LIBの下のSound_a.asmを選択します。

4.2.2　プロジェクト・ファイルの新規作成

ダウンロード・ファイルに同梱されているワークスペース・ファイルを使わずに，新規にプロジェクト・ファイルを作成してビルドする方法について説明します。

(1) ワークスペース生成

PM+を起動します。デフォルト設定にするため，他のプロジェクト使用後は立ち上げ直してください。
「ファイル」→「ワークスペースの新規作成」により，以下の設定を行います。

① ワークスペース情報の設定

・ワークスペース・ファイル名を適当に決めて入力します。
「空のワークスペース作成」にはチェックをつけません。
・ダウンロード・ファイルを解凍したフォルダ位置を指定します。
・マイクロコントローラ名（78K0）とデバイス名を選択します。デバイス名は，SM06Eボードの場合はuPD78F0537_64，SM05F3ボードの場合はuPD78F0547_80を選びます。

② 使用ツールの指定

「詳細設定」を押して使用するツールを選択します。

③ ソース・ファイルの設定

以下のソース・ファイルを追加します。

・ADPCM_LIBディレクトリ
正規版ツール　：Sound_a.asmを選択します。
フリー・ツール：Sound_sample_a.asmを選択します。一部の音声が途中で途切れます。
・Kernelディレクトリ
Kernel_a.asm，Kernel_data_a.asm，Kernel_OPT_a.asm，
UDEF_data_a.asm，UDEF_func_c.c
・TASK_Aディレクトリ
ADPCM_c.c，COM1_rx_c.c，COM1_tx_c.c，
Key_code_c.c（キーによる音量調整を行う場合）
・TASK_E／TASK_Fディレクトリ
BTimer_c.c，BTimer_INT_a.asm（TASK_Fのみ），Data_a.asm，I2C_IO_c.c，
Kernel_init_c.c，Main_c.c

(2) 関連ファイル指定

プロジェクト・ウインドウの「プロジェクト関連ファイル」を右クリックして下記を追加します。

・ADPCM_LIBディレクトリ：adpcms.lib
・TASK_ED／TASK_FDディレクトリ：SM07ED.dr／SM06FD3.dr

(3) オプション設定

「ツール」メニューで各種オプション設定を行います。

① コンパイラ・オプション

・プリプロセッサ

定義マクロ：KOS_CODE_TYPE=0

インクルード・ファイル・パス：ADPCM_LIB，Kernel，TASK_xxを追加します。

・出力

アセンブラ・ソースの出力：チェックを付けます。

② アセンブラ・オプション

・その他

インクルード・ファイル・パス：コンパイラ・オプションと同じ設定をします。

シンボル定義：コンパイラ・オプションの定義マクロと同じ設定をします。

③ リンカ・オプション

・出力１

出力ファイル名：必要に応じて指定します。

オンチップ・デバッグ：MINIQUBE2などでデバッグする場合はチェックします。

(4) ビルド

「ビルド」→「ビルド」（または「リビルド」）によりビルドを開始します。

エラーが出る原因としては，次が考えられます。

・必要なソース・ファイルの指定や関連ファイルの指定が抜けている。

・オプション設定が正しくない。特に他のアプリケーションをビルドした後で，本書指定外のオプションが残ったままになっている場合。

第5章　サンプル・アプリケーションの実行方法

この章では，プログラムの実行方法とホスト・マシンからサンプル・プログラム（ADPCM_c.c）を動作させるためのコマンド例について説明します。

デモ用ボードの設定とホスト・マシンのターミナル・ソフトウエアの設定方法については，使用するボードによって異なります。以下のマニュアルを参照してください。

（1）SM07Aボード＋SM06Eボード

・「ヒューマン・マシンI/Fデモ用ベース・ボード　ユーザーズ・マニュアル（U20117J）」
・「ヒューマン・マシンI/Fデモ用78K0ボード　ユーザーズ・マニュアル（U20118J）」

（2）SM05A2ボード＋SM06B2ボード＋SM05F3ボード

・「漢字表示デモンストレーション用ベース・ボード　ユーザーズ・マニュアル（U19207J）」
・「漢字表示デモンストレーション用78K0R/KF2ボード　ユーザーズ・マニュアル（U19208J）」
・「漢字表示デモンストレーション用拡張ボード ユーザーズ・マニュアル（U19526）」

5. 1　プログラムの書き込みと起動方法

デバッガによる起動とデバッガを使わずにボード単体で起動する方法について説明します。

5. 1. 1　デバッガで起動する場合

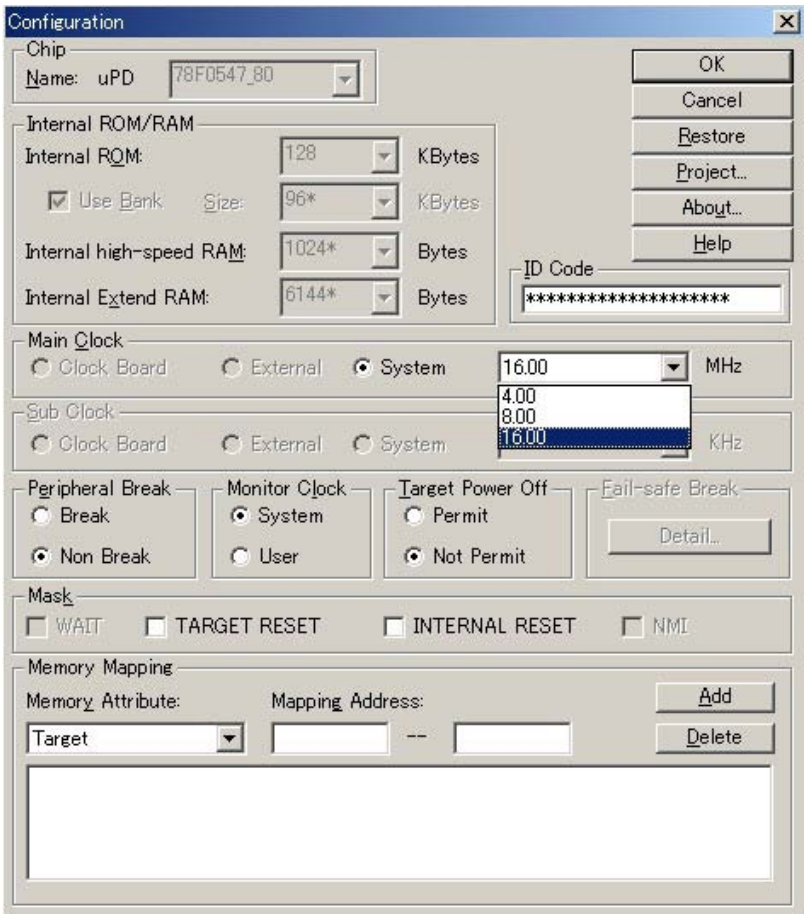
ビルド後，デバッガ（ここではID78K0-QBを使用します）を起動します。

（1）初回起動時にエラーが出た場合は消去（イレース）を行います。

初回起動時に「このデバイスではオンチップデバッグができません」とメッセージが表示されることがあります。この場合はQB-Programmerなどで消去を行ってからデバッガを起動してください。

(2) メイン・クロックの周波数設定

OCDボード上に発振器をつけていない場合は，システムクロックの選択が出来ますので，16MHzを選択してください。ここで選択した周波数は動作周波数とは異なります。動作周波数はデバイス・ボードに実装されている発振子周波数となり，SM05F3は16MHz，SM06Eは20MHzになります。



5. 1. 2　プログラマで書き込んで起動する場合

ビルドで作成されたHEXファイルをプログラマで書き込み後，ベース・ボードからプログラマを外すとデバイス・ボードが起動します。

5. 2　ターミナル・ソフトウエアの操作について

サンプル・アプリケーションが起動すると，ホスト・マシンのターミナル・ソフトウエアからコマンドを入力できます。コマンド入力時の基本操作について以下説明します。

(1) キーボードからのコマンド入力

コマンド入力時の注意点を説明します。

① コマンドの編集

コマンドはリターン・キーを押すと確定します。リターン・キーを押すまでは，バックスペースによる修正はできますが，そのほかの編集キーによる操作は無効です。

② デモボードからの表示

入力中にデモボードからのメッセージが表示されても，表示がなかったものとして，続けて入力してください。

(2) コピー＆ペーストによるコマンド入力

コマンドは，後述のコマンド例をコピー＆ペーストしても動作します。コメントも含めて複数行を一括で貼り付けしてもかまいません。

PDFファイルからのテキスト・コピーでは動作しない場合は，ダウンロード・ファイルのリリース・ノートの後ろにあるコマンド・サンプル集からコピーしてください。

5. 3　コマンドの書式

(1) パラメータ・ライト・コマンド

動作パラメータを書き込むためのコマンドです。アドレスとパラメータ内容を次の書式で入力します。

書式：$mW' {設定開始パラメータ・アドレス}　{パラメータ・データの並び}

表5－1　パラメーター覧

アドレス	内容
0	再生終了メッセージの自動報告先のユニット記号を設定します。デフォルト値は“ u ”です。 本サンプル・アプリケーションでは変更不要です。
1	bit6：1なら以下のキー操作による音量制御を行います。デフォルト値は1です。 →キー：音量アップ（スピーカ・ボリュームを6 dBアップ） ←キー：音量ダウン（スピーカ・ボリュームを6 dBダウン） ＞キー：AUXミキシング音量アップ（AUXボリュームを3 dBアップ） ＜キー：AUXミキシング音量ダウン（AUXボリュームを3 dBダウン） bit5：1ならADPCMのビット数に応じて高域イコライザを設定します。デフォルト値は1です。イコライザのカットオフ周波数と調整量は次のとおりです（ADPCM-SP2ライブラリおよびその簡易版での伸長時）。 2ビットADPCM：5.3 kHz, -12dB 4ビットADPCM：6.9 kHz, -12dB
2～3	BEEP周波数をリトル・エンディアンで設定します。デフォルト値は1000H（500 Hz）です。 上限はおおむね3kHzです。 出力周波数 = 設定値／2000H　[kHz]

(2) ADPCM再生コマンド

サンプル音声として実装しているデータの番号と繰り返し再生回数を次の書式で入力します。

書式：$mp' {データ番号}　{繰り返し回数}

なお，繰り返し回数0で1回再生となります。またFFを指定した場合は停止コマンドを受け付けるまで繰返し再生を行います。

表5－2　実装されているADPCMデータ

2ビットADPCM データ番号	4ビットADPCM データ番号	再生内容
0	1	「料金は300円です」
2	3	「現在の設定温度は25度です」（Sound_sample_a.asmを組み込んだ場合は途中までです）
4	5	「3階，家具売場です」

(3) 再生停止コマンド

ADPCM再生およびBEEP出力を次のコマンドで中止出来ます。

書式：$me

(4) BEEP出力コマンド

BEEP音を出力するには次のコマンドを使います。音の長さは10ms単位で指定します。

書式：$mb' {長さ}

(5) コーデックのレジスタ設定

レジスタ設定を行うにはI2C_IOユニットに対するコマンドを使用します。

SM06FD3の場合は，まずコーデック・アドレスを次のコマンドで設定します。

書式：$IW'0 34

SM07EDの場合は，デフォルト値が34Hになっているので設定不要です。

次のコマンドでレジスタに値を書き込みます。

書式：$ID' {レジスタ番号} {設定データ}

ただし，{レジスタ番号} の上位7ビットにはコーデックのレジスタ・アドレス7ビット，下位1ビットにはコーデックの設定データの最上位ビットを格納します。また {設定データ} にはコーデックの設定データの下位8ビットを格納します。

5.4 コマンド例

(1) ADPCM再生と停止の例

「料金は300円です」を3回再生　$mp'1 2

「3階，家具売場です」を繰り返し再生　$mp'5 FF

SM07EDでは，この状態で←キー，→キーで音量調整が出来ます。

SM06ED3では，再生していない時だけキーで音量調整が出来ます。

再生停止　$me

(2) BEEP出力例

1kHzに変更　$mW'2 0020

1秒間出力　$mb'64

(3) コーデック設定例

SM06FD3の場合　$IW'0 34

スピーカ・ボリュームを-40dBに設定　$ID'6C 11

BEEP1秒間出力　$mb'64

スピーカ・ボリュームを-25dBに設定　$ID'6C 20

BEEP1秒間出力　$mb'64

【発行】NEC**エレクトロニクス株式会社（**http://www.necel.co.jp/**）**

【問い合わせ先】http://www.necel.com/contact/ja/