AN1367

Microchip 社製 PIC32MX 32 ビット MCU への Helix MP3 デコーダの移植

Author: *Sunil Fernandes*
Microchip Technology Inc.

はじめに

MPEG-1、MPEG-2、MPEG-2.5 Layer 3 (MP3) オーディオ エンコーディング フォーマットは、コンシューマ向けのオーディオ ストレージやデジタル オーディオ プレーヤで広く用いられているオーディオ フォーマットです。各種のビットレート (可変ビットレートも可能) とオーディオ サンプリング レートが使える MP3 アルゴリズムは、幅広いマルチメディア アプリケーションに適しています。

本書では、オープンソースの Helix MP3 デコーダ アルゴリズムを Microchip 社製 PIC32MX 32 ビット マイクロコントローラ(MCU)に移植する方法について説明します。本書には、Helix MP3 デコーダを使った MP3 プレーヤ アプリケーションのデモ用ソースコードが付属します。この MP3 プレーヤ アプリケーションは、USB フラッシュ ドライブ (本書ではサムドライブと呼ぶ) から MP3 ファイルを読み出すために Microchip 社製 USB スタックを使用し、タッチスクリーン対応のグラフィカル ユーザ インターフェイス (GUI) を実装するために Microchip 社製グラフィック スタックを使用します。

場合によっては、アプリケーションの目標要件を満たすために、オープン ソースコードにお客様独自のコードを追加する事が必要です。オープン ソースコードと一緒に静的コンパイルする場合、この独自コードはオープンソースのエンドユーザ ライセンス契約の対象となる場合があります。多くの場合、これはアプリケーション所有者には受け入れられません。従って本書では、アプリケーションの知的所有権を保護可能とする RTLL (Run-Time Library Loading) 方式について説明します。

本書の構成は以下の通りです。

1. Helix MP3 デコーダ ライブラリの説明
2. デモ アプリケーションでの RTLL 方式の使用
3. デモ アプリケーション コードの説明
4. デモ アプリケーションのコンパイル / 実行手順

HELIX MP3 デコーダについて

Helix MP3 デコーダは、浮動小数点対応と固定小数点対応の双方で実装可能です。PIC32MX マイクロコントローラへのアルゴリズムの移植には、固定小数点実装を使います。このアルゴリズムは全ての 32 ビット固定小数点プロセッサで動作します。コードは全て C 言語で書かれていますが、一部のコードは最適化したアセンブリ命令に置き換える事も可能です。

Helix MP3 デコーダは MPEG1、MPEG-2、MPEG-2.5 向けの Leyser 3 をサポートします。このデコーダは可変ビットレート、固定ビットレート、ステレオおよびモノラル オーディオ フォーマットをサポートします。実装と機能の詳細については、Helix MP3 デコーダのウェブサイト (https://datatype.helixcommunity.org/Mp3dec) をご覧ください。

Helix MP3 デコーダのソースコードはオープンソースであり、付属するファイルに記載されたライセンス条件が適用されます。Helix MP3 デコーダは無償で使えるオープンソースですが、MP3 アルゴリズム自体は無償ではなく、ロイヤルティが発生します。MP3 アルゴリズムを使うには、これらのロイヤルティを支払う必要があります。詳細は www.mp3licensing.com をご覧ください。

PIC32MX マイクロコントローラへの Helix MP3 デコーダの移植

PIC32MX プラットフォームに移植する Helix MP3 デコーダ ソースコードは、Helix MP3 デコーダのウェブサイトからダウンロードできます。ウェブページの指示に従ってソースコードをダウンロードしてください。あるいは、本書に付属する Helix MP3 デコーダ ソースコードを使う事もできます。このソースコードは、PIC32MX デバイス上で Helix MP3 デコーダが動作できるように変更済みです。最新のソースコードについては、Helix MP3 デコーダのウェブサイト (https://datatype.helixcommunity.org/Mp3dec) をご覧ください。

ダウンロードしたソースコード内の各フォルダには、3 つのライセンス ファイル (`RPSL.txt`、`RCSL.txt`、`LICENSE.txt`) が保存されています。ユーザには、これらのライセンス ファイルの記載事項に同意する事が求められます。

Helix MP3 デコーダのウェブサイトからダウンロードした場合、ソースコードは「fixpt」フォルダ内にあります。デコーダ ソースコードを PIC32MX に移植する手順は以下の通りです。

1. 「\fixpt\real」フォルダ内の assembly.h ファイルを開く。
2. MIPS プラットフォーム向けの FASTABS 関数定義 (このファイルの行 337) へ移動する。
   この関数は、MIPS アセンブリで書かれた絶対関数です。
3. この関数のボディをコメント化し、C コードで置き換えた後に (例 1)、ファイルを保存する。
4. 「fixpt」フォルダ内の mp3dec.c ファイルを開き、行 47 をコメント化する (例 2 参照)。
   Helix MP3 デコーダを PIC32MX に移植する場合、この機能は不要です。

これらの手順は、バージョン 1.8 の assembly.h ファイルとバージョン 1.6 の mp3dec.c ファイルに適用されます。

以上は、Helix MP3 デコーダのウェブサイトからダウンロードしたソースコードに必要な変更内容です。本書に付属する Helix MP3 デコーダ ソースコードには、これらの変更を適用済みです。

例 1:

```
static __inline int FASTABS(int x)
{
//  int t=0; /* Really is not necessary to initialize only to avoid warning.*/
//
//  __asm__ volatile (
//      "sra %0, %1, 31 \n\t"
//      "xor %1, %1, %0 \n\t"
//      "sub %0, %1, %0 \n\t"
//      :"=&r" (t)
//      :"r" (x)
//   );
//
//  return t;

    /* Commented out the above code as it causes problems while decoding some files */
    /* on MIPS M4K core.*/
    return((x > 0) ? x :-(x));
}
```

例 2:

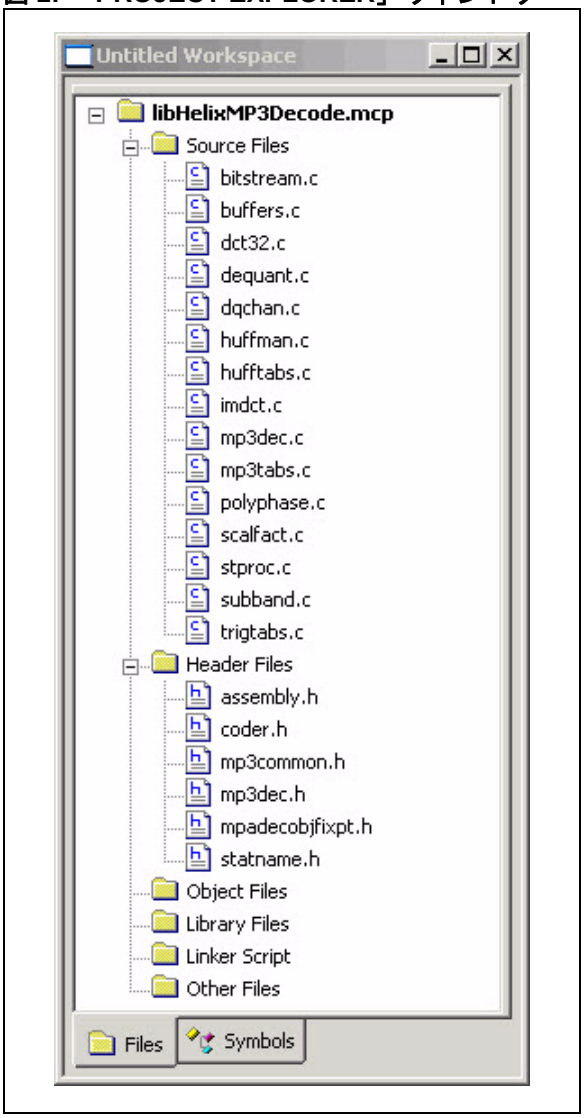
```
#include "string.h"           /* for memmove, memcpy (can replace with different */
                              /* implementations if desired) */

#include "mp3common.h"        /* includes mp3dec.h (public API) and internal, */
                              /* platform-independent API */

//#include "hxthreadyield.h"
```

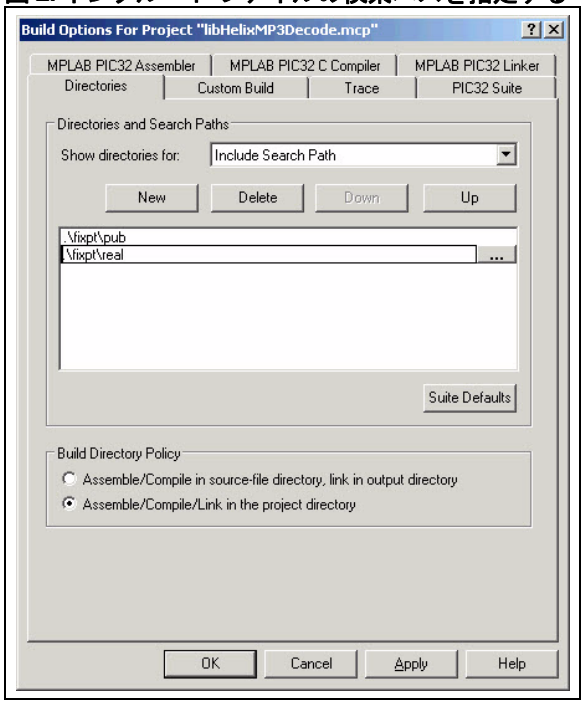
Helix ソースコード ファイルを含めたプロジェクトをビルドするには、「fixpt」および「fixpt\real」フォルダ内の全ての「.c」ソースコード ファイルをインクルードする必要があります。必要に応じて、プロジェクトに全てのヘッダファイルをインクルードできます。図 1 は、MPLAB IDE の「Project Explorer」ウィンドウに表示されたファイルリストです。

図 1: 「PROJECT EXPLORER」ウィンドウ



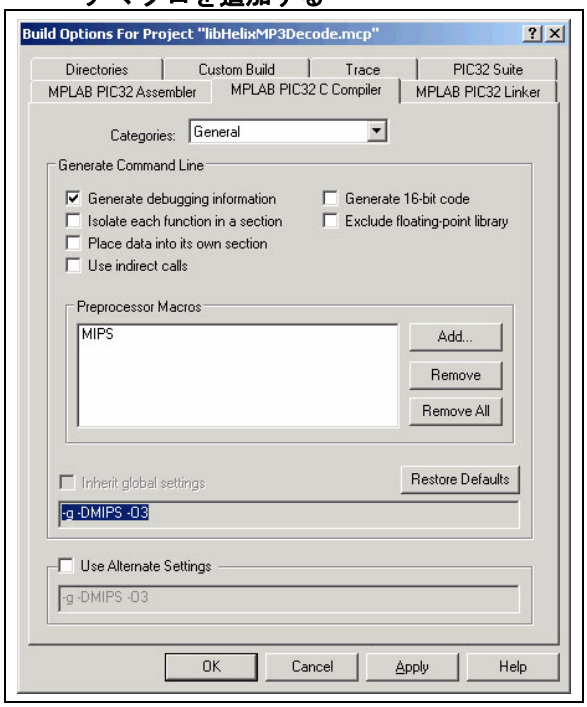
「fixpt\pub」および「fixpt\real」フォルダのパスは、プロジェクト ファイル格納位置からの相対パスで指定する必要があります。
Project>Build Options>Project をクリックし、**[Directories]** タブで「Include Search Path」を選択し、フォルダのパスを MPLAB プロジェクトの格納位置に対する相対パスで指定します (絶対パスで指定する事も可能)。プロジェクト フォルダと「fixpt」フォルダが同じフォルダ内にある場合のオプション選択の例を図 2 に示します。

図 2: インクルード ファイルの検索パスを指定する



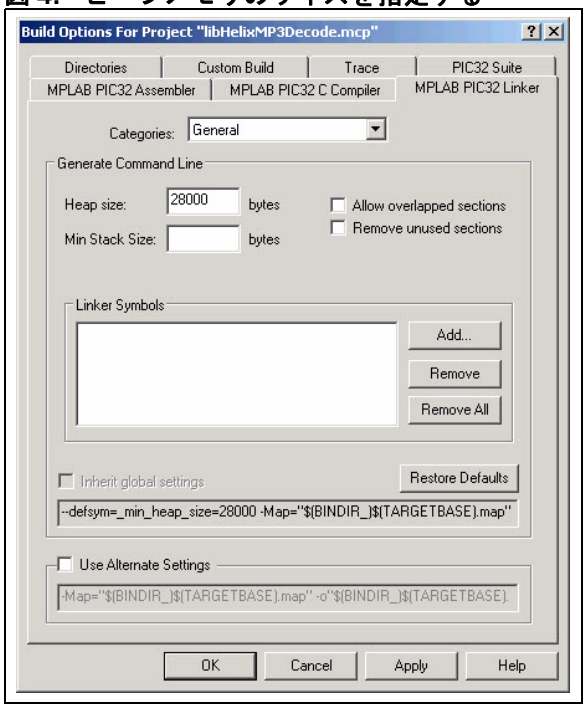
コンパイルする際に、MIPS シンボルを定義する必要があります。これにより、Helix MP3 デコーダ ソースコードを PIC32MX プラットフォーム向けにビルドします。
Project>Build Options>Project をクリックし、**「MPLAB PIC32 C Compiler」** タブで、MIPS マクロをビルドプロセスに追加します (図 3 参照)。

図 3: ビルド オプションに MIPS プリプロセッサ マクロを追加する



このデコーダの動作にはヒープメモリが必要です。デコーダに必要なヒープメモリのサイズ (入出力バッファを含まず) は 28 KB です。コンパイル時に、この値をヒープサイズとして指定する必要があります。*Project>Build Options>Project* をクリックし、**「MPLAB PIC32 Linker」**タブを開き、図 4 のように [Heap size] を指定します。

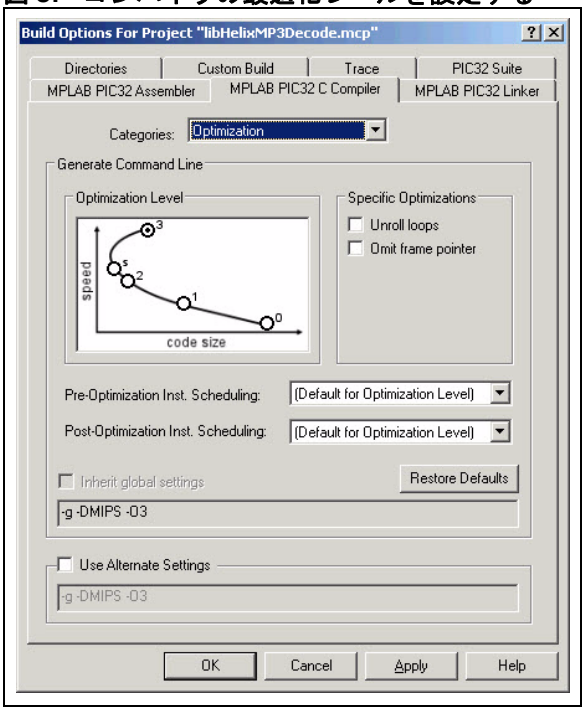
図 4: ヒープメモリのサイズを指定する



Helix MP3 デコーダのソースコードは、最適化レベルを「O3」に設定してコンパイルします。*Project>Build Options>Project* をクリックし、**「MPLAB PIC32 C Compiler」**タブを開き、[Categories] フィールドでドロップダウン リストから「Optimization」を選択します。最適化レベルは「O3」に設定します (図 5 参照)。

以上により、Helix MP3 デコーダ ソースコードを PIC32MX アプリケーションで使うための準備が完了します。

図 5: コンパイラの最適化レベルを設定する



Helix MP3 デコーダ API (Application Program Interface)

Helix MP3 デコーダ API は以下を実行するための関数を提供します。

- デコーダの初期化
- デコーダの終了
- フレーム同期の検出
- フレーム情報 (ビットレート、サンプリング レート等) の取得
- MP3 フレームのデコード

注意： デモ アプリケーションは、RTLL 方式を使って、システムローダ インターフェイス経由で間接的に Helix MP3 デコーダ API を呼び出します。このため、MEB USB Thumb Drive MP3 C32 Demo.mcp ソースコードは、Helix MP3 デコーダ API を直接呼び出すためのインターフェイスを備えていません。

Helix MP3 デコーダを使用するための最初の手順は、デコーダ API を呼び出すために、必要なヘッダファイルを全てのソースファイルにインクルードする事です。インクルードする必要があるファイルは、coder.h と mp3dec.h です (例 3 参照)。

例 3:

```
// The following files must be
// included in the source code to
// invoke the Helix MP3 decoder API

#include "coder.h"
#include "mp3dec.h"
```

Helix MP3デコーダは完全リエントラントです。これは、デコーダステートがデータ構造体内に完全に格納される事を意味します。MP3InitDecoder( ) 関数は、メモリをこのデータ構造体用に割り当てて初期化した後に、初期化済みデータ構造体へのポインタを返します。この関数は動的メモリ割り当てを使い、ヒープメモリの割り当てを必要とします。割り当てに失敗した場合、この関数は値「0」を返します (例 4 参照)。

アプリケーションでデコーダの動作を終了する場合、MP3FreeDecoder( ) 関数を使う事で、デコーダに割り当てたメモリを解放できます (例 5 参照)。この関数は、デコーダが使っていたメモリをクリアします。

通常、MP3 ファイルには、ID3 タグ形式の追加情報が書き込まれています。これらのタグは曲名、アーティスト、アルバム、ジャンル等の情報を格納します。Helix MP3 デコーダ関数へ入力されるフレームはこの情報を含みません。アプリケーションは、ファイルまたは入力ストリーム内で実際に MP3 データが始まる位置を特定する必要があります。これには MP3FindSyncWord( ) 関数を使います。この関数は、入力バッファ(MP3 ファイルからのバイト読み出しの一部) を受け取り、MP3 フレームヘッダ内で同期ワードの位置を特定し、その位置を入力バッファの先頭からのオフセットとして返します。同期ワードは PM3 フレームの先頭を示します。負の値が返された場合、入力バッファには同期ワードまたは MP3 フレームの先頭が含まれていなかった事を意味します(例6参照)。

例 4:

```
// Initialize the Helix MP3 decoder.This will point to the MP3 decoder data
// structure.

HMP3Decoder mp3Decoder;
mp3Decoder = MP3InitDecoder();
if(mp3Decoder == 0)
{
// This means the memory allocation failed.This typically happens if there is not
// enough heap memory.Recompile the code with additional heap memory.

while (1);
}
```

例 5:

```
// Close the decoder.

MP3FreeDecoder(mp3Decoder);

// At this point, memory consumed by the MP3 decoder is released.
```

例 6:

```
// Find a valid MP3 start of frame in the input stream

while(!endOfFile(mp3File))
{

        // Read the input file

        nRead = fread(mp3File,input,MAX_FRAME_SIZE);
        if(nRead == 0)
        {

        // We have reached end of file and a valid MP3 start of frame was not found.
        // Do something.

        NotValidMP3File();
        break;
        }
        else
        {
            offset = MP3FindSyncWord(input,MAX_FRAME_SIZE);
            if(offset < 0)
            {

            // The input buffer does not contain a start of frame.Read another frame.

            continue;
            }
            else
            {

            // We found a start of frame. offset contains location of the start of frame
            // within input buffer.

            foundStartOfFrame = TRUE;
            break;
            }
        }
    }
```

MP3FindSyncWord( ) 関数が同期ワードを検出したとしても、それが MP3 フレームの実際の先頭ではない可能性があります。ID3 タグ内の何らかのデータが同期ワードに一致するといったケースが生じる可能性があります。そのような場合、その入力が MP3 デコーダ関数に引き渡されると、エラーが返されます。アプリケーションは、このエラーを適切に処理する必要があります。

本書に付属する MP3 プレーヤ アプリケーションは、44.1 kHz でサンプリングされ、ステレオ形式でエンコードされた MP3 ファイルを処理するよう設計されています。Helix MP3 デコーダは、処理前の MP3 フレームのオーディオ属性を返すMP3GetNextFrameInfo( )関数を備えています。アプリケーションは、必要に応じてこの情報を使う事ができます。
MP3FindSyncWord( ) 関数は、SOF (Start of Frame) の位置を特定する事で、MP3 フレームを示します。MP3GetNextFrameInfo( ) 関数を使うと、デコード前の MP3 フレームに関する情報を抽出できます。この情報は、MP3FrameInfo 型データ構造体で返されます。MP3 フレームが無効な場合、エラーが返されます (例 7 参照)。

MP3FrameInfo 型データ構造体からは以下の情報が得られます。

- 処理されたフレームのビットレート
- オーディオ チャンネルの数 (1: モノラル、2: ステレオ)
- エンコード時のオーディオ サンプリング レート
- サンプルあたりのビット数
- デコードされたオーディオ フレームのサイズ (オーディオ サンプルの数) (ステレオ サンプルは 2 個として数える)
- MPEG レイヤ
- レイヤ バージョン

MP3GetLastFrameInfo( ) 関数は、デコード後のオーディオ フレームの情報 (内容は上記と同じ) を返します。例 8 に、MP3GetLastFrameInfo( ) 関数の使用方法を示します。

例 7:

```
// Get information about the next frame to be decoded.This assumes that the
// MP3FindSyncWord()function has been used to locate the start of a MP3 frame.

MP3FrameInfo frameInfo;

if(foundStartOfFrame == TRUE)
{
    int error;
    error = MP3GetNextFrameInfo(mp3Decoder, &frameInfo, input);
    if(error == MP3_INVALID_FRAME_HEADER)
    {
            // This means that the MP3FindSyncWord function has found the sync word,
            // but this was not a start of frame.This may have happened because
            // the sync word may have found in an ID3 tag.

            GetAnotherFrame();
    }

else if(frameInfo.sampRate != 44100)
    {
            // For this example, we want only data which
            // was sampled at 44100 Hz.Ignore this frame.

            IgnoreThisFrame();
    }

}
```

例 8:

```
// Get information about the last decoded frame.It is assumed that the frame was
// decoded by calling the decode function.

MP3FrameInfo mp3frameInfo;
int nOutputSamples;

MP3GetLastFrameInfo(mp3Decoder, &mp3FrameInfo);

// Get the size of the output raw audio frame.

nOutputSamples = mp3FrameInfo.outputSamps;
```

MP3Decode( ) 関数は、エンコードされた MP3 フレームをデコードします。この関数は、コア MP3 デコーディング アルゴリズムを呼び出します。1 回のデコード動作では、入力エンコード フレームの全体を処理できない場合があります。その場合、入力ポインタは未処理バイトの先頭へ進みます。デコーダ関数は、未処理バイトの総数も返します。MP3 アプリケーションは、この情報に基づいて入力バッファを準備できます。適正な動作を得るには、新しいデータを未処理バイトの最後に追加する必要があります。

入力フレームは、独立 MP3 フレーム向けに標準 MPEG ストリームとしてフォーマットできます。標準 MPEG ストリーム フォーマットはMP3 プレーヤで幅広く使われています。デコード後の RAW データのフォーマットは、オーディオ チャンネルの数によって異なります。モノラル オーディオデータの場合、オーディオ サンプルは連続して出力されます。ステレオ オーディオデータの場合、左 / 右チャンネルのオーディオ サンプルが交互に (LRLRLR…) 出力されます。デコード関数は、デコード処理の結果を示すエラー値を返します。MP3 アプリケーションは、エラーのタイプに応じて適切な対応を実行できます (例 9 参照)。

例 9:

```
// Decode a MP3 frame.It is assumed that the MP3FindSyncWord() function was used
// to find a start of frame.

short output[MAX_NCHAN * 1152];
int err;
int bytesLeft

bytesLeft = INPUT_BUF_SIZE;
err = MP3Decode(mp3Decoder, &input, &bytesLeft, output, 0);

// bytesLeft will have number of bytes left in the input buffer.Input buffer will
// point to the first unconsumed byte.

// This code example shows how the errors can be handled.
// This may differ between applications.

switch(err)
{
        case ERR_MP3_INDATA_UNDERFLOW:
                CloseMP3File();
                break;
        case ERR_MP3_MAINDATA_UNDERFLOW:
                ReadMoreMp3Data();
                break;
        case ERR_MP3_FREE_BITRATE_SYNC:
                CloseMP3File();
                break;
        default:
                CloseMP3File();
                break;
}

// The MP3GetLastFrameInfo() function can be used to obtain information about the
// frame.The following shows an example.

MP3GetLastFrameInfo(mp3Decoder, &mp3FrameInfo);
if(mp3FrameInfo.outputSamps != 0)
{
        // Write data to output DAC

WriteDataToDAC(output, mp3FrameInfo.,outputSamps);
}
```

表 1 に、PIC32MX マイクロコントローラ上で Helix MP3 デコーダが動作する場合のメモリ要件を示します。表 2 に、計算性能要件を示します。

表 1: HELIX MP3 デコーダのメモリ要件

メモリタイプ	容量 (バイト)	備考
プログラム メモリ	53000	—
データメモリ	28000	デコーダのみによる要求
入力バッファ	1940	最大 MP3 フレームサイズ
出力バッファ	2304	出力バッファが要求する最大容量 (ステレオ オーディオデータの場合)

表 2: HELIX MP3 デコーダの計算性能要件

関数	MIPS	備考
MP3Decode( )	26	最適化レベル「O3」でコンパイルしたコードを使って、80 MHz のプロセッサ クロックで計算

RTLL (RUNTIME LIBRARY LOADING)

アプリケーションにオープン ソースコード コンポーネントを使う場合、使う場合、お客様独自のアプリケーション コードにオープン ソースコードのライセンス条件が適用される可能性があります。これは、アプリケーションの開発者または所有者にとって不都合かもしれません。このため、本書のソースコードに付属する MP3 アプリケーション サンプルでは、以下の手法を使います。

- オープン　ソースコードをメインのアプリケーション ソースコードにリンクしない (別々にコンパイルし、互いにリンクしない)。
- ローダ ユーティリティを使って、実行中にオープン ソースコード ライブラリを一連の関数ポインタとして読み込む。

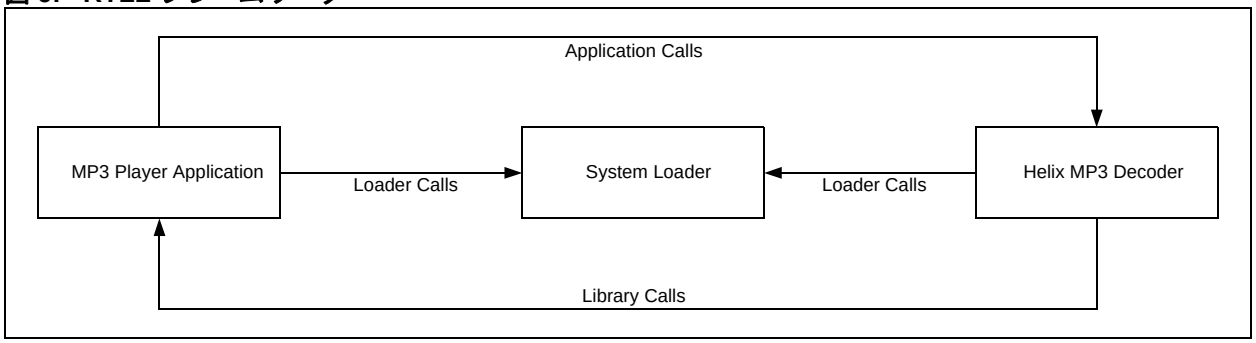
この手法を RTLL (Run-Time Library Loading) と呼びます。MP3 プレーヤ アプリケーションは、RTLL 方式を使って、実行中に Helix MP3 デコーダを読み込みます。図 6 に、RTLL 動作の概念図を示します。

RTLL 方式では、システムローダを使って Helix MP3 デコーダを読み込みます。読み込みプロセス中に RAM を初期化し、プログラムメモリから RAM へテーブルをコピーし (必須ではない)、デコーダからエクスポートされた関数への一連のポインタを取得します。MP3 プレーヤ アプリケーションは、システムローダを使って自身の関数をエクスポートする事で、Helix MP3 デコーダからそれらの関数を呼び出せるようにします。同様に、Helix MP3 デコーダ モジュールにコードを追加する事で、デコーダからシステムローダを介して一連の関数をエクスポートし、メイン アプリケーションからそれらの関数を呼び出せるようにします。

MP3 プレーヤ アプリケーションは 2 つのプロジェクトを含み、それらは 2 つの異なる HEX ファイル (プログラム イメージ) を生成します。1 つのファイルはメイン アプリケーション用 (USB およびグラフィック スタックも含む) に使い、もう 1 つのファイルは Helix MP3 デコーダ用に使います。これら 2 つのプログラム イメージは、デバイス プログラムメモリ内の別々の領域に書き込まれます。このため MP3 アプリケーションは、RTLL 方式を使って Helix MP3 デコーダにアクセスします。この方式を使うには、C32 リンカが使用する既定値リンカスクリプトを変更する必要があります。RTLL 方式を実装するプログラム フラッシュおよびデータ RAM セクションの開始位置を調整する必要があります。Helix MP3 デコーダ向けのこれらのセクションは、それぞれのメモリ領域の最後に配置されます。Helix MP3 デコーダ向けのプログラムメモリ領域は、プログラムメモリの最後から 72 K バイト手前の位置で始まるよう設定され、Helix MP3 デコーダ向けのデータ RAM 領域は、KSEG1 RAM の最後から 256 バイト手前の位置で始まるよう設定されます (Helix 用に十分なスタックメモリを確保)。同様に、MP3 プレーヤ プログラム イメージを作成する際に使うリンカスクリプトは、Helix MP3 デコーダを格納するためにサイズを調整します。その結果、MP3 アプリケーション向けのプログラムメモリは 72 K バイト減少し、KSEG1 RAM は 256 バイト減少します。

本書内の以降の説明では、「メイン アプリケーション」は MP3 プレーヤ アプリケーションを指し、「ライブラリ」は Helix MP3 デコーダを指します。これら 2 つを総称して「モジュール」と呼びます。RTLL 方式は、動的なモジュール ヘッダデータ構造体 (module_dyn_hdr) を使って、モジュールに関する情報を保持します。メイン アプリケーション、システムローダ、ライブラリはそれぞれに固有の動的モジュールヘッダを有します。例 10 に、module_dyn_hdr の定義を示します。

図 6: RTLL フレームワーク



例 10:

```
typedef struct
{
        // module signature
        const char module_sign[_MCHP_DYN_HDR_SIGNATURE_SIZE_];

        // name of the module
        const char module_name[_MCHP_MODULE_NAME_SIZE_];

        // pointer to module init
        const module_init_dcpt* module_init;

        // pointer to the export descriptor
        const export_dcpt*      module_export_tbl;

        // pointer to the import descriptor
        const import_dcpt*      module_import_tbl;

        // pointer to module data
        module_data_dcpt*       module_data;

        // module info
        const module_info_dcpt  module_info;

}module_dyn_hdr;
```

各モジュールの動的ヘッダデータ構造体は、インポートおよびエクスポートされた関数に関する情報を格納します。インポートされた関数は、モジュール インポート テーブル (module_import_tbl) に含まれます。エクスポートされた関数は、モジュール エクスポート テーブル (module_export_tbl) に含まれます。例 11 に、メイン アプリケーション モジュール向けに動的ヘッダを初期化する方法を示します。

例 11:

```
const module_dyn_hdr __attribute__((__section__(".Module_Header_Section"))) _ModuleLoadHdr =
{
        _MCHP_DYN_HDR_SIGNATURE_,           // signature
        MAIN_SERVICES_LIB,                  // lib_name
        0,                                  // module init
        &SystemExports,                     // exports
        &_DefaultModuleImports,             // imports
        &module_data,                       // private data
        _DefaultModuleInfo(0x0100),         // module info

};
```

メイン モジュールは、Helix MP3 ライブラリが必要とする malloc() および free() 関数をエクスポートします。メイン アプリケーション モジュールは、他のモジュールの関数を一切インポートしません。このため、_DefaultModuleImports テーブルは空白です。RTLL 方式を使うと、システムローダを 1 つのモジュールとして読み込む事ができます。しかし本書のアプリケーション例ではこの方法を採らず、システムローダをメイン アプリケーション モジュールへ静的にリンクしています。この選択は、メイン アプリケーション モジュールのコンパイル時に LOADER_STATIC_LINK オプションで指定します。

メイン アプリケーションは、RTLL システムを初期化するために、dlopen() 関数 (AudioUSBTasks.c ファイル内の AudioUSBInit() 関数内) を呼び出します (例 12 参照)。

dlopen() 関数は、p_modules[] 配列で指定されている全てのモジュールを読み込みます。読み込みの際に、各モジュールに対応する起動コード (存在する場合) が呼び出されます。Helix MP3 ライブラリ モジュールは、変更された C ランタイム起動コードを使って、割り当てられたメモリを初期化します。その後 dlopen() 関数は、Helix MP3 ライブラリ モジュールの動的ヘッダを指すポインタを返します。

メイン アプリケーションは、次に dlsym() 関数を使って、Helix MP3 ライブラリ モジュール エントリポイントへのハンドルを取得します (例 13 参照)。

これは、helix_lib.c ファイルで定義されている HelixLibEntry() 関数を指すポインタを返します。この関数は、指定されたアルゴリズムに基づいて、Helix MP3 デコーダ API を呼び出します。例 14 に、HelixLibEntry() 関数実装の一部を示します。

HelixLibEntry() 関数の引数 fCode は、呼び出す Helix ライブラリ関数を指定します。fCode が「0」の場合、MP3InitDecoder() 関数を呼び出します。メイン アプリケーション モジュールは、システムローダ dlsym() を呼び出す事で、HelixLibEntry() 関数へのハンドル (MP3DecoderFunctions) を取得する事に注意してください。

メイン アプリケーション モジュールは、取得した MP3DecoderFunctions ハンドルを使って、Helix MP3 デコーダ API 関数を呼び出します。

例 15 に、メイン アプリケーション モジュールから MP3DecoderFunctions ハンドルを使って MP3Decode() 関数を呼び出す方法を示します。

例 12:

```
/* Get a handle to the MP3 decoder and obtain an entry point to the decoder
 * Library functions.*/

hMP3DecoderLibrary = dlopen("Helix Library", 0)
```

例 13:

```
MP3DecoderFunctions = (void* (*)(int, int, ...))dlsym(
hMP3DecoderLibrary, "HelixLibEntry");
```

例 14:

```
void* HelixLibEntry(int fCode, int nparams, va_list args)
{
        void*   res = (void*)-1;

        // Library entry point
        switch(fCode)
        {
            case 0:
                if(nparams == 0)
                {
                    res = MP3InitDecoder();
                }
                break;

            case 1:
            if(nparams == 1)
            {
                MP3FreeDecoder(va_arg(args, HMP3Decoder));
                res= 0;
            }
            break;
```

例 15:

```
// MP3_DECODE_FUNCTION is fCode value for MP3Decode() function.
// The second argument indicates the total number of arguments that follow.

err = (int)(*MP3DecoderFunctions)(MP3_DECODE_FUNCTION,5,hMP3Decoder, &readPtr, &bytesLeft,
outBuf, 0);
```

MP3 プレーヤ アプリケーション

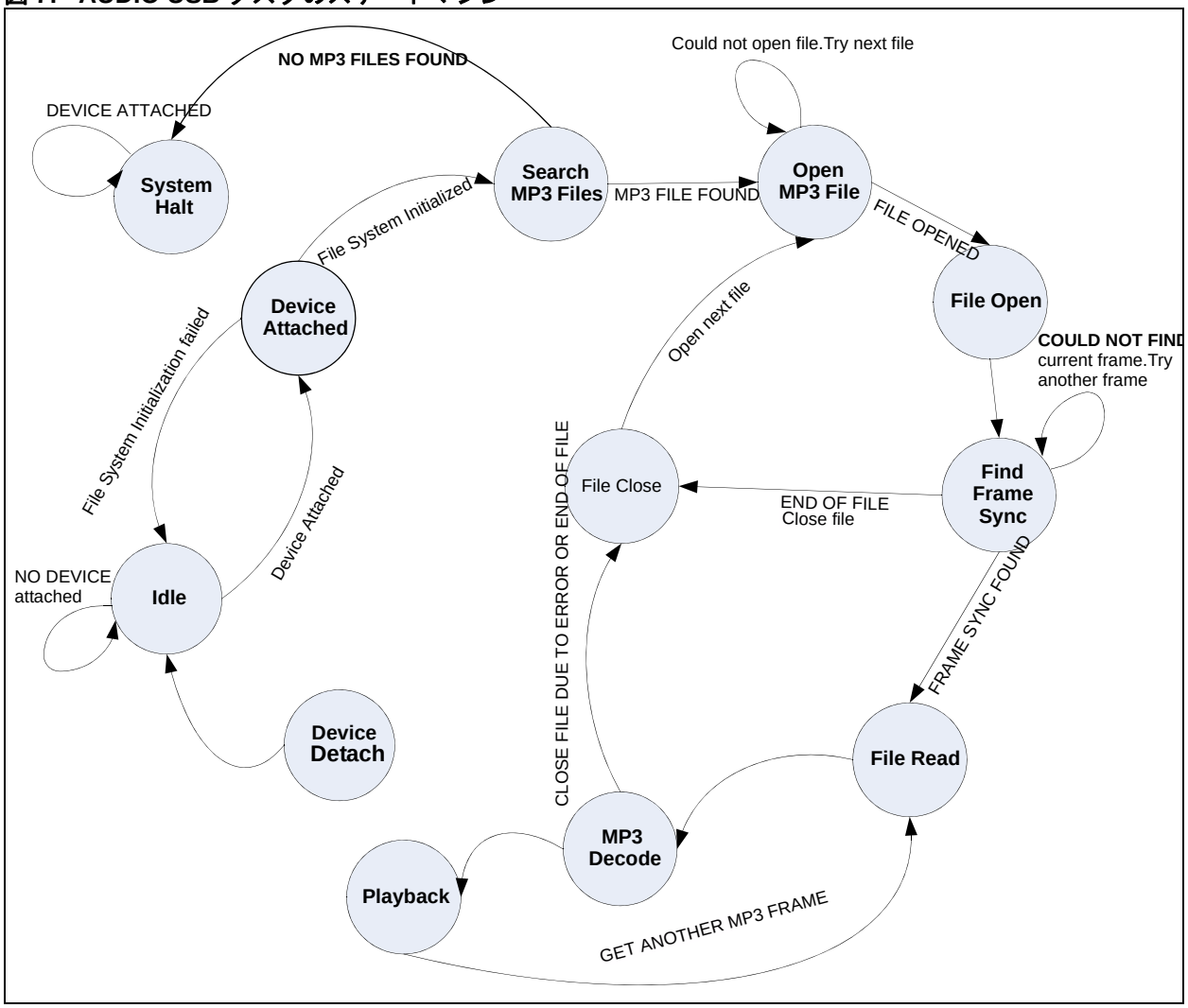
MP3 プレーヤ アプリケーションは、Helix MP3 デコーダ (RTLL 経由) と Microchip 社製 USB スタックおよびグラフィック スタックを使って、MP3 プレーヤを実装します。このアプリケーションは、USB スタックの MSD (Mass Storage Device) ホスト コンポーネントを使って、USB サムドライブを読み出します。グラフィック スタックは、タッチスクリーンおよびユーザ インターフェイス関数を提供します。

本書では、USB およびグラフィック スタックの動作について説明しません。詳細は関連文書を参照してください。MP3 レイヤのコアロジックは、AudioUSBTasks.c ファイル内の AudioUSBTasks( )関数でステートマシンとして実装されます。このステートマシンを図 7 に示します。

「Idle」ステートでは、ステートマシンは USB サムドライブが接続されるまで待機します。サムドライブが接続されると、コードはサムドライブ ファイルシステムの初期化を試みます。アプリケーションは、サムドライブ内の MP3 ファイルを検索します。MP3 ファイルが見つかると、そのファイル名が fileNames[] 配列に保存されます。最大で MAX_FILES 個のファイル名を保存できます。ファイルが見つかると、ディスプレイ上のファイルリスト ボックス内のファイル名が更新されます。最初のファイルが開かれ、コードは SOF (Start of Frame) の最初のインスタンスを検索します。SOF が見つかった場合、フレームの後続の内容がファイルから読み出され、そのフレームはデコード関数へ引き渡されます。デコード済みのオーディオ フレームは、再生のために WM8731 オーディオ DAC デバイスへ送られます。読み出し / デコード動作がファイルの最後に達すると、そのファイルは閉じられて、次のファイルが開かれます。

currentPlaybackFileIndex 変数は、現在処理中のファイルを fileNames[] 配列へのインデックスとして示します。アプリケーションは、**[Next]** または **[Previous]** ボタンの操作に応じてこの変数を変更し、「Open MP3 File」ステートへジャンプします。

図 7: AUDIO USB タスクのステートマシン



デモ アプリケーションの実行

弊社は、本書の内容に対応する MP3 プレーヤ デモ アプリケーションのソースコードも提供しています。MP3プレーヤ アプリケーションは、サンプリング レート 44.1 kHz でエンコードされたステレオ MP3 ファイル をデコードします。これとは異なるサンプリング レートでエンコードされた MP3 ファイルまたはモノラル エンコードされたMP3ファイルは無視されます。これはアプリケーションの設計による制限であり、Helix MP3 デコーダによる制限ではありません。このデコーダは、MP3 フォーマットが定める全てのビット レートとサンプリング周波数をサポートします。

MP3 プレーヤ アプリケーションは、マルチメディア拡張ボード(MEB) (製品番号: DM320005)と、PIC32MX USB スタータキット II (製品番号 : DM320003-2) または PIC32 Ethernet スタータキット (製品番号 : DM320004) の組み合わせで動作するよう設計されています。マルチメディア拡張ボード (MEB) は 24 ビット オーディオ用 ADC/DAC とタッチスクリーン ディスプレイを実装し、スタータキット コネクタを介して PIC32MX デバイスをサポートします。詳細については、マルチメディア拡張ボードのウェブページ (www.microchip.com/MEB) をご覧ください。

PIC32MX USB スタータキット II と PIC32 Ethernet スタータキットは、PIC32MX795F512L デバイスを使います。このデバイスは 512 Kbyte のプログラム フラッシュメモリと 128 Kbyte のデータ RAM を実装しています。また、USB モジュールも実装しています。スタータキットは、USB ホストおよびデバイス コネクタを備えています。

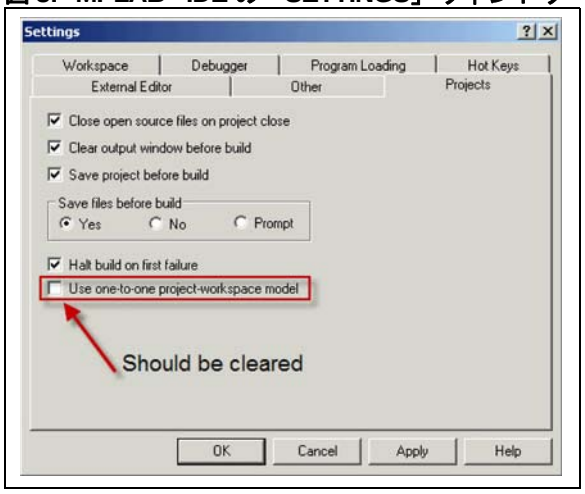
デモ ソースコードを評価するには、MP3 ファイルを保存した USB サムドライブとステレオ ヘッドフォンが必要です。MP3 ファイルは、サムドライブのルート ディレクトリに保存する必要があります。

デモ アプリケーションのコンパイルとデバイスへのプログラミング

以下では、MP3 プレーヤデモをコンパイルして PIC32MX デバイスにプログラミングするための手順について説明します。これは PIC32MX USB スタータキット II を使う場合の手順です。PIC32 Ethernet スタータキットを使う場合、MEB ENET USB Thumb Drive MP3 Demo.mcw MPLAB IDE ワークスペース ファイルを使います。デモのコンパイルおよび実行方法は、どちらのボードでも同じです。以下の説明は、ユーザがMPLAB® IDEの使用に習熟している事を前提とします。

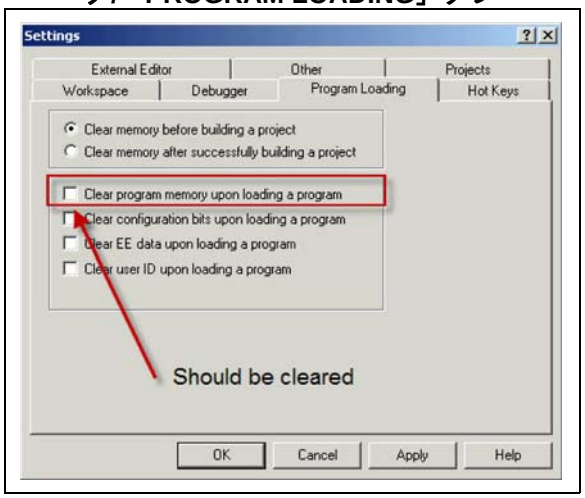
1. MPLAB IDE を開く。*Configure>Settings* をクリックする。**[Projects]** タブを開き、「Use one-to-one project-workspace model」のチェックマークを外す (図 8 参照)。**[OK]** をクリックする。

図 8: MPLAB® IDE の「SETTINGS」ウィンドウ



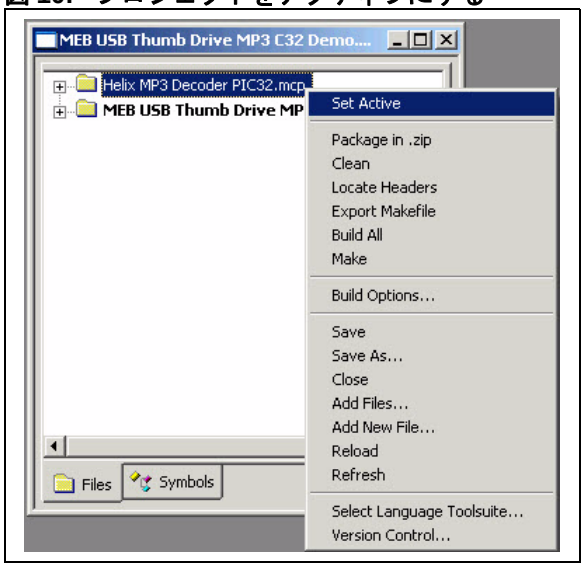
2. *Configure>Settings* をクリックする。**[Program Loading]** タブを開く。「Clear program memory upon loading a program」のチェックマークを外す (図 9 参照)。

図 9: MPLAB® IDE の「SETTINGS」ウィンドウ / 「PROGRAM LOADING」タブ



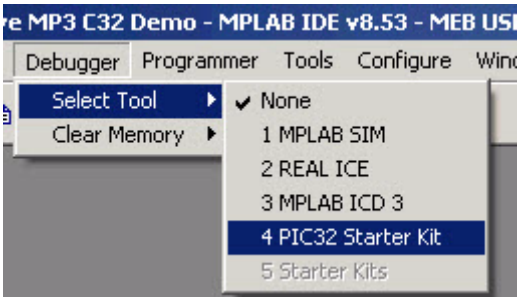
3. *File>Open Workspace* をクリックして MPLAB IDE ワークスペース ファイル「MEB USB Thumb Drive MP3 C32 Demo.mcw」を開く。アプリケーション プロジェクトが MPLAB IDE に読み込まれる。
4. 「Project Explorer」ウィンドウに、ワークスペース内の 2 つのプロジェクトが表示される。「Helix MP3 Decoder PIC32.mcp」プロジェクトは、PIC32MX マイクロコントローラ上での実行用に変更された Helix MP3 デコーダ ソースコードで構成される。
5. 「Helix MP3 Decoder PIC32.mcp」プロジェクトを右クリックし、開いたメニューから「Set Active」を選択する (図 10 参照)。

図 10: プロジェクトをアクティブにする



6. デバッガもプログラマも選択されていない事を確認する。「Debug/Release」の選択は動作に影響せず、無視される。「Build All」オプションを使って、プロジェクトをコンパイルおよびビルドする。
7. 「MEB USB Thumb Drive MP3 C32 Demo.mcp」プロジェクトを右クリックし、開いたメニューから「Set Active」を選択する。
8. PIC32 USB スタータキット II を MEB に接続する。ファスナーを使って、スタータキットをソケットに確実に固定する。
9. スタータキットのデバッグ USB ポートを PC の USB ポートに接続する。
10. MPLAB IDE で *Debugger>Select Tool>PIC32 Starter Kit* をクリックし、PIC32 スタータキットをデバッガとして選択する (図 11 参照)。

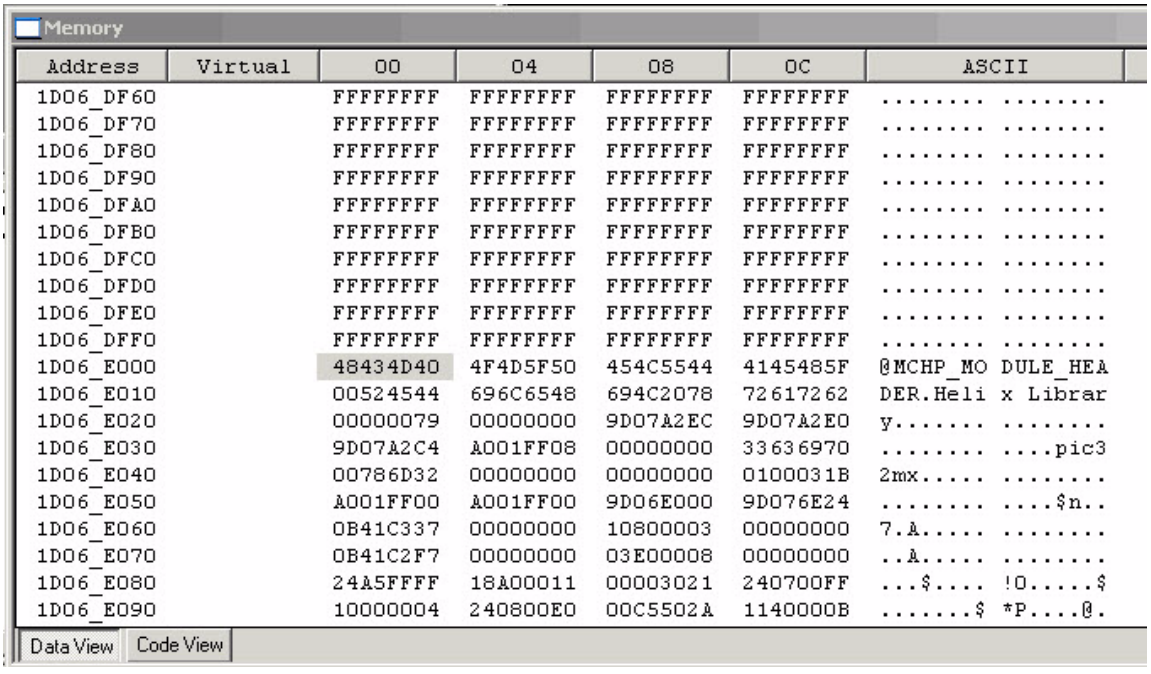
図 11: デバッガを選択する



11. プロジェクトをコンパイルおよびビルドする。
12. PIC32MX デバイスにプログラミングする。MPLAB IDE は以下の 2 つの HEX イメージをデバイスにプログラミングする。
    - `Helix MP3 Decoder PIC32.elf`
    - `MEB USB Thumb Drive MP3 C32 Demo.elf`
13. **この手順は必須ではありません** : 以下の方法で、プログラムメモリのアドレス 0x9D06E000 の内容を確認する。
    MPLAB IDE で *View>Memory* をクリックし、「Memory」ウィンドウ内の **[Data View]** をクリックする。「Memory」ウィンドウ内で右クリックして「Go To」を選択し、アドレスとして 0x9D06E000 を入力する。「Memory」ウィンドウは図 12 のように表示される。文字列「Helix Library」が表示されれば、デバイスへのプログラミングは成功した。

以上で、デモ アプリケーションの使用準備が完了します。

図 12: プログラムメモリの内容を表示する



デモ アプリケーションの実行

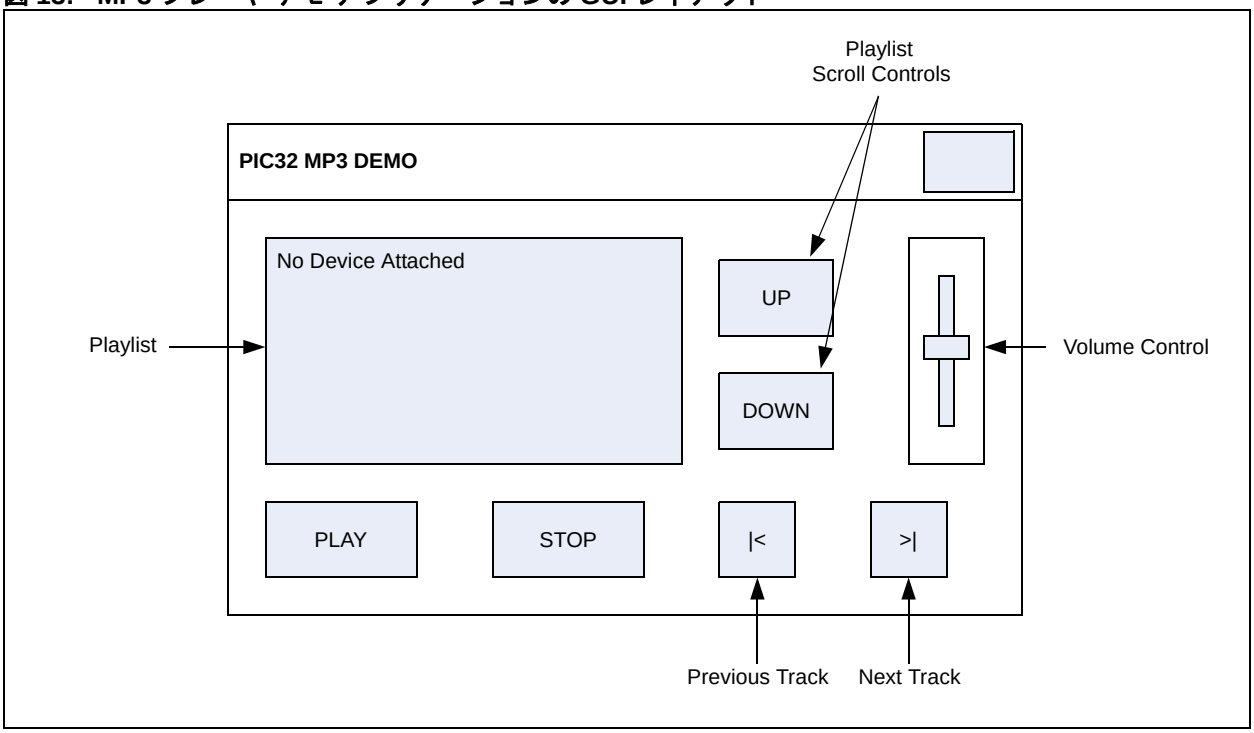
MEB 上のヘッドフォン ジャックにヘッドフォンを接続し、MPLAB IDE で **[Run]** をクリックし、MEB 上のタッチスクリーンを観察します。図 13 に、MP3 プレーヤ デモ アプリケーションの GUI を示します。

PIC32MX USB スタータキット II の USB ホスト レセプタクルに USB サムドライブを差し込みます。プレイリストは、MP3 ファイルと一緒にサムドライブのルート ディレクトリに保存されます。アプリケーションは、プレイリスト内の最初の MP3 ファイルの再生を開始します。

タッチスクリーン ボタンの機能は以下の通りです。

- 音量調整用スライダ
- [STOP] ボタン : MP3 ファイルの再生を停止する。
- [PLAY] ボタン : 再生中にこのボタンを押すと再生は一時停止し、もう一度押すと再開する。
- [UP]/[DOWN] ボタン : プレイリスト内のファイルをスクロールする。プレイリスト内のファイルにタッチする事で、再生するファイルを選択する。

図 13: MP3 プレーヤ デモ アプリケーションの GUI レイアウト



プロジェクト ファイル

本書に付属するソースコード (**補遺 A:「ソースコード」** 参照) には、PIC32MX USB スタータキット II と PIC32MX Ethernet スタータキット向けのデモ用ソースコードが含まれています。

「MEB USB Thumb Drive MP3 Demo」フォルダ内のアプリケーションは、PIC32MX USB スタータキット II を接続した MEB 上で動作します。「MEB ENET USB Thumb Drive MP3 Demo」フォルダ内のアプリケーションは、PIC32MX Ethernet スタータキットを接続した MEB 上で動作します。

表 3 に示したフォルダとファイルの説明は、両方のアプリケーション フォルダに適用されます。アプリケーションによって異なる部分は明示しています。

まとめ

オープンソースの Helix MP3 デコーダは、Microchip 社の PIC32MX 32 ビット マイクロコントローラに移植可能です。本書では Helix デコーダ API について説明し、本書に付属する MP3 プレーヤ アプリケーションを使って Helix MP3 デコーダのデモを実行する方法を紹介しました。また、RTLL方式についても解説しました。

表 3: ソースコード ファイルとフォルダ

名称	概要
h	MEB USB Thumb Drive MP3 C32 Demo プロジェクトに必要なインクルード ファイルを格納したフォルダ
lib	USBおよびグラフィック スタックのアーカイブを格納したフォルダ
obj	テンポラリ BSP (Board Support Package) オブジェクト ファイルを保存するフォルダ
src	MEB USB Thumb Drive MP3 C32 Demo プロジェクトに必要なソースファイルを格納したフォルダ
MP3 Decoder Source Code	Helix MP3 デコーダのソースコードを格納したフォルダ
procdefs.ld	MEB USB Thumb Drive MP3 C32 Demo および Helix MP3 Decoder PIC32 MPLAB IDE プロジェクトで必要な変更済みリンカスクリプト ファイル
MEB USB Thumb Drive MP3 C32 Demo.mcp または MEB ENET USB Thumb Drive MP3 Demo.mcp	MPLAB IDE プロジェクト ファイル
MEB USB Thumb Drive MP3 C32 Demo.mcw または MEB ENET USB Thumb Drive MP3 Demo.mcw	MPLAB IDE ワークスペース ファイル
CleanUp.bat	テンポラリ ファイルをクリアするためのバッチファイル
MAL BSP Files Archive.bat	BSP アーカイブを更新およびビルドするためのバッチファイル
MAL GFX Stack Archive.bat	グラフィック スタック アーカイブを更新およびビルドするためのバッチファイル
MAL USB Stack Archive.bat	USBスタック アーカイブを更新およびビルドするためのバッチファイル
Readme.txt	デモの実行に関する情報を収めたファイル
Alternative Configurations	MEB グラフィック ディスプレイに必要なインクルード ファイルを格納したフォルダ

補遺 A:　ソースコード

本書に関連する全てのソフトウェアは、1 つの WinZip ファイルに収められています。このファイルは、下記の Microchip 社ウェブサイトからダウンロードできます。

www.microchip.com

NOTE:

Microchip 社製デバイスのコード保護機能に関して次の点にご注意ください。

- Microchip 社製品は、該当する Microchip 社データシートに記載の仕様を満たしています。
- Microchip 社では、通常の条件ならびに仕様に従って使用した場合、Microchip 社製品のセキュリティ レベルは、現在市場に流通している同種製品の中でも最も高度であると考えています。
- しかし、コード保護機能を解除するための不正かつ違法な方法が存在する事もまた事実です。弊社の理解ではこうした手法は、Microchip 社データシートにある動作仕様書以外の方法で Microchip 社製品を使用する事になります。このような行為は知的所有権の侵害に該当する可能性が非常に高いと言えます。
- Microchip 社は、コードの保全性に懸念を抱くお客様と連携し、対応策に取り組んでいきます。
- Microchip 社を含む全ての半導体メーカーで、自社のコードのセキュリティを完全に保証できる企業はありません。コード保護機能とは、Microchip 社が製品を「解読不能」として保証するものではありません。

コード保護機能は常に進歩しています。Microchip 社では、常に製品のコード保護機能の改善に取り組んでいます。Microchip 社のコード保護機能の侵害は、デジタル ミレニアム著作権法に違反します。そのような行為によってソフトウェアまたはその他の著

商標

ISBN: 978-1-63276-115-6

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949 ==

Microchip 社では、Chandler および Tempe (アリゾナ州)、Gresham (オレゴン州) の本部、設計部およびウェハー製造工場そしてカリフォルニア州とインドのデザインセンターが ISO/TS-16949:2009 認証を取得しています。Microchip 社の品質システム プロセスおよび手順は、PIC® MCU および dsPIC® DSC、KEELOQ® コード ホッピング デバイス、シリアル EEPROM、マイクロペリフェラル、不揮発性メモリ、アナログ製品に採用されています。さらに、開発システムの設計と製造に関する Microchip 社の品質システムは ISO 9001:2000 認証を取得しています。

各国の営業所とサービス

北米

本社
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel:480-792-7200
Fax:480-792-7277
技術サポート :
http://www.microchip.com/support
URL:
www.microchip.com

アトランタ
Duluth, GA
Tel:678-957-9614
Fax:678-957-1455

オースティン (TX)
Tel:512-257-3370

ボストン
Westborough, MA
Tel:774-760-0087
Fax:774-760-0088

シカゴ
Itasca, IL
Tel:630-285-0071
Fax:630-285-0075

クリーブランド
Independence, OH
Tel:216-447-0464
Fax:216-447-0643

ダラス
Addison, TX
Tel:972-818-7423
Fax:972-818-2924

デトロイト
Novi, MI
Tel:248-848-4000

ヒューストン (TX)
Tel:281-894-5983

インディアナポリス
Noblesville, IN
Tel:317-773-8323
Fax:317-773-5453

ロサンゼルス
Mission Viejo, CA
Tel:949-462-9523
Fax:949-462-9608

ニューヨーク (NY)
Tel:631-435-6000

サンノゼ (CA)
Tel:408-735-9110

カナダ - トロント
Tel:905-673-0699
Fax:905-673-6509

アジア / 太平洋

アジア太平洋支社
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel:852-2943-5100
Fax:852-2401-3431

オーストラリア - シドニー
Tel:61-2-9868-6733
Fax:61-2-9868-6755

中国 - 北京
Tel:86-10-8569-7000
Fax:86-10-8528-2104

中国 - 成都
Tel:86-28-8665-5511
Fax:86-28-8665-7889

中国 - 重慶
Tel:86-23-8980-9588
Fax:86-23-8980-9500

中国 - 杭州
Tel:86-571-8792-8115
Fax:86-571-8792-8116

中国 - 香港 SAR
Tel:852-2943-5100
Fax:852-2401-3431

中国 - 南京
Tel:86-25-8473-2460
Fax:86-25-8473-2470

中国 - 青島
Tel:86-532-8502-7355
Fax:86-532-8502-7205

中国 - 上海
Tel:86-21-5407-5533
Fax:86-21-5407-5066

中国 - 瀋陽
Tel:86-24-2334-2829
Fax:86-24-2334-2393

中国 - 深圳
Tel:86-755-8864-2200
Fax:86-755-8203-1760

中国 - 武漢
Tel:86-27-5980-5300
Fax:86-27-5980-5118

中国 - 西安
Tel:86-29-8833-7252
Fax:86-29-8833-7256

中国 - 厦門
Tel:86-592-2388138
Fax:86-592-2388130

中国 - 珠海
Tel:86-756-3210040
Fax:86-756-3210049

アジア / 太平洋

インド - バンガロール
Tel:91-80-3090-4444
Fax:91-80-3090-4123

インド - ニューデリー
Tel:91-11-4160-8631
Fax:91-11-4160-8632

インド - プネ
Tel:91-20-3019-1500

日本 - 大阪
Tel:81-6-6152-7160
Fax:81-6-6152-9310

日本 - 東京
Tel:81-3-6880- 3770
Fax:81-3-6880-3771

韓国 - 大邱
Tel:82-53-744-4301
Fax:82-53-744-4302

韓国 - ソウル
Tel:82-2-554-7200
Fax:82-2-558-5932 または
82-2-558-5934

マレーシア - クアラルンプール
Tel:60-3-6201-9857
Fax:60-3-6201-9859

マレーシア - ペナン
Tel:60-4-227-8870
Fax:60-4-227-4068

フィリピン - マニラ
Tel:63-2-634-9065
Fax:63-2-634-9069

シンガポール
Tel:65-6334-8870
Fax:65-6334-8850

台湾 - 新竹
Tel:886-3-5778-366
Fax:886-3-5770-955

台湾 - 高雄
Tel:886-7-213-7830

台湾 - 台北
Tel:886-2-2508-8600
Fax:886-2-2508-0102

タイ - バンコク
Tel:66-2-694-1351
Fax:66-2-694-1350

ヨーロッパ

オーストリア - ヴェルス
Tel:43-7242-2244-39
Fax:43-7242-2244-393

デンマーク - コペンハーゲン
Tel:45-4450-2828
Fax:45-4485-2829

フランス - パリ
Tel:33-1-69-53-63-20
Fax:33-1-69-30-90-79

ドイツ - デュッセルドルフ
Tel:49-2129-3766400

ドイツ - ミュンヘン
Tel:49-89-627-144-0
Fax:49-89-627-144-44

ドイツ - プフォルツハイム
Tel:49-7231-424750

イタリア - ミラノ
Tel:39-0331-742611
Fax:39-0331-466781

イタリア - ベニス
Tel:39-049-7625286

オランダ - ドリューネン
Tel:31-416-690399
Fax:31-416-690340

ポーランド - ワルシャワ
Tel:48-22-3325737

スペイン - マドリッド
Tel:34-91-708-08-90
Fax:34-91-708-08-91

スウェーデン - ストックホルム
Tel:46-8-5090-4654

イギリス - ウォーキンガム
Tel:44-118-921-5800
Fax:44-118-921-5820

03/25/14