AN1367

Porting the Helix MP3 Decoder onto Microchip's PIC32MX 32-bit MCUs

Author: Sunil Fernandes
Microchip Technology Inc.

INTRODUCTION

The MPEG-1, MPEG-2, and MPEG-2.5 Layer 3 (MP3) audio encoding format is a popular audio format for consumer audio storage and digital audio players. Features such as multiple bit rates, variable bit rates and choice of audio sampling rates make this algorithm an attractive choice for a wide variety of multimedia applications.

This application note describes the procedure to port the open source Helix MP3 decoder algorithm onto Microchip's PIC32MX 32-bit microcontrollers (MCUs). The source code provided with this document demonstrates a MP3 player application using the Helix MP3 decoder. The MP3 player application uses Microchip's USB stack to read MP3 files from a USB flash drive, which is referred to as a thumb drive in this document, and the Microchip graphics stack to implement a Graphical User Interface (GUI) with touch screen support.

Application developers may need to add proprietary code to open source code to meet the target application requirements. When statically compiled with the open source code, this proprietary code may be subject to the open source End User License Agreement. In many cases, this may not be acceptable to application owners. Therefore, this application note describes a Run-Time Library Loading (RTLL) technique that enables the preservation of the application's intellectual property.

This application note is organized in the following order:

1. Description of the Helix MP3 decoder library.
2. RTLL technique used in the demo application.
3. Description of the demo application code.
4. Steps required to compile and run the demo application.

ABOUT THE HELIX MP3 DECODER

The Helix MP3 decoder is available as both floating point and fixed point implementations. The fixed point implementation is considered for porting the algorithm onto the PIC32MX microcontroller. The algorithm runs on any 32-bit fixed point processor and is coded entirely in the C language with options to replace certain code sections with optimized assembly instructions.

The Helix MP3 decoder provides Layer 3 support for MPEG-1, MPEG-2, and MPEG-2.5. It supports variable bit rates, constant bit rates, and stereo and mono audio formats. For details on implementation and features, visit the Helix MP3 decoder web site at: https://datatype.helixcommunity.org/Mp3dec.

The Helix MP3 decoder source code is open source and is governed by the license described in files that accompany the source code. It should be noted that even though the Helix MP3 decoder is free to use and is open source, the MP3 algorithm itself is not free and has royalties associated with it. These royalties must be paid in order to use the algorithm. For more details, visit www.mp3licensing.com.

Porting the Helix MP3 Decoder onto the PIC32MX Microcontroller

To port the Helix MP3 decoder to the PIC32MX platform, the decoder source code must be downloaded from the Helix MP3 decoder web site. Follow the instructions on the Web page to download the source code. Alternatively, the Helix MP3 decoder source code available with this application note can also be used. The source code available with this document is already modified to allow the Helix MP3 decoder to work on PIC32MX devices. For the latest source code, visit the Helix MP3 decoder web site at: https://datatype.helixcommunity.org/Mp3dec.

Each folder in the downloaded source code contains three license files: `RPSL.txt`, `RCSL.txt` and `LICENSE.txt`. Users are encouraged to read these license files and are requested to ensure compliance.

The Helix MP3 decoder source code (as downloaded from the Helix MP3 decoder web site) is located in the `fixpt` folder. The following steps need to be performed to port the decoder source code to the PIC32MX device.

1. Open the `assembly.h` file in the `\fixpt\real` folder.
2. Go to the `FASTABS` function define for the MIPS platform (line 337 of this file). This function is a MIPS assembly implementation of an absolute function.
3. Comment out the function body and replace this section with the C implementation (see Example 1), and save these changes.
4. Open the `mp3dec.c` file located in the `fixpt` folder and comment out line 47 (see Example 2). The PIC32MX port of the Helix MP3 decoder does not require this feature.

These steps apply to Version 1.8 of `assembly.h` file and Version 1.6 of the `mp3dec.c` file.

These are the changes that need to be made to the source code downloaded from the Helix MP3 decoder web site. The Helix MP3 decoder source code included with this document already contains these modifications.

EXAMPLE 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));
}
```

EXAMPLE 2:

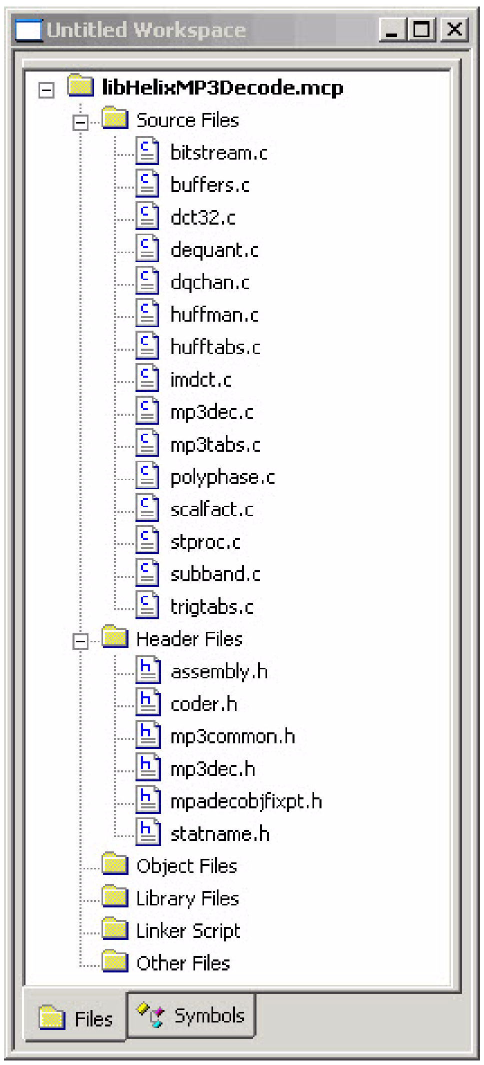
```
#include "string.h"           /* for memmove, memcpy (can replace with different */
                              /* implementations if desired) */

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

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

To build a project containing the Helix source code files, include all `.c` source code files located in the `fixpt` and `fixpt\real` folders. It is optional for the user to include all of the header files in the project. Figure 1 shown the file listing for the MPLAB IDE Project Explorer window.

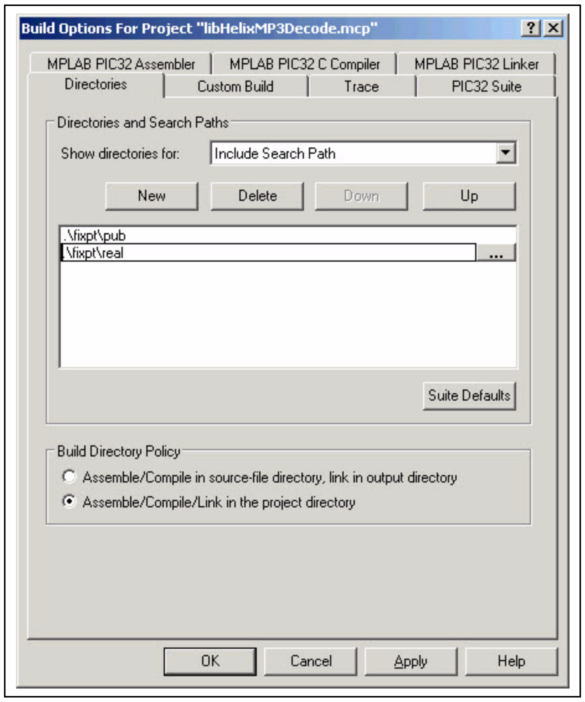
FIGURE 1: PROJECT EXPLORER WINDOW



```
libHelixMP3Decode.mcp
  Source Files
    bitstream.c
    buffers.c
    dct32.c
    dequant.c
    dqchan.c
    huffman.c
    hufftabs.c
    imdct.c
    mp3dec.c
    mp3tabs.c
    polyphase.c
    scalfact.c
    stproc.c
    subband.c
    trigtabs.c
  Header Files
    assembly.h
    coder.h
    mp3common.h
    mp3dec.h
    mpadecobjfixpt.h
    statname.h
  Object Files
  Library Files
  Linker Script
  Other Files
```

The path to the `fixpt\pub` and `fixpt\real` folders needs to be specified and should be relative to the location of the project file. Click *Project>Build Options>Project*. From the **Directories** tab, select Include Search Path, and specify the path of the folder relative to the MPLAB project location; otherwise, the absolute path can also be specified. For example, if the project and `fixpt` folders are located in the same folder, the selected options would look similar to Figure 2.

FIGURE 2: SPECIFYING INCLUDE FILE SEARCH PATH



Directories and Search Paths

Show directories for: Include Search Path

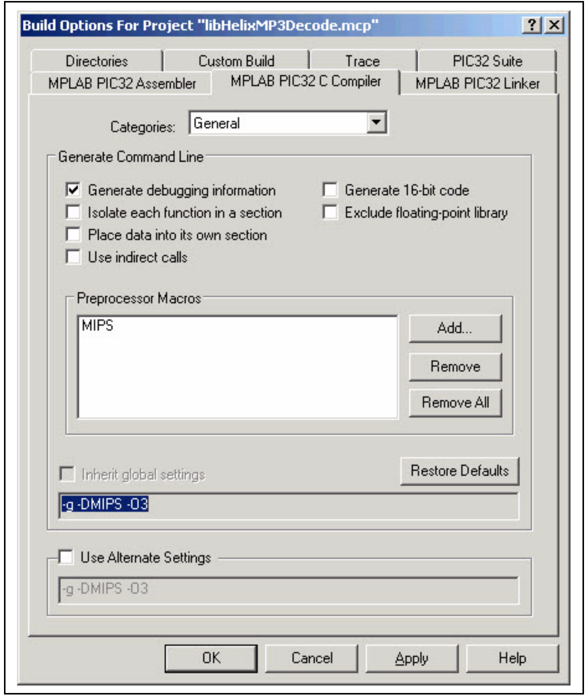
.\fixpt\pub
.\fixpt\real

Build Directory Policy

- ( ) Assemble/Compile in source-file directory, link in output directory
- (●) Assemble/Compile/Link in the project directory

When compiling, the MIPS symbol needs to be defined. This builds the Helix MP3 decoder source code for the PIC32MX platform. Click *Project>Build Options>Project*. From the **MPLAB PIC32 C Compiler** tab, add the MIPS macro to the build process, as shown in Figure 3.

FIGURE 3: ADDING MIPS PRE-PROCESSOR MACRO TO BUILD OPTIONS



Categories: General

Generate Command Line

- [✓] Generate debugging information
- [ ] Isolate each function in a section
- [ ] Place data into its own section
- [ ] Use indirect calls
- [ ] Generate 16-bit code
- [ ] Exclude floating-point library

Preprocessor Macros: MIPS

[ ] Inherit global settings

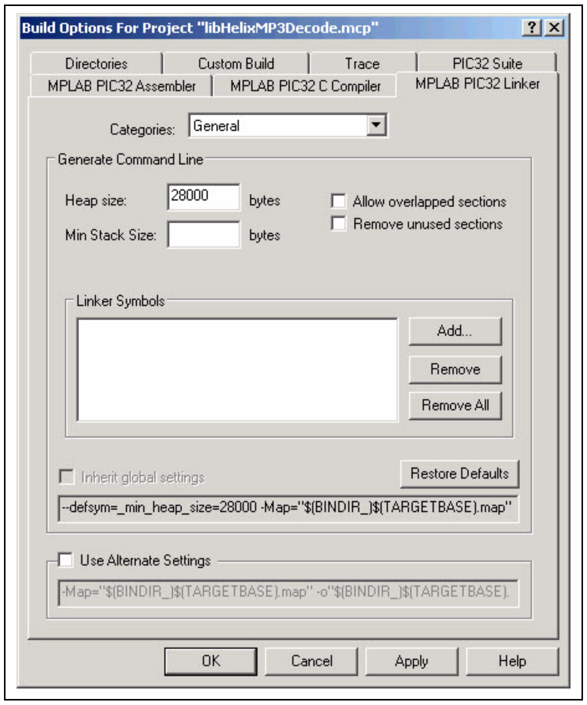
-g -DMIPS -O3

[ ] Use Alternate Settings

-g -DMIPS -O3

The decoder requires heap memory for its operation. The heap memory size required for the decoder without input and output buffers is 28 Kbytes, which must be specified at compile time. Click *Project>Build Options>Project*, and select the **MPLAB PIC32 Linker** tab. Specify the heap size as shown in Figure 4.

FIGURE 4: SPECIFYING THE HEAP MEMORY SIZE



Build Options For Project "libHelixMP3Decode.mcp"

Categories: General

Generate Command Line

Heap size: 28000 bytes

Min Stack Size: bytes

☐ Allow overlapped sections

☐ Remove unused sections

☐ Inherit global settings

--defsym=_min_heap_size=28000 -Map="$(BINDIR_)$(TARGETBASE).map"

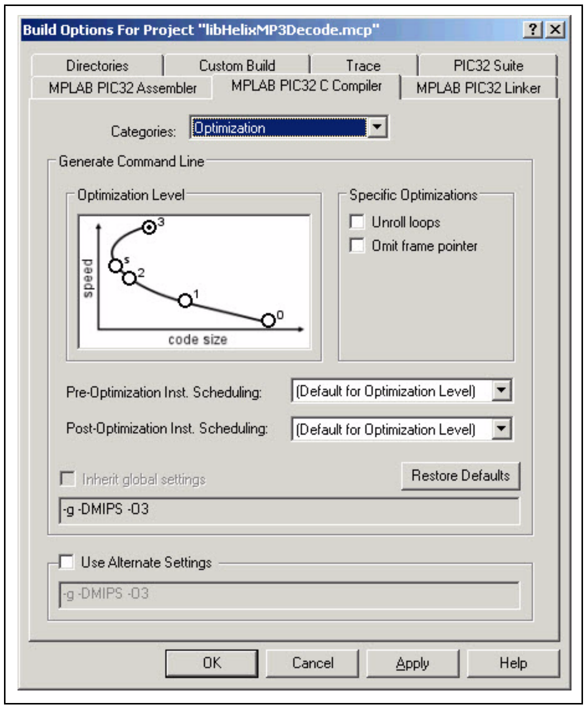
☐ Use Alternate Settings

-Map="$(BINDIR_)$(TARGETBASE).map" -o"$(BINDIR_)$(TARGETBASE).

The Helix MP3 decoder source code should be compiled with the optimization level set at O3. Click *Project>Build Options>Project*, and select the **MPLAB PIC32 C Compiler** tab. In the Categories field, select Optimization from the drop-down list. Set the optimization level to O3, as shown in Figure 5.

The Helix MP3 decoder source code is now ready for use with a PIC32MX application.

FIGURE 5: SETTING THE COMPILER OPTIMIZATION LEVEL



Build Options For Project "libHelixMP3Decode.mcp"

Categories: Optimization

Generate Command Line

☐ Unroll loops

☐ Omit frame pointer

Pre-Optimization Inst. Scheduling: (Default for Optimization Level)

Post-Optimization Inst. Scheduling: (Default for Optimization Level)

☐ Inherit global settings

-g -DMIPS -O3

☐ Use Alternate Settings

-g -DMIPS -O3

Helix MP3 Decoder Application Program Interface (API)

The Helix MP3 decoder API provides functions to:

- Initialize the decoder
- Close the decoder
- Detect frame synchronization
- Get frame information such as bit rate and sampling frequency
- Decode MP3 frames

Note:	The demo application uses the RTLL technique and invokes the Helix MP3 decoder API indirectly via the system loader interface. Therefore, the `MEB USB Thumb Drive MP3 C32 Demo.mcp` source code does not feature direct instances of the Helix MP3 decoder API.

The first step when using the Helix MP3 decoder is to include the required header files in all the source files to invoke the decoder API. The files to be included are `coder.h` and `mp3dec.h` (see Example 3).

EXAMPLE 3:

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

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

The Helix MP3 decoder is fully re-entrant, which means that the decoder state is encapsulated completely within a data structure. The `MP3InitDecoder()` function allocates the memory for this data structure, initializes it, and then returns a pointer to the initialized data structure. This function uses the dynamic memory allocation and requires the heap to be allocated. If the function returns a value of zero, the allocation was not successful (see Example 4).

If the application wants to close the decoder, memory allocated to the decoder can be deallocated using the `MP3FreeDecoder()` function (see Example 5). This function clears the memory consumed by the decoder.

MP3 files typically have additional information stored within them through ID3 tags. These tags contain information about the song, artist, album, genre, etc. An input frame provided to the Helix MP3 decoder function should not contain this information. The application must locate the start of actual MP3 data within the file or input stream. This can be done by using the `MP3FindSyncWord()` function. This function accepts an input buffer (which should be a portion of bytes read from a MP3 file), locates the Synchronization Word in the MP3 frame header, and then returns this location in terms of an offset from the start of the input buffer. The Synchronization Word indicates the start of a MP3 frame. If a negative value is returned, the input buffer does not contain a Synchronization Word or a start of MP3 frame (see Example 6).

EXAMPLE 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);
}
```

EXAMPLE 5:

```
// Close the decoder.

MP3FreeDecoder(mp3Decoder);

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

EXAMPLE 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;
            }
        }
    }
```

The MP3FindSyncWord() function may find the Synchronization Word, but this may not be the actual start of a MP3 frame. There may be instances where some data within an ID3 tag could match a Synchronization Word. In such cases, if the input is passed to the MP3 Decoder function, it returns an error. The application should process this error accordingly.

The MP3 player application, which accompanies this document, is designed to process the MP3 files that are stereo encoded and processed at a 44.1 kHz sampling rate. The Helix MP3 decoder features the MP3GetNextFrameInfo() function, which returns the audio attributes of a MP3 frame that is yet to be processed. The application can use this information as required. The MP3FindSyncWord() function is used to locate the Start of Frame (SOF) and thereby indicate a potential MP3 frame. The MP3GetNextFrameInfo() function can then be used to extract the information about the yet to be decoded MP3 frame. The information is returned in a MP3FrameInfo type data structure. An error is returned in case of an invalid MP3 frame (see Example 7).

The following information is available in the MP3FrameInfo data structure:

- Bit rate of the processed frame
- Number of audio channels (one for mono, two for stereo)
- Encoding audio sample rate
- Number of bits per sample
- Size of the decoded audio frame in audio samples (stereo samples are counted as two audio samples)
- MPEG layer
- Layer version

The MP3GetLastFrameInfo() function returns the same information, but for a decoded audio frame. Example 8 shows the usage of the MP3GetLastFrameInfo() function.

EXAMPLE 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();
    }

}
```

EXAMPLE 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;
```

The `MP3Decode()` function is used to decode the encoded MP3 frame. This function invokes the core MP3 decoding algorithm. The entire input encoded frame may not be consumed in one decode operation. In such a case, the input pointer is advanced to point to the start of unconsumed bytes. The decoder function also returns the total unconsumed bytes. The MP3 application can use this to prime the input buffer. For correct operation, the new data should be appended to the end of the unconsumed bytes.

The input frame can be formatted as a standard MPEG stream for self-contained MP3 frames. The standard MPEG stream option has been used in the MP3 player example. The format of the decoded raw data depends on the number of audio channels. In the case of mono audio data, each output audio sample is unique. In the case of stereo audio data, the output audio samples are organized as interleaved left channel and right channel (LRLRLR…) audio samples. The decode function returns an error value indicating the result of the decode process. The MP3 application can take the appropriate action based on the type of error (see Example 9).

EXAMPLE 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);
}
```

Table 1 lists the Helix MP3 decoder memory requirements when running on a PIC32MX microcontroller. Table 2 lists the computational requirement.

TABLE 1: HELIX MP3 DECODER MEMORY REQUIREMENTS

Memory Type	Size (in bytes)	Remarks
Program Memory	53000	—
Data Memory	28000	Required by the decoder only.
Input Buffer	1940	Maximum MP3 frame size.
Output Buffer	2304	Maximum size required by the output buffer for stereo audio data.

TABLE 2: HELIX MP3 DECODER COMPUTATIONAL REQUIREMENT

Function	MIPS	Remarks
`MP3Decode()`	26	Calculated with code compiled with O3 optimization and processor clock at 80 MHz.

RUNTIME LIBRARY LOADING

Application developers may consider use of open source code components for their application. The open source code license in such an instance may require proprietary application code to be covered by the terms and conditions of the open source code license. This requirement may not be convenient to an application developer or owner. For such cases, the MP3 application example accompanying this source code utilizes a technique where:

- Open source code is not linked to the main application source code. They are compiled separately and are not linked with each other.
- Open source code library is loaded at run time through a loader utility as a set of function pointers

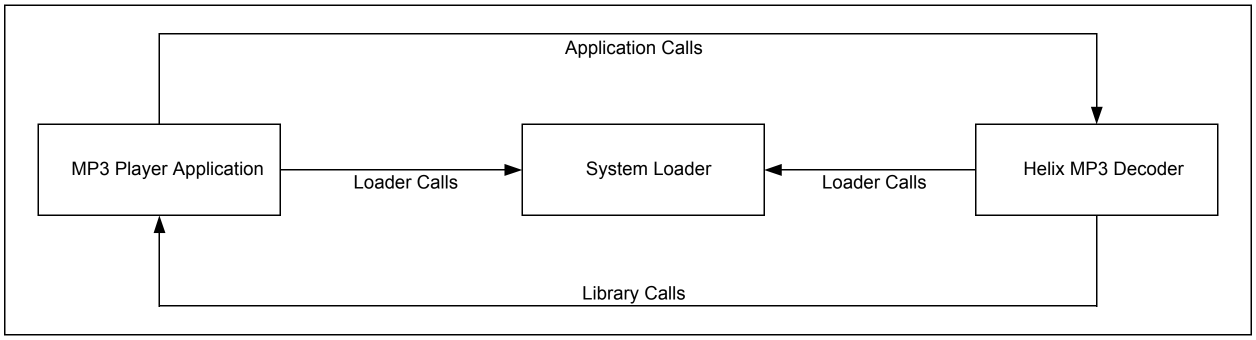
This technique is termed as Run-Time Library Loading (RTLL). The MP3 player application uses the RTLL technique to load the Helix MP3 decoder at run-time. Figure 6 shows the conceptual block diagram of the RTLL operation.

The RTLL technique uses a system loader to load the Helix MP3 decoder. The loading process involves initialization of RAM, copying of tables from program memory to RAM (if required), and obtaining a set of pointers to the functions exported by the decoder. The MP3 player application exports its functions through the system loader and these functions can be called by the Helix MP3 decoder. Similarly, code is added to the Helix MP3 decoder module to expose a set of functions through the system loader, which can be called by the main application.

The MP3 player application consists of two projects that create two distinct hex files (program images). One file for the main application (also containing the USB and graphics stacks) and the other for the Helix MP3 decoder. These two program images are programmed at separate locations in device program memory. The MP3 application thus accesses the Helix MP3 decoder using the RTLL technique. This technique requires modification to the default linker script used by the C32 linker. The start of the program Flash and data RAM sections needs to be adjusted for implementing the RTLL technique. For the Helix MP3 decoder, these sections are adjusted to be placed towards the end of the respective memory areas. The program memory area is set to start at 72 Kbytes from end of program memory, and data RAM is set to start at 256 bytes from the end of KSEG1 RAM (adequate stack memory for Helix). Similarly, the linker script used when creating the MP3 player program image adjusts the size to accommodate for the Helix MP3 decoder. The program memory is reduced by 72 Kbytes and KSEG1 RAM by 256 bytes.

Throughout the remainder of this discussion, the term “main application” refers to the MP3 player application, and the term “library” refers to the Helix MP3 decoder. The two are collectively referred to as modules. The RTLL technique uses the dynamic module header data structure, `module_dyn_hdr`, to hold information about a module. The main application, system loader and library have their own dynamic module headers. Example 10 shows the definition of `module_dyn_hdr`.

FIGURE 6: RTLL FRAMEWORK



EXAMPLE 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;
```

The dynamic header data structure for each module includes the information about the imported and exported functions. The imported functions are contained in the module import table (`module_import_tbl`). The exported functions are contained in the module export table (`module_export_tbl`). Example 11 shows how the module dynamic header is initialized for the main application module.

EXAMPLE 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

};
```

The main module exports the malloc() and free() functions required by the Helix MP3 library. The main application module does not import any other module functions and the _DefaultModuleImports table is empty. The RTLL technique allows the system loader to be loaded as a module. However, this is not performed in this particular example and the system loader is statically linked to the main application module. This selection is enforced by specifying the LOADER_STATIC_LINK option while compiling the main application module.

The main application calls the dlopen() function (in AudioUSBInit() function in AudioUSBTasks.c file) to initialize the RTLL system (see Example 12).

The dlopen() function loads all of the modules specified in the p_modules[] array. Loading invokes the start-up code (if any) associated with each module. The Helix MP3 Library module uses modified C run-time start-up code to initialize its memory. The dlopen() function then returns a pointer to the module dynamic header for the Helix MP3 library.

The main application then uses the dlsym() function to obtain a handle to the Helix MP3 library module entry point (see Example 13).

This returns a pointer to the HelixLibEntry() function defined in the helix_lib.c file. This function invokes the Helix MP3 decoder API based on specified arguments. Example 14 shows a portion of the HelixLibEntry() function implementation.

The fCode argument to the HelixLibEntry() function determines the Helix library function to be invoked. For example, if fCode is ‘0’, the MP3InitDecoder() function is invoked. It should be noted that the main application module obtains a handle (MP3DecoderFunctions) to the HelixLibEntry() function via the system loader dlsym() call.

The main application module now uses the MP3DecoderFunctions handle to invoke the Helix MP3 decoder API functions. Example 15 shows the main application module invoking the MP3Decode() function through the MP3DecoderFunctions handle.

EXAMPLE 12:

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

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

EXAMPLE 13:

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

EXAMPLE 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;
```

EXAMPLE 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);
```

THE MP3 PLAYER APPLICATION

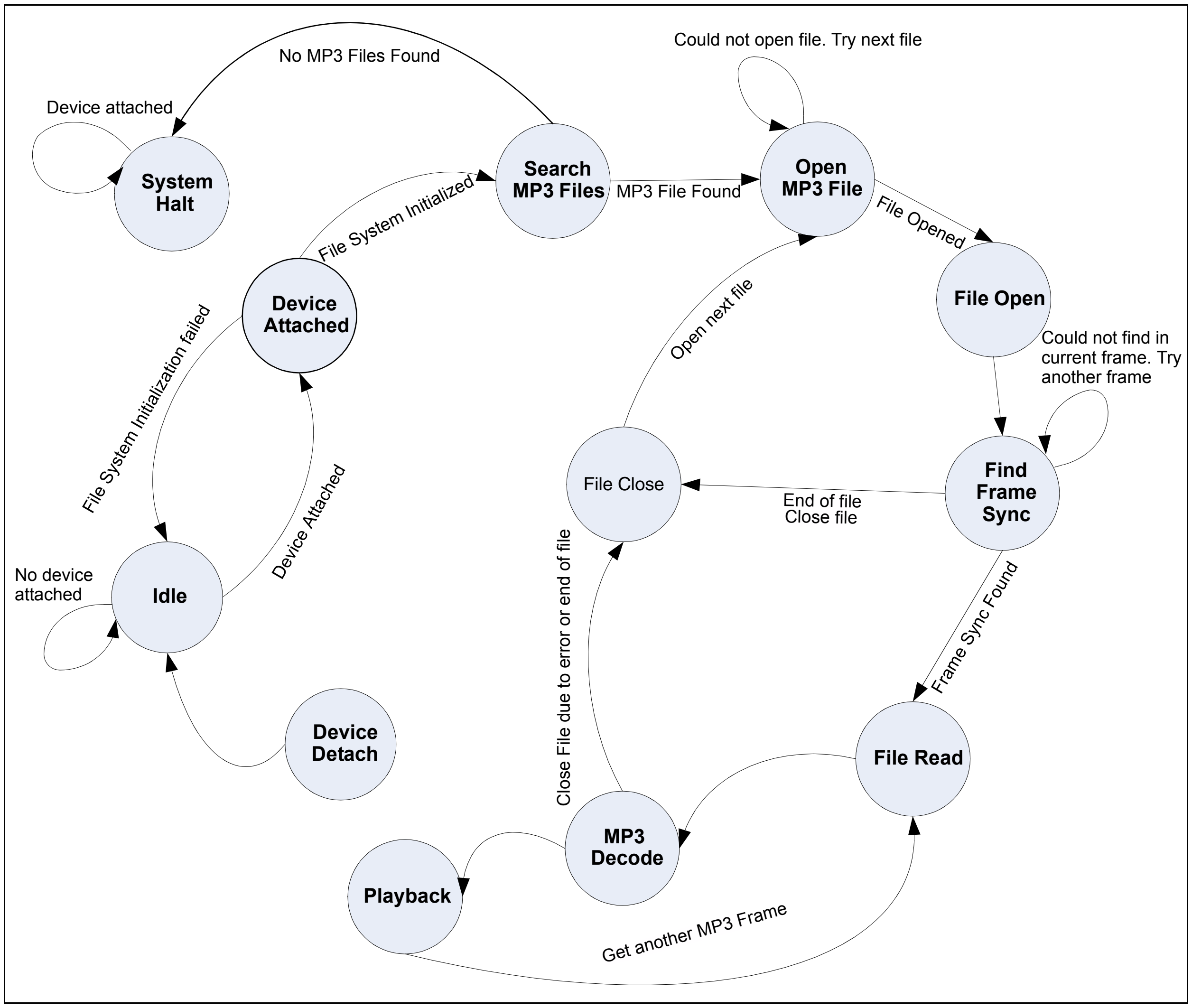
The MP3 player application uses the Helix MP3 decoder (via RTLL), Microchip's USB stack and graphics stack to implement a MP3 player. The application uses the Mass Storage Device (MSD) host component of the USB stack to read a USB thumb drive. The graphics stack provides the touch screen and user interface functions.

The USB and graphics stack operations are documented well and will not be discussed in this application note. The core logic of the MP3 player is implemented as a state machine in the `AudioUSBTasks()` function in the `AudioUSBTasks.c` file. Figure 7 shows a pictorial representation of this state machine.

When in the Idle state, the state machine waits for a USB thumb drive to be attached. When the device is attached, the code attempts to initialize the thumb drive file system. The application searches the thumb drive for the MP3 files. If files are found, the file names are stored in `fileNames[]` array. A maximum of `MAX_FILES` filenames can be stored. The file list box on the display is updated with the file names. The first file is opened, and the code searches for the first instance of start of frame. If a start of frame is found, the remainder of the frame is read from the file, and the frame is passed to the decode function. The decoded audio frame is sent for playback to the WM8731 Audio DAC device. The file read and decode operations continue until an end of file is encountered; the file is then closed and the next file is opened.

The `currentPlaybackFileIndex` variable tracks the current file being processed as an index into the `fileNames[]` array. The application changes this variable and jumps to the Open MP3 File state in response to an activity on the **Next** or **Previous** buttons.

FIGURE 7: AUDIO USB TASKS STATE MACHINE



RUNNING THE DEMO APPLICATION

This application note is accompanied by MP3 player demo application source code. The MP3 player application decodes stereo MP3 files, which were encoded at a 44.1 kHz sampling rate. MP3 files encoded at other sample rates or mono-encoded are ignored. This is purely an application design choice and not a limitation of the Helix MP3 decoder. The decoder supports the entire range of bit rates and sampling frequencies as specified by the MP3 format.

The MP3 player application is designed to run on the Multimedia Expansion Board (MEB) (Part Number: DM320005) along with a PIC32MX USB Starter Kit II (Part Number: DM320003-2) or PIC32 Ethernet Starter Kit (Part Number: DM320004). The MEB features a 24-bit audio ADC/DAC, touch screen display and supports the PIC32MX device via a starter kit connector. For more details, visit the web page dedicated to the Multimedia Expansion Board at www.microchip.com/MEB.

The PIC32MX USB Starter Kit II and PIC32 Ethernet Starter Kit use the PIC32MX795F512L device. This device features 512 Kbytes of program Flash memory and 128 Kbytes of data RAM. The device also features a USB module. The starter kit includes the USB host and device connectors.

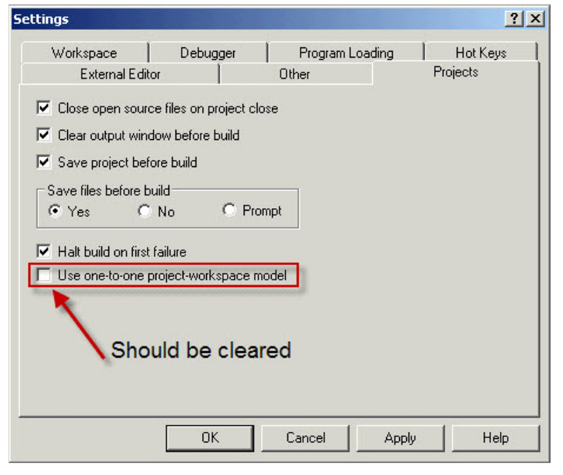
A pair of headphones and a USB thumb drive containing MP3 files are required for testing the demo source code. MP3 files must be placed in the root directory of the thumb drive.

Compiling and Programming the Demo Application

This section describes the steps required to compile the MP3 player demo and program the PIC32MX device. The following steps specify use of the PIC32MX USB Starter Kit II. When using the PIC32 Ethernet Starter Kit, the `MEB ENET USB Thumb Drive MP3 Demo.mcw` MPLAB IDE workspace file should be used. The instructions for compiling and running the demo are the same for both boards. Familiarity with MPLAB® IDE by the user is assumed.

1. Open MPLAB IDE. Click *Configure>Settings*. Click the **Projects** tab and clear “Use one-to-one project-workspace model”, as shown in Figure 8. Click **OK**.

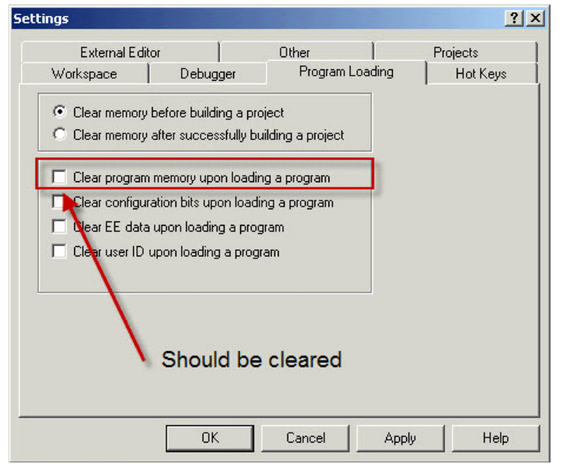
FIGURE 8: MPLAB® IDE SETTINGS



☑ Close open source files on project close
☑ Clear output window before build
☑ Save project before build
Save files before build: [✓ Yes] No Prompt
☑ Halt build on first failure
☐ Use one-to-one project-workspace model

2. Click *Configure>Settings*. Click the **Program Loading** tab. The “Clear program memory upon loading a program” option should be cleared, as shown in Figure 9.

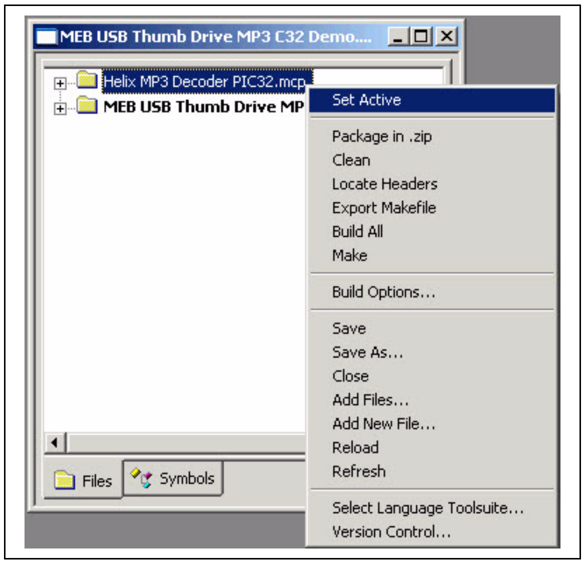
FIGURE 9: MPLAB® IDE PROGRAM LOADING SETTINGS



[✓ Clear memory before building a project] Clear memory after successfully building a project
☐ Clear program memory upon loading a program
☐ Clear configuration bits upon loading a program
☐ Clear EE data upon loading a program
☐ Clear user ID upon loading a program

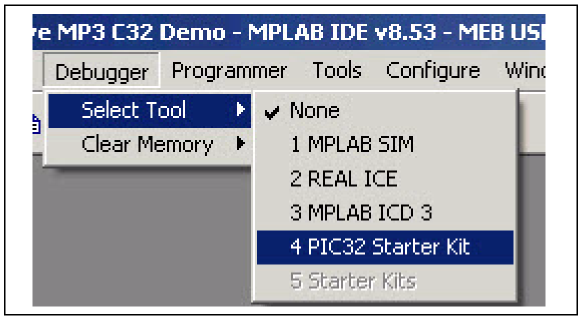
3. Click *File>Open Workspace* and open the `MEB USB Thumb Drive MP3 C32 Demo.mcw` MPLAB IDE workspace file. The application project is loaded in the MPLAB IDE.
4. The Project Explorer window displays two projects within the workspace. The `Helix MP3 Decoder PIC32.mcp` project is composed of the Helix MP3 decoder source code with modifications made for running the code on PIC32MX microcontrollers.
5. Right click the `Helix MP3 Decoder PIC32.mcp` project, and then select Set Active from the menu, as shown in Figure 10.

FIGURE 10: MAKING A PROJECT ACTIVE



6. Ensure that no debugger or programmer is selected. The Debug/Release choice does not affect the operation and can be ignored. Compile and build the project using the "Build All" option.
7. Right click the `MEB USB Thumb Drive MP3 C32 Demo.mcp` project, and then select Set Active from the menu.
8. Connect the PIC32 USB Starter Kit II to the MEB. Use the fastener to ensure that the starter kit is securely locked into the socket.
9. Connect the starter kit debug USB port to an available USB port on the PC.
10. In MPLAB IDE, click *Debugger>Select Tool>PIC32 Starter Kit*, as shown in Figure 11 to load the PIC32 Starter Kit.

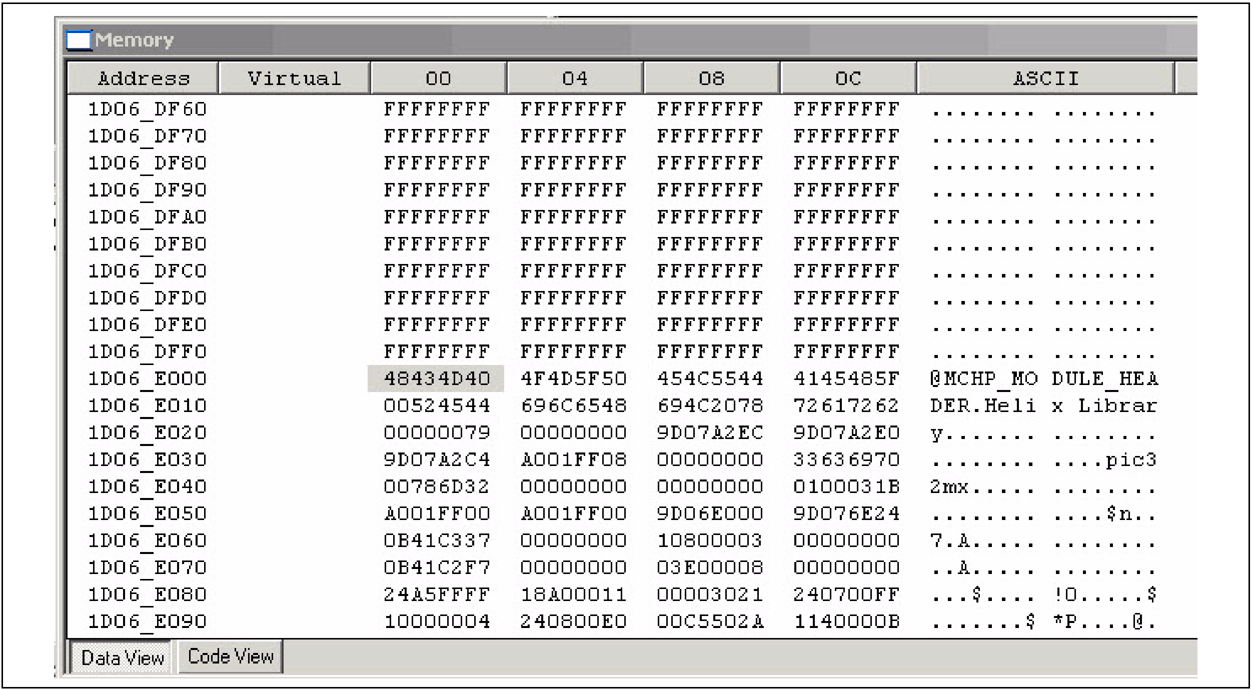
FIGURE 11: SELECTING THE DEBUGGER



11. Compile and build the project.
12. Program the PIC32MX device. The MPLAB IDE then programs two hex images onto the device:
    - `Helix MP3 Decoder PIC32.elf`
    - `MEB USB Thumb Drive MP3 C32 Demo.elf`
13. *This step is optional*. Check the program memory location at address 0x9D06E000. In MPLAB IDE, click *View>Memory*, and then click **Data View** in the Memory window. Right click in the Memory window, select Go To, and then enter the address as 0x9D06E000. The resultant Memory window is displayed, as shown in Figure 12. The string "Helix Library" indicates that the programming operation is successful.

The demo application is now ready for use.

FIGURE 12: PROGRAM MEMORY VIEW



```
 Address    Virtual    00        04        08        0C          ASCII
1D06_DF60            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_DF70            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_DF80            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_DF90            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_DFA0            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_DFB0            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_DFC0            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_DFD0            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_DFE0            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_DFF0            FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF  ........ ........
1D06_E000            48434D40  4F4D5F50  454C5544  4145485F  @MCHP_MO DULE_HEA
1D06_E010            00524544  696C6548  694C2078  72617262  DER.Heli x Librar
1D06_E020            00000079  00000000  9D07A2EC  9D07A2E0  y....... ........
1D06_E030            9D07A2C4  A001FF08  00000000  33636970  ........ ....pic3
1D06_E040            00786D32  00000000  00000000  0100031B  2mx..... ........
1D06_E050            A001FF00  A001FF00  9D06E000  9D076E24  ........ ....$n..
1D06_E060            0B41C337  00000000  10800003  00000000  7.A..... ........
1D06_E070            0B41C2F7  00000000  03E00008  00000000  ..A..... ........
1D06_E080            24A5FFFF  18A00011  00003021  240700FF  ...$.... !0.....$
1D06_E090            10000004  240800E0  00C5502A  1140000B  .......$ *P....@.
```

Running the Demo Application

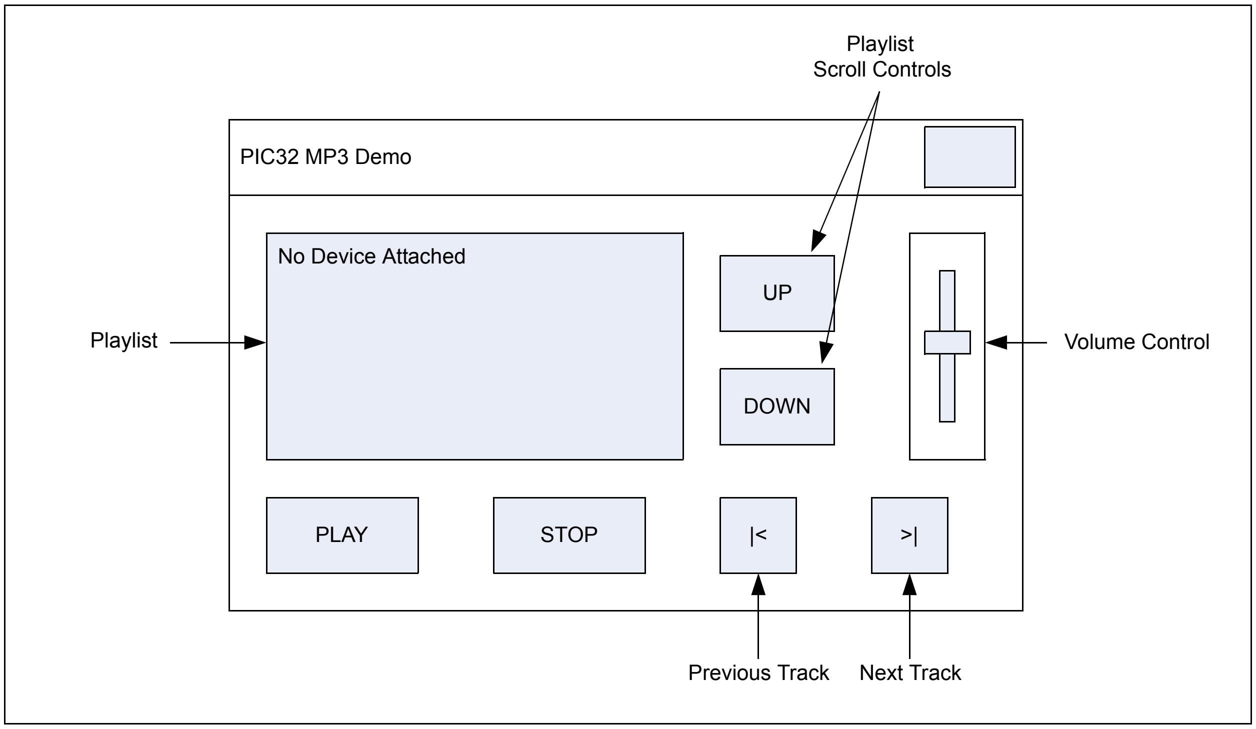
Insert the headphones into the headphone jack on the MEB. In MPLAB IDE click **Run**. Observe the MEB touch screen. Figure 13 illustrates the MP3 player demo application GUI.

Insert the USB thumb drive into the USB host receptacle on the PIC32MX USB Starter Kit II. The Playlist is populated with MP3 files located in the root directory of the thumb drive. The application starts to play the first MP3 file in the Playlist.

The functionality of the touch screen buttons are as follows:

- The Volume Control slider is used to adjust the volume.
- The STOP button is used to stop the play of a MP3 file.
- Pressing the PLAY button while playback is in progress causes the playback to pause. Pressing it again resumes the playback.
- The UP and DOWN buttons allow the user to scroll through the files in the Playlist. Touching a file in the Playlist causes the file to be selected for playback.

FIGURE 13: MP3 PLAYER DEMO APPLICATION GUI LAYOUT



PROJECT FILES

The source code (see **Appendix A: "Source Code"**) accompanying this application note contains demonstration source code for the PIC32MX USB Starter Kit II and PIC32MX Ethernet Starter Kit.

The application in the `MEB USB Thumb Drive MP3 Demo` folder runs on MEB while using the PIC32MX USB Starter Kit II. The application in the `MEB ENET USB Thumb Drive MP3 Demo` folder runs on MEB while using the PIC32MX Ethernet Starter Kit.

The folder and file descriptions provided in Table 3 apply to both the application folders. Any differences are explicitly called out.

CONCLUSION

The Helix MP3 decoder is an open source MP3 decoder and can be ported onto Microchip's PIC32MX 32-bit microcontrollers. The decoder API has been described and a MP3 player application is provided along with this application note to demonstrate the use of the MP3 decoder. Additionally, the RTLL technique is described and demonstrated.

TABLE 3: SOURCE CODE FILE AND FOLDER DESCRIPTIONS

Item Name	Description
`h`	Folder containing Include files required by the `MEB USB Thumb Drive MP3 C32 Demo` project.
`lib`	Folder containing USB and graphics stack archives.
`obj`	Folder to store temporary Board Support Package (BSP) object files.
`src`	Folder containing source files required by the `MEB USB Thumb Drive MP3 C32 Demo` project.
`MP3 Decoder Source Code`	Folder containing the Helix MP3 decoder source code.
`procdefs.ld`	Modified linker script file required by the `MEB USB Thumb Drive MP3 C32 Demo` project and the `Helix MP3 Decoder PIC32 MPLAB IDE` project.
`MEB USB Thumb Drive MP3 C32 Demo.mcp` or `MEB ENET USB Thumb Drive MP3 Demo.mcp`	`MPLAB IDE` project file.
`MEB USB Thumb Drive MP3 C32 Demo.mcw` or `MEB ENET USB Thumb Drive MP3 Demo.mcw`	`MPLAB IDE` workspace file.
`CleanUp.bat`	Batch file to clean up temporary files.
`MAL BSP Files Archive.bat`	Batch file to update and build the BSP archive.
`MAL GFX Stack Archive.bat`	Batch file to update and build the graphics stack archive.
`MAL USB Stack Archive.bat`	Batch file to update and build the USB stack archive.
`Readme.txt`	File containing information on running the demo.
`Alternative Configurations`	Folder containing include files required by MEB graphics display.

APPENDIX A: SOURCE CODE

All of the software covered in this application note is available as a single WinZip archive file. This archive can be downloaded from the Microchip corporate web site at:

www.microchip.com

NOTES:

ISBN: 978-1-60932-630-2

MICROCHIP

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-6578-300
Fax: 886-3-6578-370

Taiwan - Kaohsiung
Tel: 886-7-213-7830
Fax: 886-7-330-9305

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

08/04/10