REALTEK

Realtek Ameba M4A_AAC Codec

This document highlights the use of the m4a codec in AMEBA

Table of Contents

1 Introduction

The support for M4A and AAC decoding has been introduced in the AMEBA SDK and can be used for decoding the related codecs on the AMEBA board and listening to their output using audio shields.

1.1 Description of M4A

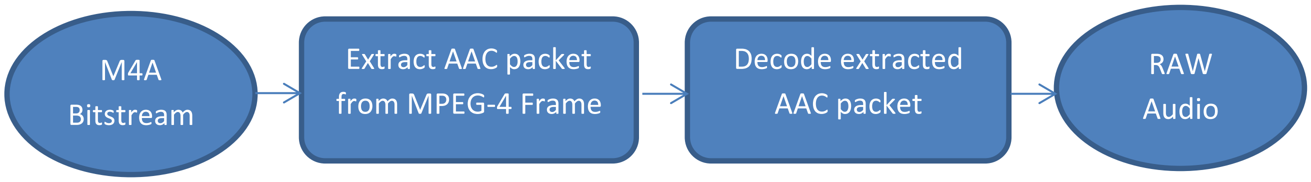
M4A is a file extension for an audio file encoded with advanced audio coding (AAC). M4A stands for MPEG-4 Audio, which is an MPEG4 container that contains AAC encoded audio.

Due to this unique nature of M4A files, there has been a need to port more than one codec to conduct the process of decoding the m4a file. The details of the codecs used have been highlighted in the next section.

1.2 Procedure to decode M4A

The M4A file stands for MPEG-4 Audio. The MPEG-4 container is a video container but here in this case it is used to encapsulate only audio data hence he M4A file has an MPEG-4 container which carries encoded audio data of the AAC format. Due to this nature of M4A files it needs to be decoded in 2 stages before obtaining the RAW audio data that can be supplied to an audio shield for playback.

The M4A decoding procedure is highlighted by the following diagram:-



As seen in the above diagram there are 2 intermediate stages before the final RAW data is obtained from the M4A file. These 2 stages are performed by 2 different open source codecs that have been ported to AMEBA SDK. The details of these open source codecs have beem detailed in the next section.

2 Open Source Codecs

In order to build an M4A decoder some open source software has been used. In order to build an M4A decoder, 2 codecs needed to be ported to AMEBA SDK.

2.1 LIBAV codec

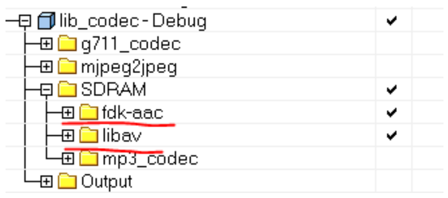
The libav codec has been ported to the AMEBA SDK in order to be able to decode the m4a MPEG-4 header and extract the aac packets from the m4a data.

2.2 FDK-AAC codec

Once the aac data is ectracted from the MPEG-4 container an AAC decoder is needed to decode the aac encoded data into its raw form PCM to play using various audio shields. In order to do this we ported the FDK-AAC codec to AMEBA.

3 Location in project

The files related to the codec can be found in the SDK in form of a library under lib_codec as shown below.



4 API Documentation for M4A

4.1 av_register_all

This function is part of libav and is used to register the handles of all the codecs that are supported by the libav codec. In case of AMEBA this call has been modified only to register and initialize the m4a header decoder.

4.2 avformat_open_input

This function is also part of libav and is used to open the m4a audio file from SDCARD which is used to extract the m4a header to initialize extraction of the aac packets. This call is also used to extract the extradata present in the m4a header which is used further in the call “aacDecoder_ConfigRaw.

4.3 aacDecoder_Open

This function call is part of FDK-AAC and is used to open an instance of the AAC decoder. The parameters are used to specify the format type of the audio that is encapsulated inside audio bitstream in the case of m4a files it is MPEG-4 data hence we use the parameter “TT_MP4_RAW”. For more details regarding this function refer to the FDK-AAC documentation*****.

4.4 aacDecoder_ConfigRaw

Explicitly configure the decoder by passing a raw AudioSpecificConfig (ASC) or a StreamMuxConfig (SMC), contained in a binary buffer. This is required for MPEG-4 and Raw Packets file format bitstreams. The parameters for this function call are the extradata that was extracted using the "avformat_open_input call". For more details regarding this function refer to the FDK-AAC documentation*.

4.5 av_read_frame

This function is part of libav and is used to extract the aac encoded data from each frame of the m4a data. The parameters are the AVFormatContext which was initialized using the function "avformat_open_input" and the Avpacket variable which will return the pointer to the data and size of the aac data that is encapsulated within the mpeg-4 frame.

4.6 aacDecoder_Fill

This function is used to fill the internal buffers of the FDK-AAC decoder with the packet data and the packet size that was extracted by the api "av_read_frame". This function is called before the actual decode functions so that the input buffers of the FDK-AAC decoder are filled with the data before the actual decoding begins. For more details regarding this function refer to the FDK-AAC documentation*.

4.7 aacDecoder_DecodeFrame

This function is part of the FDK-AAC decoder and is used to initiate the decoder to begin decoding the aac packet that has been supplied to it in the previous call of "aacDecoder_Fill". This function is given the parameter of the aac decoder handle and the output buffer that will contain the final raw audio data that is obtained after decoding the aac packet. For more details regarding this function refer to the FDK-AAC documentation*.

4.8 aacDecoder_GetStreamInfo

This API is used to extract the type of stream in order to find out the stream parameters like frequency, number of channels etc. This API is for indicative purposes only. For more details regarding this function refer to the FDK-AAC documentation*.

→The FDK-AAC documentation can be found at: - https://github.com/mstorsjo/fdk-aac/blob/master/documentation/aacDecoder.pdf

5 Running the M4A example

There is an example that demonstrates how to play an m4a encoded file from SDCARD and play the output through an audio shield.

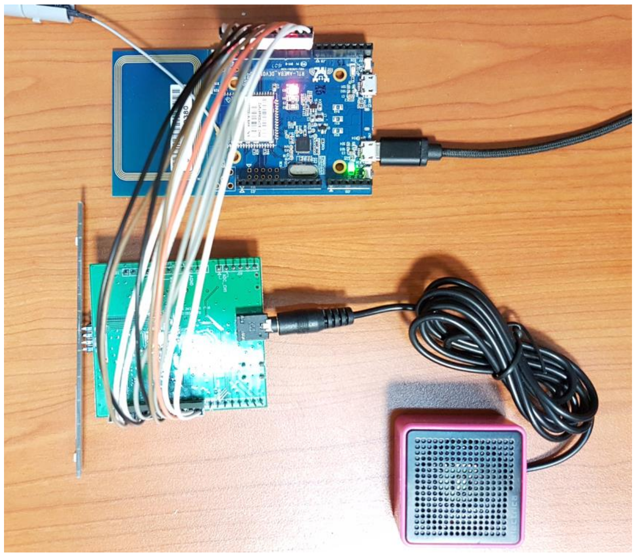
5.1 Hardware requirements

- AMEBA 1 Board
- Audio Shield(ALC5651/ALC5680/SGTL5000)
- Connectors
- 3.5mm earphones/speaker
- SDCARD sniffer

5.2 Hardware Setup

5.2.1 Connect Audio Shield

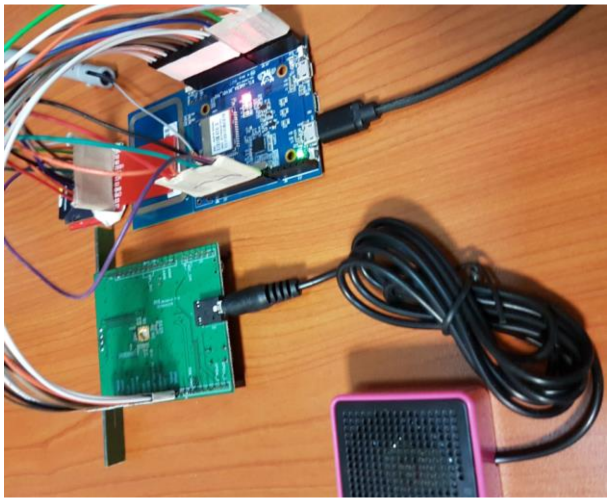
The audio shield is connected as shown in the image below. The audio shield in the image given below is the ALC5680 but connections are similar for the case of ALC5651 and SGTL5000.



The audio shield cannot be directly connected to AMEBA as we need the other slots on the AMEBA board to connect the SDCARD which shall be shown in the next step. A speaker or any headphone can be connected to the 3.5mm jack for audio output.

5.2.2 Connecting SDCARD sniffer

The SDCARD sniffer must be connected to the AMEBA board as shown below and an SDCARD must be inserted with the appropriate m4a file.

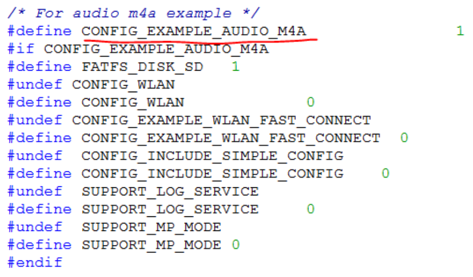


5.3 Software Setup

The following steps need to be ensured in the SDK in order to make the example run smoothly.

5.3.1 Enable example in platform_opts.h

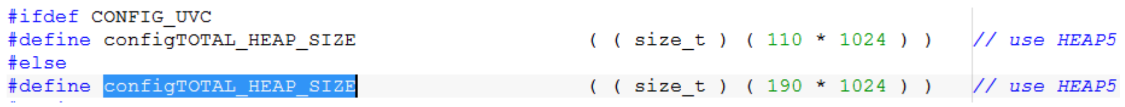
In the file platform_opts.h enable the macro CONFIG_EXAMPLE_AUDIO_M4A to 1 as shown below.



```
/* For audio m4a example */
#define CONFIG_EXAMPLE_AUDIO_M4A              1
#if CONFIG_EXAMPLE_AUDIO_M4A
#define FATFS_DISK_SD    1
#undef CONFIG_WLAN
#define CONFIG_WLAN                 0
#undef CONFIG_EXAMPLE_WLAN_FAST_CONNECT
#define CONFIG_EXAMPLE_WLAN_FAST_CONNECT  0
#undef  CONFIG_INCLUDE_SIMPLE_CONFIG
#define CONFIG_INCLUDE_SIMPLE_CONFIG     0
#undef  SUPPORT_LOG_SERVICE
#define SUPPORT_LOG_SERVICE      0
#undef  SUPPORT_MP_MODE
#define SUPPORT_MP_MODE  0
#endif
```

5.3.2 Increase heap size in FreeRTOSConfig.h

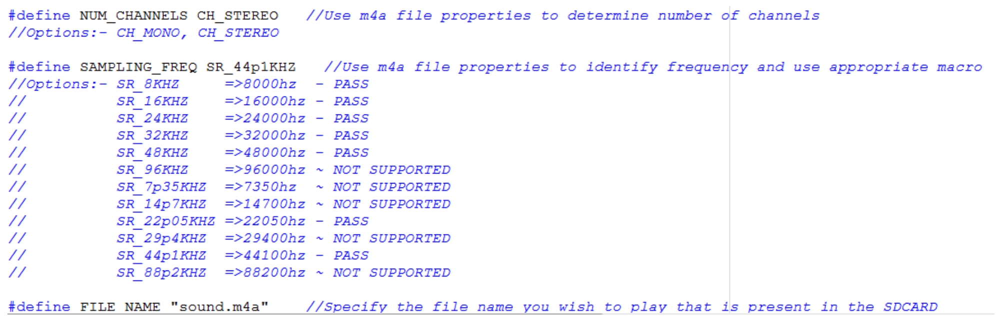
In order to run the example smoothly, the heap size (configTOTAL_HEAP_SIZE) needs to be increased to 190 x 1024 as shown below.



```
#ifdef CONFIG_UVC
#define configTOTAL_HEAP_SIZE                ( ( size_t ) ( 110 * 1024 ) )   // use HEAP5
#else
#define configTOTAL_HEAP_SIZE                ( ( size_t ) ( 190 * 1024 ) )   // use HEAP5
```

5.3.3 Set the frequency and number of channels.

In order to achieve smooth playback the frequency of the audio file, the number of channels in the audio file and the name of the audio file must be accurately set in the config parameters of the example_audio_m4a.c file as shown below.



```
#define NUM_CHANNELS CH_STEREO   //Use m4a file properties to determine number of channels
//Options:- CH_MONO, CH_STEREO

#define SAMPLING_FREQ SR_44p1KHZ   //Use m4a file properties to identify frequency and use appropriate macro
//Options:- SR_8KHZ     =>8000hz  - PASS
//          SR_16KHZ    =>16000hz - PASS
//          SR_24KHZ    =>24000hz - PASS
//          SR_32KHZ    =>32000hz - PASS
//          SR_48KHZ    =>48000hz - PASS
//          SR_96KHZ    =>96000hz ~ NOT SUPPORTED
//          SR_7p35KHZ  =>7350hz  ~ NOT SUPPORTED
//          SR_14p7KHZ  =>14700hz ~ NOT SUPPORTED
//          SR_22p05KHZ =>22050hz - PASS
//          SR_29p4KHZ  =>29400hz ~ NOT SUPPORTED
//          SR_44p1KHZ  =>44100hz - PASS
//          SR_88p2KHZ  =>88200hz ~ NOT SUPPORTED

#define FILE_NAME "sound.m4a"    //Specify the file name you wish to play that is present in the SDCARD
```

5.3.4 Transfer file to SDCARD and flash software

Once all the above changes are made, proceed to flash the binary onto the AMEBA and put the correct m4a file on the SDCARD and reset the board to hear the playback.

6 Running M4A example with WLAN enabled

Due to the high memory consumption of the various AUDIO codecs mentioned in the previous sections the LOG service and WLAN have been disabled, in order to use the example with WLAN and log service active, the following changes need to be made.

All the previous steps mentioned in the previous section must be followed as mentioned and apart from that the steps below must also be followed.

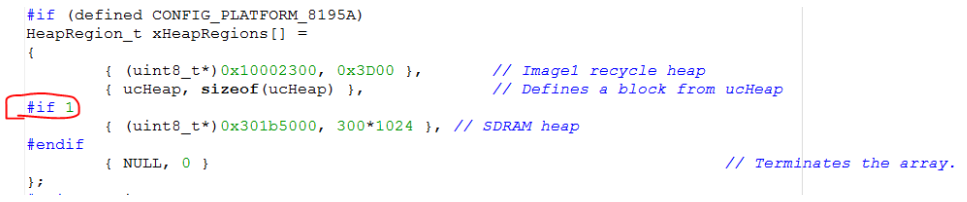
6.1 Move WLAN to SDRAM

Use the following wiki documentation to move the whole of the WLAN library to SDRAM.

https://wiki.realtek.com/pages/viewpage.action?pageId=85521501

6.2 Enable SDRAM heap

Once WLAN is enabled, more heap is required. To facilitate this SDRAM heap needs to be enabled.



```
#if (defined CONFIG_PLATFORM_8195A)
HeapRegion_t xHeapRegions[] =
{
        { (uint8_t*)0x10002300, 0x3D00 },        // Image1 recycle heap
        { ucHeap, sizeof(ucHeap) },              // Defines a block from ucHeap
#if 1
        { (uint8_t*)0x301b5000, 300*1024 }, // SDRAM heap
#endif
        { NULL, 0 }                                              // Terminates the array.
};
```

7 Running M4A and MP3 decoding on single image with WLAN.

An example with both mp3 decoding and m4a decoding has been made to work in a single image with WLAN running.

Follow the hardware setup as shown in section 5.2.

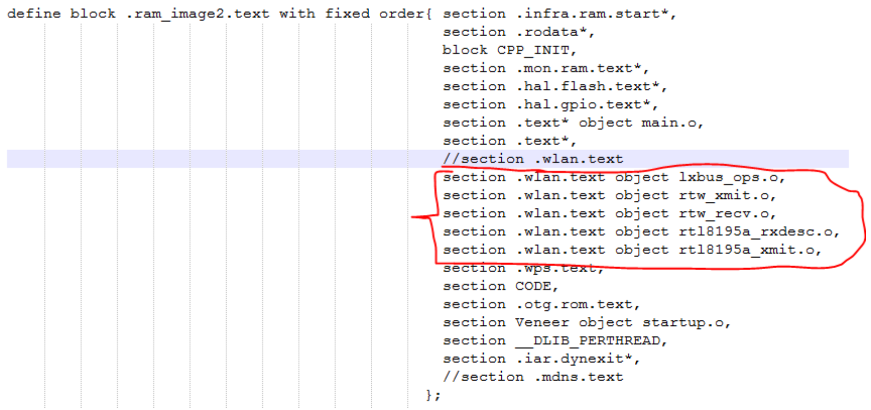
7.1 Move WLAN to SDRAM

In order to move the WLAN files to SDRAM and maintain the performance of the WLAN driver, the “image2.icf” file needs to be modified in order to keep some essential components of the WLAN driver in SRAM while the others are moved to SDRAM.

The “image2.icf” file can be found under “project\realtek_ameba1_va0_example\EWARM-RELEASE” or the respective RELEASE directory. The following modifications need to be made to the file.

Under “**define block .ram_image2.text with fixed order**” Comment out the “**section .wlan.text**” and add the following lines as shown below.

section .wlan.text object lxbus_ops.o,
section .wlan.text object rtw_xmit.o,
section .wlan.text object rtw_recv.o,
section .wlan.text object rtl8195a_rxdesc.o,
section .wlan.text object rtl8195a_xmit.o,

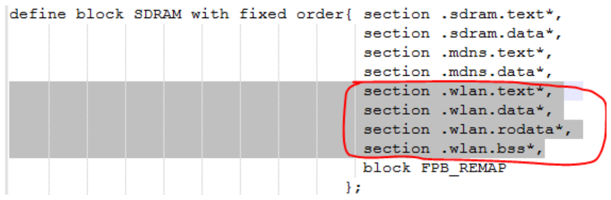


```
define block .ram_image2.text with fixed order{ section .infra.ram.start*,
                                                section .rodata*,
                                                block CPP_INIT,
                                                section .mon.ram.text*,
                                                section .hal.flash.text*,
                                                section .hal.gpio.text*,
                                                section .text* object main.o,
                                                section .text*,
                                                //section .wlan.text
                                                section .wlan.text object lxbus_ops.o,
                                                section .wlan.text object rtw_xmit.o,
                                                section .wlan.text object rtw_recv.o,
                                                section .wlan.text object rtl8195a_rxdesc.o,
                                                section .wlan.text object rtl8195a_xmit.o,
                                                section .wps.text,
                                                section CODE,
                                                section .otg.rom.text,
                                                section Veneer object startup.o,
                                                section __DLIB_PERTHREAD,
                                                section .iar.dynexit*,
                                                //section .mdns.text
                                              };
```

Apart from this add the components of WLAN under the “**define block SDRAM with fixed order**” with the following lines as shown below.

section .wlan.text*,
section .wlan.data*,

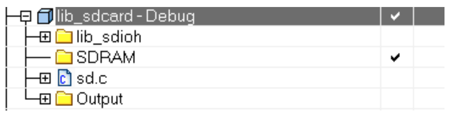
section .wlan.rodata*,
section .wlan.bss*,



```
define block SDRAM with fixed order{ section .sdram.text*,
                                     section .sdram.data*,
                                     section .mdns.text*,
                                     section .mdns.data*,
                                     section .wlan.text*,
                                     section .wlan.data*,
                                     section .wlan.rodata*,
                                     section .wlan.bss*,
                                     block FPB_REMAP
                                   };
```

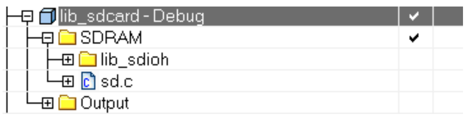
7.2 Move the lib_sdcard to SDRAM.

In order to do so open the library in the project as shown below.



```
lib_sdcard - Debug     ✓
├ lib_sdioh
├ SDRAM                ✓
├ sd.c
└ Output
```

Move the folder “lib_sdioh” and the file “sd.c” into the SDRAM folder that has already been created as shown.

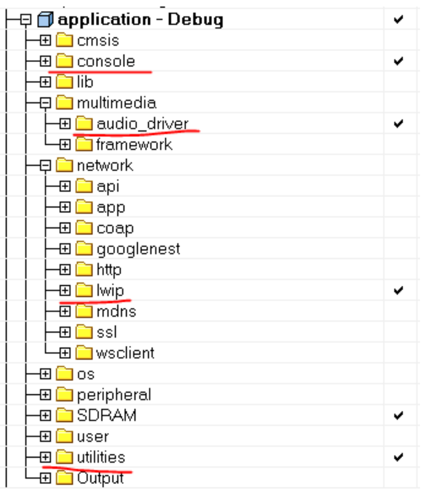


```
lib_sdcard - Debug     ✓
├ SDRAM                ✓
│ ├ lib_sdioh
│ └ sd.c
└ Output
```

Once both are moved to SDRAM folder, rebuild the lib_sdcard library.

7.3 Move other non-essential files to SDRAM.

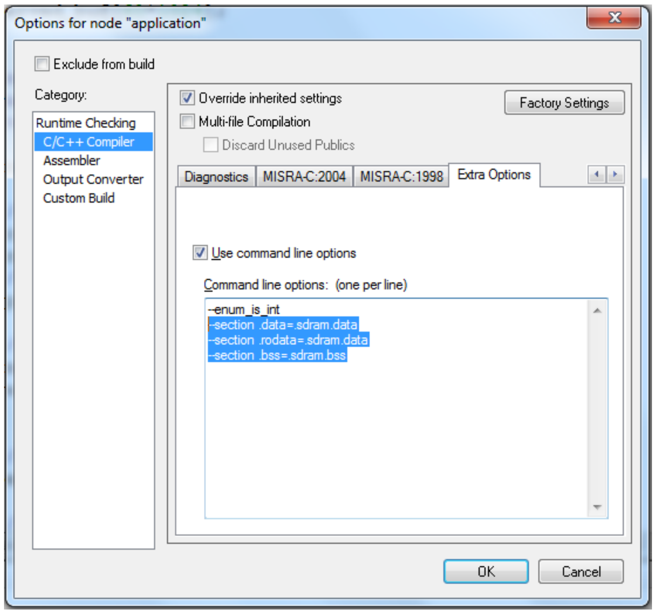
Since mp3, m4a, and WLAN take up a lot of memory, a lot of other files and folders need to be moved to SDRAM from the application part. In order to fit mp3, m4a and WLAN along with smooth playback and connectivity, the following fodders in the application project need to be moved to SDRAM as shown below.



```
application - Debug    ✓
├ cmsis
├ console              ✓
├ lib
├ multimedia
│ ├ audio_driver       ✓
│ └ framework
├ network
│ ├ api
│ ├ app
│ ├ coap
│ ├ googlenest
│ ├ http
│ ├ lwip               ✓
│ ├ mdns
│ ├ ssl
│ └ wsclient
├ os
├ peripheral
├ SDRAM                ✓
├ user
├ utilities            ✓
└ Output
```

In order to move a particular folder to SDRAM the following steps must be followed.

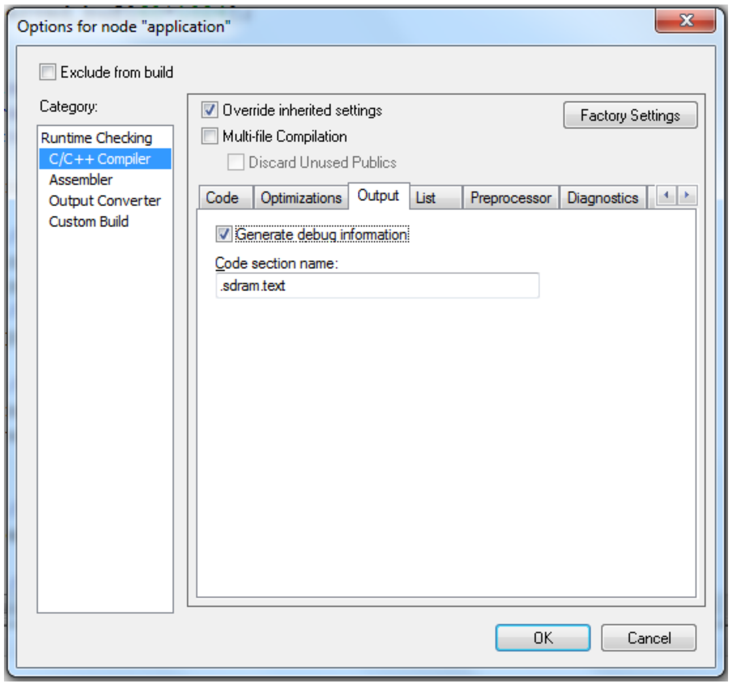
- Right click on the folder and click options.
- Inside options window click c/c++ compiler
- In order to edit the options, enable the checkbox "Override inherited settings"
- Once this is done go to the "Extra Options" tab and paste the following mappings to SDRAM as shown below.



[✓] Override inherited settings
[ ] Multi-file Compilation
[ ] Discard Unused Publics
[✓] Use command line options
Command line options: (one per line)

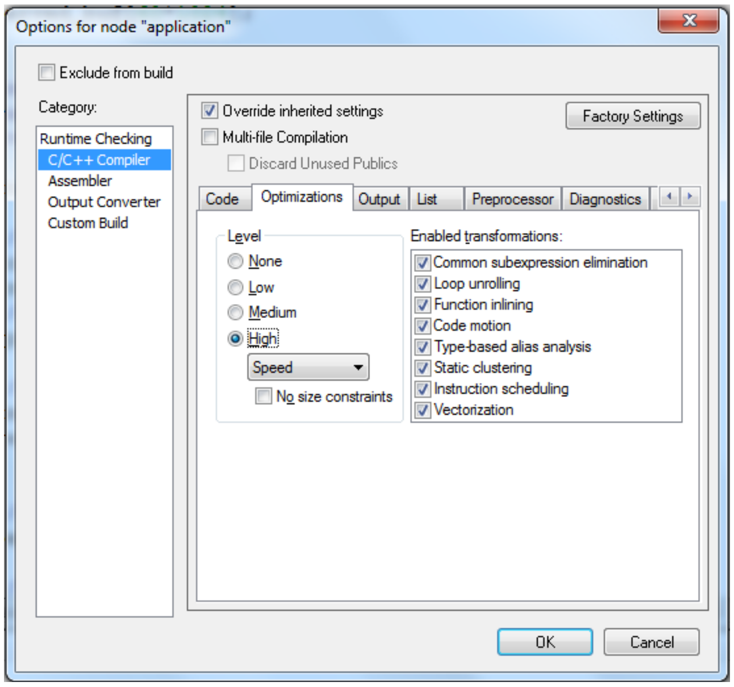
```
--enum_is_int
--section .data=.sdram.data
--section .rodata=.sdram.data
--section .bss=.sdram.bss
```

- Once this is done use the arrow keys and navigate to the "output" tab and map the .text to SDRAM as shown below.



[✓] Override inherited settings
[ ] Multi-file Compilation
[ ] Discard Unused Publics
[✓] Generate debug information
Code section name: .sdram.text

- After this navigate to the “Optimizations” tab and set the optimization level to high, select “speed” from the drop down menu and set the vectorization to on. This is done to speed up the performance even though the code is placed in SDRAM.


Options for node "application"

☐ Exclude from build

Category:
Runtime Checking
C/C++ Compiler
Assembler
Output Converter
Custom Build

☑ Override inherited settings
☐ Multi-file Compilation
☐ Discard Unused Publics

Tabs: Code | Optimizations | Output | List | Preprocessor | Diagnostics

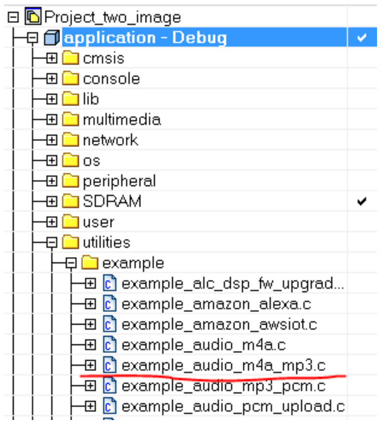
Level: None / Low / Medium / [✓ High]
Speed
☐ No size constraints

Enabled transformations:
☑ Common subexpression elimination
☑ Loop unrolling
☑ Function inlining
☑ Code motion
☑ Type-based alias analysis
☑ Static clustering
☑ Instruction scheduling
☑ Vectorization

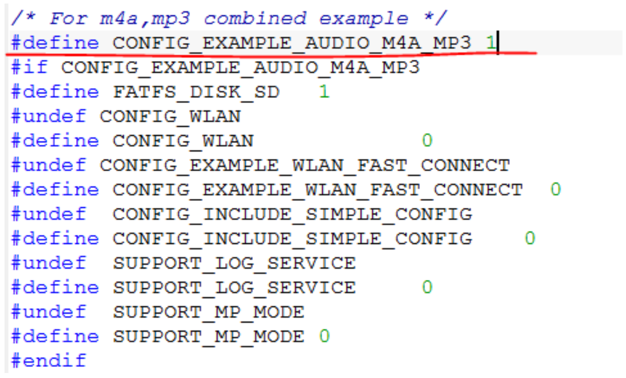
Repeat the above steps for all the folders shown in the previous image that need to be moved to SDRAM. Once this is done, save changes and rebuild the application.

7.4 Example Location

The location of the example is present under application>utilities>example_audio_m4a_mp3.c


```
Project_two_image
  application - Debug                ✓
    cmsis
    console
    lib
    multimedia
    network
    os
    peripheral
    SDRAM                            ✓
    user
    utilities
      example
        example_alc_dsp_fw_upgrad...
        example_amazon_alexa.c
        example_amazon_awsiot.c
        example_audio_m4a.c
        example_audio_m4a_mp3.c
        example_audio_mp3_pcm.c
        example_audio_pcm_upload.c
```

7.5 Enable the example options in platform_opts.h

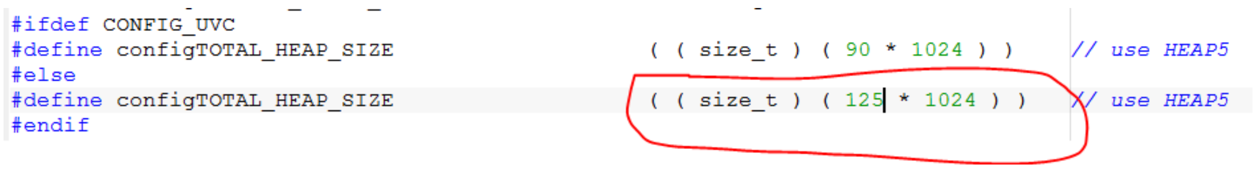


```
/* For m4a,mp3 combined example */
#define CONFIG_EXAMPLE_AUDIO_M4A_MP3 1
#if CONFIG_EXAMPLE_AUDIO_M4A_MP3
#define FATFS_DISK_SD    1
#undef CONFIG_WLAN
#define CONFIG_WLAN               0
#undef CONFIG_EXAMPLE_WLAN_FAST_CONNECT
#define CONFIG_EXAMPLE_WLAN_FAST_CONNECT  0
#undef  CONFIG_INCLUDE_SIMPLE_CONFIG
#define CONFIG_INCLUDE_SIMPLE_CONFIG     0
#undef  SUPPORT_LOG_SERVICE
#define SUPPORT_LOG_SERVICE       0
#undef  SUPPORT_MP_MODE
#define SUPPORT_MP_MODE 0
#endif
```

After enabling the macro that enables the example proceed to the next step.

7.6 Increase heap size in FreeRTOSConfig.h

As mentioned before for running this example as well, the free heap size (configTOTAL_HEAP_SIZE) needs to be increased to 125 x 1024 as shown below.

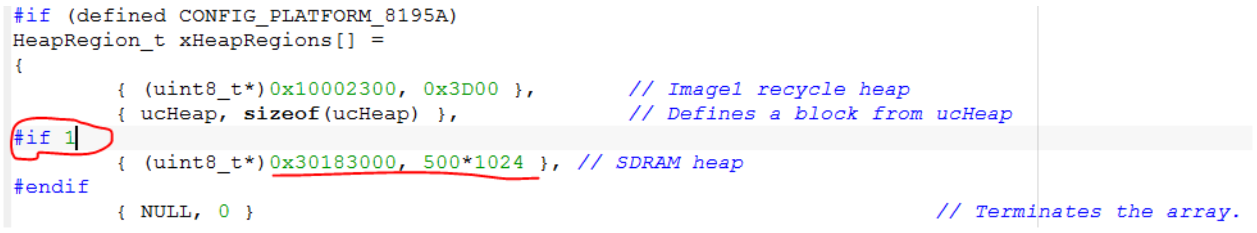


```
#ifdef CONFIG_UVC
#define configTOTAL_HEAP_SIZE                    ( ( size_t ) ( 90 * 1024 ) )    // use HEAP5
#else
#define configTOTAL_HEAP_SIZE                    ( ( size_t ) ( 125 * 1024 ) )   // use HEAP5
#endif
```

The increased heap is required in order to allow for the various memory allocations that are required by the MP3 and M4A decoder and WLAN library.

7.7 Enable SDRAM heap

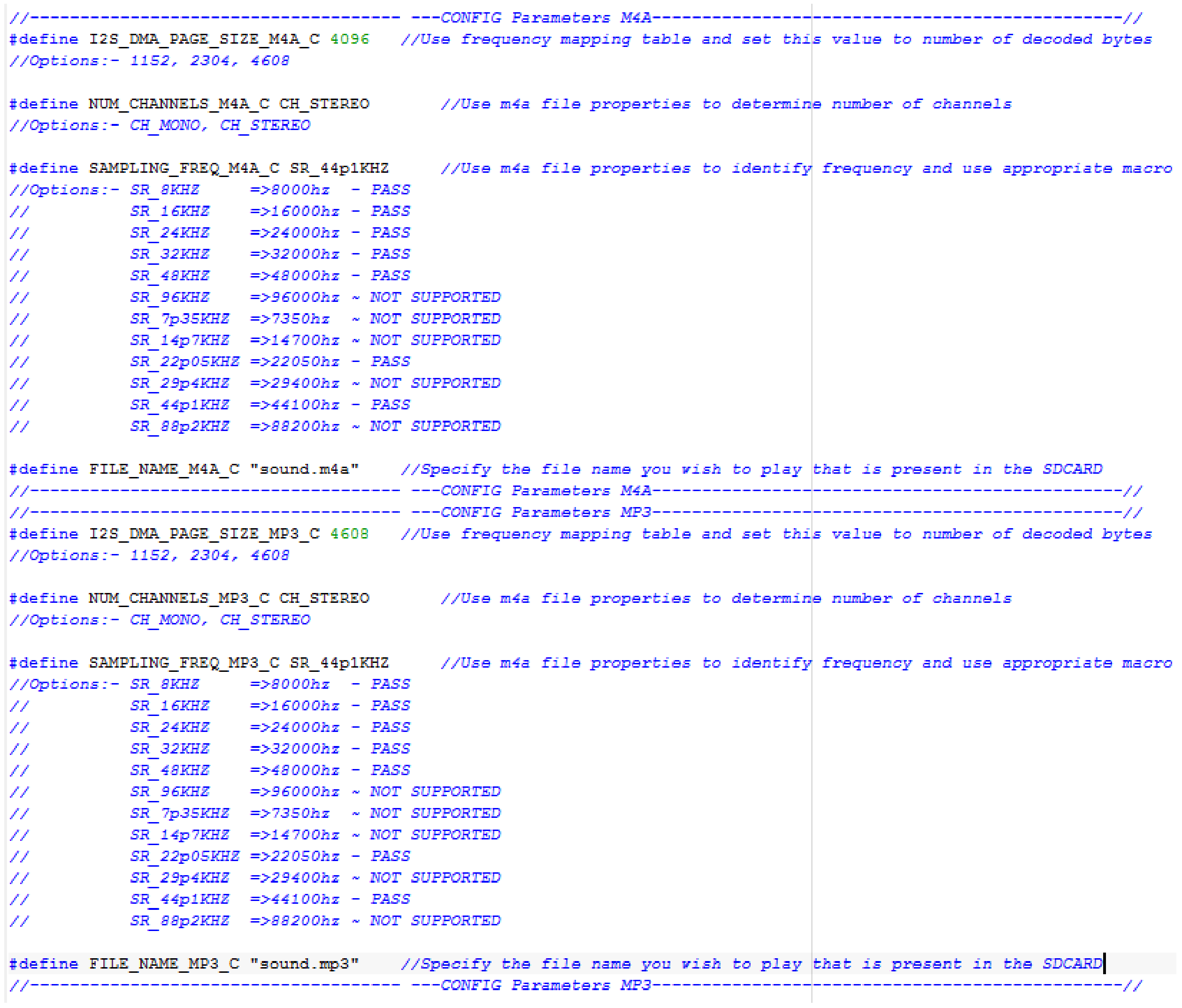
Once WLAN is enabled, more heap is required to perform certain function calls within WLAN and AUDIO Codecs. To facilitate this SDRAM heap needs to be enabled. In order to enable SDRAM heap open the file heap_5.c and make the condition shown below as true. Also increase the amount of memory allocated to heap as shown below.



```
#if (defined CONFIG_PLATFORM_8195A)
HeapRegion_t xHeapRegions[] =
{
        { (uint8_t*)0x10002300, 0x3D00 },        // Image1 recycle heap
        { ucHeap, sizeof(ucHeap) },              // Defines a block from ucHeap
#if 1
        { (uint8_t*)0x30183000, 500*1024 }, // SDRAM heap
#endif
        { NULL, 0 }                                              // Terminates the array.
```

7.8 Set frequency and number of channels in the config file.

In the mp3 and m4a example, there are 2 sets of config parameters are present in the file example_audio_m4a_mp3.c as shown below.



```
//------------------------------------- ---CONFIG Parameters M4A-----------------------------------------------//
#define I2S_DMA_PAGE_SIZE_M4A_C 4096   //Use frequency mapping table and set this value to number of decoded bytes
//Options:- 1152, 2304, 4608

#define NUM_CHANNELS_M4A_C CH_STEREO       //Use m4a file properties to determine number of channels
//Options:- CH_MONO, CH_STEREO

#define SAMPLING_FREQ_M4A_C SR_44p1KHZ     //Use m4a file properties to identify frequency and use appropriate macro
//Options:- SR_8KHZ     =>8000hz  - PASS
//          SR_16KHZ    =>16000hz - PASS
//          SR_24KHZ    =>24000hz - PASS
//          SR_32KHZ    =>32000hz - PASS
//          SR_48KHZ    =>48000hz - PASS
//          SR_96KHZ    =>96000hz ~ NOT SUPPORTED
//          SR_7p35KHZ  =>7350hz  ~ NOT SUPPORTED
//          SR_14p7KHZ  =>14700hz ~ NOT SUPPORTED
//          SR_22p05KHZ =>22050hz - PASS
//          SR_29p4KHZ  =>29400hz ~ NOT SUPPORTED
//          SR_44p1KHZ  =>44100hz - PASS
//          SR_88p2KHZ  =>88200hz ~ NOT SUPPORTED

#define FILE_NAME_M4A_C "sound.m4a"    //Specify the file name you wish to play that is present in the SDCARD
//------------------------------------- ---CONFIG Parameters M4A-----------------------------------------------//
//------------------------------------- ---CONFIG Parameters MP3-----------------------------------------------//
#define I2S_DMA_PAGE_SIZE_MP3_C 4608   //Use frequency mapping table and set this value to number of decoded bytes
//Options:- 1152, 2304, 4608

#define NUM_CHANNELS_MP3_C CH_STEREO       //Use m4a file properties to determine number of channels
//Options:- CH_MONO, CH_STEREO

#define SAMPLING_FREQ_MP3_C SR_44p1KHZ     //Use m4a file properties to identify frequency and use appropriate macro
//Options:- SR_8KHZ     =>8000hz  - PASS
//          SR_16KHZ    =>16000hz - PASS
//          SR_24KHZ    =>24000hz - PASS
//          SR_32KHZ    =>32000hz - PASS
//          SR_48KHZ    =>48000hz - PASS
//          SR_96KHZ    =>96000hz ~ NOT SUPPORTED
//          SR_7p35KHZ  =>7350hz  ~ NOT SUPPORTED
//          SR_14p7KHZ  =>14700hz ~ NOT SUPPORTED
//          SR_22p05KHZ =>22050hz - PASS
//          SR_29p4KHZ  =>29400hz ~ NOT SUPPORTED
//          SR_44p1KHZ  =>44100hz - PASS
//          SR_88p2KHZ  =>88200hz ~ NOT SUPPORTED

#define FILE_NAME_MP3_C "sound.mp3"    //Specify the file name you wish to play that is present in the SDCARD
//------------------------------------- ---CONFIG Parameters MP3-----------------------------------------------//
```

Since the example plays both mp3 and m4a files the config parameters for both the files need to be mentioned in the config parameters as shown above. There are sample files present in the example folder that can be used to test the example. The parameters that need to be set are frequency and the number of channels present on the file.

7.9 Transfer file to SDCARD and flash software

Once all the above changes are made, proceed to flash the binary onto the AMEBA and put the correct m4a and mp3 file on the SDCARD and reset the board to hear the playback.

8 Common Troubleshooting Issues

In case any of the examples do not work properly before running the examples please ensure that the following checks are made.

8.1 Check if heap size is increased.

The mp3 and m4a decoders require large amounts of free heaps and if the free heap size is not extended in FreeRTOSConfig.h. the heap size should at least be set at 190 x1024 bytes.

8.2 Check if SDRAM heap is enabled.

In case the examples need to be used with WLAN, the SDRAM heap should also be enabled to ensure that all mallocs are successful.

8.3 Verify correct frequency

For both mp3 and m4a the correct frequency should be set in the example code else the i2s driver will play the raw audio output at different frequency and the output will sound distorted.