Contents

Pre-requisites

- Follow the Getting Started steps found here [MIMXRT685-EVK Start Now]
- DSP Build environment: Xtensa Xplorer 8.10 + RI-2019.1
- Arm Build environment: MCUXpresso V11.1.1

Objectives

In this lab, you will learn:

- An overview of the Xtensa Audio Framework (XAF) from Cadence and it's integration into the NXP RT600 SDK.

Hardware

- Micro USB Cable
- MIMXRT685-EVK Rev E
- Headphones with 3.5 mm audio jack
- Female-to-female jumper wire

Lab high level description

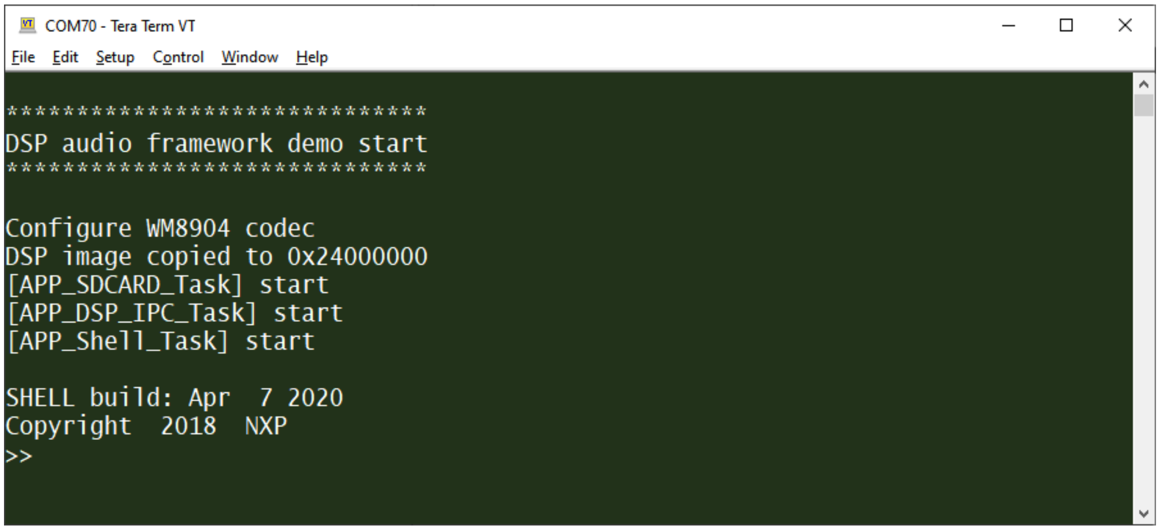
In this lab, we'll go through the Xtensa Audio Framework (XAF) to understand how to use it and leverage its components for creating custom audio applications. We'll use the SDK example of *dsp_xaf_demo* and see how the different components are interacting in the audio pipeline.

Xtensa Audio Framework SDK example walkthrough

In the SDK, there is the *dsp_xaf_demo* example. This example application demonstrates audio processing using the DSP core, the Xtensa Audio Framework (XAF) middleware library, and select Xtensa audio codecs.

When the application is started, a shell interface is displayed on the terminal that executes from the ARM® application. User can control this with shell commands which are relayed via RPMsg-Lite IPC to the DSP where they are processed and response is returned.

The ARM M33 core handles the application level user interaction and sending messages to the DSP to start and send data to the different audio processing applications running in the DSP. The DSP runs the XAF and runs audio pipeline or audio processing chain.



```
******************************
DSP audio framework demo start
******************************

Configure WM8904 codec
DSP image copied to 0x24000000
[APP_SDCARD_Task] start
[APP_DSP_IPC_Task] start
[APP_Shell_Task] start

SHELL build: Apr  7 2020
Copyright  2018  NXP
>>
```

Figure 1. XAF demo shell console

Shell commands

"help":	List all the registered commands
"exit":	Exit program
"version":	Query DSP for component versions
"aac":	Perform AAC decode on DSP
"mp3":	Perform MP3 decode on DSP
"opusdec":	Perform OPUS decode on DSP
"opusenc":	Perform OPUS encode on DSP
"vorbis":	Perform VORBIS decode on DSP
"file":	Perform audio file decode and playback on DSP
"src":	Perform sample rate conversion on DSP
"gain":	Perform PCM gain adjustment on DSP
"record_dmic":	Record DMIC audio and playback on WM8904 codec

RPMsg-Lite

For communication between the cores, the RT600 uses RPMsg-Lite which is a software layer that uses the Message Unit peripheral (MU). The following figure shows an example of the messages sent between the CM33 and the DSP for the “version” shell command.

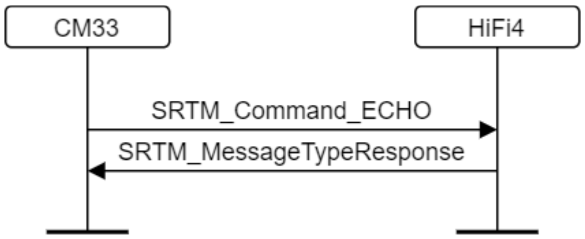
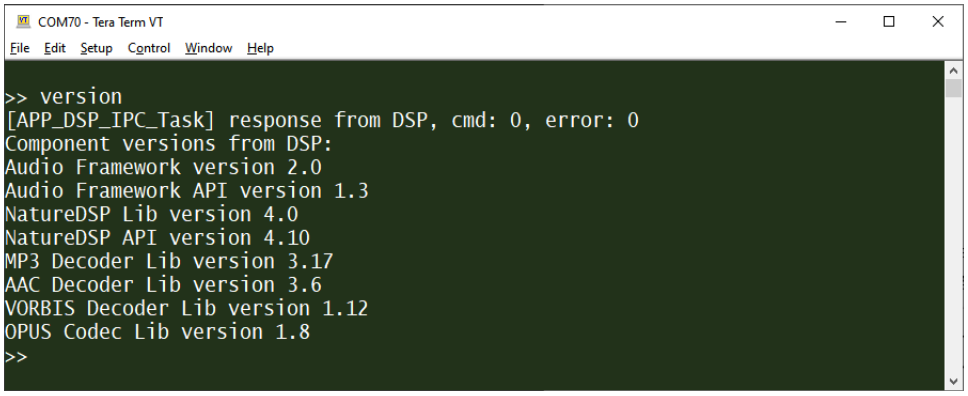


Figure 2. RPMsg for "version" command

Both the CM33 and the DSP have a dedicated task for listening for messages and trigger the required action based on the command received. In the above example, the shell task in the CM33 running the “version” command sends the message to the DSP and is blocked until it receives a response from the DSP. After the response is received, the CM33 displays the received information in the terminal.



```
>> version
[APP_DSP_IPC_Task] response from DSP, cmd: 0, error: 0
Component versions from DSP:
Audio Framework version 2.0
Audio Framework API version 1.3
NatureDSP Lib version 4.0
NatureDSP API version 4.10
MP3 Decoder Lib version 3.17
AAC Decoder Lib version 3.6
VORBIS Decoder Lib version 1.12
OPUS Codec Lib version 1.8
>>
```

Figure 3. "version" command response

Xtenxa Audio Framework (XAF)

Xtensa Audio Framework (XAF) is responsible for creating, configuring, and running the processing chains through XAF Developer API. Memory management of components, data movement between components, and scheduling of components is all done by XAF internally and is completely abstracted from the application.

For detailed documentation on XAF and the XAF developer API, you can refer to Cadence documentation located in:

"<RT600 SDK>\middleware\dsp\audio_framework\libxa_af_hostless\doc\HiFi-AF-Hostless-ProgrammersGuide.pdf"

XAF Terminology

The following terms are used within this lab. For a complete list of XAF terminology, refer to the *XAF Programmer's Guide* document.

Audio Device: The software abstraction of a digital signal processor (DSP) core.

Component: A software module that conforms to a specified interface and runs on the audio device. It would implement some audio processing functionality.

Chain: A graph formed by connecting different components by links.

Framework: A software entity that enables the creation of an audio processing chain. It manages the transfer of buffers between components as well as the scheduling of different components in the chain.

Application: A software entity that uses the framework to create a chain. It is the responsibility of the application to provide input data to the chain and consume the output data generated by the chain.

Audio pipeline

The audio pipeline or processing chain is made of different components such as: capturer, renderer, decoder, encoder and pre/post processing components. The following figure show an example of a typical audio pipeline in an audio application.

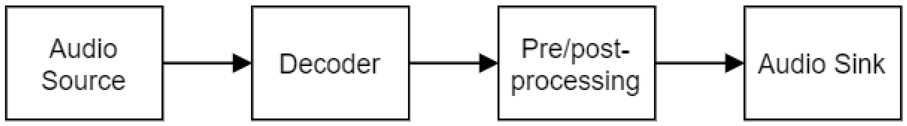


Figure 4. Audio pipeline example

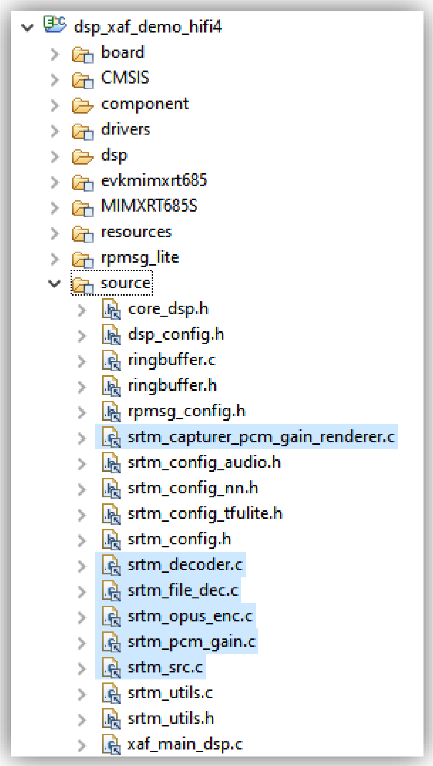
Audio components

Audio components are the actual data processing modules. XAF interacts with audio components using Cadence Audio Codec API (DSP Developer API). The following table lists the components supported by XAF.

Component type	Component description
Decoder	Decodes input compressed data to generate output PCM data.
Encoder	Encodes input PCM data to generate output compressed data.
Mixer	Combines input PCM data from multiple ports to generate one output PCM data.
Pre-processing	Pre-processes input PCM data to generate output PCM data.
Post-processing	Post-processes input PCM data to generate output PCM data.
Renderer	Plays input PCM data to a speaker/headphone.
Capturer	Captures output PCM data from a microphone.
MIMO	Multi-Input Multi-Output (MIMO) component process input PCM data to generate output PCM data.

Audio pipeline applications in XAF demo

The *dsp_xaf_demo* example creates different audio pipelines for each command in the shell console. The following figure shows the files in the HiFi4 project where each pipeline creation can be found. These files make use of the XAF API for the component creation and processing.



```
dsp_xaf_demo_hifi4
  board
  CMSIS
  component
  drivers
  dsp
  evkmimxrt685
  MIMXRT685S
  resources
  rpmsg_lite
  source
    core_dsp.h
    dsp_config.h
    ringbuffer.c
    ringbuffer.h
    rpmsg_config.h
    srtm_capturer_pcm_gain_renderer.c
    srtm_config_audio.h
    srtm_config_nn.h
    srtm_config_tflite.h
    srtm_config.h
    srtm_decoder.c
    srtm_file_dec.c
    srtm_opus_enc.c
    srtm_pcm_gain.c
    srtm_src.c
    srtm_utils.c
    srtm_utils.h
    xaf_main_dsp.c
```

Figure 5. Files implementing audio pipeline applications

Decoder

The commands “aac”, “mp3”, “opusdec” and “vorbis” create a similar audio pipeline each. They take a compressed input file for decoding and use the Renderer component to output I²S data to the external codec for playback or alternatively it can save the decoded PCM output directly. The audio pipeline for these commands is shown below.

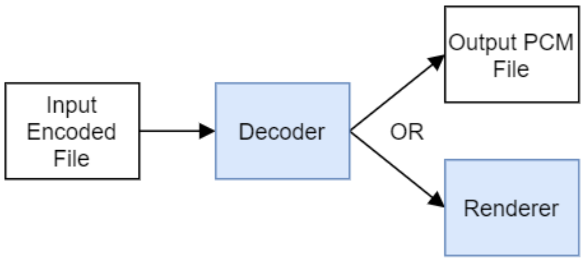


Figure 6. Audio pipeline for decoder commands in XAF demo.

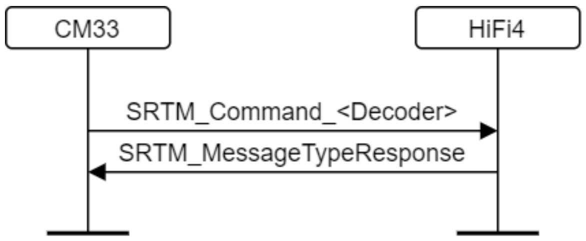


Figure 7. RPMsg for decoder commands in XAF demo.

Encoder

The command “opusenc” takes an uncompressed input PCM file, encodes it and saves the compressed output. The audio pipeline for this command is shown below.

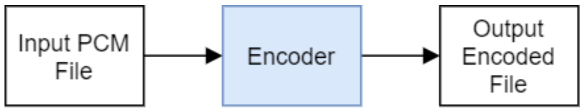


Figure 8. Audio pipeline for encoder commands in XAF demo.

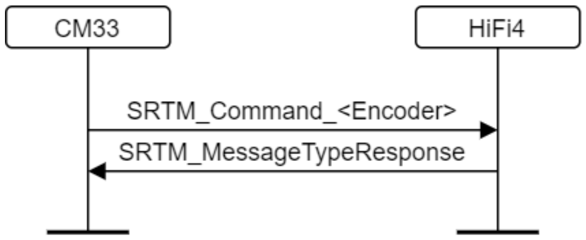


Figure 9. RPMsg for encoder commands in XAF demo.

SRC

The command “src” takes an uncompressed input PCM file uses the Sample Rate Converter component and saves the output PCM data. The audio pipeline for this command is shown below.

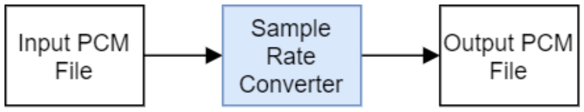


Figure 10. Audio pipeline for “src” command in XAF demo.

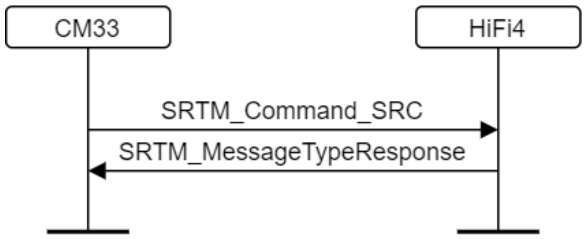


Figure 11. RPMsg for “src” command in XAF demo.

PCM gain

The command “gain” takes an uncompressed input PCM file uses the PCM Gain component and saves the output PCM data. The audio pipeline for this command is shown below.

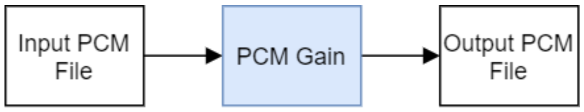


Figure 12. Audio pipeline for “gain” command in XAF demo.

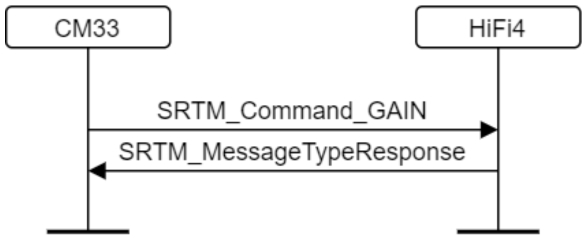


Figure 13. RPMsg for “gain” command in XAF demo.

Capturer gain renderer

The command "record_dmic" uses the Capturer component to use the DMIC for audio input. It then uses the PCM Gain component and use the Renderer component to output I^{2}S data to the external codec for playback. Note that this command doesn't return the shell console, the DSP enters an infinite loop executing the audio pipeline. The audio pipeline for this command is shown below.

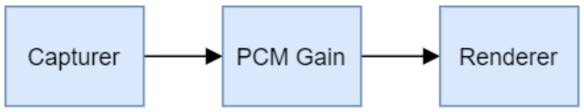


Figure 14. Audio pipeline for "record_dmic" command in XAF demo.

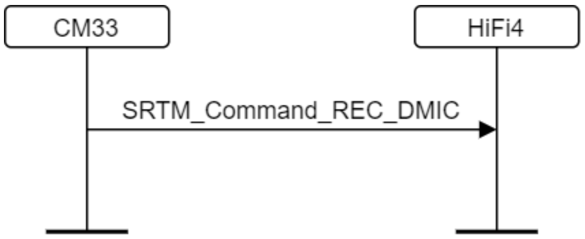


Figure 15. RPMsg for "record_dmic" command in XAF demo.

File decoder

The command "file" takes .mp3, .aac and .ogg files from the mounted SD card filesystem. It then decodes them and use the Renderer component to output I^{2}S data to the external codec for playback. The DSP in this application needs to continuously request data from the CM33 for continuous playback. The DSP creates two tasks for this, one for managing the audio buffer and request the CM33 for more data when a specific threshold is reached, and the other task is for processing the audio pipeline and checking for status and feeding data when needed. The audio pipeline for this command is shown below.

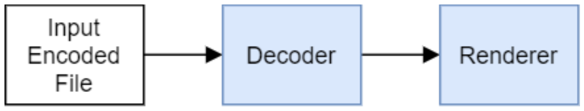


Figure 16. Audio pipeline for "file" command in XAF demo.

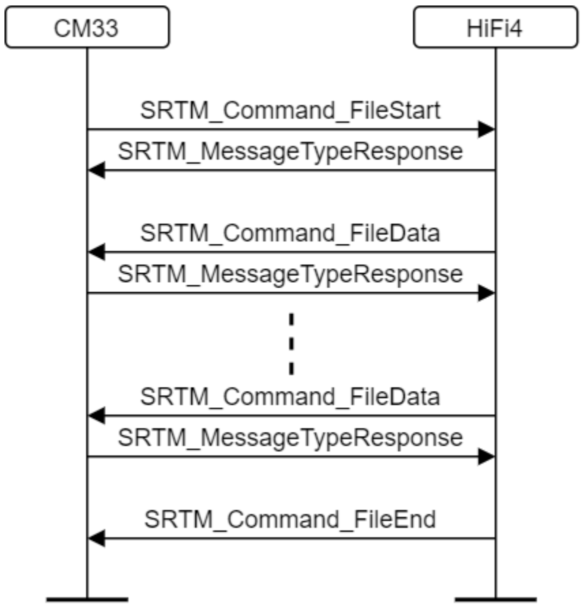
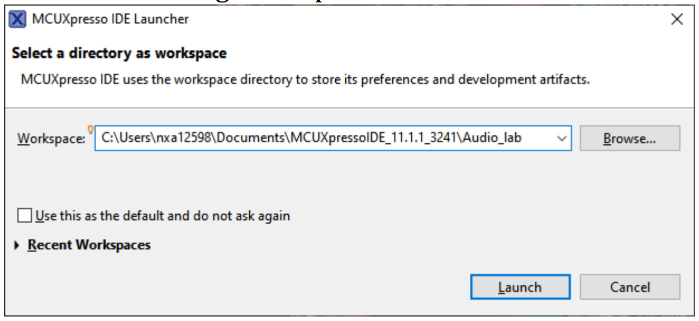


Figure 17. RPMsg for "file" command in XAF demo.

Running the XAF example in MCUXpresso

1. Open **MCUXpresso IDE v11.1.1**.

2. Select the existing workspace or create a new one.

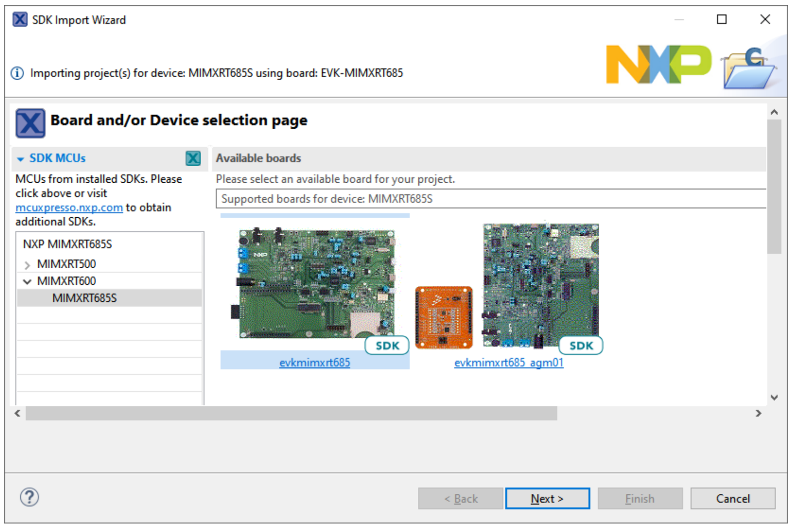


Select a directory as workspace

MCUXpresso IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\nxa12598\Documents\MCUXpressoIDE_11.1.1_3241\Audio_lab

3. Click on **Import SDK example(s)…** in the Quickstart Panel.
4. In **SDK Import Wizard** select evkmimxrt685 and click **Next**.


SDK Import Wizard

Importing project(s) for device: MIMXRT685S using board: EVK-MIMXRT685

Board and/or Device selection page

SDK MCUs

MCUs from installed SDKs. Please click above or visit mcuxpresso.nxp.com to obtain additional SDKs.

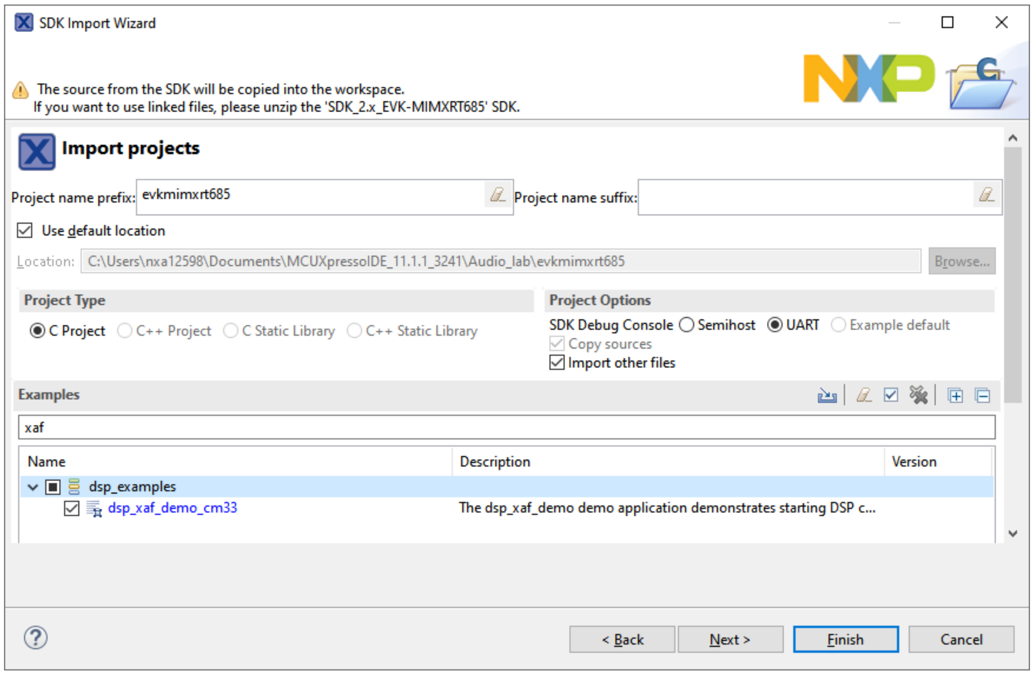
NXP MIMXRT685S
- MIMXRT500
- MIMXRT600
  - MIMXRT685S

Available boards

Please select an available board for your project.

Supported boards for device: MIMXRT685S

5. In **SDK Import Wizard** select the “dsp_xaf_demo_cm33” example and click **Finish**.


SDK Import Wizard

The source from the SDK will be copied into the workspace.
If you want to use linked files, please unzip the 'SDK_2.x_EVK-MIMXRT685' SDK.

Import projects

Project name prefix: evkmimxrt685 Project name suffix:

☑ Use default location

Location: C:\Users\nxa12598\Documents\MCUXpressoIDE_11.1.1_3241\Audio_lab\evkmimxrt685

Project Type: [✓ C Project] C++ Project C Static Library C++ Static Library

Project Options: SDK Debug Console Semihost [✓ UART] Example default
☑ Copy sources
☑ Import other files

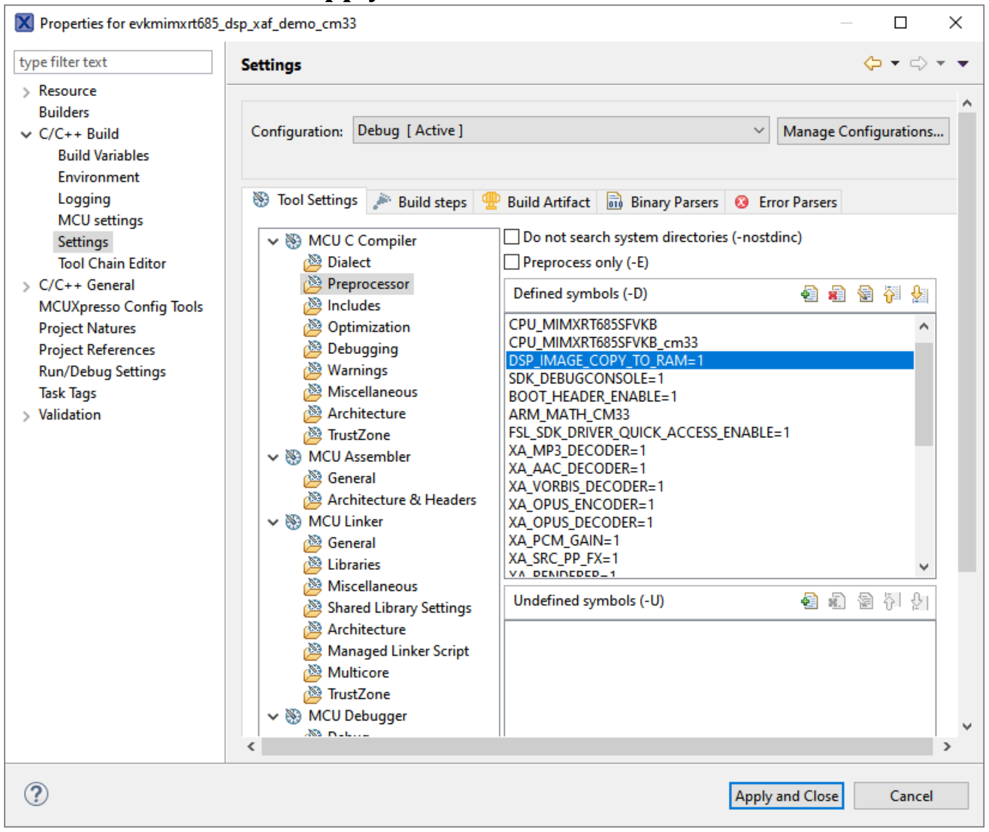
Examples

xaf

Name	Description	Version
dsp_examples		
☑ dsp_xaf_demo_cm33	The dsp_xaf_demo demo application demonstrates starting DSP c...	

6. The project “evkmimxrt685_dsp_xaf_demo_cm33” will appear in the Project Explorer window.

7. Open the project properties and change the symbol DSP_IMAGE_COPY_TO_RAM from 0 to **1** and click on **Apply and close**.



Configuration: Debug [Active]

Do not search system directories (-nostdinc)

Preprocess only (-E)

Defined symbols (-D)

```
CPU_MIMXRT685SFVKB
CPU_MIMXRT685SFVKB_cm33
DSP_IMAGE_COPY_TO_RAM=1
SDK_DEBUGCONSOLE=1
BOOT_HEADER_ENABLE=1
ARM_MATH_CM33
FSL_SDK_DRIVER_QUICK_ACCESS_ENABLE=1
XA_MP3_DECODER=1
XA_AAC_DECODER=1
XA_VORBIS_DECODER=1
XA_OPUS_ENCODER=1
XA_OPUS_DECODER=1
XA_PCM_GAIN=1
XA_SRC_PP_FX=1
```

With this change, the ARM build will use the prebuilt HiFi4 binaries and will load the image into RAM and will initialize the HiFi4 to run from the RAM image. For loading and debugging the HiFi4 image separately please refer to the "<RT600 SDK>\Getting Started with Xplorer for EVK-MIMXRT685.pdf" document.

8. Select your project and click on **Build** and wait for the build to finish.

9. Select your project and click on **Debug** to start the debug session.

10. Select the on-board debug probe and click **OK**.

11. The debug session will start. Click on the **Resume** button to start the application.

12. The XAF shell console will appear on the serial terminal and the user can interact with the different commands.

References

- “<RT600 SDK>\Getting Started with Xplorer for EVK-MIMXRT685.pdf”
- "<RT600 SDK>\middleware\dsp\audio_framework\libxa_af_hostless\doc\HiFi-AF-Hostless-ProgrammersGuide.pdf"