NXP Semiconductors
User's Guide

Document Number: DSPUG
Rev. 1, 01/2019

DSP User's Guide

1. Introduction

This document provides an overall introduction to the DSP including system architecture, file organization, DSP-related toolchain, and so on. This document helps with the overall understanding of the DSP-related code. Currently, the DSP is used to decode and encode audio streams on the i.MX8 QXP platform. The current DSP framework can support up to 64 clients. They support these codecs:

Decoder:

- AAC-LC
- AAC plus(HE-AAC/HE-AACv2)
- BSAC
- DAB+
- MP2
- MP3
- DRM
- SBC
- OGG

Encoder:

- SBC

Contents

1. Introduction 1
2. System architecture 2
3. File organization 3
3.1. DSP driver 3
3.2. DSP framework 3
3.3. DSP wrapper and unit test 3
3.4. Interface header files 4
4. Building DSP framework on Linux OS 4
4.1. Installing Xtensa development toolchain 4
4.2. Building DSP framework 5
5. Building DSP framework on Windows OS 5
5.1. Adding new configuration packages 5
5.2. Creating the DSP framework Xplorer project 7
5.3. Building DSP framework 9
6. Building DSP wrapper and unit test 11
6.1. Installing Linaro compiler toolchain 11
6.2. Building the code 11
7. Usage of DSP binary files 11
7.1. Getting DSP binary files 11
7.2. Binary files in Linux OS rootfs 12
7.3. Unit test and playing 12
8. Making codec wrapper and codec library 13
8.1. Finding custom LSPs 13
8.2. Source code modifying and compiling 13
8.3. Linking the library code 14
9. Revision history 14
Appendix A. Memory allocation for DSP 15

© 2019 NXP B.V.

2. System architecture

Figure 1 provides the overall system architecture of the DSP-related code.

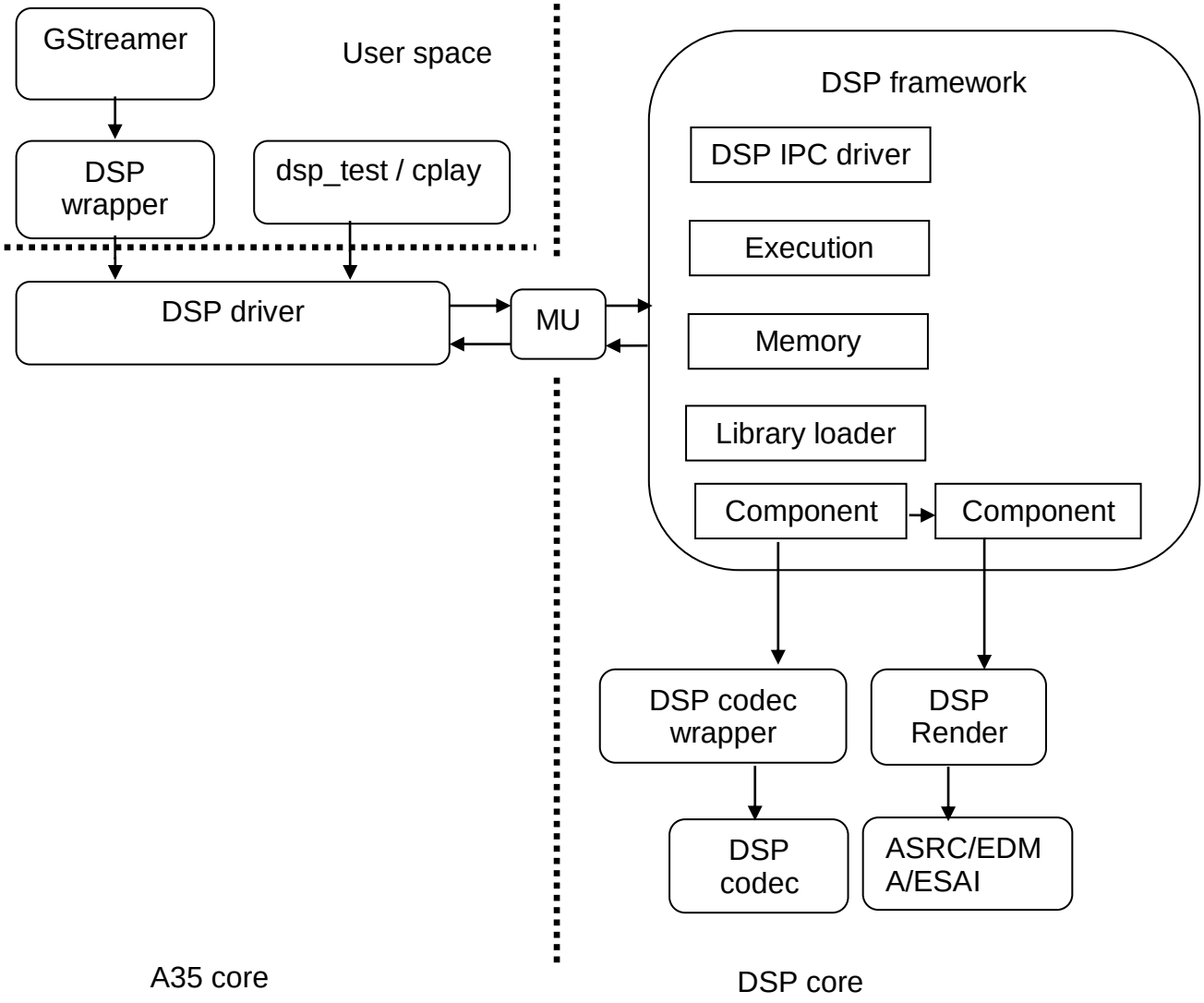

Figure 1. System architecture

The DSP-related code includes the DSP framework, DSP driver, DSP wrapper, unit test, DSP codec wrapper, and DSP codec. The DSP framework is a firmware code which runs on the DSP core. The DSP driver is used to load the DSP firmware into the memory and transfer messages between the user space and the DSP framework. A shared ring buffer is used to transfer messages between the A35 and the DSP core. The MU (Message Unit) is used to trigger interrupts between A35 and DSP core when messages are placed into the ring buffer. The DSP wrapper and the unit test are the application code in the user space, which uses the *ioctl()* interface to transfer messages between the DSP driver and the user space. In addition, the DSP wrapper is used to provide unified interfaces for the GStreamer. The DSP codec provides the actual decoding and encoding functions. The DSP codec wrapper is a wrapping code for the DSP codec and provides unified interfaces for the DSP framework. The DSP driver also offers a

compressed sound card interface to the user space applications. The DSP framework supports direct rendering of the decoded PCM data to an external audio device.

3. File organization

The DSP framework, DSP wrapper, and unit test code are in the *imx-audio-framework* package. The DSP driver code belongs to the Linux OS kernel. The DSP codec wrapper and DSP codec are license-restricted code; a license authorization is required to get them.

3.1. DSP driver

The DSP driver code is in the Linux OS kernel. It includes the following files:

- *linux-kernel/sound/soc/fsl/fsl_dsp.c*
- *linux-kernel/sound/soc/fsl/fsl_dsp.h*
- *linux-kernel/sound/soc/fsl/fsl_dsp_proxy.c*
- *linux-kernel/sound/soc/fsl/fsl_dsp_proxy.h*
- *linux-kernel/include/uapi/linux/mxc_dsp.h*
- *linux-kernel/sound/soc/fsl/fsl_dsp.c*
- *linux-kernel/sound/soc/fsl/fsl_dsp_cpu.c*
- *linux-kernel/sound/soc/fsl/fsl_dsp_cpu.h*
- *linux-kernel/sound/soc/fsl/fsl_dsp_pool.c*
- *linux-kernel/sound/soc/fsl/fsl_dsp_pool.h*
- *linux-kernel/sound/soc/fsl/fsl_dsp_library_load.c*
- *linux-kernel/sound/soc/fsl/fsl_dsp_library_load.h*
- *linux-kernel/sound/soc/fsl/fsl_dsp_xaf_api.c*
- *linux-kernel/sound/soc/fsl/fsl_dsp_xaf_api.h*
- *linux-kernel/sound/soc/fsl/fsl_dsp_platform_compress.c*
- *linux-kernel/sound/soc/fsl/fsl_dsp_platform.h*
- *linux-kernel/sound/soc/fsl/imx-dsp.c*

3.2. DSP framework

The DSP framework code is in this folder:

- *imx-audio-framework/dsp_framework*

3.3. DSP wrapper and unit test

The DSP wrapper and unit test are in these folders:

- *imx-audio-framework/dsp_wrapper*
- *imx-audio-framework/unit_test*

3.4. Interface header files

The DSP-related code includes these four interface header files:

- *imx-audio-framework/include/mxc_dsp.h*
- *imx-audio-framework/dsp_framework/plugins/audio_codec/dsp_codec_interface.h*
- *imx-audio-framework/dsp_wrapper/include/uni_audio/fsl_unia.h*
- *imx-audio-framework/dsp_wrapper/include/uni_audio/fsl_types.h*

The *mxc_dsp.h* file is the same as the header file in the Linux OS kernel. This file includes the interfaces and command definitions that are used by the DSP wrapper and unit test. The *dsp_codec_interface.h* file wraps the DSP codec's header files. It includes unified interfaces and command definitions which can be used by the DSP framework. The *fsl_unia.h* and *fsl_types.h* header files include the interfaces and command definitions which can be used by GStreamer.

4. Building DSP framework on Linux OS

Before you compile the DSP-related code, set up the DSP-related toolchains. The DSP framework, DSP codec wrapper, and DSP codec use Xtensa® development toolchain.

4.1. Installing Xtensa development toolchain

The Xtensa development toolchain consists of two components which are installed separately in the Linux OS, including:

- Configuration-independent Xtensa Tool
- Configuration-specific core files and Xtensa Tool

The configuration-independent Xtensa Tool is released by Cadence®. For the current code, the version of this tool is *XtensaTools_RF_2016_4_linux.tgz*. You may use two ways to get this package. The first is to download it from the Xtensa Xplorer™ and the second is to get it from other people who installed the Xtensa Xplorer.

The configuration-specific core files and the Xtensa Tool are released by NXP. The current version of this tool is *hifi4_nxp_v3_3_1_2_dev_linux.tgz*. It is recommended to get the configurable memory map linker files from NXP. These files are in the *memmap/mainsim* folder.

When you have these two components, you can set up the toolchain as follows:

- Open the *imx-audio-framework* folder and execute these commands:

```
mkdir -p ./imx-audio-toolchain/Xtensa_Tool/tools
mkdir -p ./imx-audio-toolchain/Xtensa_Tool/builds
```

- Set up the configuration-independent Xtensa Tool:

```
cd imx-audio-toolchain/Xtensa_Tool
tar zxvf XtensaTools_RF_2016_4_linux.tgz -C ./tools
```

- Set up the configuration-specific core files and the Xtensa Tool:

```
cd imx-audio-toolchain/Xtensa_Tool
tar zxvf hifi4_nxp_v3_3_1_2_dev_linux.tgz -C ./builds
```

- Copy the configurable memory map files to this path:

```
cd imx-audio-toolchain/Xtensa_Tool
cp -r memmap/mainsim ./builds/ RF-2016.4-linux/hifi4_nxp_v3_3_1_2_dev/xtensa-elf/lib
```

- Install the Xtensa development toolchain:

```
cd imx-audio-toolchain/Xtensa_Tool
./ builds/RF-2016.4-linux/hifi4_nxp_v3_3_1_2_dev/install --xtensa-tools ./tools/RF-2016.4-linux/XtensaTools --registry ./tools/RF-2016.4-linux/XtensaTools/config
```

- Set the PATH environment variable:

```
export PATH= ./imx-audio-toolchain/Xtensa_Tool/tools/RF-2016.4-linux/XtensaTools/bin:$PATH
```

- Set the LM_LICENSE_FILE environment variable.

The Xtensa development tools use FLEXlm for license management. The FLEXIm licensing is required for tools such as the Xtensa Xplorer, TIE Compiler, and Xtensa C and C++ compiler. If you want to use a floating license, install the FLEXIm license manager and set the LM_LICENSE_FILE environment variable. In case of any problems, you can find useful information in the *Xtensa Development Tools Installation Guide User's Guide.doc* document provided by Cadence.

After the above steps, the Xtensa development toolchain is set up successfully. In addition, the Xtensa Tools and additional tools are provided as 32-bit (x86) binaries. They are supported on 32-bit (x86) systems, and also on recent 64-bit (x86-64) systems that have appropriate 32-bit compatibility packages installed. If you use a 64-bit system (for example; Ubuntu 16.04), install the 32-bit compatibility packages first. Use these commands:

```
sudo apt-get install lib32ncurses5 lib32z1
sudo dpkg --add-architecture i386
sudo apt-get install libc6:i386 libstdc++6:i386
```

4.2. Building DSP framework

After installing the DSP-related toolchains on your Linux OS server, you can compile the DSP framework. Execute the "make" command in the *imx-audio-framework* folder to compile the DSP framework. This way also builds the DSP wrapper and unit test. If you want to compile the DSP framework separately, see the REAME file in the *imx-audio-framework* folder. After the compiling process, you can find the binary files in the *imx-audio-framework/release* folder.

For the DSP framework:

- *imx-audio-framework/release/hifi4.bin*

5. Building DSP framework on Windows OS

The DSP framework can be built also on Windows OS. The Xplorer software can be used to build the DSP framework on Windows OS. This chapter explains how to use Xplorer to build the DSP framework. It is assumed that you have installed the Xplorer software on your Windows system and you acquired the Xplorer licence from Cadence. The Xplorer 7.0.8 version is used as an example and its default installing folder is *C:\usr\xtensa*.

5.1. Adding new configuration packages

Currently, the *hifi4_nxp_v3_3_1_2_dev_win32.tgz* configuration package is used to build the DSP framework on Linux OS, so you shall add this configuration package into Xplorer before building the code. You can get this configuration package and the corresponding memory map linker files from NXP. The required files are as following:

- *hifi4_nxp_v3_3_1_2_dev_win32.tgz*
- *memmap/mainsim folder*

When you have the DSP configuration package and memory map linker files, you can add a new configuration package into Xplorer as follows:

- Download and install Xtensa Tools for Xplorer.

  If you do not have the Xtensa Tools, you shall download and install it using Xplorer. Currently, the Xtensa Tool that we use is *XtensaTools_RF_2016_4_win32.tgz*. You can first open the Xplorer software and click the "RF-2016.4" option in the "XPG View" panel and select the "tools->Xtensa Tools->Xtensa Tools 11.04 for Windows" option. After you select it, you can click the download button to start the downloading process.

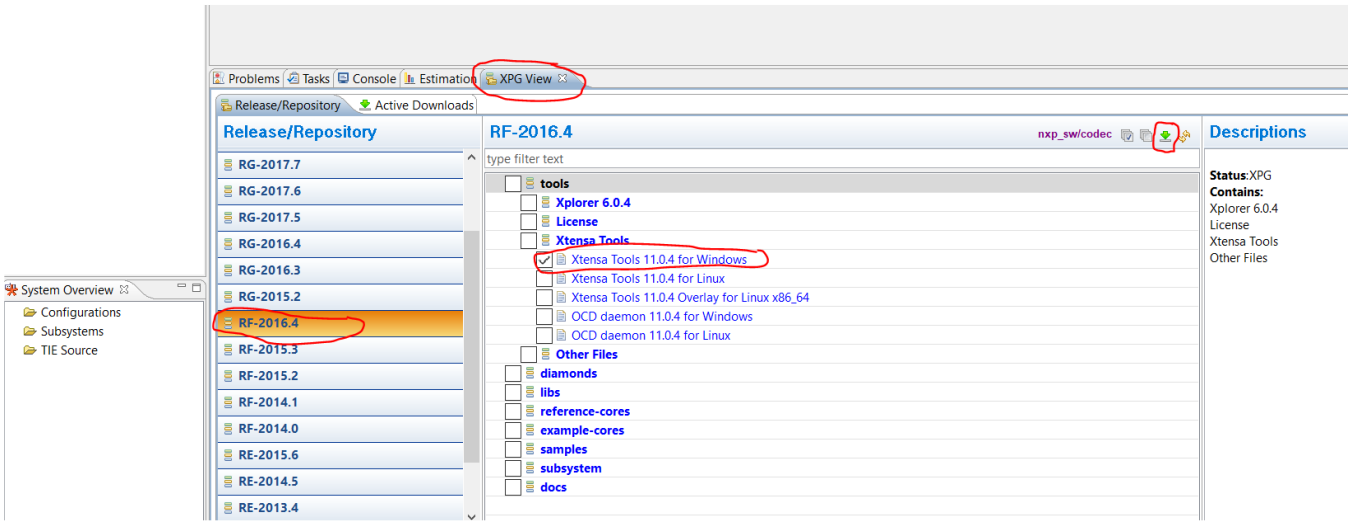

Figure 2. "XPG View" panel

  After the download finishes, right click the "Xtensa Tools 11.0.4 for Windows" option and select the "Install Xtensa Tools…" option in the new dialog. The installing process takes some time. The Xtensa Tool is installed successfully after this step. You can see this folder in the Xplorer's installing folder if everything is ok:

  C:\usr\xtensa\XtDevTools\install\tools\RF-2016.4-win32

- Add the configuration package into Xplorer.

  When you have the *hifi4_nxp_v3_3_1_2_dev_win32.tgz* package from NXP, you can add it into Xplorer. The first thing to do is to create a new folder called *build* in Xplorer's installing path if the *build* folder is not created already. The total path after this operation is as follows:

  C:\usr\xtensa\XtDevTools\downloads\RF-2016.4\build

- Place the *hifi4_nxp_v3_3_1_2_dev_win32.tgz* package into the new *build* folder.

  C:\usr\xtensa\XtDevTools\downloads\RF-2016.4\build\ hifi4_nxp_v3_3_1_2_dev_win32.tgz

- After you have performed the above steps, you can click the refresh button in the "XPG View" panel and find the "build" option in this panel.

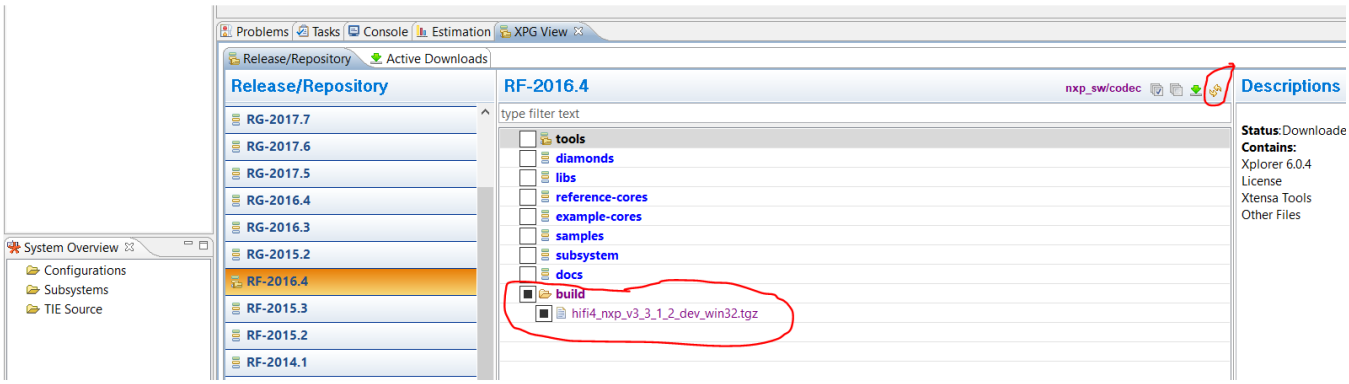

Figure 3. The "build" option

- Right click the *build->hifi4_nxp_v3_3_1_2_dev_win32.tgz* package and click the "Install Build…" option in the new dialog to start the installing process. This takes some time. You can see the following folder in the Xplorer's installing folder if everything is OK.

  C:\usr\xtensa\XtDevTools\install\builds\RF-2016.4-win32\hifi4_nxp_v3_3_1_2_dev

- Add the new memmap linker files into Xplorer.

  After you add the *hifi4_nxp_v3_3_1_2_dev_win32.tgz* configuration package into Xplorer, you can add the new memmap linker files. You can copy the *mainsim* folder into Xplorer's install folder to finish this process. The complete folder after this process is as follows:

  C:\usr\xtensa\XtDevTools\install\builds\RF-2016.4-win32\hifi4_nxp_v3_3_1_2_dev\xtensa-elf\lib\mainsim

After you complete the above three steps, the new configuration package and the corresponding memory map linker files are successfully added into Xplorer.

5.2. Creating the DSP framework Xplorer project

The DSP framework project must be created before using Xplorer to build it. The DSP framework code is in the *imx-audio-framework* package:

- *imx-audio-framework\dsp_framework*

You can create the DSP framework as follows:

- Open Xplorer and click the "File->New->Xtensa C/C++ project" option in the menu bar. You will see this dialog:

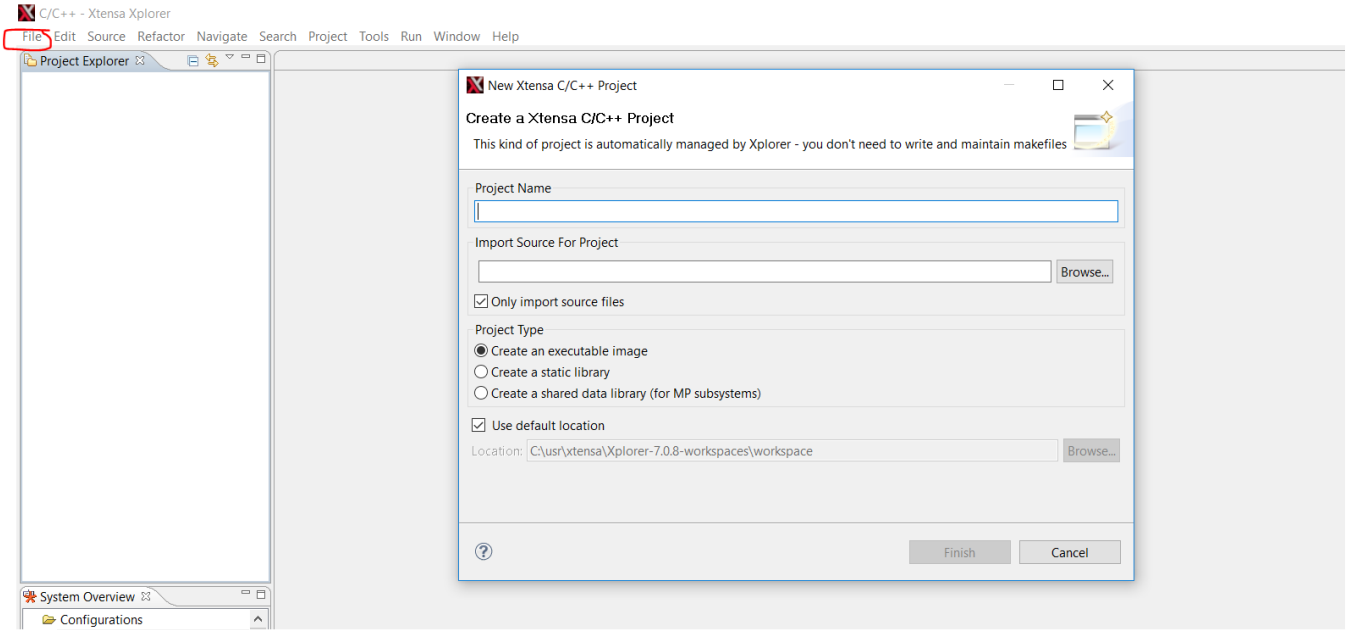

Figure 4. New Xtensa project

- Enter the project name and import DSP framework source code into the "New Xtensa C/C++ Project" dialog. Then click the "Finish" button.

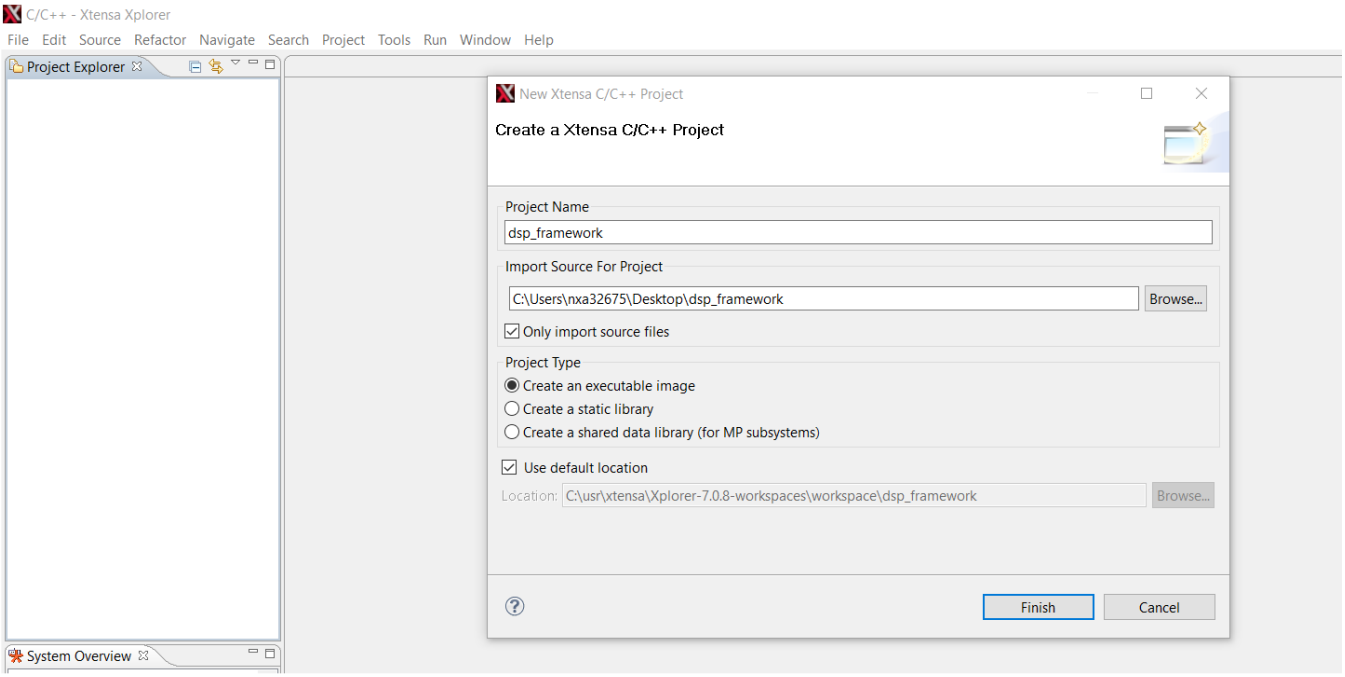

Figure 5. The "Finish" button

After the above two steps, the DSP framework project is successfully created. You can see the project in Figure 6.

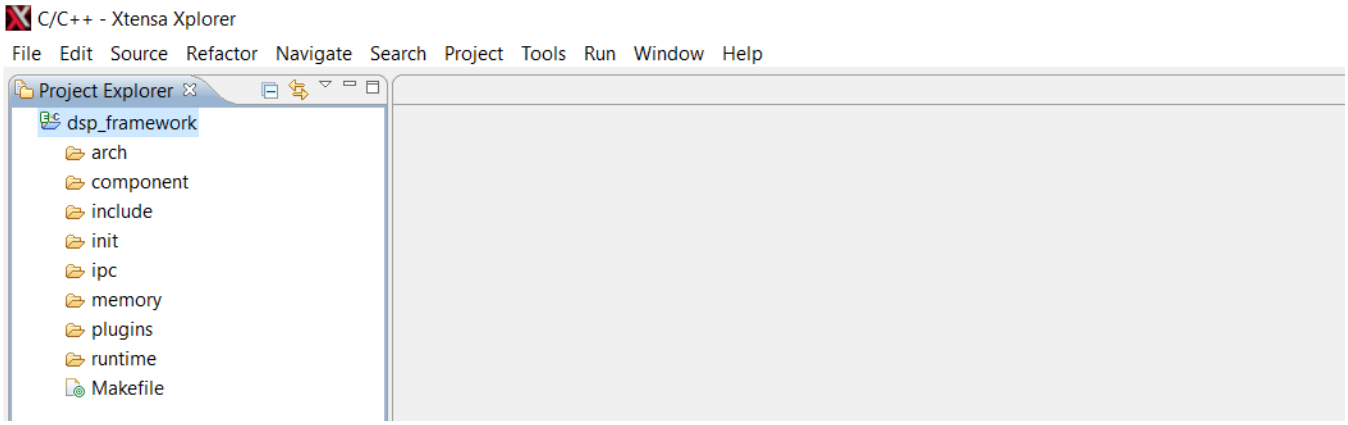

Figure 6. DSP framework project

5.3. Building DSP framework

When you created the DSP framework project, you can build its code. Choose the *memmap* linker files before the building process.

- Right-click the name of the DSP framework project in the "Project Explorer" panel and choose the "Build Properties…" option. You will see the "Build Properties for dsp_framework" dialog. The dialog is shown in Figure 7.

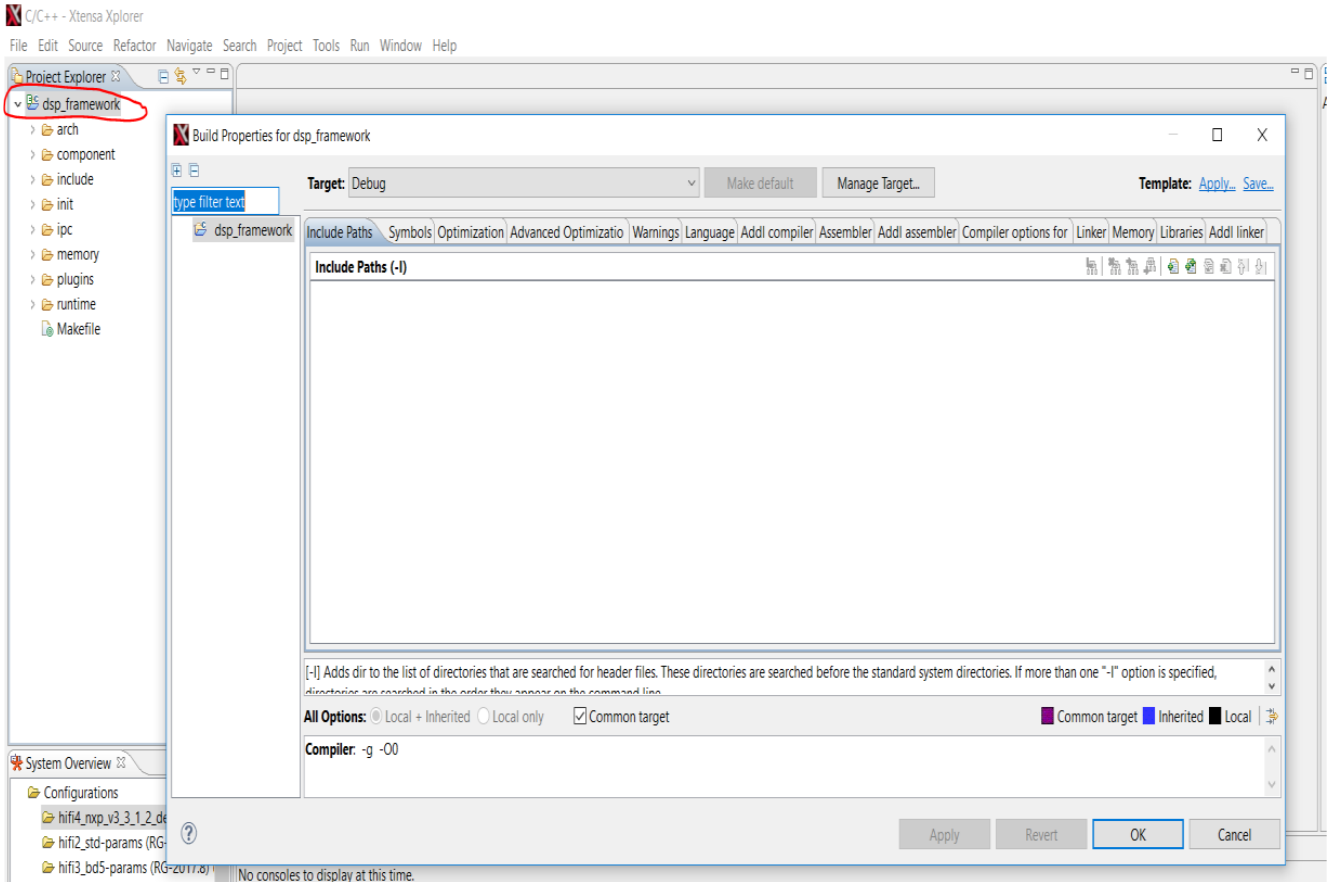

Figure 7. "Build Properties" dialog

- Click the “Linker” option and configure the custom LSP path as shown in Figure 8. Click the “OK” button to finish this process.

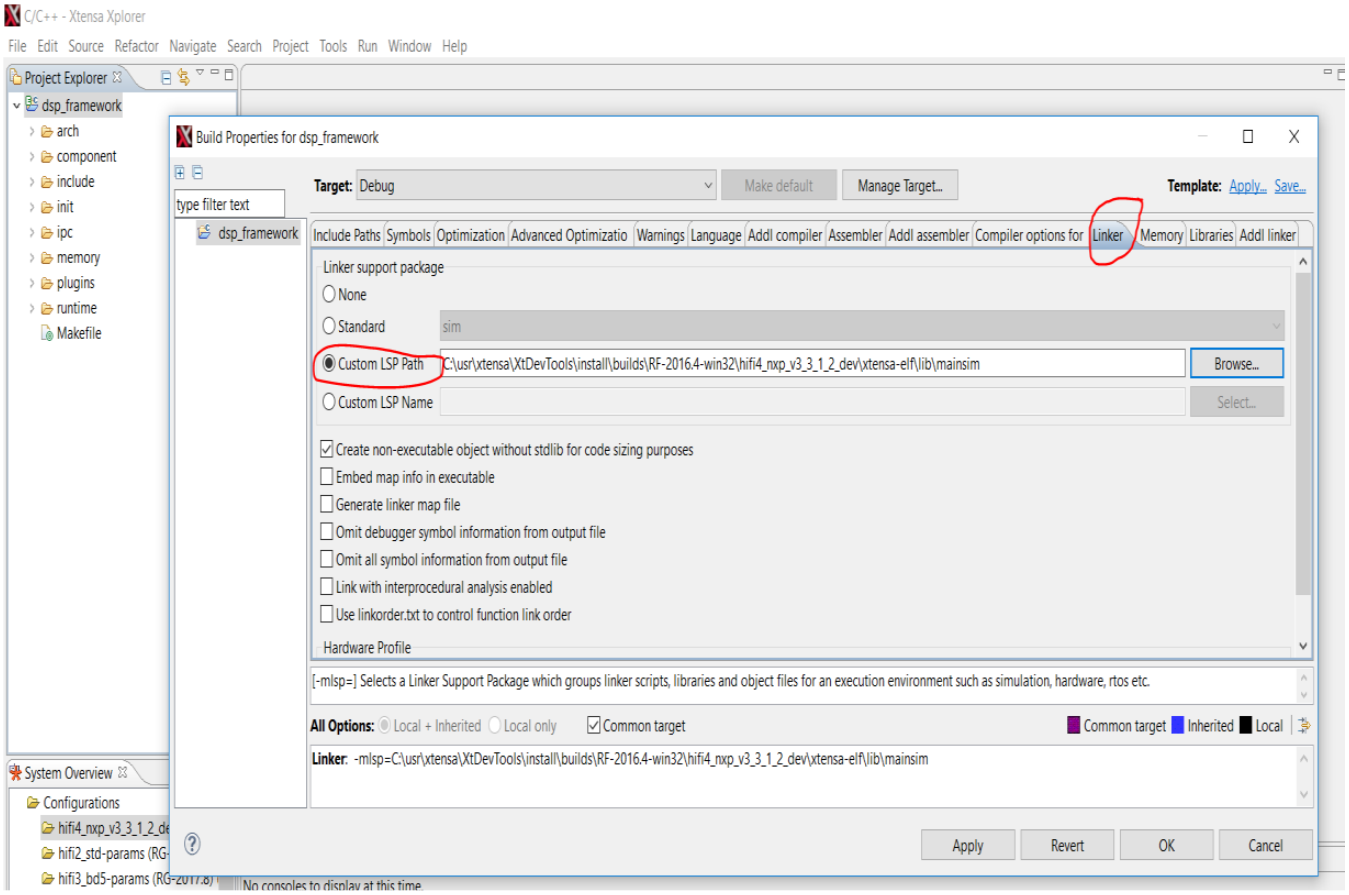

Figure 8. “Linker” option

- When you configured the *memmap* linker files, you can choose the *dsp_framework* project and the required DSP configuration to start the building process. The configuration is as follows:

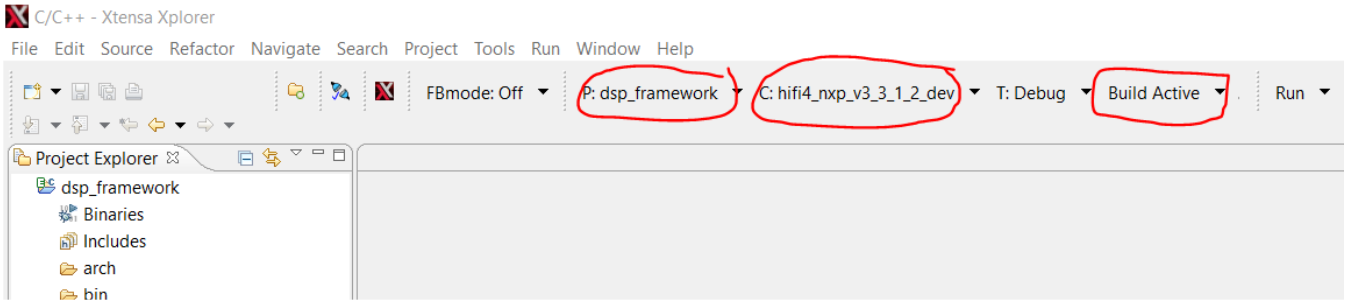

Figure 9. Build process configuration

- Click the “Build Active->Build Active” option to start building the DSP framework. This takes some time.

- After performing the above steps, you get the binary file called *dsp_framework* (which is the firmware of the DSP) in the following folder.

  C:\usr\xtensa\Xplorer-7.0.8-workspaces\workspace\dsp_framework\bin\hifi4_nxp_v3_3_1_2_dev\Debug\dsp_framework

If you want to use this binary file to run on a real board, rename the *dsp_framework* binary file as *hifi4.bin* and place it to a right place of "rootfs".

6. Building DSP wrapper and unit test

Before you compile the DSP wrapper and the unit test, set up the related toolchain. The DSP wrapper and the unit test use the Linaro compiler toolchain for the Yocto platform.

6.1. Installing Linaro compiler toolchain

Currently, the "gcc-linaro-4.9-2015.02-3-x86_64_aarch64-linux-gnu" toolchain is used to compile the DSP wrapper and the unit test's code for the Yocto platform. This toolchain shall be placed into the */usr* folder of your Linux OS server. If you want to successfully build the code, you can get more information from the *Makefile* file of the DSP wrapper and the unit test.

6.2. Building the code

When the Linaro toolchain is successfully installed on your server, you can compile the DSP-related code. You can execute the "make" command in the *imx-audio-framework* folder to compile the DSP wrapper and the unit test. If you want to compile them separately, see the *README* file in the *imx-audio-framework* folder. After the compiling process, you can find the binary files in the *imx-audio-framework/release* folder.

For the DSP wrapper:

- *imx-audio-framework/release/wrapper/lib_dsp_wrap_arm_elinux.so*

For the unit test:

- *imx-audio-framework/release/exe/dsp_test*

7. Usage of DSP binary files

7.1. Getting DSP binary files

You can get the DSP binary files of the DSP framework, DSP wrapper, and unit test directly from NXP or compile the source code to produce them yourself. Authorization is needed for the DSP codec wrapper and DSP codec binary files.

7.2. Binary files in Linux OS rootfs

To run these binary files, place them into the Linux OS rootfs. The location of the DSP framework is determined by the DSP driver, so you shall keep it in the specified place. The location of the DSP wrapper is determined by the GStreamer and you shall keep it in the specified place. You can change the location of the unit test. The binary files are in these folders:

- The unit test is here (default path):
  /unit_tests/DSP/dsp_test.out
- The DSP framework is here:
  /lib/firmware/imx/dsp/hifi4.bin
- The DSP wrapper is here:
  /usr/lib/imx-mm/audio-codec/wrap/lib_dsp_wrap_arm_elinux.so
- You can keep the DSP codec wrapper and the DSP codec in these folders of the Linux OS rootfs:
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_codec_wrap.so
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_mp3_dec.so
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_aac_dec.so
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_bsac_dec.so
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_dabplus_dec.so
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_drm_dec.so
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_mp2_dec.so
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_sbc_dec.so
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_sbc_enc.so
- Add DSP codec wrapper:
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_codec_wrap_ext.so
- Add DSP codec:
  /usr/lib/imx-mm/audio-codec/dsp/lib_dsp_ogg_dec.so
- cplay utility (used to play compressed audio files):
  /usr/bin/cplay

7.3. Unit test and playing

7.3.1. dsp_test

After placing the binary files into the correct location of the rootfs, you can decode or encode audio streams directly using the unit test binary file. To decode one **.mp3* file, use this command:

```
./dsp_test -f1 -d16 -itest.mp3 -otest.pcm
```

For more information about the *dsp_test*, use this command:

```
./dsp_test
```

To play one music file using the GStreamer and DSP wrapper, use this command:

```
gplay-1.0 test.mp3
```

7.3.2. cplay

cplay is a standard Linux utility used to play compressed audio files over compressed audio interfaces. Currently DSP framework only supports playing mp3 files.

```
ls /snd/comprC1D0
cplay -c 1 -d 0 test.mp3
```

8. Making codec wrapper and codec library

The library of the DSP codec wrapper and DSP codec is the loadable library. This chapter describes how to make the loadable library for the DSP.

The DSP loadable library is available as two different types: a fixed-location overlay and a position-independent library. For a fixed-location overlay, you can load the code into a predetermined location in the memory. For a position-independent library, you can load the code at an address determined during run time. You can link the loadable library using a special LSP named “piload” or “pisplitload” (see the *Xtensa Linker Support Packages (LSPs) Reference Manual*). The binary files that are used by the DSP framework belong to the position-independent library, so this chapter briefly discusses how to generate the position-independent library. For more detailed information, see Chapter 4 of the *Xtensa System Software Reference Manual*.

A position-independent library can be loaded and run at any address that supports both code and data, like a normal system RAM. Alternatively, you can use the “pisplitload” LSP to load the code and data into separate memory blocks located in local RAMs. The library location must be decided before the run time.

The Xtensa development toolchain must be installed before making a loadable library. After that, you can follow the steps below.

8.1. Finding custom LSPs

The loadable libraries must be linked to a custom linker support package. For the position-independent libraries, you don’t have to generate or edit an LSP. Instead, you must link your position-independent library using the standard “pisplitload” LSP that is provided as a part of your configuration.

8.2. Source code modifying and compiling

The API only allows the main program to directly access a single symbol in the library, the “_start” symbol. The library cannot access any symbols in the main program directly. Any other symbol’s address must be passed to or from the library as an argument to the “_start” function. This code is an example:

```
#include <stdio.h>
/* declare a printf function pointer */
int (*printf_ptr)(const char *format, …);
/* replace all calls to printf with calls through the pointer */
#define printf printf_ptr
/* This is the function provided by the library */
```

```
char * interface_func(unsigned int input)
{
       printf("executing function interface_func\n"); 13

       return "this is string returned from interface_func";
}
void * _start(int (*printf_func)(const char *format, …))
{
       printf_ptr = printf_func;
       /* The main application wants to call the function interface_func, but can't
       directly reference it. Therefore, this function returns a pointer to it, and
       the main application will be able to call it via this pointer. */
       return interface_func;
}
```

The main application calls the "_start" function, passes a pointer to "printf", and takes a pointer to *interface_func()* in return. If the library and the main program must communicate a value of more than one symbol, then the "_start" function call can return arrays of pointers, rather than just single pointers.

After finishing your source code, you can use "xt-xcc" of the Xtensa development toolchain to compile the code. Because the position-independent libraries can be loaded at any address, make sure that the code in the library is position-independent using the "-fpic" flag along with your normal compile options, as shown here:

```
xt-xcc -O3 -o library.o -c library.c
```

8.3. Linking the library code

In this step, link the library code into a loadable library using the appropriate LSP. For position-independent library, you can use this command:

```
xt-xcc -mlsp=pisplitload -Wl,--shared-pagesize=128 -Wl,-pie -lgcc -lc -o library.so
library.o
```

After this command, you can get a position-independent library with the code and data loadable separately. If you want to get a contiguous position-independent library, you can use this command.

```
xt-xcc -mlsp=piload -Wl,--shared-pagesize=128 -Wl,-pie -lgcc -lc -o library.so
library.o
```

After the linking stage, you can get a loadable library which can be loaded by the DSP framework. The current DSP framework only supports loading the code and data sections separately.

9. Revision history

Table summarizes the changes done to this document since the initial release.

Table 1. Revision history

Revision number	Date	Substantive changes
0	06/2018	Initial release
1	01/2019	Added details about using the sound card feature that allows users to play mp3 files over ALSA compressed interface.

Appendix A. Memory allocation for DSP

The DSP firmware is loaded into the memory by the DSP driver. The loading address is defined by the memory map linker files of the Xtensa development toolchain. You may change the loading address based on the memory map list of i.MX8 QXP, shown in Table 2.

Table 2. Memory allocation

Cortex-A35/Cortex-M4	DSP	Content
—	0x80000000 ~ 0x806FFFFF	Reserved (cannot be used)
0x59700000 ~ 0x5971FFFF	0x80700000 ~ 0x8071FFFF	DSP OCRAM-system RAM
0x59720000 ~ 0x5973FFFF	0x80720000 ~ 0x8073FFFF	DSP OCRAM-system ROM
—	0x80740000 ~ 0x80FFFFFF	Reserved (cannot be used)
0x80700000 ~ 0x8073FFFF	—	Linux OS kernel (not visible from DSP)
0x81000000 ~ 0x9FFFFFFF	0x81000000 ~ 0x9FFFFFFF	SDRAM

Currently, the Linux OS kernel reserves the memory for the DSP in the SDRAM separately. The range of the reserved memory is 0x92400000 ~ 0x943fffff (32 MB). You may set this reserved memory by changing the *fsl-imx8qxp.dtsi* file in the *linux-kernel/arch/arm64/boot/dts/freescale* folder.

```
reserved-memory {
        …….
        dsp_reserved: dsp@0x92400000 {
                no-map;
                reg = <0 0x92400000 0 0x2000000>;
        };
        ……..….
}
```

The DSP driver splits the current reserved memory into two parts. One part is used to store the DSP firmware and the other part is a scratch memory for the DSP framework. The detailed information about these two parts is shown in Table 3.

Table 3. Two memory parts

0x92400000 ~ 0x933FFFFF	DSP firmware (16 MB)
0x93400000 ~ 0x943FFFFF	Scratch memory (16 MB)

NOTE

If you make changes in the memory map linker files of the Xtensa development toolchain, make the related changes for the DSP driver.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C 5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, AMBA, Arm Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and µVision are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. Arm7, Arm9, Arm11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, Mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2019 NXP B.V.

Document Number: DSPUG
Rev. 1
01/2019