GSXMIMXRT600UG
Getting Started with Xplorer for MIMXRT600

Rev. 1 — 10 July 2021 User Guide

Contents

1 Overview

Cadence® Tensilica® Xplorer is a complete development environment that helps users create application code for high-performance Tensilica® processors. Xplorer is the interface to powerful software development tools such as the XCC compiler, assembler, linker, debugger, code profiler, and full set of GUI tools.

Xplorer, including both GUI and command-line environment, is the only available development IDE for the DSP core of MIMXRT600.

2 Install Xplorer Toolchains

This section provides information on Xtensa Software Tools Platform Supportand the steps to Install the Xtensa Xplorer IDE and Tools, Install License Key, Install RT600 DSP Build Configuration, Install Xtensa On Chip Debugger Daemon, Program LPC-Link2 with SEGGER J-Link, and Install Xtensa Software Tools without IDE.

2.1 Xtensa Software Tools Platform Support

The Xtensa Software Tools are officially supported on the following platforms:

Windows: Win 10 64-bit, Win 8 64-bit, Win 7 64-bit.

Linux: RHEL 6 64-bit (with ‘Desktop’ package installed).

There may be compatibility issues with other versions of Linux or Windows, especially when using the IDE.

NOTE

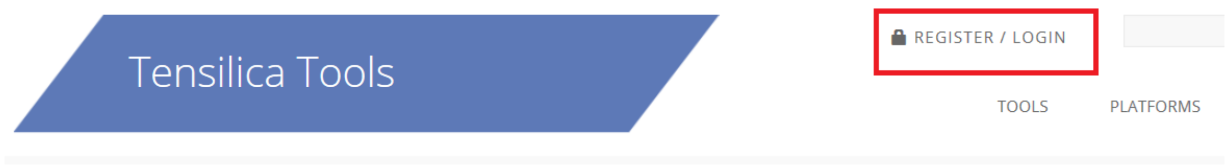
The security-enhanced Linux (SELinux) is not a supported platform because the OS can prevent different shared libraries (including Xtensa Tools) from loading.

For more information on platform support and installation guidelines, see the Xtensa Development Tools Installation Guide.

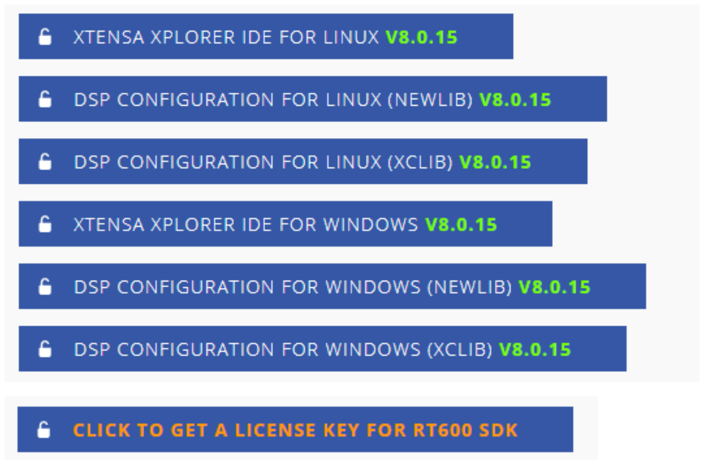
2.2 Install the Xtensa Xplorer IDE and Tools

To install the Xtends Xplorer IDE and tools, perform the following steps:

1. Go to the URL https://tensilicatools.com/download/rt600-download-page/ and login. If you are accessing the site for the first time, make sure to register.
2. Make sure to use your corporate email address to register.



3. Once registered you should receive an email confirmation with an activation link from ‘Tensilica Tools’ no-reply@tensilicatools.com. Make sure to check the spam folder if this email does not show up in the inbox.
4. To complete the registration, click the activation link.
5. Once registered, log in to see the material available for download:


XTENSA XPLORER IDE FOR LINUX V8.0.15
DSP CONFIGURATION FOR LINUX (NEWLIB) V8.0.15
DSP CONFIGURATION FOR LINUX (XCLIB) V8.0.15
XTENSA XPLORER IDE FOR WINDOWS V8.0.15
DSP CONFIGURATION FOR WINDOWS (NEWLIB) V8.0.15
DSP CONFIGURATION FOR WINDOWS (XCLIB) V8.0.15
CLICK TO GET A LICENSE KEY FOR RT600 SDK

- Download and install the **Xplorer IDE 8.0.15** for your operating system (Windows or Linux).
- Download the **DSP Configuration** for your operating system – installed later through the IDE, see Install RT600 DSP Build Configuration.

NOTE
NXP recommends version **8.0.15** of the Xtensa Xplorer IDE and tools for use with the RT600 DSP.

2.3 Install License Key

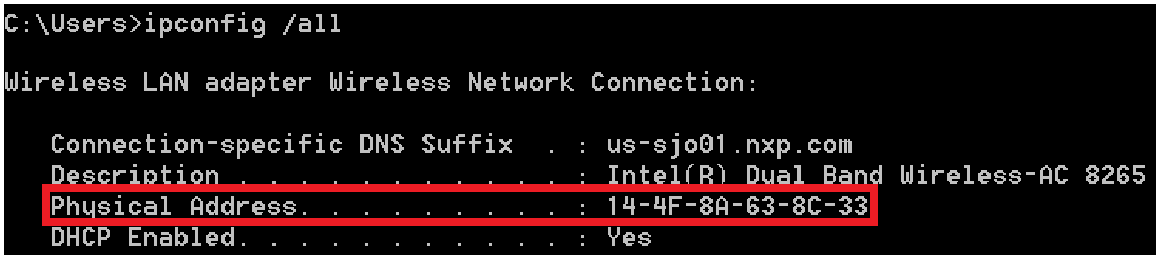
Xtensa development tools use FLEXlm for license management. FLEXlm licensing is required for tools such as the Xtensa Xplorer IDE, Xtensa C and C++ compiler, and Instruction Set Simulator (ISS).

Currently RT600 supports node-locked license for Xtensa tools. A node-locked license permits tools to run on a specific computer, tied to the MAC address of the primary network interface permanently attached to the machine.

2.3.1 Identify PC MAC Address

To generate the correct license file, you should first identify the appropriate MAC for the computer you plan to run Xtensa tools on. Remove ‘-‘ or ‘:’ symbols in the MAC address.

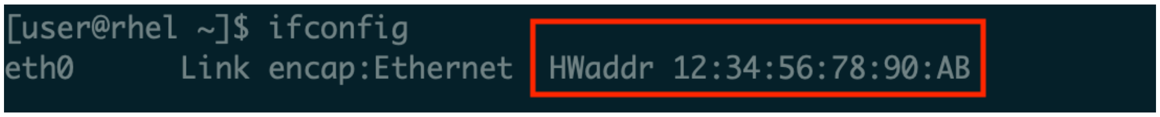
Windows:



```
C:\Users>ipconfig /all

Wireless LAN adapter Wireless Network Connection:

   Connection-specific DNS Suffix  . : us-sjo01.nxp.com
   Description . . . . . . . . . . . : Intel(R) Dual Band Wireless-AC 8265
   Physical Address. . . . . . . . . : 14-4F-8A-63-8C-33
   DHCP Enabled. . . . . . . . . . . : Yes
```

Figure 1. Physical Address for Windows

Linux:



```
[user@rhel ~]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 12:34:56:78:90:AB
```

Figure 2. MAC address

NOTE

For Linux: MAC address must be associated with eth0 interface. If not, Flexlm cannot perform the license checkout, compilation, or simulation of code is not possible. If the host has the MAC address associated with another interface, for example em1, the following approach may be used.

```
# Add udev rule for naming interface
$ sudo vim /etc/udev/rules.d/70-persistent-net.rules

# udev rule (replace 'XX' with the MAC address of your PC):
SUBSYSTEM=="net", ACTION=="add",
ATTR{address}=="XX:XX:XX:XX:XX:XX", NAME="eth0"

# Change "em1" to "eth0" in your interfaces file.
$ sudo vim /etc/network/interfaces

# Restart udev or reboot machine
$ sudo reboot
```

Alternatively, you can use the approach recommended by your IT team to rename the interface to eth0.

2.3.2 Download License Key

To download the license key:

1. Reload or return to the Tensilica URL: https://tensilicatools.com/download/rt600-download-page/.
2. Click the button and get the license key for RT600 SDK.

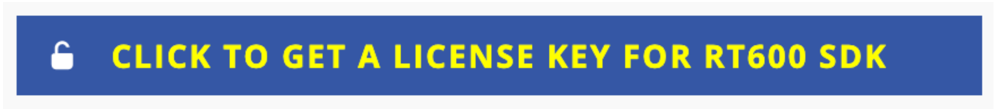
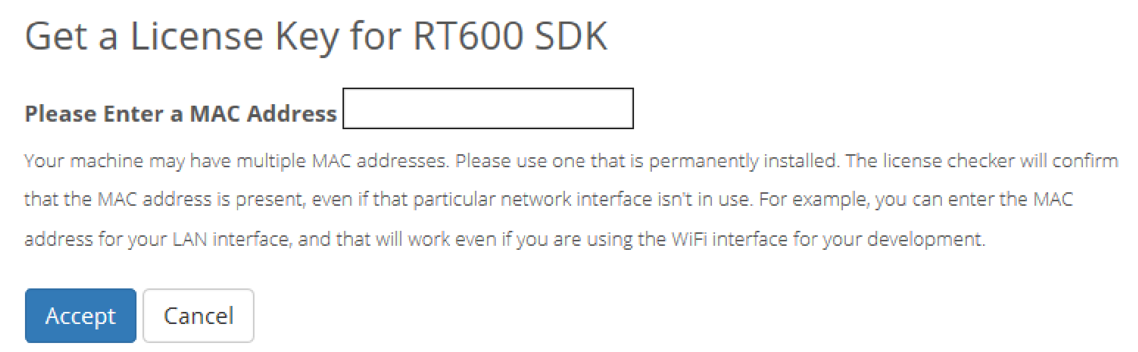


Figure 3. Get a license key for RT600 SDK

3. Provide the address.



Get a License Key for RT600 SDK

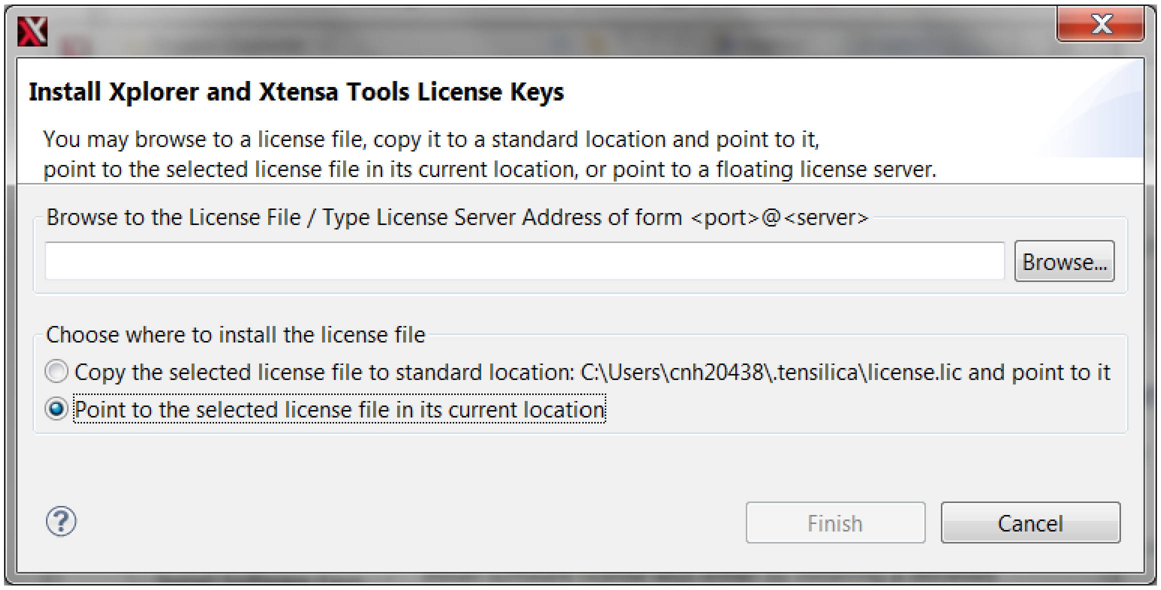
Please Enter a MAC Address

Your machine may have multiple MAC addresses. Please use one that is permanently installed. The license checker will confirm that the MAC address is present, even if that particular network interface isn't in use. For example, you can enter the MAC address for your LAN interface, and that will work even if you are using the WiFi interface for your development.

Figure 4. Provide the address

4. Once the license file has been generated and downloaded, open your recently installed Xplorer 8.0.15.

5. Select menu **Help > Xplorer License Keys > Install Software Keys**.
6. Select the license key file.



Install Xplorer and Xtensa Tools License Keys

You may browse to a license file, copy it to a standard location and point to it, point to the selected license file in its current location, or point to a floating license server.

Browse to the License File / Type License Server Address of form <port>@<server>

Choose where to install the license file

○ Copy the selected license file to standard location: C:\Users\cnh20438\.tensilica\license.lic and point to it

● Point to the selected license file in its current location

Figure 5. Select the license key

7. Click the **Finish** button.

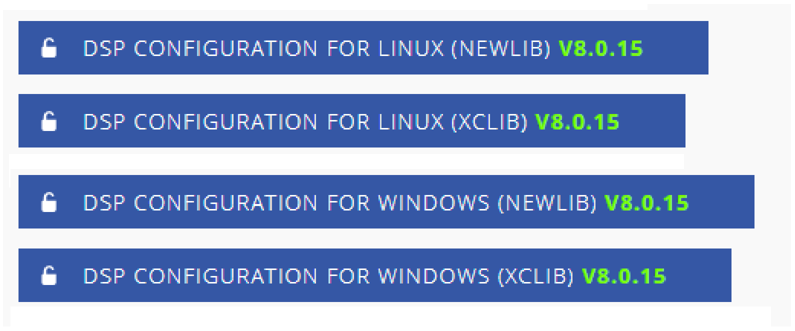
NOTE

The generated license file only supports debug/run on the RT600 device target. It does not support software simulation/Xplorer ISS. If you have special requirements to run the software simulations, contact Cadence directly.

2.4 Install RT600 DSP Build Configuration

'*Build Configuration*' is a term that describes all parameters and necessary build includes for the Tensilica processor implementation for development. It is mandatory to install a specific build configuration before starting development on RT600.

The build configuration is provided by NXP as a binary file and is imported into the Xplorer IDE. The binary file for the OS is available at the Tensilica URL: https://tensilicatools.com/download/rt600-download-page/.



DSP CONFIGURATION FOR LINUX (NEWLIB) V8.0.15

DSP CONFIGURATION FOR LINUX (XCLIB) V8.0.15

DSP CONFIGURATION FOR WINDOWS (NEWLIB) V8.0.15

DSP CONFIGURATION FOR WINDOWS (XCLIB) V8.0.15

Figure 6. Binary files for OS

The build configuration is installed into the IDE using the **System Overview** panel available in the lower left corner, by default. If the **System Overview** panel is not visible, toggle using the **Window > Show View > System Overview** menu item.

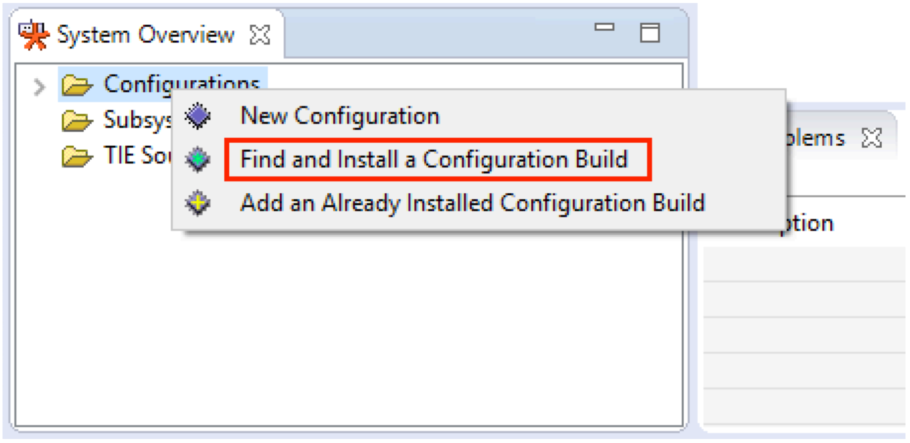
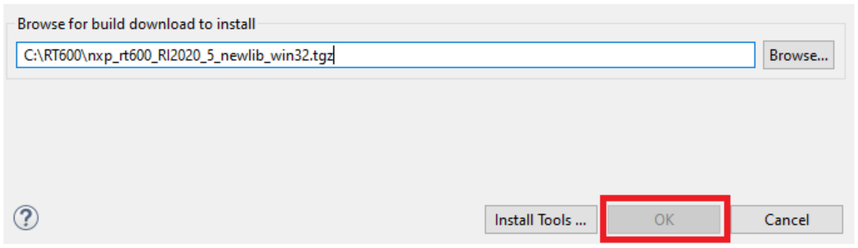

System Overview
Configurations
Subsys
TIE Sou
New Configuration
Find and Install a Configuration Build
Add an Already Installed Configuration Build
blems
ption

Figure 7. System Overview panel

Click **OK** for build download to install.



```
C:\RT600\nxp_rt600_RI2020_5_newlib_win32.tgz
```

2.5 Install Xtensa On Chip Debugger Daemon

The Xtensa On Chip Debugger Daemon (xt-ocd), is a gdb-based debugging tool but is not installed by default with the Xplorer IDE. For installation, a self-extracting executable installer is included with the IDE, which can be found at the following location:

Windows:

```
C:\usr\xtensa\XtDevTools\downloads\RI2020.5\tools\xt-ocd-14.05-windows64-installer.exe
```

Linux:

```
~/xtensa/XtDevTools/downloads/RI2020.5/tools/xt-ocd-14.05-linux64-installer
```

Currently, xt-ocd supports J-Link and Arm® RVI/DSTREAM probes over Serial Wire Debug (SWD) for RT600. xt-ocd installs support for J-Link probes but does not install the required J-Link drivers which must be installed separately. Make sure that the latest version of J-Link software is installed.

NOTE

For Linux: When installing xt-ocd on Linux, ensure that the symlink is manually added to the installed J-Link driver: `ln -s <jlink-install-dir>libjlinkarm.so.6 <xocd-install-dir>/modules/libjlinkarm.so.6`.

xt-ocd is configured with an XML input file ‘topology.xml’ that modifies to fit the debugger hardware. Using J-link as an example, use the content below to replace the original template.

NOTE

It is a must to replace the ‘usbser’ section with the 9-digits number J-Link serial number on the back of the J-Link hardware in use.

```
<configuration>
  <controller id='Controller0' module='jlink' usbser='600100000' type='swd' speed='1000000'
locking='1'/>
  <driver id='XtensaDriver0' dap='1' xdm-id='12' module='xtensa' step-intr='mask,stepover,setps' />
  <chain controller='Controller0'>
    <tap id='TAP0' irwidth='4' />
  </chain>
  <system module='jtag'>
    <component id='Component0' tap='TAP0' config='trax' />
  </system>
  <device id='Xtensa0' component='Component0' driver='XtensaDriver0' ap-sel='3' />
  <application id='GDBStub' module='gdbstub' port='20000' sys-reset='0'>
    <target device='Xtensa0' />
  </application>
</configuration>
```

Below is another topology.xml example if Arm RealView ICE (RVI) and DSTREAM debug probes is used.

```
<configuration>
  <controller id='Controller0' module='rvi' />
  <driver id='XtensaDriver0' debug='' inst-verify='mem' module='xtensa' step-
intr='mask,stepover,setps'/>
  <driver id='TraxDriver0'   module='trax' />
  <chain controller='Controller0'>
    <tap id='TAP0' irwidth='4' />
  </chain>
  <system module='jtag'>
    <component id='Component0' tap='TAP0' config='trax' />
  </system>
  <device id='Xtensa0' component='Component0' driver='XtensaDriver0' xdm-id='12' />
  <device id='Trax0'   component='Component0' driver='TraxDriver0' xdm-id='12' />
  <application id='GDBStub' module='gdbstub' port='20000' >
    <target device='Xtensa0' />
  </application>
  <application id='TraxApp' module='traxapp' port='11444'>
    <target device='Trax0' />
  </application>
</configuration>
```

Congratulations! All Xplorer toolchains are installed.

For more details on Xtensa software tools, build configurations, or xt-ocd daemon, see the full set of documents in Xplorer menu **Help > PDF Documentation**.

2.6 Program LPC-Link2 with SEGGER J-Link

In addition to standalone probes, the onboard LPC-Link2 debug probe is used to debug HiFi4 DSP over SWD ports. The RT600 EVK has an LPC4300 MCU (top right corner on EVK) and by default has been pre-programmed as CMSIS-DAP probe. CMSIS-DAP is not compatible with HiFi4. Therefore, a few extra steps are required to flash it with J-Link firmware and support both Arm® CM33 core and HiFi4 DSP core.

The steps are:

1. Install LPCScrypt, a command-line tool for programming onboard LPC-Link2 debug probe with the latest CMSIS-DAP and J-Link firmware. LPCScrypt is available for download from:https://www.nxp.com/design/microcontrollers-developer-resources/lpc-microcontroller-utilities/lpcscrypt-v2.1.0:LPCSCRYPT.
2. After the download is complete, run the installer.

NOTE

During the installation, the DFU, and VCOM drivers are automatically installed for all platforms.

3. To update the LPC-Link2 debug circuit firmware, unplug the USB cable on J5 and then connect to the DFULink jumper. In the MIMXRT685-EVK, JP1 is the LPCXpresso DFU jumper as shown in Figure 8.
4. Connect JP1 using the jumper.

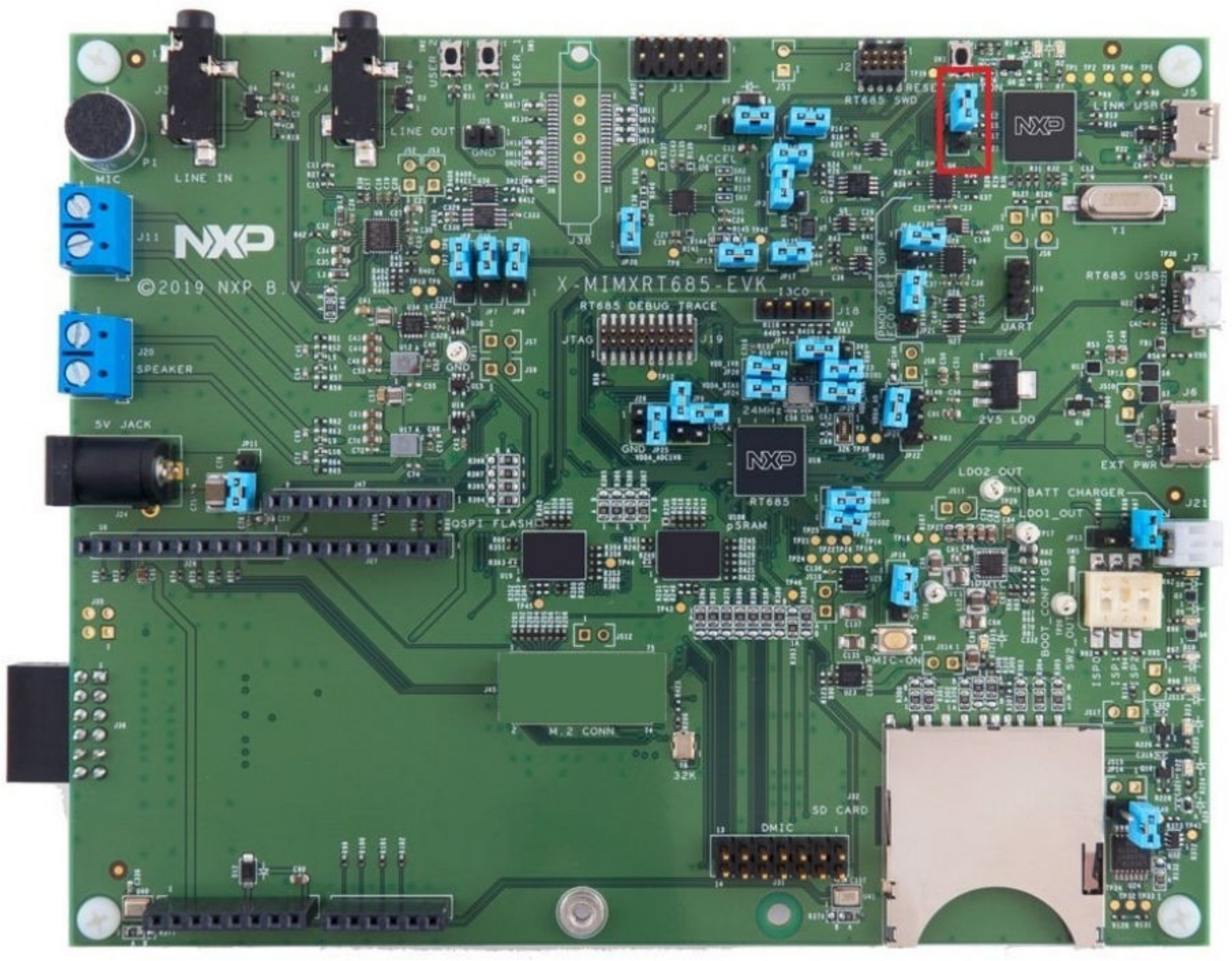
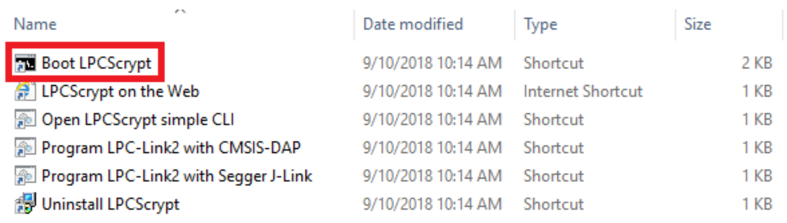


Figure 8. Connect JP1

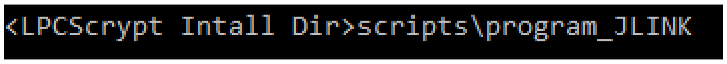
5. Reconnect the board to the host computer over the debug link USB connector J5.
6. Launch LPCScrypt by double-clicking the *Boot LPCScrypt* file in the LPCScrypt install location: `C:\ProgramData\Microsoft\Windows\Start Menu\Programs\LPCScrypt.`



Name	Date modified	Type	Size
Boot LPCScrypt	9/10/2018 10:14 AM	Shortcut	2 KB
LPCScrypt on the Web	9/10/2018 10:14 AM	Internet Shortcut	1 KB
Open LPCScrypt simple CLI	9/10/2018 10:14 AM	Shortcut	1 KB
Program LPC-Link2 with CMSIS-DAP	9/10/2018 10:14 AM	Shortcut	1 KB
Program LPC-Link2 with Segger J-Link	9/10/2018 10:14 AM	Shortcut	1 KB
Uninstall LPCScrypt	9/10/2018 10:14 AM	Shortcut	1 KB

Figure 9. Locate and double-click on the Boot LPCScrypt file

7. In that command shell, run the program_JLINK script to install the JLink debug firmware.



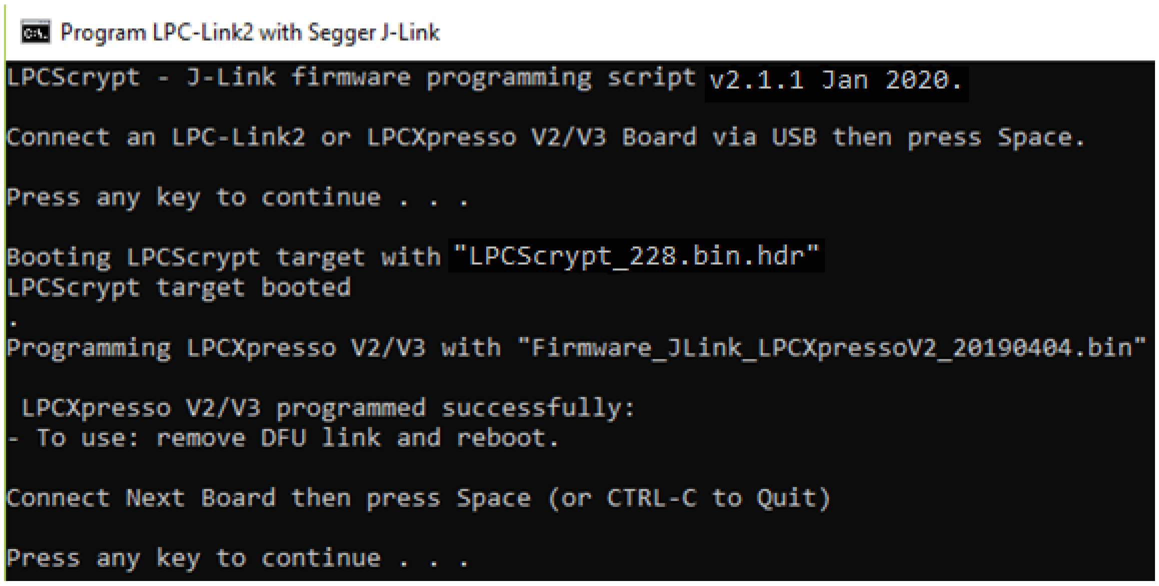
```
<LPCScrypt Intall Dir>scripts\program_JLINK
```

Figure 10. Run program_JLINK

NOTE

File paths in this document use the Windows directory separator, on Linux or Mac OSX. The file paths must be replaced with '/.' For Windows users, shortcuts to the scripts are available from the LPCScrypt entry on the Start menu.

8. Verify once you select the firmware (in this case J-Link), LPCScrypt. The console appears as shown in Figure 11.



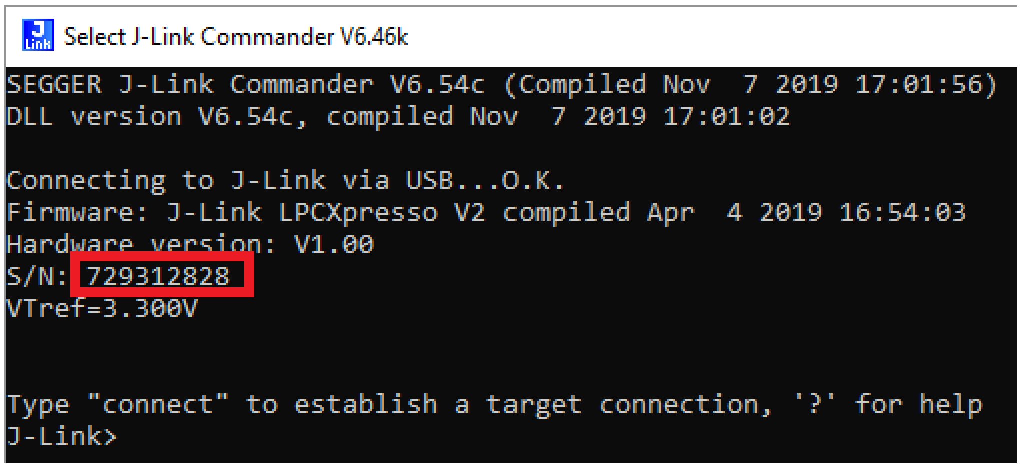
Program LPC-Link2 with Segger J-Link

```
LPCScrypt - J-Link firmware programming script v2.1.1 Jan 2020.

Connect an LPC-Link2 or LPCXpresso V2/V3 Board via USB then press Space.

Press any key to continue . . .

Booting LPCScrypt target with "LPCScrypt_228.bin.hdr"
LPCScrypt target booted
.
Programming LPCXpresso V2/V3 with "Firmware_JLink_LPCXpressoV2_20190404.bin"

 LPCXpresso V2/V3 programmed successfully:
- To use: remove DFU link and reboot.

Connect Next Board then press Space (or CTRL-C to Quit)

Press any key to continue . . .
```

Figure 11. Console after selecting the firmware

9. Open/ Disconnect JP1 and power cycle the board. The onboard LPC-Link2 is ready to be used as SEGGER J-Link probe.

   Every EVK/ LPC-Link2 has a different J-Link S/N. Therefore, make sure to write down the S/N for xt-ocd and topology.xml as indicated in Install Xtensa On Chip Debugger Daemon.



```
SEGGER J-Link Commander V6.54c (Compiled Nov  7 2019 17:01:56)
DLL version V6.54c, compiled Nov  7 2019 17:01:02

Connecting to J-Link via USB...O.K.
Firmware: J-Link LPCXpresso V2 compiled Apr  4 2019 16:54:03
Hardware version: V1.00
S/N: 729312828
VTref=3.300V


Type "connect" to establish a target connection, '?' for help
J-Link>
```

Figure 12. Select J-Link Commander

10. Another benefit is that LPC-Link2 debug probe creates a virtual serial port over USB, so the extra UART2USB cable for debugging is not required.

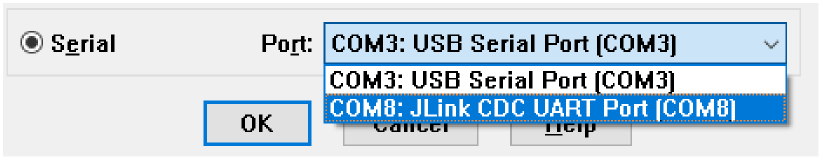

Serial
Port:
COM3: USB Serial Port (COM3)
COM3: USB Serial Port (COM3)
COM8: JLink CDC UART Port (COM8)
OK
Cancel
Help

Figure 13. Select Port

The download link provides the document, demo videos and details on LPC-Link2. If you have any questions, or have difficulties to program the probe, see https://www.nxp.com/design/microcontrollers-developer-resources/lpc-microcontroller-utilities/lpcscrypt-v2.1.0:LPCSCRYPT.

2.7 Install Xtensa Software Tools without IDE

The Xtensa Software Tools can also be installed without the use of the IDE. The installation without the IDE is useful in a command-line only Linux environment, or for better compatibility with an unsupported Linux environment.

The command-line tools package is available as a redistributable zip file that is extracted with an Xplorer IDE install. To gain access to the tools package, the IDE must be installed once in the organization. The tools package is available at: `~/xtensa/XtDevTools/downloads/RI2020.5/tools/XtensaTools_RI_2020_5_linux.tgz`.

With the tools package and the DSP Build Configuration package available from the Tensilica Tools download site, the toolchain can be set up as follows:

```
# Create Xtensa install root
mkdir -p ~/xtensa/tools
mkdir -p ~/xtensa/builds

# Set up the configuration-independent Xtensa Tool:
tar zxvf XtensaTools_RI_2020_5_linux.tgz -C ~/xtensa/tools

# Set up the configuration-specific core files:
tar zxvf nxp_rt600_RI2020_5_newlib_linux_redist.tgz -C ~/xtensa/builds

# Install the Xtensa development toolchain:
cd ~/xtensa
./builds/RI-2020.5-linux/nxp_rt600_RI2020_5_newlib/install \
  --xtensa-tools./tools/RI-2020.5-linux/XtensaTools \
  --registry ./tools/RI-2020.5-linux/XtensaTools/config
```

3 Install MCUXpresso SDK

This section list the steps to:

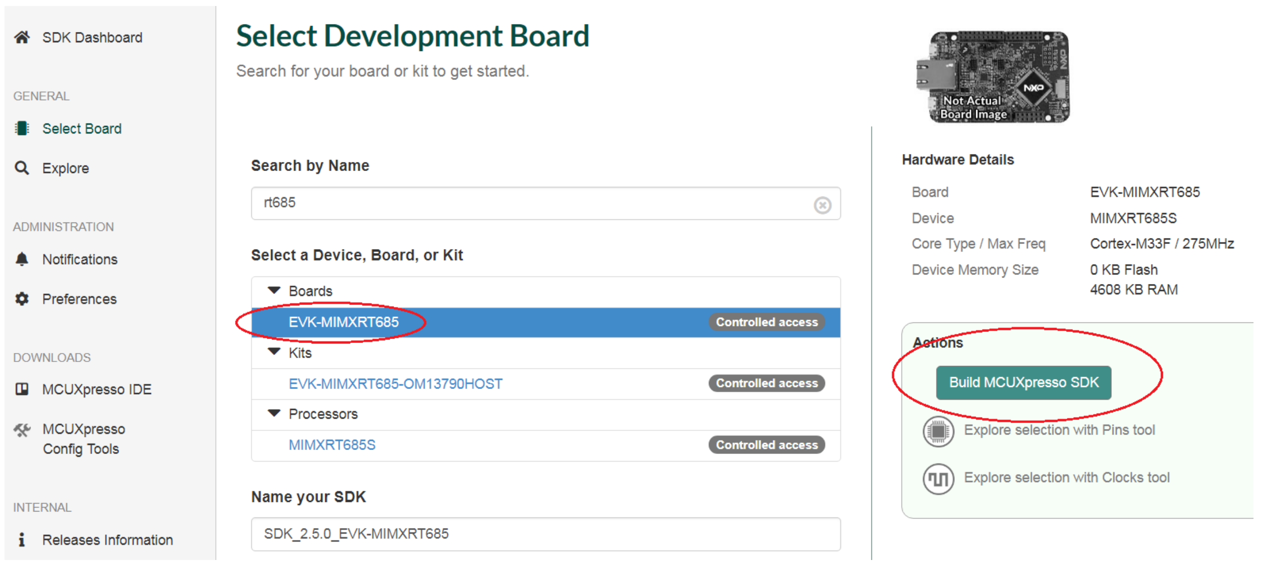
- Download MCUXpresso SDK for RT600
- Enable MCUXpresso SDK DSP
- Initialize DSP Core
- Link DSP Profiles

3.1 Download MCUXpresso SDK for RT600

DSP enablement for RT600, including drivers, middleware libraries, and demo applications are included with the latest RT600 SDK available for download from https://mcuxpresso.nxp.com. If you are accessing the site for the first time, make sure to register.

Once logged in, perform the following steps to use the SDK builder.

1. Click the link **Select Board /Processor** on the left panel.
2. In the **Search by Name** field, enter the name of the board. For example, *RT685*.
3. From the search results, select *EVK-MIMXRT685*.
4. Click the **Build MCUXpresso SDK** button on the right panel.



Search by Name: rt685

Select a Device, Board, or Kit

- Boards
  - EVK-MIMXRT685 — Controlled access
- Kits
  - EVK-MIMXRT685-OM13790HOST — Controlled access
- Processors
  - MIMXRT685S — Controlled access

Name your SDK: SDK_2.5.0_EVK-MIMXRT685

Hardware Details

Board	EVK-MIMXRT685
Device	MIMXRT685S
Core Type / Max Freq	Cortex-M33F / 275MHz
Device Memory Size	0 KB Flash 4608 KB RAM

Figure 14. Select development board

5. In the Build SDK for EVK-MIMXRT685 page, select the required environment settings.
6. Click the **Download SDK** button at the bottom of the page.
7. Save the archive in a local directory.

3.2 Enable MCUXpresso SDK DSP

The following DSP-specific enablements are available inside the MCUXpresso SDK release package for RT600.

-
  ```
  <SDK_ROOT>/devices/MIMXRT685S/
  ```

Unified device and peripheral driver source code that can be compiled for both Arm and DSP cores.

NOTE

Only a limited subset of peripheral drivers and components is supported on the DSP.

-
  ```
  <SDK_ROOT>/boards/evkmimxrt685/dsp_examples/
  ```

  DSP example applications

-
  ```
  <SDK_ROOT>/middleware/multicore/rpmsg_lite/
  ```

  Unified RPMsg-Lite multicore communication library, with porting layers for Arm and DSP cores

-
  ```
  <SDK_ROOT>/middleware/dsp/audio_framework/
  ```

  Xtensa Audio Framework (XAF) for DSP core

-
  ```
  <SDK_ROOT>/middleware/dsp/audio_framework/libxa_af_hostless/
  ```

  Source code and documentation for the core XAF framework

-
  ```
  <SDK_ROOT>/middleware/dsp/audio_framework/testxa_af_hostless/
  ```
- Utilities and tests for developing applications with the XAF framework

  ```
  <SDK_ROOT>/middleware/dsp/audio_framework/testxa_af_hostless /plugins/
  ```
- XAF components and codec binaries

-
  ```
  <SDK_ROOT>/middleware/dsp/naturedsp_hifi4/
  ```

  NatureDSP Math Library for HiFi4 DSP

3.3 Initialize DSP Core

To minimize the power consumption, the DSP core is not powered when RT600 boots up. To initialize, run, or debug DSP applications, you must execute some code on the Arm core.

The DSP management interface library provided in the SDK is available at the location: `<SDK_ROOT>/devices/MIMXRT685S/drivers/fsl_dsp.c`.

```
/* Initialize DSP core. */
void DSP_Init(void);
/* Deinit DSP core. */
void DSP_Deinit(void);
/* Copy DSP image to destination address. */
void DSP_CopyImage(dsp_copy_image_t *dspCopyImage);
/* Start DSP core. */
void DSP_Start(void);
/* Stop DSP core. */
void DSP_Stop(void);
```

The SDK includes a helper function used by the DSP example applications at: `<SDK_ROOT>/boards/evkmimxrt685/dsp_examples/dsp_support.c`.

```
/* Prepare DSP core for code execution:
   - Setup PMIC for DSP
   - Initialize DSP clock and core
```

```
   - (Optional) Copy DSP binary image into RAM
   - Start DSP core
 */
void BOARD_DSP_Init(void);
```

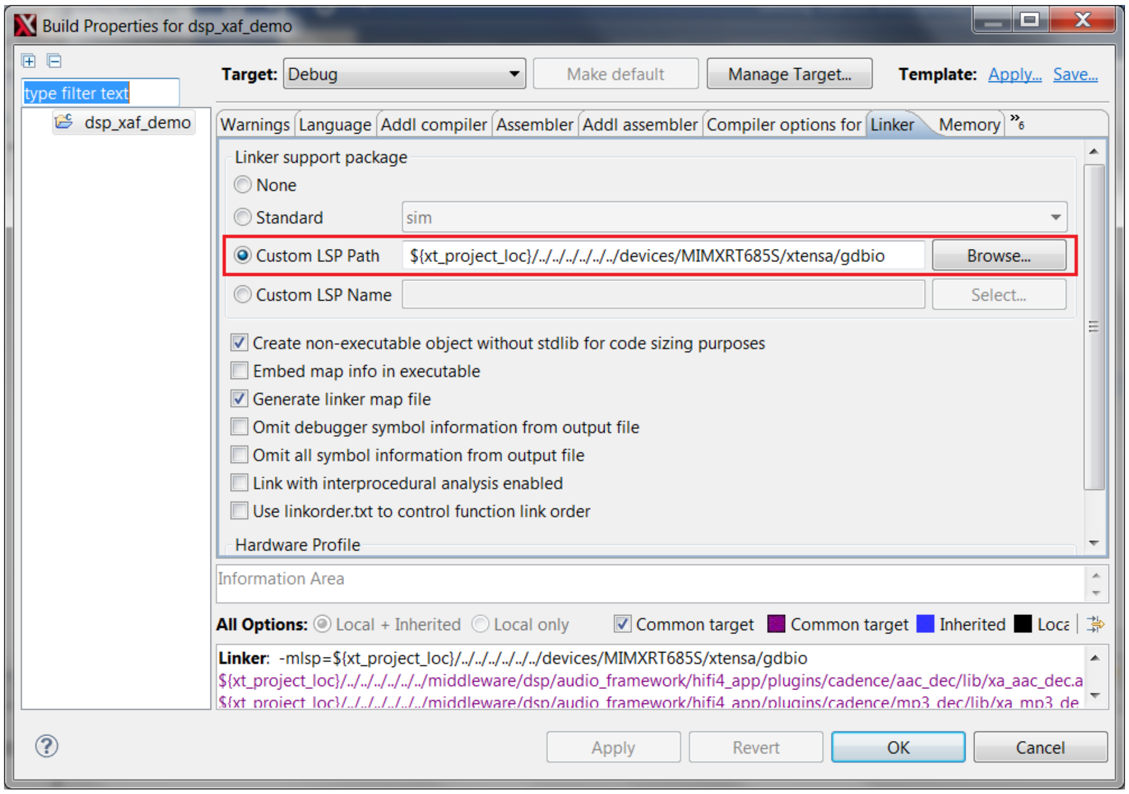
After executing this function during your Arm application startup, the DSP is initialized and ready to run. From here, code is loaded and debugged on the DSP with Xplorer IDE and tools.

3.4 Link DSP Profiles

The Xtensa Software Tools use linker support packages (LSPs) to link a HiFi4 DSP application for the RT600. An LSP includes both a system memory map and a collection of libraries to include into the final binary. The LSPs are provided in the MCUXpresso SDK and are available at: `<SDK_ROOT>/devices/MIMXRT685S/xtensa/`.

DSP sample applications are configured to link against one of these custom LSPs. By default, *Debug* targets are linked against the gdbio LSP intended for use with an attached debugger and captures I/O requests (printf) through gdb. The *Release* target links against the min-rt LSP, which includes minimal runtime support.

To change the LSP used by a project target in the Xplorer IDE, use the **Linker** tab in the project **Build Properties** dialog box.



Build Properties for dsp_xaf_demo

Target: Debug

Template: Apply... Save...

Linker support package
- None
- Standard: sim
- [✓ Custom LSP Path: ${xt_project_loc}/../../../../../../devices/MIMXRT685S/xtensa/gdbio]
- Custom LSP Name

- ✓ Create non-executable object without stdlib for code sizing purposes
- Embed map info in executable
- ✓ Generate linker map file
- Omit debugger symbol information from output file
- Omit all symbol information from output file
- Link with interprocedural analysis enabled
- Use linkorder.txt to control function link order

Hardware Profile

All Options: Local + Inherited Local only ✓ Common target ■ Common target ■ Inherited ■ Loca

Linker: -mlsp=${xt_project_loc}/../../../../../../devices/MIMXRT685S/xtensa/gdbio
${xt_project_loc}/../../../../../../middleware/dsp/audio_framework/hifi4_app/plugins/cadence/aac_dec/lib/xa_aac_dec.a
${xt_project_loc}/../../../../../../middleware/dsp/audio_framework/hifi4_app/plugins/cadence/mp3_dec/lib/xa_mp3_de

Figure 15. Build Properties dialog box

The MCUXpresso SDK ships with other standard LSPs for RT600. For more information on LSPs and creating a custom memory map using Xtensa software tools, see the Cadence Linker Support Packages (LSPs) Reference Manual.

4 Run and Debug DSP Demo using Xplorer IDE

This section lists the steps to:

- Prepare Arm Core for ‘Hello World’
- Start Xtensa Debugger Daemon

- Prepare DSP Core for 'Hello World'
- Run and Debug DSP Audio Framework
- Launch DSP Application from Arm Core

4.1 Prepare Arm Core for 'Hello World'

Each of the DSP demos included in the MCUXpresso SDK consists of two separate applications that run on the Arm core and DSP core. The Arm core application initializes the DSP core in the manner described in Initialize DSP Core and executes other application-specific functionality.

To debug the 'Hello World' DSP application, you must first set up and execute the Arm application using an environment of your choice.

- Build and execute the 'Hello World' Arm demo located in: `<SDK_ROOT>/boards/evkmimxrt685/dsp_examples/hello_world_usart/cm33/`.

Preparing an Arm core development environment is outside the scope of this document. For information on how to use the SDK for Arm core development, see the *Getting Started with MCUXpresso SDK for MIMXRT600.pdf* document located at `<SDK_ROOT>/docs/`.

NOTE

IAR Embedded Workbench may require a patch to enable compatibility with RT600. For details on the patch, contact NXP directly.

NOTE

If you are using MCUXpresso, it is highly recommended to upgrade to latest version of the SDK and match the latest EVK board.

4.2 Start Xtensa Debugger Daemon

To debug DSP applications on RT600, ensure that the xt-ocd daemon is running. This application runs a gdb server that the Xtensa core debugger connects to.

Go to the command-line window and change directory (cd) to xt-ocd daemon installation path. The default path on Windows is `C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.0.5`.

Execute the daemon with the custom topology.

```
xt-ocd.exe -c topology.xml

XOCD 14.05 2020-10-12 17:38:02
(c) 1999-2021 Cadence Design Systems Inc. All rights reserved.
[Debug Log 2021-06-17 08:12:29]
Loading module "gdbstub" v2.0.0.12
Loading module "jlink" v2.0.2.0
Using JLINK lib v.68803
Jlink USB Serial Number: 726415538
Connected to Jlink Device:
  Name:'J-Link LPCXpresso V2 compiled Apr  4 2019 16:54:03'
  S/N:XXXXXXXX
  Firmware: J-Link LPCXpresso V2 compiled Apr  4 2019 16:54:03
  Requested/Set TCK: 2000kHz/65534kHz
Jlink: Select SWD
SWD-DP with ID 0x6BA02477
Loading module "jtag" v2.0.0.20
Loading module "xtensa" v2.0.0.48
Starting thread 'GDBStub'
Opened GDB socket at port 20000
```

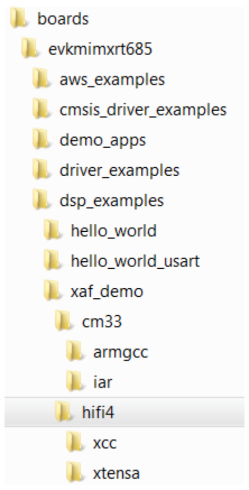
```
Initialize XDM driver
Warning: Warning: DAP Reset request failed! Ignoring...
```

NOTE

Some warning messages are expected and can be ignored. If you receive an error initializing the XDM driver, initialize and start the DSP core before debugging. For details on initializing and debugging, see Initialize DSP Core and Link DSP Profiles. For details on xt-ocd runtime options and configuration, see Chapter 7 of the Xtensa Debug Guide, available in **Help > PDF Documentation**.

4.3 Prepare DSP Core for ‘Hello World’

The RT600 SDK provides a collection of DSP example applications located in `boards/evkmimxrt685/dsp_examples/`. Each DSP example has two source directories, one for the Arm® Cortex®-M33 core (‘cm33’) and one for the DSP HiFi4 core (‘dsp’).



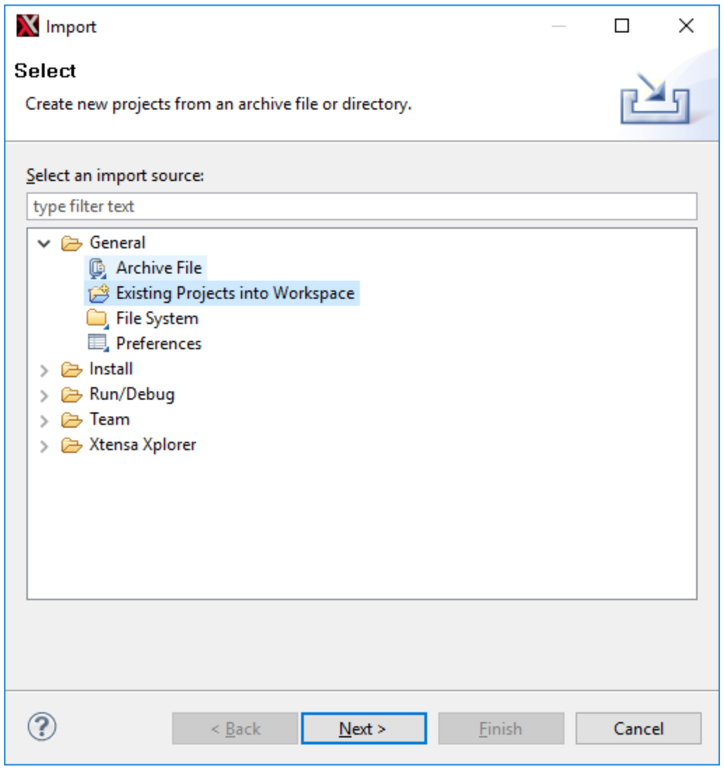
```
boards
  evkmimxrt685
    aws_examples
    cmsis_driver_examples
    demo_apps
    driver_examples
    dsp_examples
      hello_world
      hello_world_usart
      xaf_demo
        cm33
          armgcc
          iar
        hifi4
          xcc
          xtensa
```

Figure 16. Location of DSP example application

The projects for different supported toolchains are build in the above directories. For the DSP example above, the ‘xcc’ project builds on the command line and the ‘xtensa’ directory is an Xplorer IDE project.

To run the ‘Hello World’ demo, import the SDK sources into the Xplorer IDE.

1. Select **File > Import > General > Existing Projects into Workspace**.


Import

Select

Create new projects from an archive file or directory.

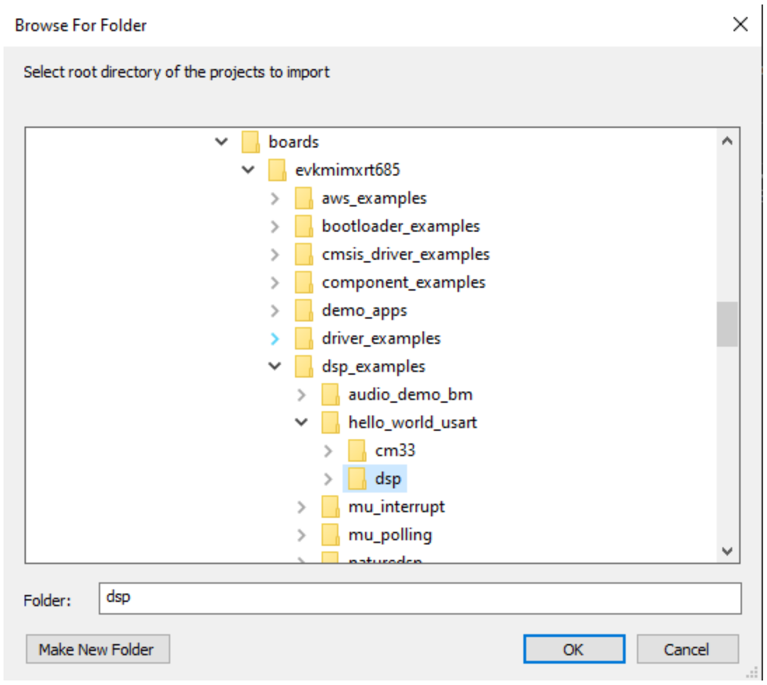
Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- Install
- Run/Debug
- Team
- Xtensa Xplorer

Figure 17. Select an import source

2. Select the SDK directory `<SDK_ROOT>\boards\evkmimxrt685\dsp_examples\hello_world_usart\dsp\xtensa` as root directory.


Browse For Folder

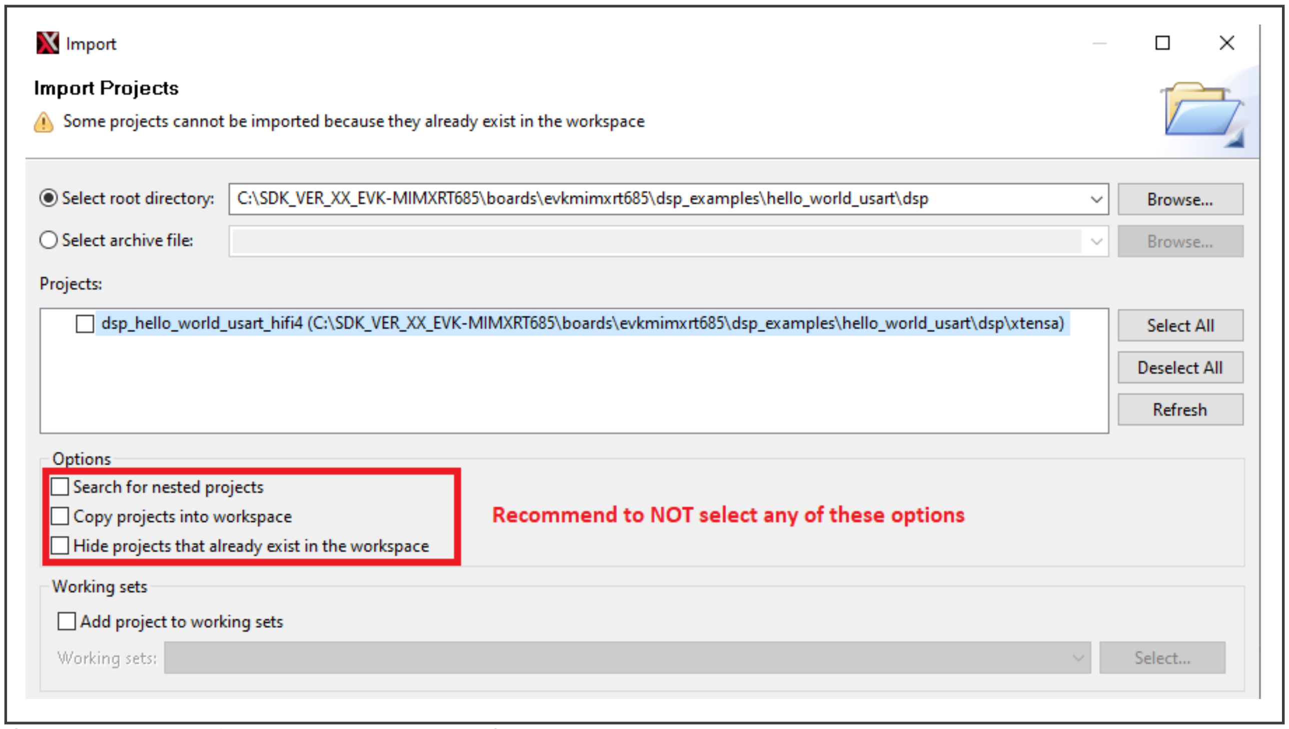
Select root directory of the projects to import

- boards
  - evkmimxrt685
    - aws_examples
    - bootloader_examples
    - cmsis_driver_examples
    - component_examples
    - demo_apps
    - driver_examples
    - dsp_examples
      - audio_demo_bm
      - hello_world_usart
        - cm33
        - dsp
      - mu_interrupt
      - mu_polling

Folder: dsp

Figure 18. Select the SDK directory

3. Leave all the other check boxes blank.



Select root directory: C:\SDK_VER_XX_EVK-MIMXRT685\boards\evkmimxrt685\dsp_examples\hello_world_usart\dsp

Select archive file:

Projects:

dsp_hello_world_usart_hifi4 (C:\SDK_VER_XX_EVK-MIMXRT685\boards\evkmimxrt685\dsp_examples\hello_world_usart\dsp\xtensa)

Options

- Search for nested projects
- Copy projects into workspace
- Hide projects that already exist in the workspace

Recommend to NOT select any of these options

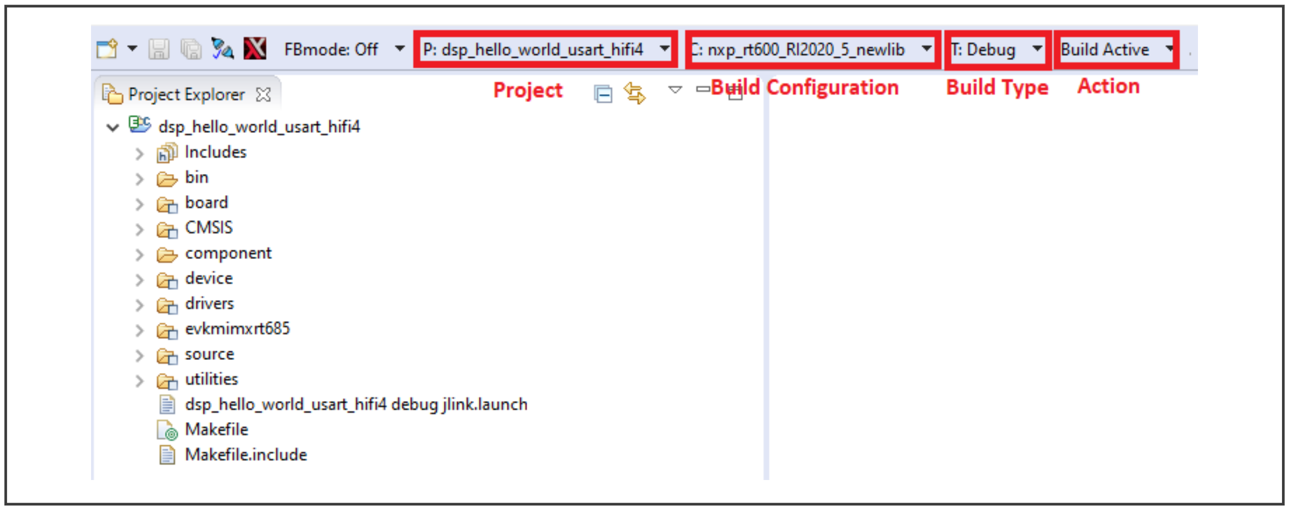
Working sets

- Add project to working sets

Working sets:

Once imported, the 'dsp_hello_world_usart_hifi4' appears in the Project Explorer.

4. To make a build selection for the project and hardware target configuration, use the drop-down buttons on the menu bar.



5. To build the project for debug, profile, or trace, use the action buttons on the right side of the menu bar.

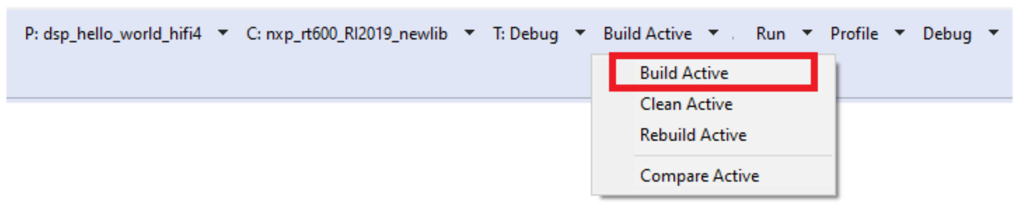


Figure 19. Build project

A default debug configuration is provided by the SDK project, which utilizes the on-chip debugger.

6. To choose the configuration, select the **Debug Configurations** menu item.

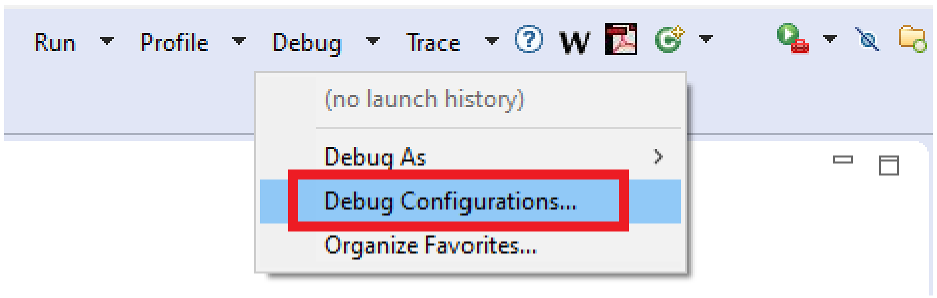
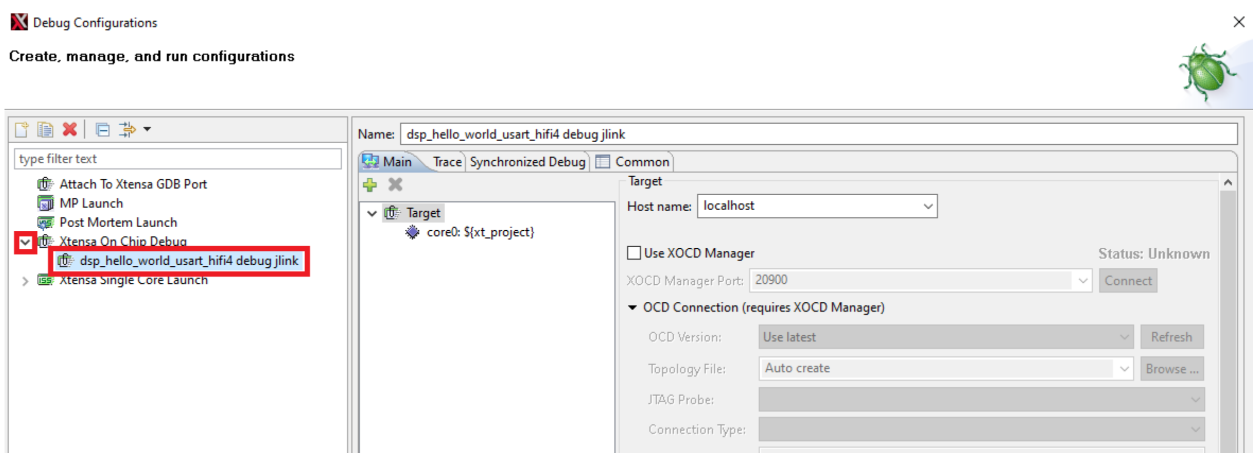


Figure 20. Debug configurations

The **Debug Configurations** dialog box appears.

7. Expand the **Xtensa On Chip Debug** tree option and select ‘*dsp_hello_world_hifi4_debug_jlink*’.



Name: dsp_hello_world_usart_hifi4 debug jlink

Host name: localhost

Use XOCD Manager

Status: Unknown

XOCD Manager Port: 20900

OCD Connection (requires XOCD Manager)

OCD Version: Use latest

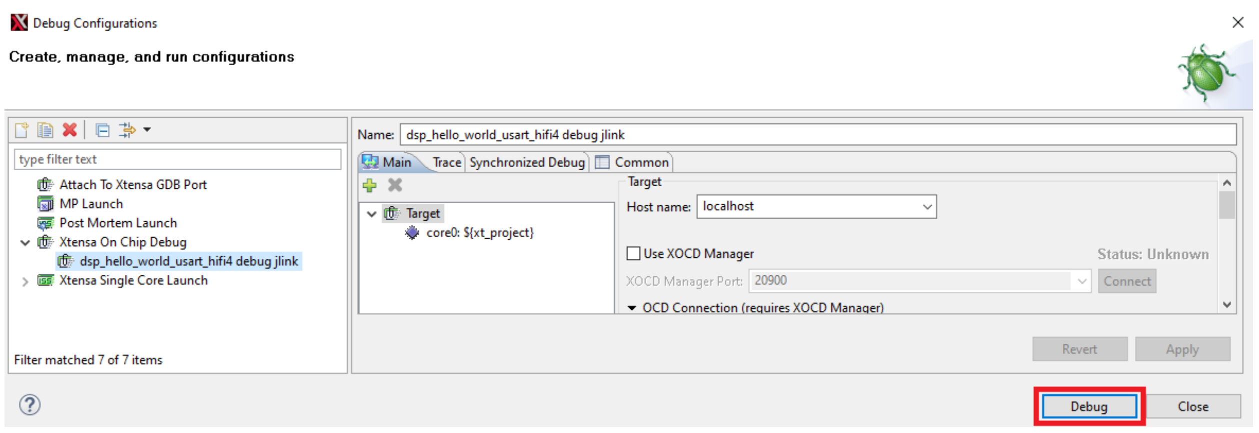
Topology File: Auto create

JTAG Probe:

Connection Type:

Figure 21. Select dsp_hello_world_hifi4_debug_jlink

8. Click the **Debug** button. The actual debug on the chip initiates.



Name: dsp_hello_world_usart_hifi4 debug jlink

Host name: localhost

Use XOCD Manager

Status: Unknown

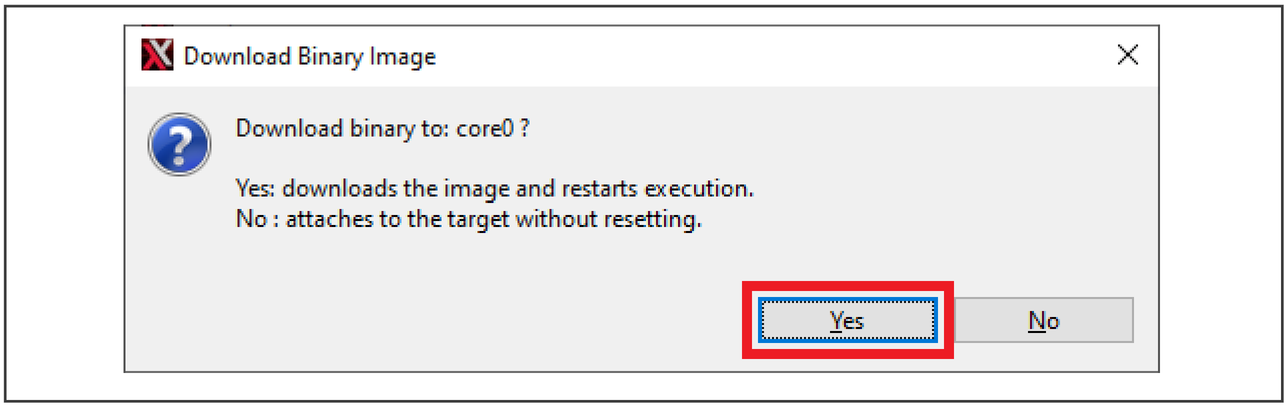
XOCD Manager Port: 20900

OCD Connection (requires XOCD Manager)

Figure 22. Click Debug

The Xplorer IDE prompts whether binaries should download to the hardware.

9. Select **Yes**.



Download binary to: core0 ?

Yes: downloads the image and restarts execution.
No : attaches to the target without resetting.

10. The Xplorer IDE transitions to the **Debug** perspective after the binary download.

NOTE

After initial configuration, it is possible to select the same debug configuration as in Figure 23.

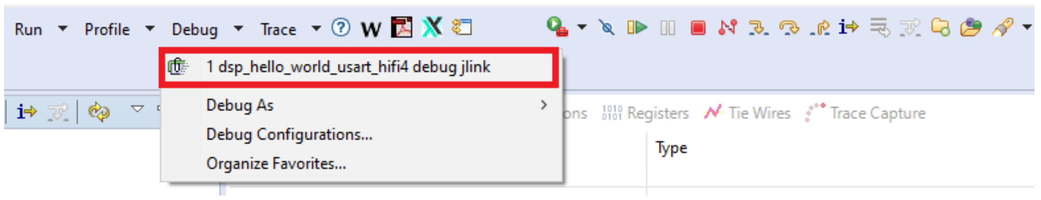
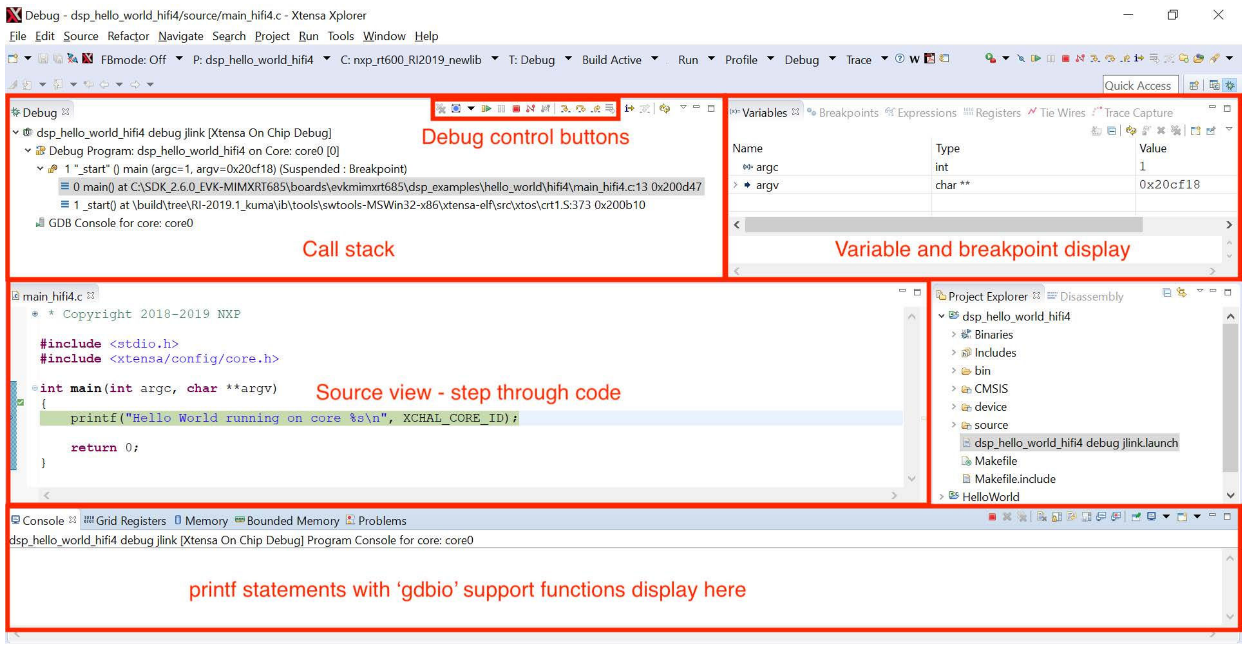


Figure 23. Select debug configuration

The **Debug** perspective appears as in Figure 24.



```
dsp_hello_world_hifi4 debug jlink [Xtensa On Chip Debug]
  Debug Program: dsp_hello_world_hifi4 on Core: core0 [0]
    1 "_start" () main (argc=1, argv=0x20cf18) (Suspended : Breakpoint)
      0 main() at C:\SDK_2.6.0_EVK-MIMXRT685\boards\evkmimxrt685\dsp_examples\hello_world\hifi4\main_hifi4.c:13 0x200d47
      1 _start() at \build\tree\RI-2019.1_kuma\ib\tools\swtools-MSWin32-x86\xtensa-elf\src\xtos\crt1.S:373 0x200b10
  GDB Console for core: core0
```

Name	Type	Value
argc	int	1
argv	char **	0x20cf18

```
 * Copyright 2018-2019 NXP

#include <stdio.h>
#include <xtensa/config/core.h>

int main(int argc, char **argv)
{
    printf("Hello World running on core %s\n", XCHAL_CORE_ID);

    return 0;
}
```

```
dsp_hello_world_hifi4 debug jlink [Xtensa On Chip Debug] Program Console for core: core0
```

Figure 24. Debug perspective

The program stops at the start of main() function. To run the program, click the **Resume** or **Stepping through** icon button as shown in Figure 25.

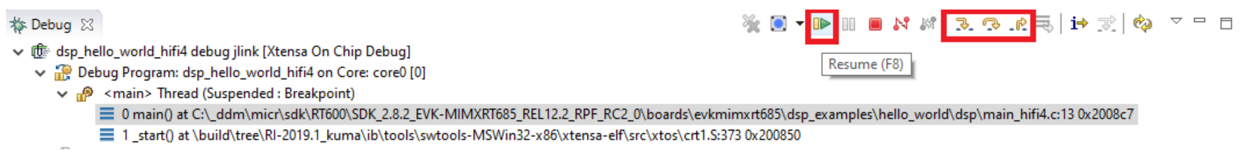

```
Debug
dsp_hello_world_hifi4 debug jlink [Xtensa On Chip Debug]
  Debug Program: dsp_hello_world_hifi4 on Core: core0 [0]
    <main> Thread (Suspended : Breakpoint)
      0 main() at C:\_ddm\micr\sdk\RT600\SDK_2.8.2_EVK-MIMXRT685_REL12.2_RPF_RC2_0\boards\evkmimxrt685\dsp_examples\hello_world\dsp\main_hifi4.c:13 0x2008c7
      1 _start() at \build\tree\RI-2019.1_kuma\ib\tools\swtools-MSWin32-x86\xtensa-elf\src\xtos\crt1.S:373 0x200850
```

Figure 25. Run the program

11. After resuming / stepping through the 'printf' statement, the following output appears in the **Console** view of the IDE.

```
Hello World running on core nxp_rt600_RI2020_5_newlib
```

To return to the default IDE layout after the debugging completes, select the previous code perspective.

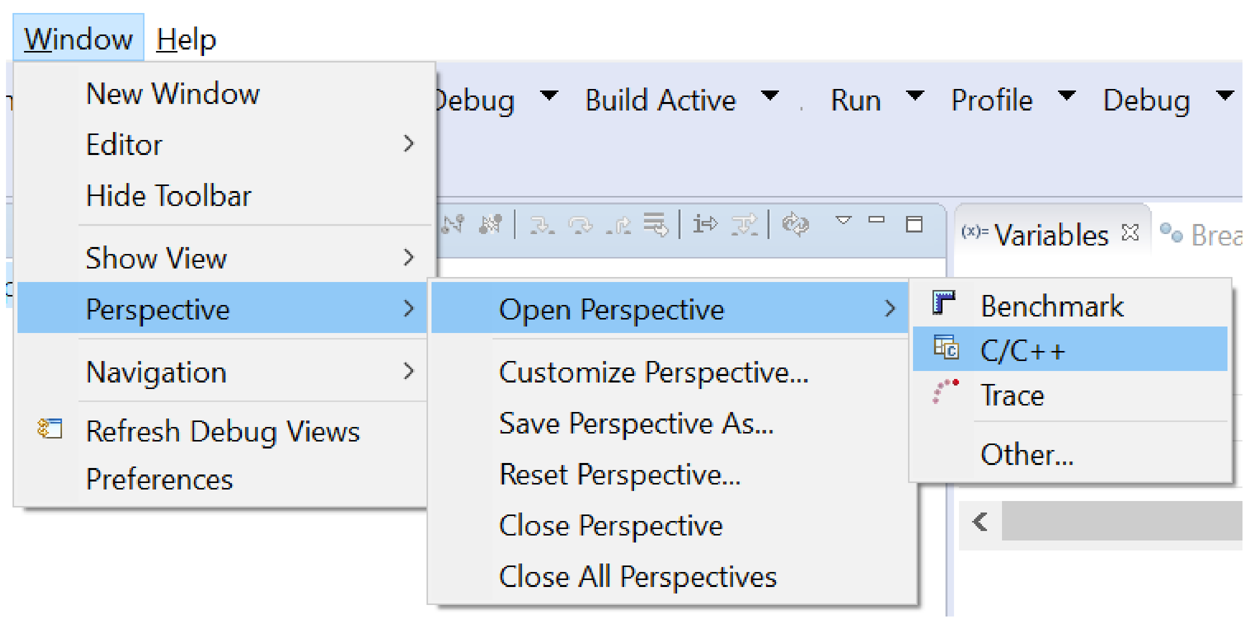


Figure 26. Select perspective

4.4 Run and Debug DSP Audio Framework

The DSP audio framework demo consists of separate applications that run on the Arm core and DSP core. The Arm application runs a command shell and relays the input requests to the DSP application using RPMsg-Lite.

4.4.1 EVK Board Setup for Audio Demo

The DSP audio demo is tested against EVK-MIMXRT685 Rev E and requires UART for serial console.

For the codec to output audio properly, attach one jumper on the board as follows:

JP7-1 <==> JP7-2

JP8-1 <==> JP8-2

NOTE

DSP operates DMA and audio peripherals same as CM33 side. Jumper settings, I2S configs, or pin mux settings are all same with CM33 side. For information on steps to operate I2S, DMIC and DMA, see the SDK CM33 side driver examples. SDK\boards\evkmimxrt685\driver_examples\i2s;

NOTE

Jumper settings are different on different version of EVKs or validation boards. The above mentioned setting is for SDK 2.8.0 and Rev E EVK or later only. For old version of boards or SDKs, double check the SDK driver examples for correct settings for the pin mux and jumper settings.

The demo uses the UART for console input and output. Connect the EVK board to a PC via the USB debug interface (J5) and open up a serial interface on your PC using a terminal tool such as PuTTY on Windows or screen on Linux.

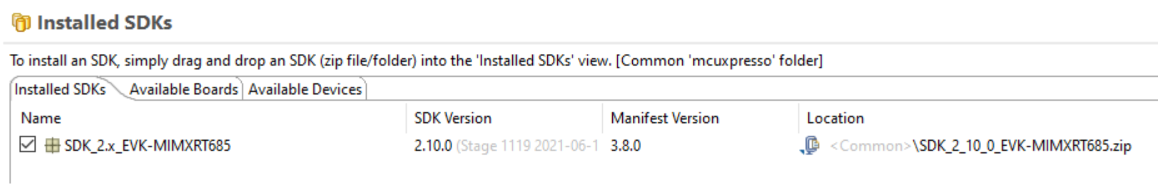
4.4.2 Debug Audio Demo

To debug this DSP application, set up and execute the Arm application in a desired environment. For more information on Arm development environment options, see 'Getting Started with MCUXpresso SDK for EVK-MIMXRT685.pdf.

The following example uses the NXP MCUXpresso IDE for the Arm environment.

To debug the audio demo, perform the following steps.

1. Install the MCUXpresso SDK for RT600 into the MCUXpresso IDE using the Installed SDK panel as shown in



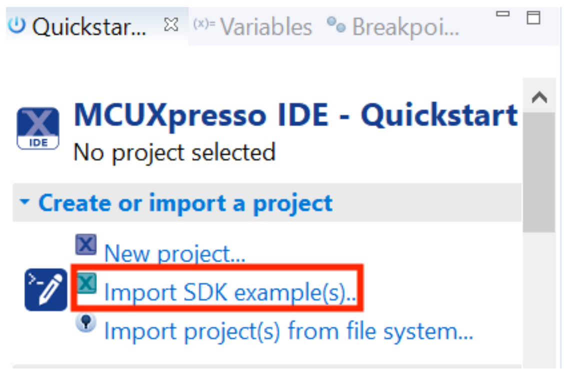
Installed SDKs

To install an SDK, simply drag and drop an SDK (zip file/folder) into the 'Installed SDKs' view. [Common 'mcuxpresso' folder]

Name	SDK Version	Manifest Version	Location
☑ SDK_2.x_EVK-MIMXRT685	2.10.0 (Stage 1119 2021-06-1	3.8.0	<Common>\SDK_2_10_0_EVK-MIMXRT685.zip

Figure 27. Install SDK

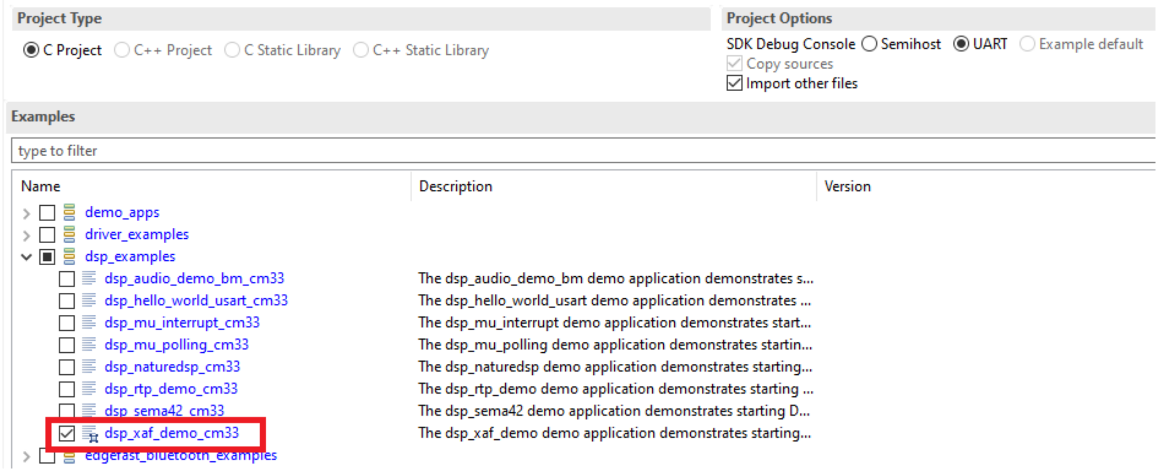
2. To import an example from the installed SDK., use the **Import SDK examples** link in the **Quickstart** panel on the lower left of the screen.



Quickstar...
Variables
Breakpoi...
MCUXpresso IDE - Quickstart
No project selected
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...

Figure 28. Import SDK examples

3. Select the '*dsp_xaf_demo_cm33* example for Cortex-M33 core.



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

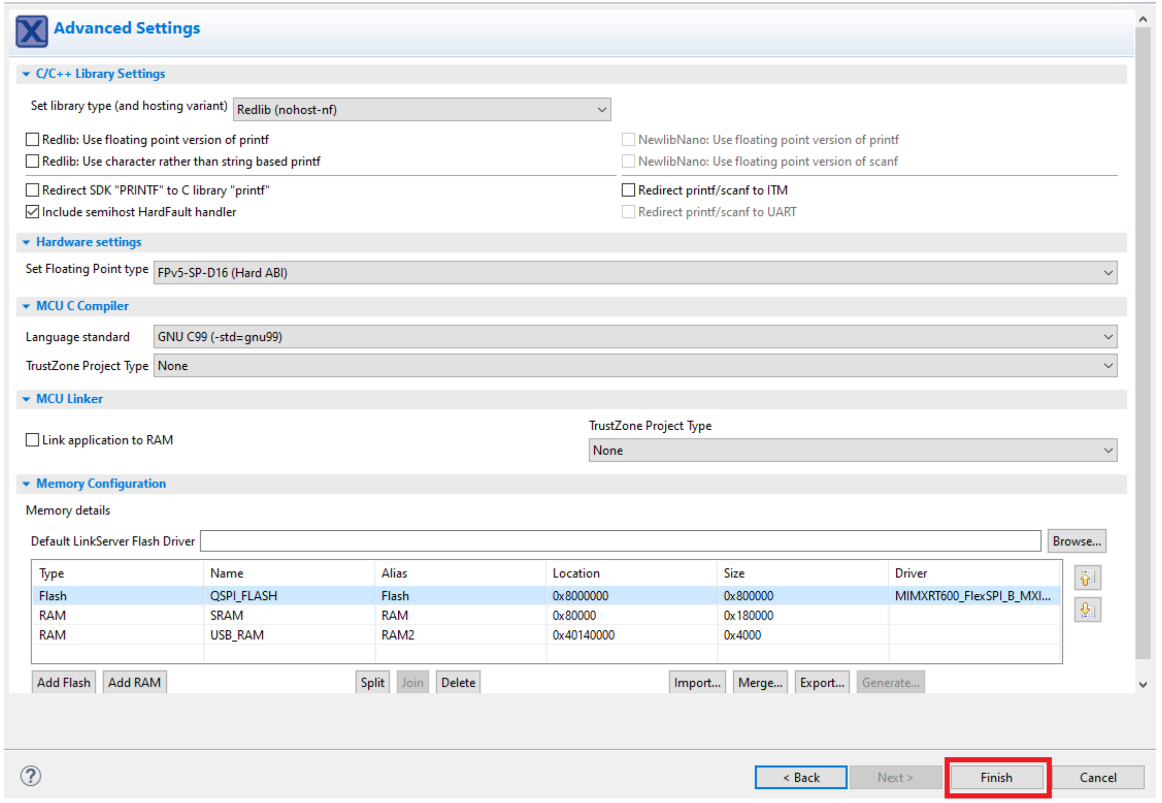
Examples

type to filter

Name	Description	Version
☐ demo_apps		
☐ driver_examples		
■ dsp_examples		
☐ dsp_audio_demo_bm_cm33	The dsp_audio_demo_bm demo application demonstrates s...	
☐ dsp_hello_world_usart_cm33	The dsp_hello_world_usart demo application demonstrates ...	
☐ dsp_mu_interrupt_cm33	The dsp_mu_interrupt demo application demonstrates start...	
☐ dsp_mu_polling_cm33	The dsp_mu_polling demo application demonstrates startin...	
☐ dsp_naturedsp_cm33	The dsp_naturedsp demo application demonstrates starting...	
☐ dsp_rtp_demo_cm33	The dsp_rtp_demo demo application demonstrates starting ...	
☐ dsp_sema42_cm33	The dsp_sema42 demo application demonstrates starting D...	
☑ dsp_xaf_demo_cm33	The dsp_xaf_demo demo application demonstrates starting...	
☐ edgefast_bluetooth_examples		

Figure 29. Select example

4. Click the **Finish** button.



Advanced Settings

C/C++ Library Settings

Set library type (and hosting variant): Redlib (nohost-nf)

☐ Redlib: Use floating point version of printf

☐ Redlib: Use character rather than string based printf

☐ NewlibNano: Use floating point version of printf

☐ NewlibNano: Use floating point version of scanf

☐ Redirect SDK "PRINTF" to C library "printf"

☑ Include semihost HardFault handler

☐ Redirect printf/scanf to ITM

☐ Redirect printf/scanf to UART

Hardware settings

Set Floating Point type: FPv5-SP-D16 (Hard ABI)

MCU C Compiler

Language standard: GNU C99 (-std=gnu99)

TrustZone Project Type: None

MCU Linker

☐ Link application to RAM

TrustZone Project Type: None

Memory Configuration

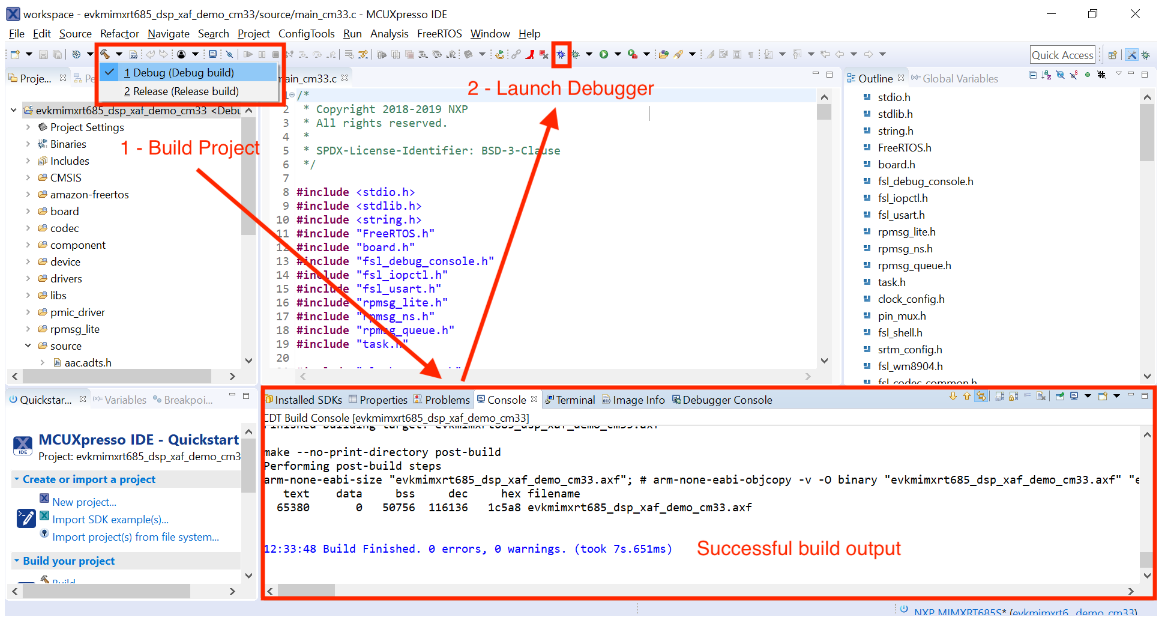
Memory details

Default LinkServer Flash Driver:

Type	Name	Alias	Location	Size	Driver
Flash	QSPI_FLASH	Flash	0x8000000	0x800000	MIMXRT600_FlexSPI_B_MXI...
RAM	SRAM	RAM	0x80000	0x180000	
RAM	USB_RAM	RAM2	0x40140000	0x4000	

Figure 30. Click Finish

5. Build the project and launch the debugger on success.


workspace - evkmimxrt685_dsp_xaf_demo_cm33/source/main_cm33.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run Analysis FreeRTOS Window Help
1 Debug (Debug build)
2 Release (Release build)
1 - Build Project
2 - Launch Debugger
Successful build output
Installed SDKs Properties Problems Console Terminal Image Info Debugger Console
Outline Global Variables
MCUXpresso IDE - Quickstart
Create or import a project
Build your project

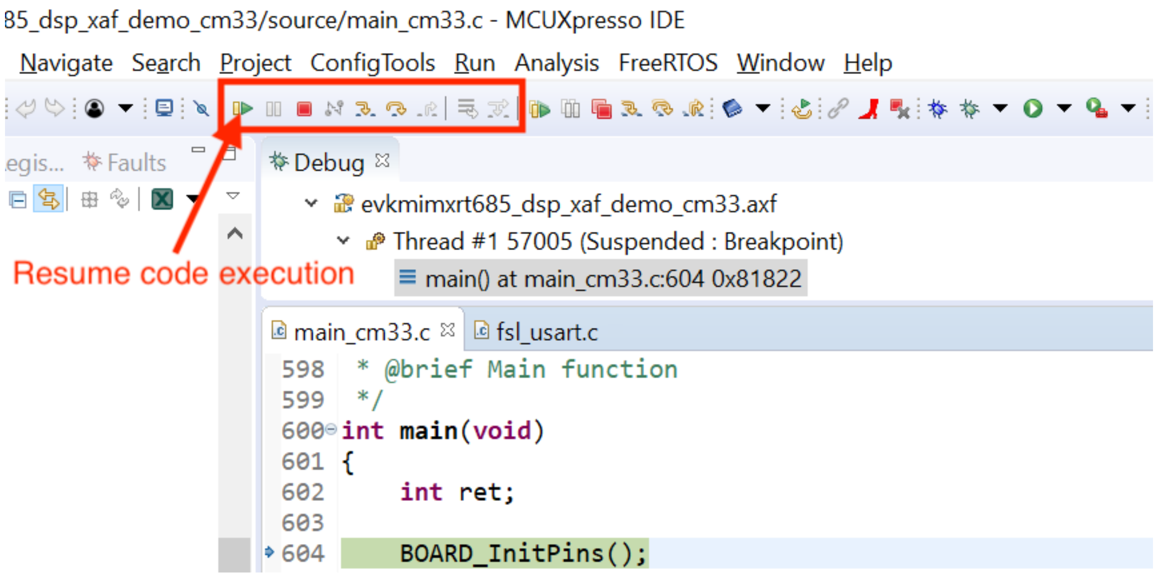
```
/*
 * Copyright 2018-2019 NXP
 * All rights reserved.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "FreeRTOS.h"
#include "board.h"
#include "fsl_debug_console.h"
#include "fsl_iopctl.h"
#include "fsl_usart.h"
#include "rpmsg_lite.h"
#include "rpmsg_ns.h"
#include "rpmsg_queue.h"
#include "task.h"
```

CDT Build Console [evkmimxrt685_dsp_xaf_demo_cm33]

```
make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "evkmimxrt685_dsp_xaf_demo_cm33.axf"; # arm-none-eabi-objcopy -v -O binary "evkmimxrt685_dsp_xaf_demo_cm33.axf" "e
   text    data     bss     dec     hex filename
  65380       0   50756  116136   1c5a8 evkmimxrt685_dsp_xaf_demo_cm33.axf


12:33:48 Build Finished. 0 errors, 0 warnings. (took 7s.651ms)
```

Figure 31. Build and debug the project

6. Use the debug toolbar to resume the code execution.


85_dsp_xaf_demo_cm33/source/main_cm33.c - MCUXpresso IDE
Navigate Search Project ConfigTools Run Analysis FreeRTOS Window Help
egis... Faults
Debug
Resume code execution
main_cm33.c
fsl_usart.c

```
evkmimxrt685_dsp_xaf_demo_cm33.axf
  Thread #1 57005 (Suspended : Breakpoint)
    main() at main_cm33.c:604 0x81822
```

```
 * @brief Main function
 */
int main(void)
{
    int ret;

    BOARD_InitPins();
```

Figure 32. Resume code execution

7. Observe serial terminal output with shell prompt.

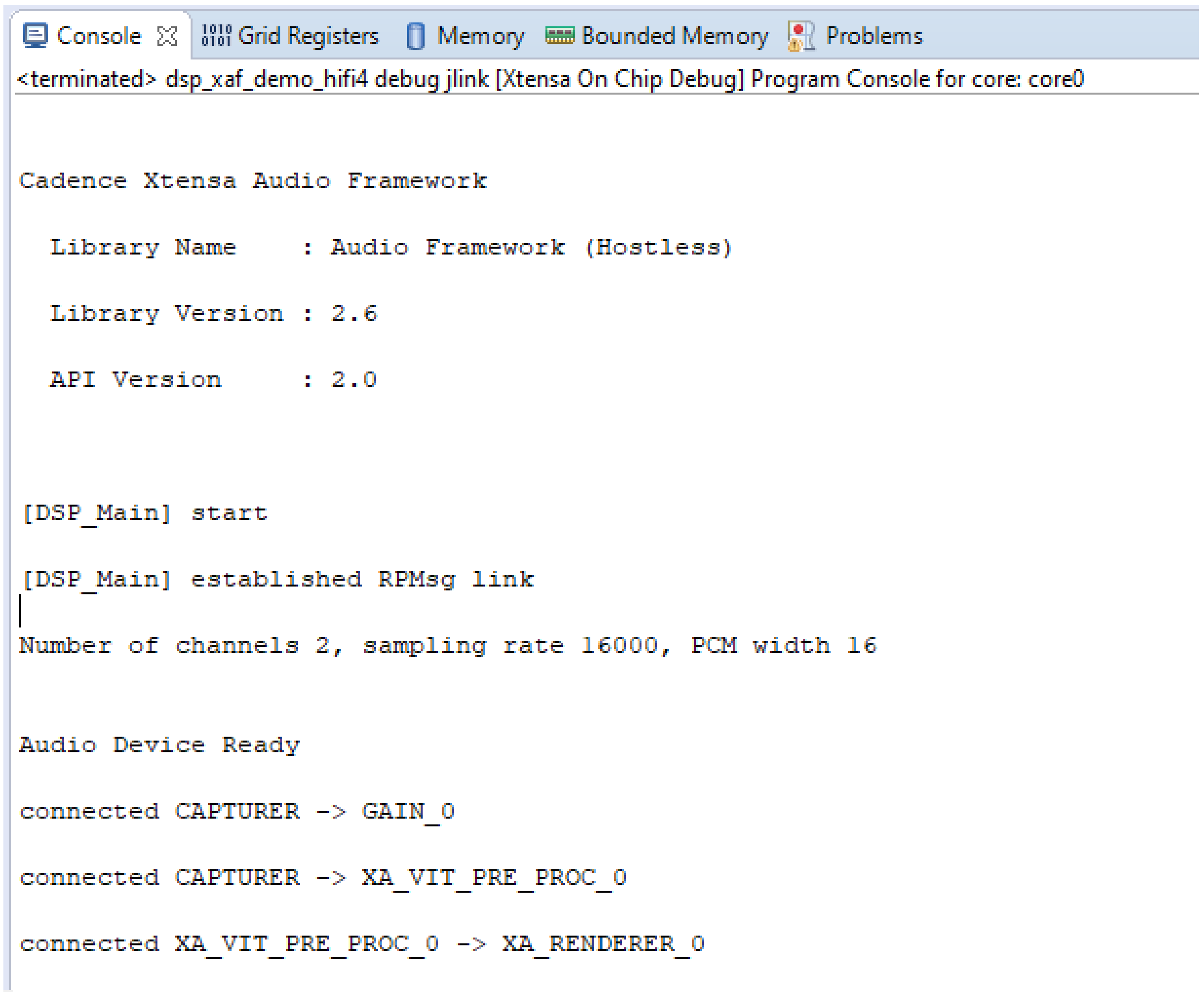
```
******************************
DSP audio framework demo start
******************************

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

```
SHELL build: Jun 17 2021
Copyright  2020  NXP
>>
```

8. Using the Xplorer IDE, load and execute xaf_demo using the steps described in Prepare DSP Core for 'Hello World'.
9. After the DSP application runs, use the serial shell to invoke the 'record_dmic [en | cn]' command. Using the serial shell creates an audio pipeline that captures microphone audio, perform voice recognition (VIT), and playback via the codec speaker line out (J4 on the EVK).

```
>> help
"help": List all the registered commands
"exit": Exit program
"version": Query DSP for component versions
…
"record_dmic [en|cn]": Record DMIC audio and playback on WM8904 codec
…
>> record_dmic en
```

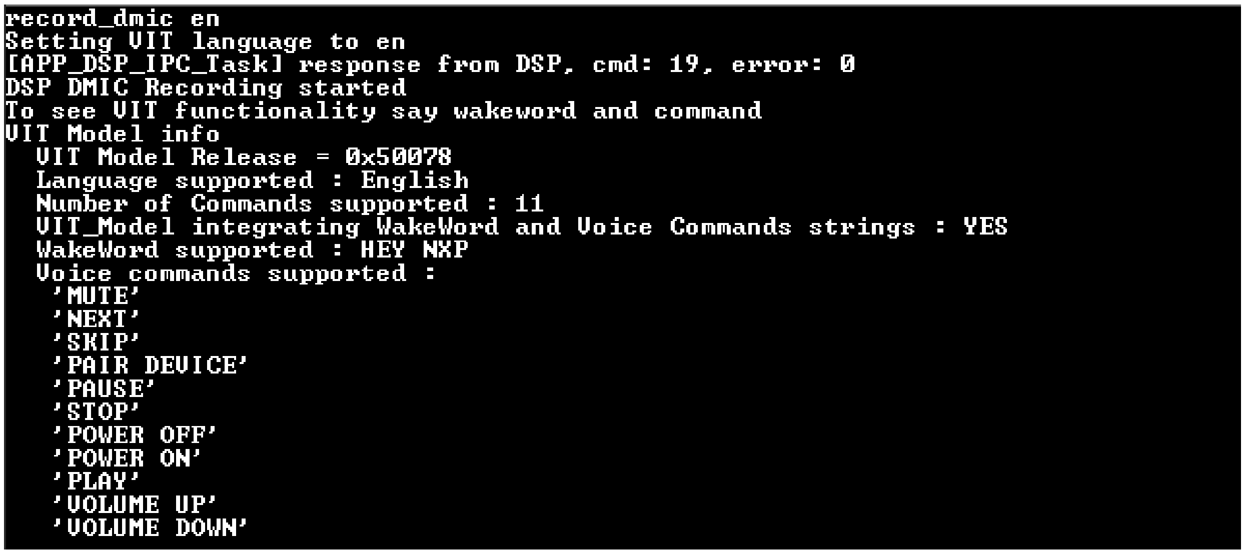
10. The Xplorer IDE console shows the output of the audio framework pipeline initializing.



```
<terminated> dsp_xaf_demo_hifi4 debug jlink [Xtensa On Chip Debug] Program Console for core: core0


Cadence Xtensa Audio Framework

  Library Name    : Audio Framework (Hostless)

  Library Version : 2.6

  API Version     : 2.0



[DSP_Main] start

[DSP_Main] established RPMsg link

Number of channels 2, sampling rate 16000, PCM width 16


Audio Device Ready

connected CAPTURER -> GAIN_0

connected CAPTURER -> XA_VIT_PRE_PROC_0

connected XA_VIT_PRE_PROC_0 -> XA_RENDERER_0
```

Figure 33. Xplorer IDE console

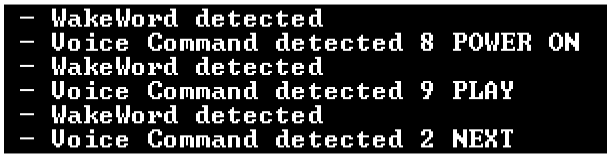
The VIT wake-word and supported commands appear in the serial console.



```
record_dmic en
Setting VIT language to en
[APP_DSP_IPC_Task] response from DSP, cmd: 19, error: 0
DSP DMIC Recording started
To see VIT functionality say wakeword and command
VIT Model info
  VIT Model Release = 0x50078
  Language supported : English
  Number of Commands supported : 11
  VIT_Model integrating WakeWord and Voice Commands strings : YES
  WakeWord supported : HEY NXP
  Voice commands supported :
   'MUTE'
   'NEXT'
   'SKIP'
   'PAIR DEVICE'
   'PAUSE'
   'STOP'
   'POWER OFF'
   'POWER ON'
   'PLAY'
   'VOLUME UP'
   'VOLUME DOWN'
```

Figure 34. VIT wake-word and supported commands

The serial console prints the detected wake-word and commands.



```
 - WakeWord detected
 - Voice Command detected 8 POWER ON
 - WakeWord detected
 - Voice Command detected 9 PLAY
 - WakeWord detected
 - Voice Command detected 2 NEXT
```

Figure 35. Serial console output

For more information on configuration and using the Audio Framework demo, see the file `<SDK_ROOT>\boards\evkmimxrt685\dsp_examples\xaf_demo\readme.txt`.

For more details about VIT which supports English and Mandarin models, see `<SDK_ROOT>\middleware\vit\HIFI4\Doc\VIT_Integration_Guide.pdf`.

4.5 Launch DSP Application from Arm Core

In the previous example, the Arm application and DSP application are independently loaded and debugged. This section list the steps to produce one Arm application binary that includes and starts the DSP application without the use of a debugger or a loader.

The Arm core application for each DSP demo uses a global preprocessor macro to control loading of the DSP binary application:

DSP_IMAGE_COPY_TO_RAM

When this macro is set to ‘1’/TRUE, it instructs the DSP demo application to do the following:

- Link the DSP application binary images into the Arm binary
- Copy the DSP application images into RAM on program boot
- Initialize the DSP to run from the RAM image

NOTE

When modifying the Arm project, make sure to supply the global preprocessor macro to both the C compiler and assembler.

To build the DSP application image for the Arm application, select the **Release** target option in the Xplorer IDE. For more information on building with min-rt LSP, see section Link DSP Profiles.

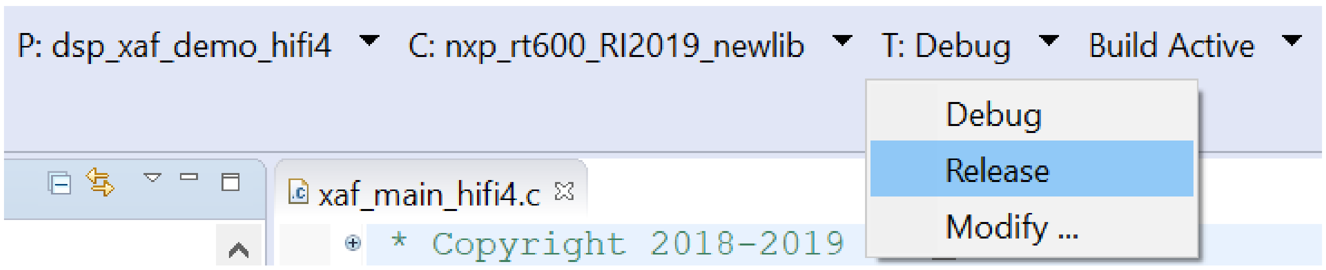


Figure 36. Select the Release target

Three DSP binaries are generated and are loaded into different TCM or SRAM address segments:

- `<SDK_ROOT>/boards/evkmimxrt685/dsp_examples/xaf_demo/dsp/binary/`**`dsp_data_release.bin`**
- `<SDK_ROOT>/boards/evkmimxrt685/dsp_examples/xaf_demo/dsp/binary/`**`dsp_text_release.bin`**
- `<SDK_ROOT>/boards/evkmimxrt685/dsp_examples/xaf_demo/dsp/binary/`**`dsp_ncache_release.bin`**

NOTE

Depending on the environment, the binary images may need to be copied into your Arm application workspace.

5 Run and Debug from Command-Line Environment / LINUX

RT600 SDK has been configured to be as flexible to support multiple toolchains, including the Arm GCC and XCC command-line environments. The principles and essentials are still the same as with the IDE. The command-line environment settings are nearly identical between WIN32 and LINUX, with few different path settings.

5.1 Build and Debug Arm Application

The Arm application requires the GNU Arm Embedded Toolchain and CMake version 3.x for command-line compile and linking. For more information on installation and configuration of required build tools for command-line development, see the *Getting Started with MCUXpresso SDK for EVK_MIMXRT685.pdf* in the *<SDK_ROOT>/docs/* directory.

To build and debug:

1. Launch a command prompt / terminal and change directory to the xaf_demo application.

```
user@linux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_demo/cm33/armgcc$ ls -1
build_all.bat
build_all.sh
build_debug.bat
build_debug.sh
build_flash_debug.bat
build_flash_debug.sh
build_flash_release.bat
build_flash_release.sh
build_release.bat
build_release.sh
clean.bat
clean.sh
CMakeLists.txt
```

2. Use .bat files to build the configuration on Windows, and .sh files on Linux/UNIX.

```
user@linux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_demo/cm33/armgcc$ ./build_debug.sh
...
```

```
[100%] Linking C executable debug/dsp_xaf_demo_cm33.elf
[100%] Built target dsp_xaf_demo_cm33.elf
```

3. Launch the GDB server.

```
user@linux:/opt/JLink$ ./JLinkGDBServerCLExe -device MIMXRT685S_M33 -if SWD
SEGGER J-Link GDB Server V6.46j Command Line Version
...
Listening on TCP/IP port 2331
Connecting to target...Connected to target
Waiting for GDB connection...
```

4. Connect with GDB to the device and load Arm application.

```
user@jlinux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_demo/cm33/armgcc$ arm-none-eabi-gdb debug/
dsp_xaf_demo_cm33.elf
...
Reading symbols from debug/dsp_xaf_demo_cm33.elf...
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x1301ec7a in ?? ()
(gdb) mon reset
Resetting target
(gdb) load
Loading section .flash_config, size 0x200 lma 0x7f400
Loading section .interrupts, size 0x130 lma 0x80000
Loading section .text, size 0xe330 lma 0x80130
Loading section CodeQuickAccess, size 0x52c lma 0x8e460
Loading section .ARM, size 0x8 lma 0x8e98c
Loading section .init_array, size 0x4 lma 0x8e994
Loading section .fini_array, size 0x4 lma 0x8e998
Loading section .data, size 0x104 lma 0x8e99c
Start address 0x801e4, load size 60576
Transfer rate: 272 KB/sec, 5506 bytes/write.
(gdb) b main
Breakpoint 1 at 0x808f2: file /SDK/boards/src/dsp_examples/xaf_demo/cm33/main_cm33.c, line 161.
(gdb) c
Continuing.

Breakpoint 1, main ()
    at /SDK/boards/src/dsp_examples/xaf_demo/cm33/main_cm33.c:161
161     BOARD_InitHardware();
```

5.2 Build and Debug DSP Application

The Xtensa command-line toolchain is installed as part of the Xplorer IDE. The tools can be optionally installed on a new Windows or Linux system without the IDE using the redistributable compressed file found at: *<XTENSA_ROOT>/XtDevTools/downloads/RI-2020.5/tools/*. For more information, see Install Xtensa On Chip Debugger Daemon.

In order to use the command-line tools, some environment variables must be set up for use with the cmake build scripts:

```
# Add tools binaries to PATH.  Assume ~/xtensa/ is install root - please adjust accordingly.
export PATH=$PATH:~/xtensa/XtDevTools/install/tools/RI-2020.5-linux/XtensaTools/bin
# (Optional) Use environment variable to control license file
# NOTE: ~/.flexlmrc will override this selection.  Please delete that file before proceeding.
export LM_LICENSE_FILE=~/RT600.lic

# Setup env vars needed for compile and linking
export XCC_DIR=~/xtensa/XtDevTools/install/tools/RI-2020.5-linux/XtensaTools
export XTENSA_SYSTEM=~/xtensa/XtDevTools/install/builds/RI-2020.5-linux/
nxp_rt600_RI2020_5_newlib/config
export XTENSA_CORE=nxp_rt600_RI2020_5_newlib
```

NOTE

On Windows, you can use the ‘setx’ command instead of the ‘export’ command to set the environment variables.

1. Use the batch/shell script to build out the DSP application from the command line, in the ‘xcc’ directory:

```
user@linux:~/SDK/boards/evkmimxrt685/dsp_examples/xaf_demo/dsp/xcc$ ./build_debug.sh
...
[100%] Built target dsp_xaf_demo_hifi4.elf
```

NOTE

Some warnings during the linking process (floating point ABI) may appear. However, the warnings may be ignored.

2. Launch xt-ocd debugging server (replace topology.xml with your custom version – see section 1.5 of this document):

```
user@linux:/opt/Tensilica/xocd-14.05$ ./xt-ocd.exe -c topology.xml
```

NOTE

If the Arm core fails to initialize the DSP, the xt-ocd daemon may fail to start. Therefore, the Arm core must initialize the DSP first.

3. Connect with Xtensa GDB to the device and execute the DSP application:

```
user@linux:/SDK/boards/evkmimxrt685/dsp_examples/xaf_demo/dsp/xcc$ xt-gdb
debug/dsp_xaf_demo_hifi4.elf
GNU gdb (GDB) 7.11.1 Xtensa Tools 14.05
...
Reading symbols from debug/dsp_xaf_demo_hifi4.elf...done.
(xt-gdb)
(xt-gdb) target remote localhost:20000
Remote debugging using localhost:20000
_DoubleExceptionVector ()
    at /home/xpgcust/tree/RI-2020.5/ib/tools/swtools-x86_64-linux/xtensa-
elf/src/xos/src/xos_vectors.S:216
216 /home/xpgcust/tree/RI-2020.5/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xos/src/
xos_vectors.S: No such file or directory.
(xt-gdb) reset
_ResetVector ()
    at /home/xpgcust/tree/RI-2020.5/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xtos/xea2/reset-
vector-xea2.S:71
71  /home/xpgcust/tree/RI-2020.5/ib/tools/swtools-x86_64-linux/xtensa-elf/src/xtos/xea2/reset-
vector-xea2.S: No such file or directory.
(xt-gdb) load
Loading section .rtos.rodata, size 0x80 lma 0x200000
Loading section .rodata, size 0x17d50 lma 0x200080
Loading section .text, size 0x633f0 lma 0x217dd0
Loading section .rtos.percpu.data, size 0x4 lma 0x27b1c0
Loading section .data, size 0x110c lma 0x27b1d0
Loading section NonCacheable, size 0x2960 lma 0x20040000
Loading section .Level3InterruptVector.literal, size 0x4 lma 0x24000000
Loading section .DebugExceptionVector.literal, size 0x4 lma 0x24000004
Loading section .NMIExceptionVector.literal, size 0x4 lma 0x24000008
Loading section .ResetVector.text, size 0x13c lma 0x24020000
Loading section .WindowVectors.text, size 0x16c lma 0x24020400
Loading section .Level2InterruptVector.text, size 0x1c lma 0x2402057c
Loading section .Level3InterruptVector.text, size 0xc lma 0x2402059c
Loading section .DebugExceptionVector.text, size 0xc lma 0x240205bc
Loading section .NMIExceptionVector.text, size 0xc lma 0x240205dc
Loading section .KernelExceptionVector.text, size 0xc lma 0x240205fc
Loading section .UserExceptionVector.text, size 0x18 lma 0x2402061c
Loading section .DoubleExceptionVector.text, size 0x8 lma 0x2402063c
Start address 0x24020000, load size 520016
Transfer rate: 8 KB/sec, 10612 bytes/write.
(xt-gdb) b main
Breakpoint 1 at 0x21ab5b: file /SDK/boards/src/dsp_examples/xaf_demo/hifi4/xaf_main_hifi4.c,
line 366.
(xt-gdb) c
Continuing.

Breakpoint 1, main ()
    at /SDK/boards/src/dsp_examples/xaf_demo/hifi4/xaf_main_hifi4.c:366
366     xos_start_main("main", 7, 0);
(xt-gdb) c
Continuing.
```

NOTE

You can use the gdb command ‘set substitute-path’ to map the missing symbols from the toolchain libraries. For example:

```
set substitute-path /home/xpgcust/tree/RI-2020.5/ib/tools/swtools-x86_64-linux
~/xtensa/tools/RI-2020.5-linux/XtensaTools
```

For more information on xt-gdb, see the Cadence GNU Debugger User’s Guide and the Cadence Xtensa Debug Guide. The documents are available at:

- `~/xtensa/XtDevTools/downloads/RI-2020.5/docs/gnu_gdb_ug.pdf`
- `~/xtensa/XtDevTools/downloads/RI-2020.5/docs/xtensa_debug_guide.pdf`

6 HiFi4 System Programming

This section provides more examples, tips, and some best practices about HiFi4 programming on RT600 EVKs. It focuses more on RT6xx and SDK. For general HiFi programming, see the Xtensa IDE documents. You can find them in the directory **Xtensa Xplorer IDE menu Help > PDF Documentation**.

The following are some frequently used references:

- Xtensa Instruction Set Architecture (ISA) Reference Manual: Architecture/ high-level overview.
- HiFi 4 DSP User's Guide: Most useful reference manual for DSP programmer. It has all details about HiFi4 instructions and intrinsics, as well as some algorithm optimization techniques.
- Xtensa XOS Reference Manual: XOS is the default & native embedded kernel for Xtensa HiFi cores. SDK examples use XOS as well.
- Xtensa System Software Reference Manual: XTOS, also known as the basic runtime and handlers, has been depreciated and moved towards XOS. However, the Xtensa Processor Hardware Abstraction Layer/ HAL is still very useful in many perspectives. SDK examples use HAL functions.
- Xtensa Linker Support Packages (LSPs) Reference Manual and GNU Linker User's Guide: Both documents are useful for understanding the HiFi program linker and the memory map.

Specifically for RT6xx, the user manual and the datasheet are the most important references. All product specifications are found in the user guide and datasheet documents.

6.1 HiFi4 Boot Loader and Memory Map

This section provides an overview of the HiFi4 Boot Loader, Linker and Memory Map, Cache and Data Exchange Memory Partitions, and Boot or Run from Flash.

6.1.1 HiFi4 Boot Loader

For better power efficiency, by default the HiFi4 DSP is powered off when RT6xx powers up. To boot up, Cortex M33 acts as master core to configure DSP local memories, clocks, load DSP images, and so on. SDK has wrapped up this part as HiFi4 Boot Loader or the DSP driver. The essentials are located in: `<SDK path>\devices\MIMXRT685S\drivers\fsl_dsp.h & fsl_dsp.c.`

For each DSP example, it also provides two more implementation files for detailed configurations: `<SDK path>\boards\evkmimxrt685\dsp_examples\<any dsp example>\cm33\dsp_support.h & dsp_support.c`

Below is the DSP boot procedure with more elaborations.

To run DSP at full power, for example, 600 MHz, SoC Vddcore is set to 1.1 V. If full power is not required, for example, to run DSP at the half speed at 300 Mhz, then Vddcore only requires 0.8 V. RT6xx EVK integrates NXP PCA9420 PMIC for power management and by default Vddcore is set to 1.0 V.

Therefore, PMIC is initialized for better power management. For details on Vddcore and DSP frequency operating conditions, see any `dsp example\cm33\pmic_support.c BOARD_SetPmicVoltageForFreq()`, or see the Datasheet, section 13.1, General Operating Conditions.

```
/* Initialize PMIC PCA9420 */
BOARD_InitPmic();
/* Configure PMIC Vddcore value according to main/dsp clock. */
BOARD_SetPmicVoltageForFreq(CLOCK_GetMainClkFreq(), CLK_600MHZ);
```

It is possible to clock DSP from various clock sources. The DSP PLL for full power and can also run at FFRO low-frequency clocks to save the power.

```
/* Enable DSP PLL clock 594MHz. */
CLOCK_InitSysPfd(kCLOCK_Pfd1, 16);
/*Let DSP run on DSP PLL clock with divider 1 (594Mhz). */
CLOCK_AttachClk(kDSP_PLL_to_DSP_MAIN_CLK);
CLOCK_SetClkDiv(kCLOCK_DivDspCpuClk, 1);
```

As Cortex M33 and SRAM are clocked at lower speed/ max frequency at 300 MHz, set the DSP AHB bus clock divider as 2.

```
CLOCK_SetClkDiv(kCLOCK_DivDspRamClk, 2);
```

If DSP clock is running at 300 MHz or lower, it is more efficient to use divider as 1. For divider as 1, note that an extra register SYSCTL0_PACKERENABLE is set. For more details, see the user manual section 4.5.2.18 DSP Main Ram Clock Divider and section 4.5.5.3 Packer Enable.

```
/* This is a quick register setting example for secure mode */
/* SYSCTL0->PACKERENABLE = 0x4 */
```

Power up TCM/ DSP local memories and cache, supply clock, and reset peripherals.

```
/* Initializing DSP core */
DSP_Init();
```

For SDK DSP examples, split the DSP images into three parts.

One is for vectors and critical sections sitting on TCM/ DSP local memories. The second one is for normal code and data sections sitting on SRAM, and the final is for non-cached DSP initialized data in SRAM.

Here, Cortex M33 load those binaries to its destination. When the DSP program is debugged, it is possible to load DSP binaries from the Xtensa Xplorer IDE, as described in Prepare DSP Core for 'Hello World'. To load binaries, remove the

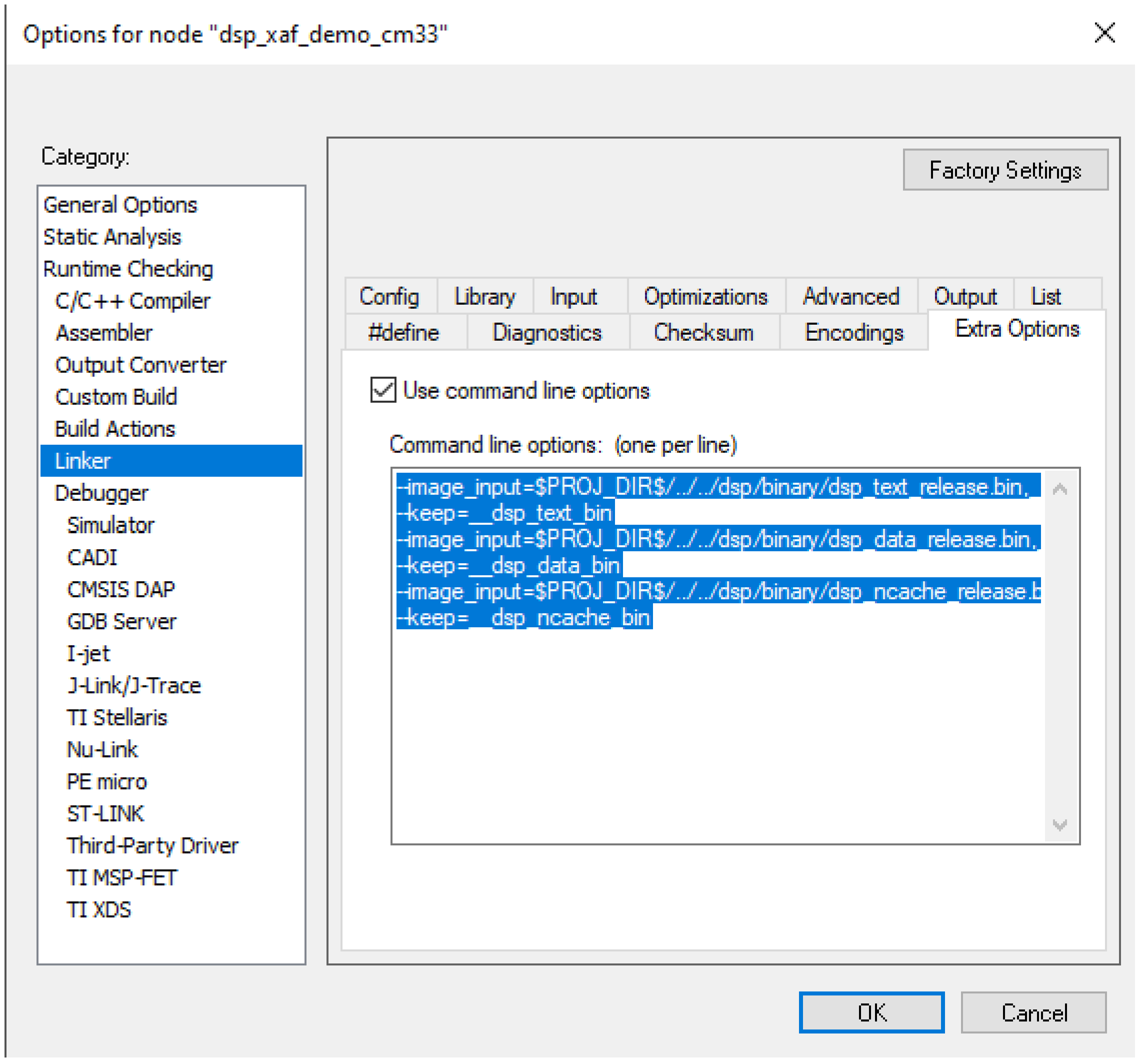
DSP_IMAGE_COPY_TO_RAM compilation flag or set it to 0. By default, the compilation flag is set to 1 and always load the DSP images.

```
#if DSP_IMAGE_COPY_TO_RAM
    /* Copy application from RAM to DSP_TCM */
    DSP_CopyImage(&tcm_image);

    /* Copy application from RAM to DSP_RAM */
    DSP_CopyImage(&sram_image);

    /* Copy application from RAM to DSP_Uncached RAM */
    DSP_CopyImage(&ncache_image);
#endif
```

The DSP stall register SYSCTL0_DSPSTALL controls the HiFi4 operation. Start the DSP and run it.

```
/* Run DSP core */
DSP_Start();
```

The post build scripts create the DSP images. For details, see Makefile.include in any DSP example. To reduce the image size and make image copy more efficient, they are split into SRAM part; TCM part, and uncached SRAM part. DSP images are set and linked into Cortex M33 side. For IAR, the linker is set in **Project Options > Linker > Extra Options**. For ArmGCC or Linux environment, set in any `DSP example\cm33\incbin_gcc.S`.



☑ Use command line options

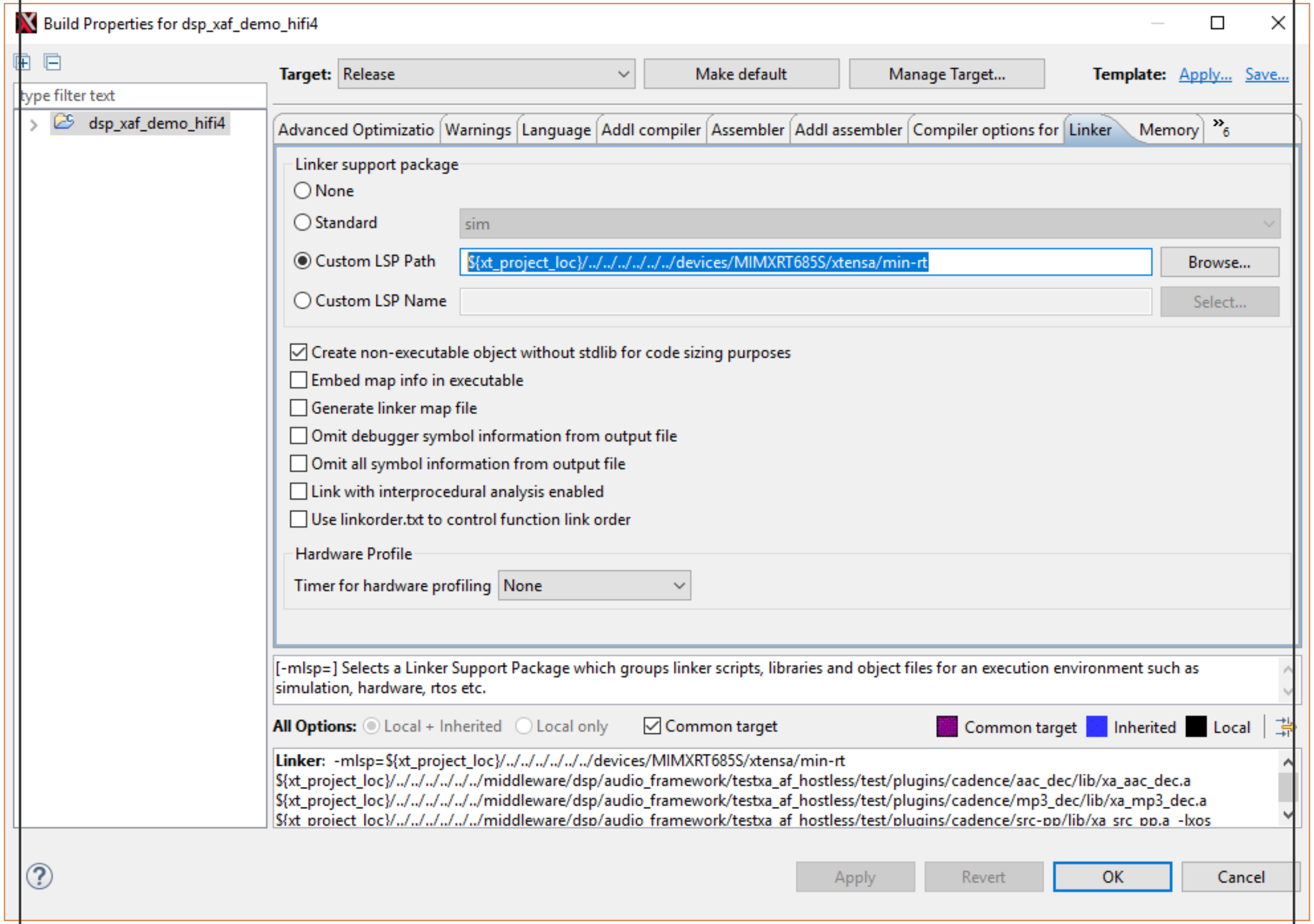
Command line options: (one per line)

```
--image_input=$PROJ_DIR$/../../dsp/binary/dsp_text_release.bin,
--keep=__dsp_text_bin
--image_input=$PROJ_DIR$/../../dsp/binary/dsp_data_release.bin,
--keep=__dsp_data_bin
--image_input=$PROJ_DIR$/../../dsp/binary/dsp_ncache_release.b
--keep=__dsp_ncache_bin
```

Figure 37. Linker options

6.1.2 Linker and Memory Map

When importing the SDK DSP examples, by default the mode is set as Release. The default mode means that the images must be built in Release mode with ‘min-rt’ Linker Support Package/ LSP. To double-check, open **SDK DSP examples > Build Properties > Linker**.



Target: Release

Linker support package

None
Standard sim
(●) Custom LSP Path ${xt_project_loc}/../../../../../devices/MIMXRT685S/xtensa/min-rt
Custom LSP Name

☑ Create non-executable object without stdlib for code sizing purposes
☐ Embed map info in executable
☐ Generate linker map file
☐ Omit debugger symbol information from output file
☐ Omit all symbol information from output file
☐ Link with interprocedural analysis enabled
☐ Use linkorder.txt to control function link order

Hardware Profile

Timer for hardware profiling: None

[-mlsp=] Selects a Linker Support Package which groups linker scripts, libraries and object files for an execution environment such as simulation, hardware, rtos etc.

All Options: Local + Inherited Local only ☑ Common target   ■ Common target ■ Inherited ■ Local

```
Linker: -mlsp=${xt_project_loc}/../../../../../devices/MIMXRT685S/xtensa/min-rt
${xt_project_loc}/../../../../../middleware/dsp/audio_framework/testxa_af_hostless/test/plugins/cadence/aac_dec/lib/xa_aac_dec.a
${xt_project_loc}/../../../../../middleware/dsp/audio_framework/testxa_af_hostless/test/plugins/cadence/mp3_dec/lib/xa_mp3_dec.a
${xt_project_loc}/../../../../../middleware/dsp/audio_framework/testxa_af_hostless/test/plugins/cadence/src-pp/lib/xa_src_pp.a -lxos
```

Figure 38. Linker option

SDK provides three different LSPs.

- 'min-rt' for Release mode - 'min-rt' eliminates all unnecessary debug info and reduces image size.
- 'gdbio' for Debug mode - gdbio' support standard 'printf'/ log output back to Xtensa Xplorer debug console, as well as other debug utilities, perfect for debug purpose but not appropriate for official deployment nor loading directly from Cortex M33 side.
- 'sim' for simulations - 'sim' only works for software simulation and does not fit on device debugging.

The memory map is identical for different LSPs. It sits with linker scripts in `SDK path\devices\MIMXRT685S\xtensa\'LSP name'\ldscripts\elf32xtensa.x`. It specifies how HiFi4 DSP organizes image sections on the memory. For example:

- 0x0020 0000 ~ 0x0048 0000, size 2.5 M bytes, for code and data.
- Stack and Heap are located at the top of the segment and count top down from 0x0048 0000 to lower.
- 0x2400 0000 ~ 0x2400 FFFF, size 64 K bytes, for Data TCM. By default, it is empty.
- 0x2402 0000 ~ 0x2402 FFFF, size 64 K bytes, for instruction TCM. By default it only contains essential vectors and left around 62 K for applications.
- 0x2004 0000 ~ 0x2007 FFFF, size 256 K bytes. This is non-cached area for Cortex M33 and HiFi4 DSP data exchange.

Note that both Cortex M33 and HiFi4 DSP have access to all SRAM partitions. It means that a unified memory map is necessary at system level and both cores must not affect each other's memory map. For SDK examples, see that HiFi4 memory map starts from 0x0020 0000 and Cortex M33 side sits under this address. Using IAR environment, as an example, its memory map sits in `SDK path\boards\evkmimxrt685\dsp_examples\ any example\cm33\iar\MIMXRT685Sxxxx_ram.icf`.

- 0x0008 0000 ~ 0x0017 FFFF, for interrupt vectors and code.
- 0x0018 0000 ~ 0x001F FFFF, for data.

The memory map is flexible and can be adjusted as per the requirement of the application. Note that modifying the core's memory map might affect another. Changes to both the cores must be made accordingly. For example, when allocating more SRAM partitions to DSP, you must reduce the memory taken at Cortex M33 side. Otherwise, Cortex M33 might not work properly. Also, when loading DSP image directly from Cortex M33 side, the image still sits in Cortex M33 data section before booting up. Therefore, it raises the bar for application data section requirements. You may consider running the application from FLASH.

6.1.3 Cache and Data Exchange Memory Partitions

You may have noticed that HiFi4 DSP has a small non-cached area that starts from 0x20040000. The non-cached are used for data exchange between two cores. As both M33 and HiFi DSP have shared access to all SRAM partitions, shared memory access is the most effective way to exchange data between two cores. the given physical addresses are read/ written by both cores at same address, no memory mapping or address converting is required. For example, if a piece of data array is passed from Cortex M33 to HiFi4 DSP, only the start pointer and the size of the array is passed, and conversely. It is convenient for system programming and simplifies the inter-core communications.

Consider cache here. Cortex M33 has no cache, the entire SRAM is considered as its local memory. Therefore, any memory write is flushed immediately. HiFi4 has 32 K instruction cache and 64 K data cache, and both cache are enabled by default. Therefore, the memory write is not flushed immediately. To make a tradeoff between performance and IPC convenience, set the non-cached area for data exchange memory partitions.

The above memory map has specified the non-cached region, and in DSP code, and HAL functions are called to disable the cache. For details, see the audio framework example in `SDK path\boards\evkmimxrt685\dsp_examples\xaf_demo\dsp\xaf_main_dsp.c`. For more details about HAL cache function, see the Xtensa System Software Reference Manual, section 3.11 Cache

```
/* Disable DSP cache for RPMsg-Lite shared memory. */
xthal_set_region_attribute((void *)RPMSG_LITE_SHMEM_BASE, RPMSG_LITE_SHMEM_SIZE, XCHAL_CA_BYPASS, 0);

/* Disable DSP cache for noncacheable sections. */
xthal_set_region_attribute((uint32_t *)&NonCacheable_start,
                               (uint32_t)&NonCacheable_end - (uint32_t)&NonCacheable_start,
XCHAL_CA_BYPASS, 0);
```

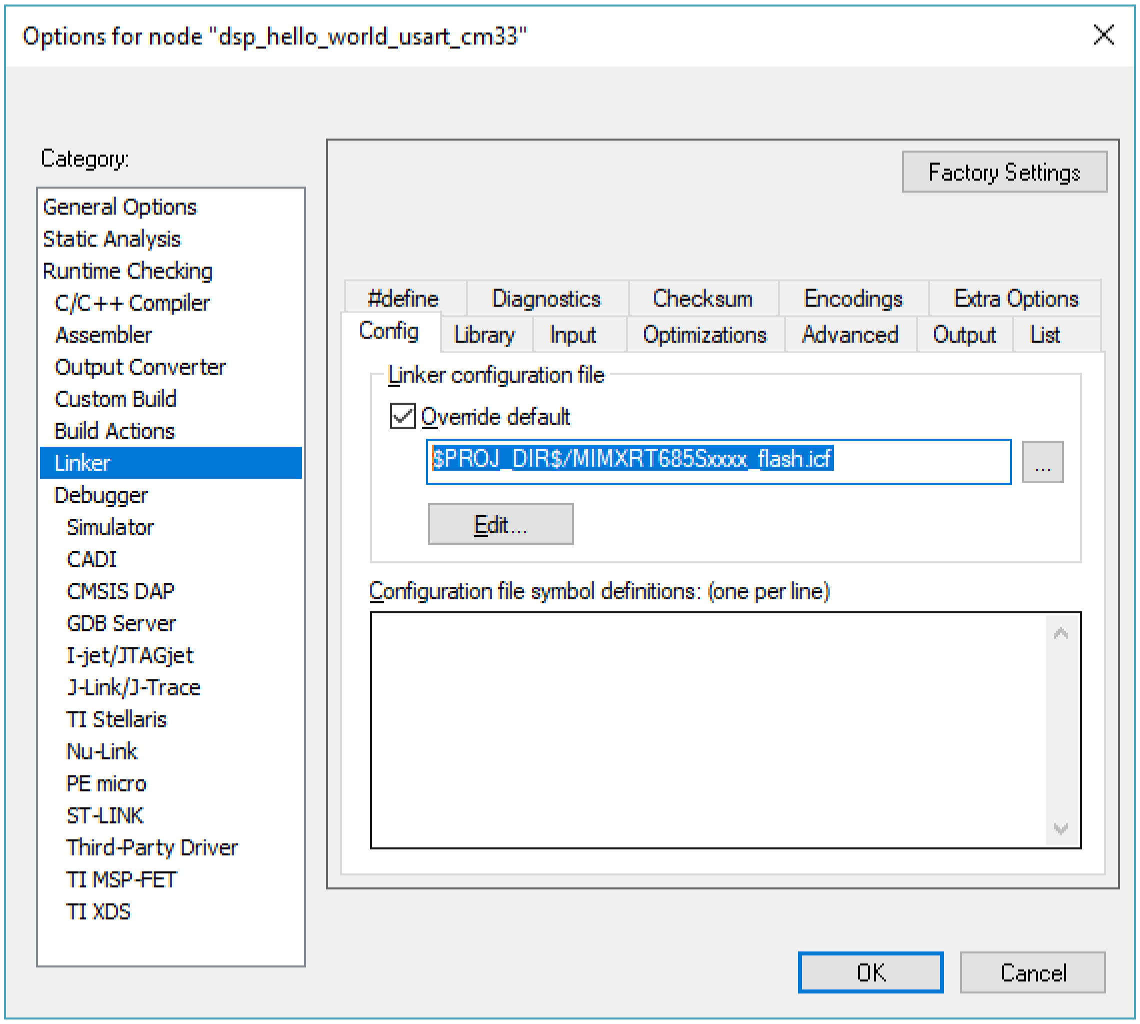
Note that the XHAL call sets cache attribute of the whole region/ 512 M bytes even if the set size is passed. This is also one reason why non-cached attribute is set on the overlapping SRAM address and starts from 0x2000 0000. This also distinguished the physical SRAM addresses starting from 0x0000 0000, which is cacheable area for HiFi4.

Data exchange memory partitions are flexible and can be configured as per application's requirement. However, to mitigate the possible AHB arbitration between the two cores, use of the first eight 32 K and following four 64 K memory partitions is recommended. DSP Data TCM is also used as data exchange area for those data have high demand on timing performance. You must avoid accessing same partition at same time for frequent data exchange between two cores. You can keep one core in Sleep or Wait for Interrupt while another core operating, or set up a ping pong data exchange/ DMA such that when one core fills one partition, another core fetches another partition, and conversely.

For more details about the RT6xx memory map, see the user manual, section 2 Memory Map, and section 2.1.11 HiFi4 memory map.

6.1.4 Boot or Run from Flash

Boot from Flash is straight forward when using IAR environment. SDK provides either two or four different build configurations based on the project: debug (from SRAM)/ release (from SRAM)/ flash_debug/ flash_release. Flash configurations use different memory map in project linker options. For details, see Figure 39.



Linker configuration file

☑ Override default

$PROJ_DIR$/MIMXRT685Sxxxx_flash.icf

Configuration file symbol definitions: (one per line)

Figure 39. Linker settings

To enable booting from flash, change the ISP mode/ SW5 switches on the EVK.

If using MCUXpresso/ armgcc build environment, note that by default build environment is set to boot from flash. Using MCUXpresso as an example:

1. Make sure to use the latest version of MCUXpresso.
2. Import the SDK examples.
3. Once completed, double click the last file/ J-Link Debug.

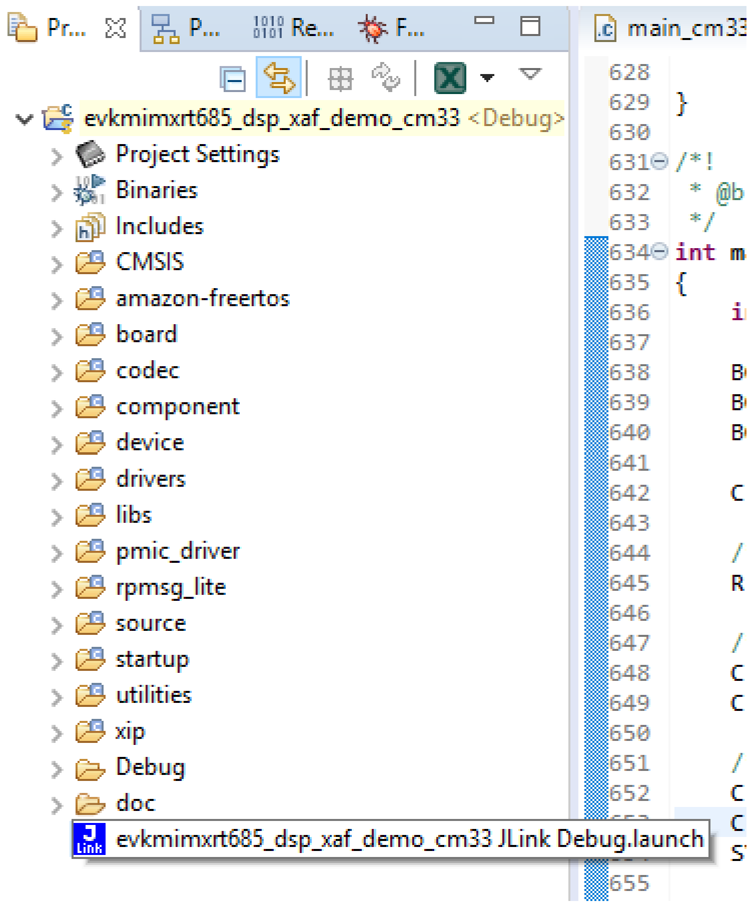
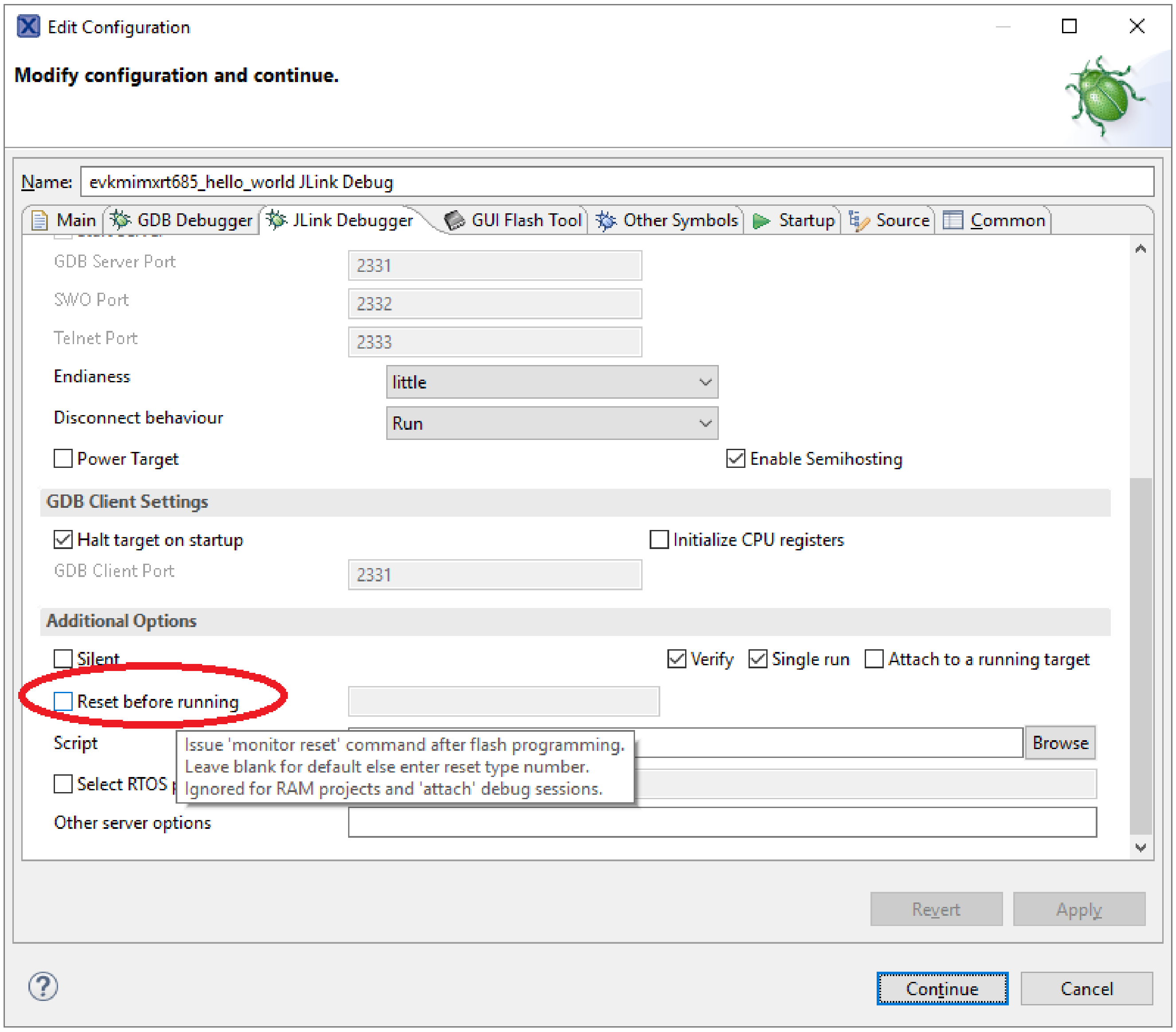


Figure 40. DOuble click J-Link Debug

4. Launch to modify J-Link debugger setting.
5. Make sure to deselect **Reset before running**. This helps the flash-based program get into main function.


Edit Configuration

Modify configuration and continue.

Name: evkmimxrt685_hello_world JLink Debug

Main | GDB Debugger | JLink Debugger | GUI Flash Tool | Other Symbols | Startup | Source | Common

GDB Server Port: 2331

SWO Port: 2332

Telnet Port: 2333

Endianess: little

Disconnect behaviour: Run

☐ Power Target ☑ Enable Semihosting

GDB Client Settings

☑ Halt target on startup ☐ Initialize CPU registers

GDB Client Port: 2331

Additional Options

☐ Silent ☑ Verify ☑ Single run ☐ Attach to a running target

☐ Reset before running

Issue 'monitor reset' command after flash programming.
Leave blank for default else enter reset type number.
Ignored for RAM projects and 'attach' debug sessions.

Script

☐ Select RTOS

Other server options

Figure 41. Modify configuration

- Make sure to:
  - — Modify DSP_IMAGE_COPY_TO_RAM and Define to 1 in **Project settings > C/C++ General > Paths and Symbols > Symbols**.
  - — Have the right compilation flag as C/C++. Compilation flags do not work on *.S files.
  - — Have #define DSP_IMAGE_COPY_TO_RAM 1 as the first line of source/incbin.S to include the DSP binaries.
  - — Use the correct DSP binaries at correct path (must be release binaries).
  - — Provide the right image path to incbin.S.
- Rebuild and debug. For debug/ first run, make sure you are using SW5 switch ISP0 ON, ISP2 off to enable flash write.
- For further runs/ boot from flash, switch ISP0 off ISP2 ON to boot from flash.

6.2 Peripheral Drivers and Interrupts

This section provides information on the basics of peripheral drivers, interrupts, DMA, and how peripheral drivers work on HiFi4 DSP.

6.2.1 Basics

As both cores use shared SRAM and all digital peripherals, SDK drivers work the same on HiFi4 DSP. User Manual section 1.3 Block Diagram shows the system architect on this perspective. For applications, it is the same to use SDK drivers on HiFi4 DSP with Cortex M33 core. SDK has a DSP hello world UART example showing how easy to use UART on DSP side, same as Cortex M33 programming:

```
#include <xtensa/config/core.h>
#include "fsl_debug_console.h"

    /* Init board hardware. */
    BOARD_InitDebugConsole();

    PRINTF("Hello World running on DSP core '%s', %s_%s\r\n", XCHAL_CORE_ID);
```

Compare with Hello world programming on Cortex M33 side. There are two major differences:

- Pin initialization is not necessary for DSP. It is doable but it is highly recommended managing all pins from Cortex M33 side to make pins all-in-one place. It is also to avoid possible conflicts when you set pins on two different cores;
- Clock setting is not necessary for DSP. It is also doable but same it is highly recommended managing all clock sources all-in-one place;

6.2.2 DMA

RT6xx has two DMA controllers. Each has the same DMA request and trigger input possibilities. The two intended scenarios for their use are:

- One DMA controller (DMA0) is used by the CM33, the other (DMA1) is used by the HiFi4. This case can apply to systems where there is no need to differentiate security between the CPUs or between different tasks.
- One DMA controller (DMA0) is secured and has access to secure spaces and peripherals. The other (DMA1) is not secured and does not have access to secure spaces and peripherals. In this scenario, only the secure code running on the CM33 has access to the secure DMA controller. The other code and the HiFi4 share the non-secure DMA controller (if needed).

HiFi4 DSP always uses DMA1 controller. Again, it is the same to call DMA drivers at HiFi4 DSP side. The DMA destination buffer and DMA descriptor have to be in non-cached area to ensure that each transaction flushes to memory immediately. The below example code showing how to create a DMIC DMA channel:

```
AT_NONCACHEABLE_SECTION_ALIGN(
    static uint8_t s_buffer[BUFFER_SIZE * BUFFER_NUM], 4
);
AT_NONCACHEABLE_SECTION_ALIGN(
dma_descriptor_t s_dmaDescriptorPingpong[2], 16
);

#define DEMO_DMA (DMA1)
#define DEMO_DMIC_RX_CHANNEL DMAREQ_DMIC0

DMA_Init(DEMO_DMA);
DMA_EnableChannel(DEMO_DMA, DEMO_DMIC_RX_CHANNEL);
DMA_SetChannelPriority(DEMO_DMA, DEMO_DMIC_RX_CHANNEL, kDMA_ChannelPriority2);
DMA_CreateHandle(&s_dmicRxDmaHandle, DEMO_DMA, DEMO_DMIC_RX_CHANNEL);
```

Macro `AT_NONCACHEABLE_SECTION_ALIGN` is defined in SDK driver layer and refer to non-cached area specified by memory map.

For more details about DMA, see the User Manual Chapter 11 RT6xx DMA Controller and SDK DMA examples in `<SDK path>\boards\evkmimxrt685\driver_examples\dma`.

For audio peripherals/ DMIC/ I2S DMA, see `<SDK path>\boards\evkmimxrt685\driver_examples\dmic` and `<SDK path>\boards\evkmimxrt685\driver_examples\i2s`.

6.2.3 Interrupts

HiFi4 DSP has total 32 interrupts and 4 interrupt levels. Besides the first five interrupts, the rest interrupt 5 ~ 31 are multiplexed to allow more control over priorities and more general flexibility. To see the full list, see the User Manual section 8.6.3 DSP Interrupt Input Multiplexers.

It only requires few lines of code to enable an interrupt by calling XOS function calls, and peripheral controls are identical as Cortex M33 side. For example, the below code shows how to enable a UTick timer at DSP side:

First, include XOS system header files and libraries to enable necessary XOS functions.

To include libraries, go to Build **Properties > Libraries > Add libraries > Select 'xos' & 'xtutil'** and **OK** to accept.

```
#include <xtensa/config/core.h>
#include <xtensa/xos.h>
```

Set up the UTick callback and delay functions. This part is identical with Cortex M33 side. For Cortex M33 side implementation, see `SDK path\\boards\evkmimxrt685\driver_examples\utick`

```
#define UTICK_TIME_1S                 (1000000UL)
#define EXAMPLE_UTICK                 UTICK0

static volatile bool utickExpired;

static void UTickCallback(void)
{
utickExpired = true;
}

static void UTickDelay(uint32_t usec)
{
/* Set the UTICK timer to wake up the device from reduced power mode */
UTICK_SetTick(EXAMPLE_UTICK, kUTICK_Onetime, usec - 1, UTickCallback);

while (!utickExpired)
{
}

utickExpired = false;
}
```

Initialize the XOS and start UTick timer. The XOS function calls are the differences with Cortex M33 side:

```
/* Initialize XOS thread and start scheduler. Priority 7*/
xos_start_main("main", 7, 0);

/* Init board hardware. */
CLOCK_AttachClk(kLPOSC_to_UTICK_CLK);
CLOCK_EnableClock(kCLOCK_InputMux);
UTICK_Init(EXAMPLE_UTICK);

INPUTMUX_AttachSignal(INPUTMUX, 10U, kINPUTMUX_Utick0ToDspInterrupt);
/* To register interrupt callback */
xos_register_interrupt_handler(15, (XosIntFunc *)UTICK0_DriverIRQHandler, NULL);
```

```
/* To enable the interrupt */
xos_interrupt_enable(15);
while (1)
{
UTickDelay(UTICK_TIME_1S);
PRINTF("DSP UTICK TIMER every 1s\r\n");
}
```

Pay attention to the instant numbers. Pick interrupt 15, which maps to DSP_INT0_SEL10 as a L1 interrupts, lowest priority level. For more details about RT6xx HiFi4 DSP interrupt configuration, see User Manual section 51.7 Interrupt. Hence, for Inputmux, attach UTick interrupt to 10. 15-5=10 as first five interrupts are reserved, not user configurable. To get it to highest priority level, for example L3, configure interrupt as follows.

```
INPUTMUX_AttachSignal(INPUTMUX, 24U,
kINPUTMUX_Utick0ToDspInterrupt);
xos_register_interrupt_handler(29, (XosIntFunc
*)UTICK0_DriverIRQHandler, NULL);
xos_interrupt_enable(29);
```

kINPUTMUX_Utick0ToDspInterrupt specifies DSP interrupt multiplexing value. It has been defined in SDK and matching User Manual section 8.6.3 DSP Interrupt Input Multiplexers.

For more details about XOS interrupt handling, see the Xtensa XOS Reference Manual Chapter 18 Interrupt and Exception Handling, and Chapter 26 Interrupt Handler Restrictions.

6.2.4 Complete Example

To present better how peripheral drivers work on HiFi4 DSP, below list a bare-metal DSP example program that transfers data from DMIC to codec on RT6xx EVKs. The full workspace located in `<SDK path>\boards\evkmimxrt685\dsp_examples\audio_demo_bm`. This example is derived from Cortex M33 driver `<SDK path>\boards\evkmimxrt685\driver_examples\dmic\dmic_i2s_dma` with below slightly modifications to adapt to HiFi4 DSP.

- Move DMA buffer and descriptors into non-cached memory partitions;
- Using DMA1 for DSP;
- Calling XOS functions to enable interrupts.
- This example does not contain any pin mux initializing or clock configurations. See the above Cortex M33 example dmic_i2s_dma to set up Cortex M33 side. Once Cortex M33 side ready, this example is compiled and run same as any other SDK DSP examples.

Connect headphone/earphone on audio out of the board, speak on DMIC, or play song nearby the DMIC, you can hear sound on the left channel of headphone/earphone.

```
    /* Start XOS */
    xos_start_main("main", 7, 0);

    /* Disable DSP cache for noncacheable sections. DMA MUST run on none-cacheable/ cache
bypass area*/
    xthal_set_region_attribute((uint32_t *)&NonCacheable_start,
                               (uint32_t)&NonCacheable_end - (uint32_t)&NonCacheable_start,
XCHAL_CA_BYPASS, 0);
    xthal_set_region_attribute((uint32_t *)&NonCacheable_init_start,
                               (uint32_t)&NonCacheable_init_end -
(uint32_t)&NonCacheable_init_start, XCHAL_CA_BYPASS,
                               0);

    PRINTF("Configure DMA\r\n");
    /* DSP_INT0_SEL18 = DMA1 */
    INPUTMUX_AttachSignal(INPUTMUX, 18U, kINPUTMUX_Dmac1ToDspInterrupt);
    /* Map DMA IRQ handler to INPUTMUX selection DSP_INT0_SEL18
     * EXTINT19 = DSP INT 23 */
    xos_register_interrupt_handler(XCHAL_EXTINT19_NUM, (XosIntFunc *)DMA_IRQHandle, DMA1);
    xos_interrupt_enable(XCHAL_EXTINT19_NUM);

    /* The rest DMA & DMIC operations are identical with CM33 side */
```

6.3 Messaging Unit, Semaphore, and IPC

Message Unit/ MU and Semaphore/ SEMA42 are essential for DSP programming. The inter-core communications a.k.a IPC on RT600 is based on MU and SEMA42. SDK provides three simple examples about standalone MU and SEMA 42. SDK path \boards\evkmimxrt685\dsp_examples\mu_interrupt, mu_polling, and sema42. These bare-metal examples show how IPC work between two cores. Furthermore, the audio framework demo/ SDK path \boards\evkmimxrt685\dsp_examples\xaf_demo is complete. It uses rpmsg_lite as IPC protocol, which based on MU to transfer messages between the two cores.

First Cortex M33 side initializes rpmsg master, main_cm33.c in app_task()

```
g_my_rpmsg = rpmsg_lite_master_init((void *)RPMSG_LITE_SHMEM_BASE, RPMSG_LITE_SHMEM_SIZE,
RPMSG_LITE_LINK_ID, RL_NO_FLAGS);
    g_my_queue = rpmsg_queue_create(g_my_rpmsg);
    g_my_ept   = rpmsg_lite_create_ept(g_my_rpmsg, LOCAL_EPT_ADDR, rpmsg_queue_rx_cb, g_my_queue);
```

Rpmsg initialize MU interrupts for MUA/ master in rpmsg_platform.c platform_init_interrupt()

```
    /* Register ISR to environment layer */
    env_register_isr(vector_id, isr_data);

    env_lock_mutex(platform_lock);

    RL_ASSERT(0 <= isr_counter);
    if (isr_counter == 0)
    {
        MU_EnableInterrupts(APP_MU, (1UL << 27U) >> RPMSG_MU_CHANNEL);
```

For DSP side, it also initializes rpmsg client and tries to hook with the master, in xaf_main_dsp.c:DSP_Main()

```
/* Initialize standard SDK demo application pins */
my_rpmsg = rpmsg_lite_remote_init((void *)RPMSG_LITE_SHMEM_BASE, RPMSG_LITE_LINK_ID, RL_NO_FLAGS,
&rpmsg_ctxt);

while (!rpmsg_lite_is_link_up(my_rpmsg))
{
```

It has rpmsg porting as well, initialize MU interrupts for MUB/ client in rpmsg_platform.c platform_init_interrupt() and enable it in platform_interrupt_enable()

```
xos_register_interrupt_handler(6, MU_B_IRQHandler, ((void *)0));
xos_interrupt_enable(6);
```

Once they hooked up both sides are ready for message exchange. SDK defines various SRTM message structures to pass the commands/ messages. In SDK example, it shows how to pass an MP3 decoding message, a AAC decoding message, a DMIC recording message and so on. Using MP3 decoding message as an example, It fills SRTM structure with input buffer address pointer, input buffer size, output buffer address pointer, output buffer size and so on. And then calling rpmsg to send it to DSP and waiting for the response.

```
    rpmsg_lite_send(g_my_rpmsg, g_my_ept, g_remote_addr, (char *)msg, sizeof(THE_MESSAGE), RL_BLOCK);
    rpmsg_queue_recv(g_my_rpmsg, g_my_queue, (unsigned long *)&g_remote_addr, (char *)msg,
sizeof(THE_MESSAGE), len,
                     RL_BLOCK);
```

DSP side listens to rpmsg event.

```
my_ept = rpmsg_lite_create_ept(my_rpmsg, DSP_EPT_ADDR, my_ept_read_cb, (void
*)&rpmsg_user_data, &my_ept_context);
```

Once received, it handles message and proceeds the command. For MP3 decoding case, it decodes the MP3 data in the input buffer, and flushes the output buffer. It also fills out the SRTM message structure with actual read data size and actual write data size, and eventually send the response message back to Cortex M33 side.

```
/*Send response message*
    /rpmsg_lite_send(my_rpmsg, my_ept, remote_addr, (char *)&msg, sizeof(THE_MESSAGE), RL_DONT_BLOCK);
```

For this example, DSP is considered as a coprocessor and rpmsg has been used for a light-weight IPC for both cores. All modules are open-sourced and could be easily adapted to whatever application needs.

6.4 NatureDSP Library

To facilitate application development on RT6xx HiFi4 DSP, NXP licensed NatureDSP signal processing library and embedded as is in source code format. It could be found in `<SDK path>\middleware\dsp\naturedsp_hifi4`

This is an extensive library, containing the most commonly used signal processing functions: FFT, FIR, vector, matrix, and common mathematics. API and programing guide in `.\doc\NatureDSP_Signal_Library_Reference_HiFi4.pdf`, and `performance data in .\doc\NatureDSP_Signal_Library_Performance_HiFi4.pdf`

As this is a huge library, it is impossible to build an all-in-one example on RT6xx hardware. Fortunately, this library is in source-code format and each function/ filter are wrapped in standalone source file. They could be integrated to any application as needed.

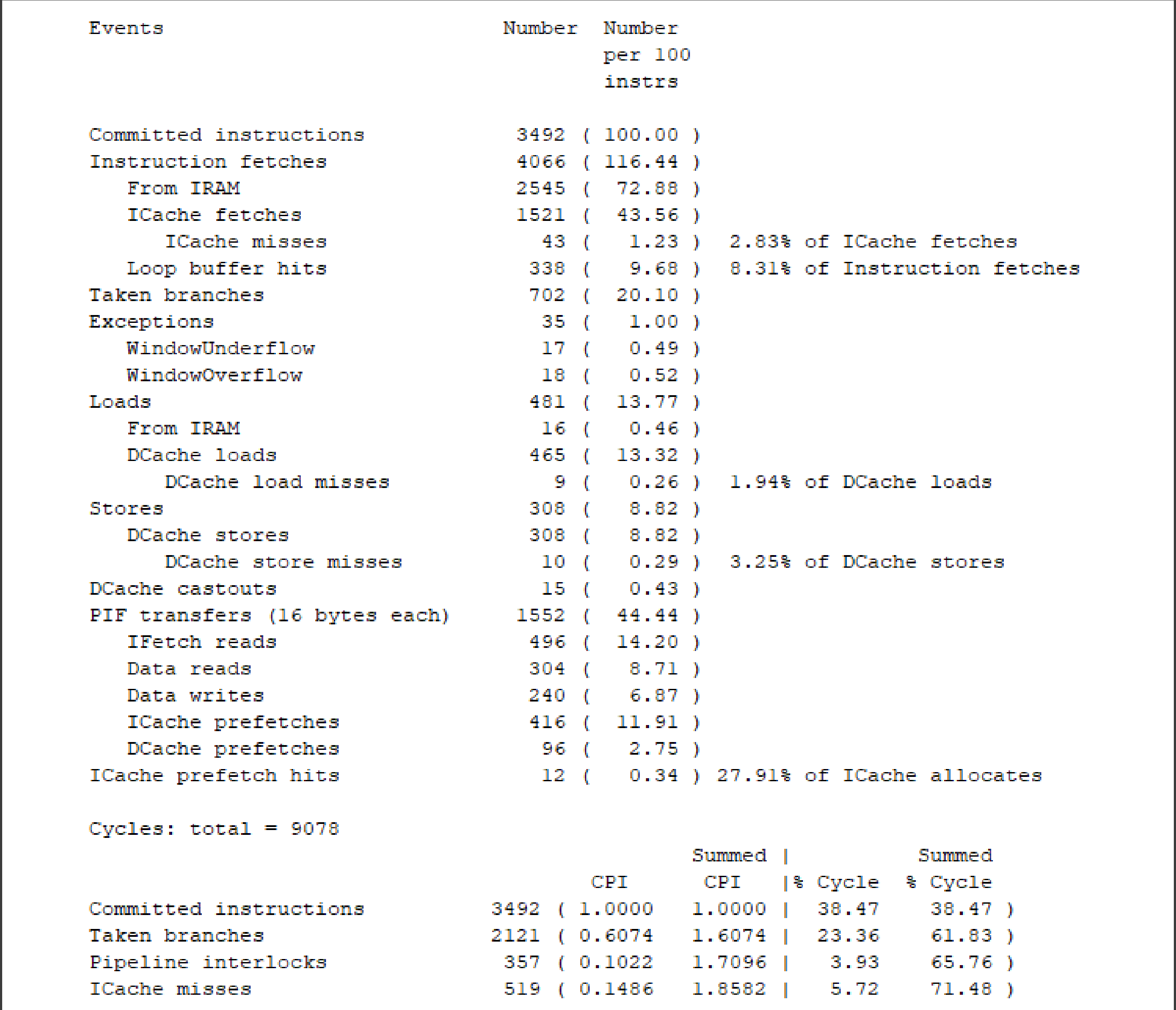
6.5 System Optimization

Performance and power efficiency are key for embedded systems. The following sections list some tips and best practices for system optimization from this perspective.

6.5.1 Profiling

The Xtensa Xplorer IDE tool can run software simulation and profile the application directly. Both simulation and profiling are cycle accurate. This is convenient for algorithm or heavy application developers.

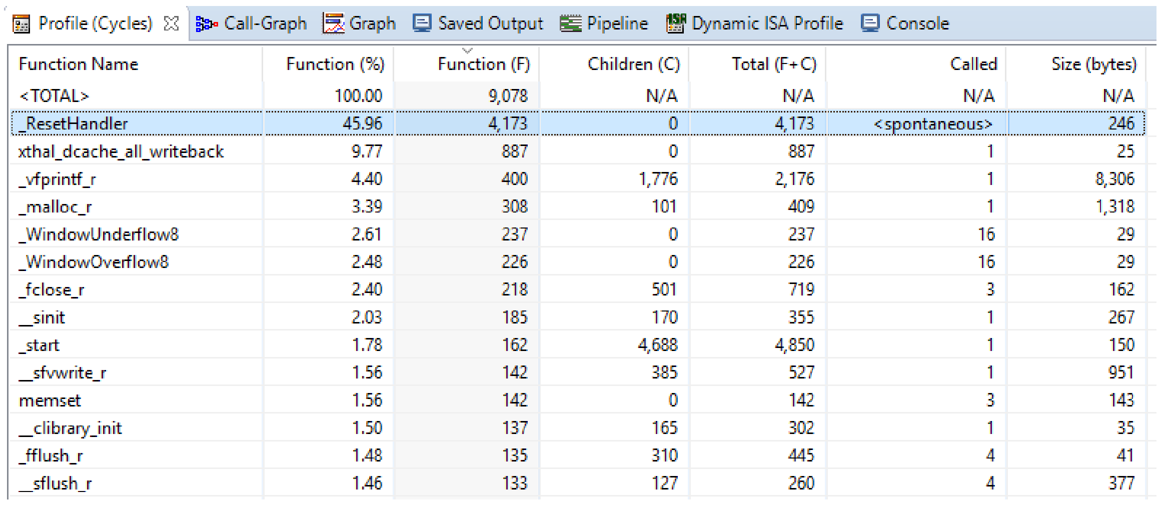
Figure 42 shows profiling result of the helloworld program on simulation console.



```
Events                                Number  Number
                                              per 100
                                              instrs

Committed instructions                 3492 ( 100.00 )
Instruction fetches                    4066 ( 116.44 )
   From IRAM                           2545 (  72.88 )
   ICache fetches                      1521 (  43.56 )
      ICache misses                      43 (   1.23 )  2.83% of ICache fetches
   Loop buffer hits                     338 (   9.68 )  8.31% of Instruction fetches
Taken branches                          702 (  20.10 )
Exceptions                               35 (   1.00 )
   WindowUnderflow                       17 (   0.49 )
   WindowOverflow                        18 (   0.52 )
Loads                                   481 (  13.77 )
   From IRAM                             16 (   0.46 )
   DCache loads                         465 (  13.32 )
      DCache load misses                  9 (   0.26 )  1.94% of DCache loads
Stores                                  308 (   8.82 )
   DCache stores                        308 (   8.82 )
      DCache store misses                10 (   0.29 )  3.25% of DCache stores
DCache castouts                          15 (   0.43 )
PIF transfers (16 bytes each)          1552 (  44.44 )
   IFetch reads                         496 (  14.20 )
   Data reads                           304 (   8.71 )
   Data writes                          240 (   6.87 )
   ICache prefetches                    416 (  11.91 )
   DCache prefetches                     96 (   2.75 )
ICache prefetch hits                     12 (   0.34 ) 27.91% of ICache allocates

Cycles: total = 9078
                                                     Summed |          Summed
                                            CPI       CPI   |% Cycle  % Cycle
Committed instructions               3492 ( 1.0000   1.0000 |  38.47   38.47 )
Taken branches                       2121 ( 0.6074   1.6074 |  23.36   61.83 )
Pipeline interlocks                   357 ( 0.1022   1.7096 |   3.93   65.76 )
ICache misses                         519 ( 0.1486   1.8582 |   5.72   71.48 )
```

Figure 42. Profiling result of the helloworld program

Figure 42 shows the profiling chart, partially.


Profile (Cycles) | Call-Graph | Graph | Saved Output | Pipeline | Dynamic ISA Profile | Console

Function Name	Function (%)	Function (F)	Children (C)	Total (F+C)	Called	Size (bytes)
<TOTAL>	100.00	9,078	N/A	N/A	N/A	N/A
_ResetHandler	45.96	4,173	0	4,173	<spontaneous>	246
xthal_dcache_all_writeback	9.77	887	0	887	1	25
_vfprintf_r	4.40	400	1,776	2,176	1	8,306
_malloc_r	3.39	308	101	409	1	1,318
_WindowUnderflow8	2.61	237	0	237	16	29
_WindowOverflow8	2.48	226	0	226	16	29
_fclose_r	2.40	218	501	719	3	162
__sinit	2.03	185	170	355	1	267
_start	1.78	162	4,688	4,850	1	150
__sfvwrite_r	1.56	142	385	527	1	951
memset	1.56	142	0	142	3	143
__clibrary_init	1.50	137	165	302	1	35
_fflush_r	1.48	135	310	445	4	41
__sflush_r	1.46	133	127	260	4	377

Figure 43. Partial profiling chart

The generated license file only supports debug/run on the RT6xx device target. It does not support software simulation/Xplorer ISS. If there are special requirements to run the software simulations, contact Cadence directly.

It is also common to measure exact cycle counts for specific processing/ timing measurements. The following is an example code to show how to do the cycle counts.

```
/* Cycle counts inline function */
static unsigned long inline get_ccount(void)
{
    unsigned long r;
    __asm__ volatile ("rsr.ccount %0" : "=r" (r));
    return r;
}
tic = get_ccount();
processing_function();
toc = get_ccount();
printf("Processing takes %d cycles \r\n", toc - tic);
```

6.5.2 Using Local Memories

RT6xx HiFi4 DSP has 64 K data TCM and 64 K instruction TCM. The TCM is filled with less than 2 K of kernel vectors and the rest is available for application needs. They are the fastest RAM available with no access latency, and can improve critical data/ instruction performance considerably. Consider using TCM as much as possible.

To program code/ data to TCM area:

1. Define macros for sections in TCM. Reuse existing .drm0 and .iram0 sections. Both sections are default TCM sections for every Xplorer project/ memory map. The following are the sections in fsl_common.h.

   ```
   #define DRAM0_DATA __attribute__((section(".dram0.data")))
   #define DRAM0_BSS __attribute__((section(".dram0.bss")))
   #define IRAM0_TEXT __attribute__((section(".iram0.text")))
   #define ALIGNED(alignbytes) __attribute__((aligned(alignbytes)))
   ```

2. Use macros to declare code and data to be placed in TCM. The data and bss/ uninitialized data are different sections. The following is an example for an FFT function call.

   ```
   DRAM0_DATA const static int32_t fft_in_ref[FFT_LENGTH] = {
   ```

```
DRAM0_DATA const static int32_t fft_out_ref[FFT_LENGTH] = {

DRAM0_BSS static int32_t fft_out[FFT_LENGTH];

IRAM0_TEXT int TEST_FFT()
{
    …
        fft_cplx32x32(fft_out, fft_in, FFT_HANDLE, 3);
    …
}
```

3. The TCM addresses start from 0x2400 0000, which is too far away to main(). Therefore, the project enables long calls to ensure that main() is able to call sub functions. To enable, select **Build Properties > Optimization > Enable long calls** and select the **Yes** checkbox. Alternatively, add -mlongcalls to the compiler flags.

6.5.3 Power Efficiency

RT6xx HiFi4 DSP has been equipped as a powerful processing core that can run at 600 MHz, but it may not be necessary at all time. It is always recommended to optimize the power efficiency at system level.

Voltage level and core frequency have huge impact on the power efficiency. Using FFT processing as example, continuous FFT may take ~130 mW at Vddcore 1.1 V / 452 MHz, and ~4 mW at Vddcore 0.7 V / 29 MHz, at room temperature. Profile or do cycle counts for software workload. If it only requires 300 MHz at peak, then the run at full power is not required. Instead, it can run with Vddcore 0.8 V 300 MHz. For more information on the Vddcore and DSP frequency operating conditions, see any *dsp example\cm33\pmic_support.c BOARD_SetPmicVoltageForFreq()*, or see the datasheet, section 13.1 General Operating Conditions.

Some other tips for better power efficiency:

- Turn off DSP/ set DSPSTALL, if needed.
- If possible, make DSP clock adapts to its workload.
- Call XT_WAITI when DSP is in while loop waiting for interrupt. Similar to Arm side __WFI(), it suspends some processor operations to reduce the power consumption.

  ```
  #include <xtensa/tie/xt_interrupt.h >
   extern void XT_WAITI(immediate s);
  ```

  The immediate value passed is the interrupt level or lower to be IGNORED. For example, if you call XT_WAITI(2) both L1 and L2 interrupts are ignored and only L3 interrupts can wake up DSP.

  Current can further reduce when clocks are turned off to unused memory partitions. For more information, see User Manual 4.5.5.13 DSP SRAM access disable, Register SYSCTL0_DSP_SRAM_ACCESS_DISABLE.
- PLLs consume power. Consider FFRO for low-power use cases.

7 Revision history

This table summarizes revisions to this document.

Table 1. Revision history

Revision number	Date	Substantive changes
0	February 2018	Initial Release
1	15 January 2021	Updated for MCUXpresso SDK v2.9.0
2	10 July 2021	Updated for MCUXpresso SDK v2.10.0

Date of release: 10 July 2021

Document identifier: GSXMIMXRT600UG