Getting Started with Xplorer for MIMXRT600

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.

Contents

1 Install Xplorer Toolchains

1.1 Xtensa Software Tools Platform Support

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

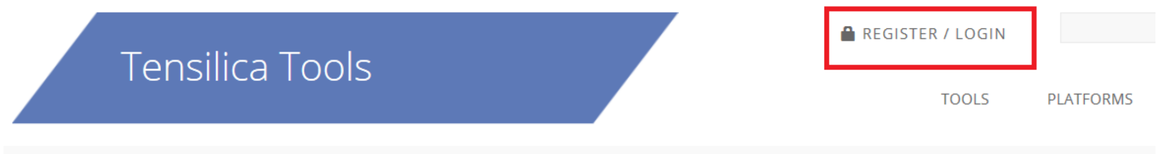
Windows: Win 10 64-bit, Win 8 64-bit, Win 7 64-bit
Linux: RHEL 6 64-bit (with 'Desktop' package installed)

There may be compatibility issues with other versions of Linux or Windows, especially when using the IDE. Also note that security-enhanced Linux (SELinux) is not a supported platform because the OS can prevent different shared libraries (including Xtensa Tools) from loading.

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

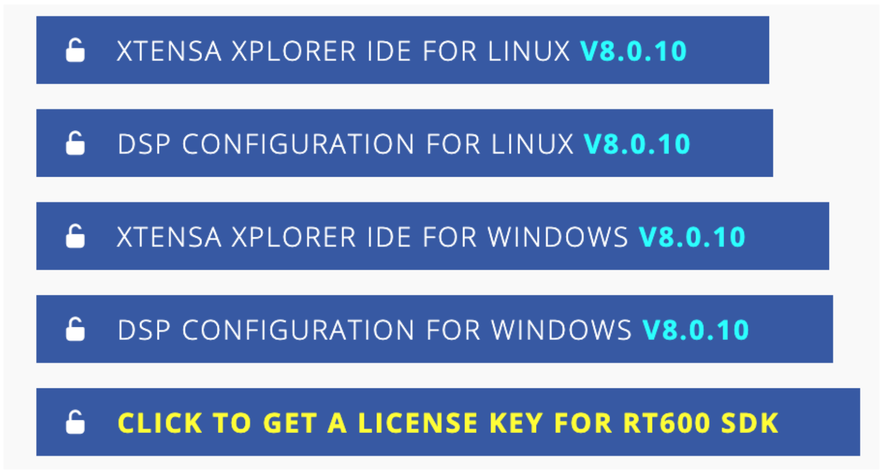
1.2 Install the Xtensa Xplorer IDE and Tools

Go to the URL https://tensilicatools.com/download/rt600-download-page/ and login. If this is the first time to access, please register first. Please make sure to use corporate email address to register.



Once registered you should receive an email confirmation with an activation link from 'Tensilica Tools' no-reply@tensilicatools.com. Please also check the spam folder if this email doesn't show up in the inbox. Please click the activation link to complete the registration.

Once registered please login and you will see available materials for download:


XTENSA XPLORER IDE FOR LINUX V8.0.10
DSP CONFIGURATION FOR LINUX V8.0.10
XTENSA XPLORER IDE FOR WINDOWS V8.0.10
DSP CONFIGURATION FOR WINDOWS V8.0.10
CLICK TO GET A LICENSE KEY FOR RT600 SDK

➔ Download and install the **Xplorer IDE 8.0.10** for your operating system (Windows or Linux).

➔ Download the **DSP Configuration** for your operating system – this will be installed later through the IDE (see section 1.4).

NXP recommends version **8.0.10** of the Xtensa Xplorer IDE and tools for use with the RT600 DSP.

1.3 Install License Key

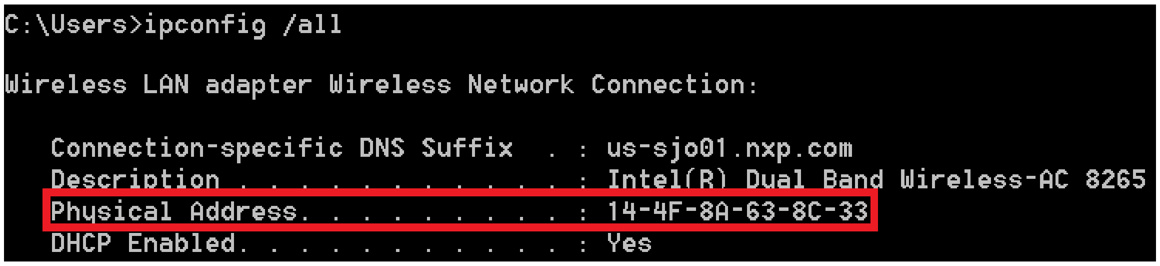
Xtensa development tools use FLEXlm for license management. FLEXlm licensing is required for tools such as the Xtensa Xplorer IDE, Xtensa C and C++ compiler, and Instruction Set Simulator (ISS).

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 that is permanently attached to the machine.

1.3.1 Identify PC MAC Address

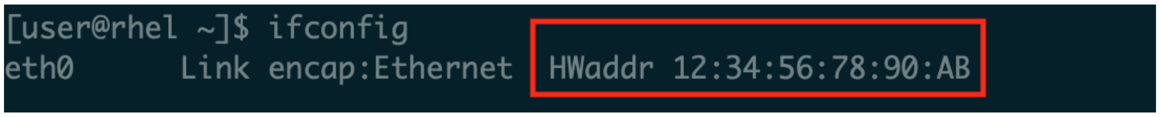
To generate the correct license file, you should first identify the appropriate MAC for the computer you plan to run Xtensa tools on. Please remove '-' or ':' symbols in the MAC address.

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
```

Linux:



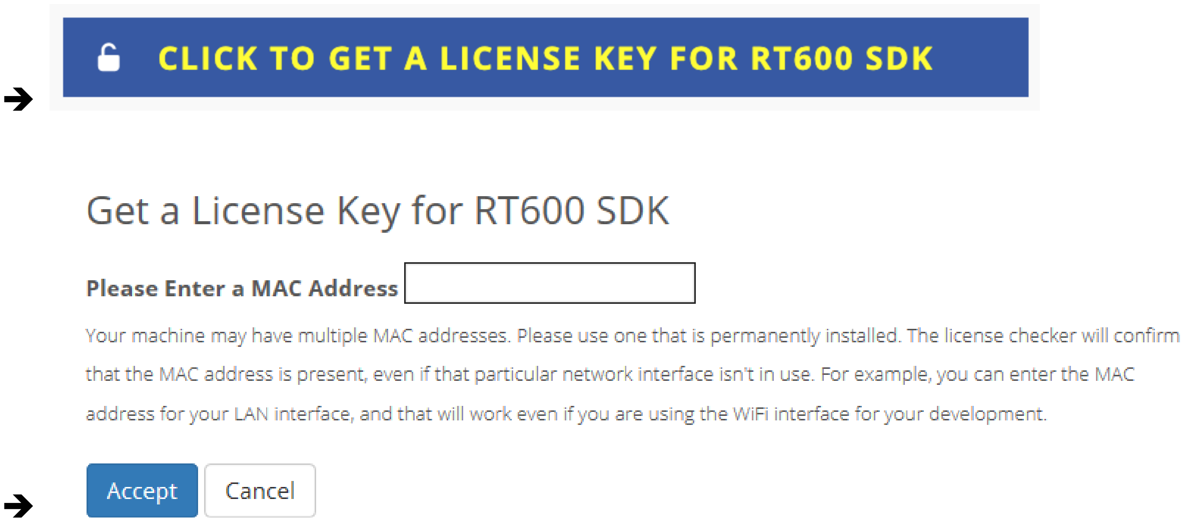
```
[user@rhel ~]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 12:34:56:78:90:AB
```

Linux NOTE: MAC address MUST be associated with eth0 interface. If not, Flexlm cannot perform the license checkout and you will not be able to compile or simulate you code. If your host has the MAC address associated with another interface, for example em1, you may use the following approach, or another approach recommended by your IT team to rename the interface to eth0:

```
# 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
```

1.3.2 Download License Key

Reload or return to the Tensilica URL: https://tensilicatools.com/download/rt600-download-page/

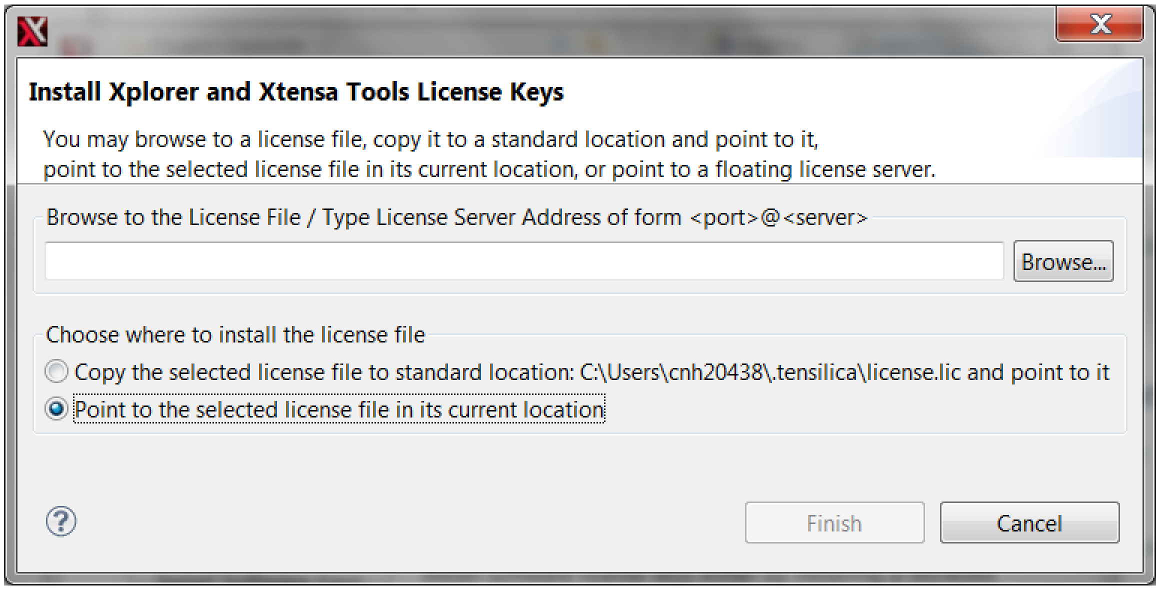


Get a License Key for RT600 SDK

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.

Once the license file has been generated and downloaded, open your recently installed Xplorer 8.0.10, select menu Help -> Xplorer License Keys -> Install Software Keys, select the license key file, and click 'Finish':



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

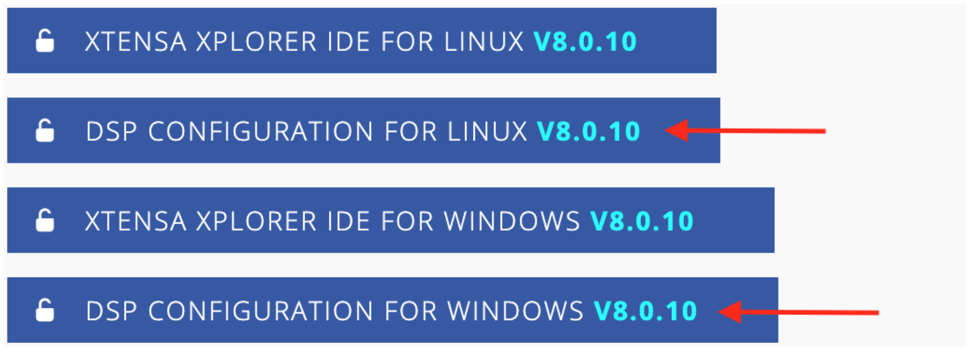
- [ ] Copy the selected license file to standard location: C:\Users\cnh20438\.tensilica\license.lic and point to it
- [✓ Point to the selected license file in its current location]

NOTE: The generated license file only supports debug/run on the RT600 device target. It does not support software simulation/Xplorer ISS. Please contact Cadence directly if you have special needs to run software simulations.

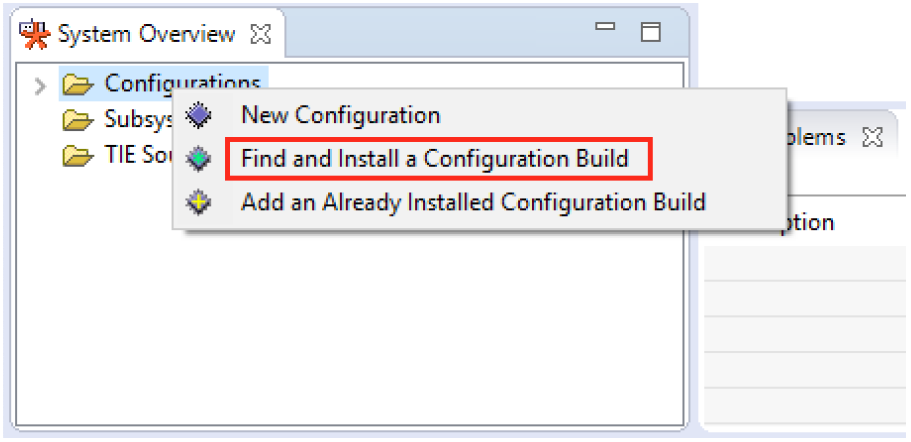
1.4 Install RT600 DSP Build Configuration

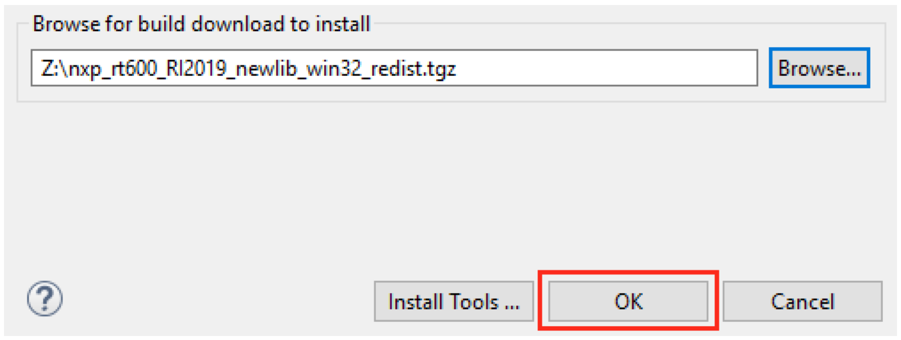
'Build Configuration' is a term that describes all parameters and necessary build includes for the Tensilica processor implementation you are developing with. It's mandatory to install a specific build configuration before starting development on RT600.

The build configuration is provided by NXP as a binary file that can be imported into the Xplorer IDE. This file can be downloaded for your OS from the Tensilica URL:


XTENSA XPLORER IDE FOR LINUX V8.0.10
DSP CONFIGURATION FOR LINUX V8.0.10
XTENSA XPLORER IDE FOR WINDOWS V8.0.10
DSP CONFIGURATION FOR WINDOWS V8.0.10

The build configuration can be installed into the IDE using the 'System Overview' panel which is in the lower left corner by default. If this panel is not visible, it can be toggled using menu item 'Window -> Show View -> System Overview'.


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


Browse for build download to install
Z:\nxp_rt600_RI2019_newlib_win32_redist.tgz
Browse...
Install Tools ...
OK
Cancel

1.5 Install Xtensa On Chip Debugger Daemon

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

Windows:

```
C:\usr\xtensa\XtDevTools\downloads\RI2019.1\tools\xt-ocd-14.0.1-windows64-installer.exe
```

Linux:

```
~/xtensa/XtDevTools/downloads/RI2019.1/tools/xt-ocd-14.0.1-linux64-installer
```

At this moment 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. Please make sure you are using the latest version of J-Link software.

Linux NOTE: When installing xt-ocd on Linux, you must manually add a symlink 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 you will need to modify to fit your debugger hardware. Using J-link as example, please use below content to replace the original template. Please note that you need to replace ‘usbser’ section to your own J-Link serial number (9 digits number on the back of the J-Link hardware).

```
<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 showing another topology.xml example for ARM RealView ICE (RVI) and DSTREAM debug probes:

```
<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! Now you have all Xplorer toolchains installed.

For more details about Xtensa software tools, build configurations, or xt-ocd daemon, please refer to full set of documents in Xplorer menu Help -> PDF Documentation.

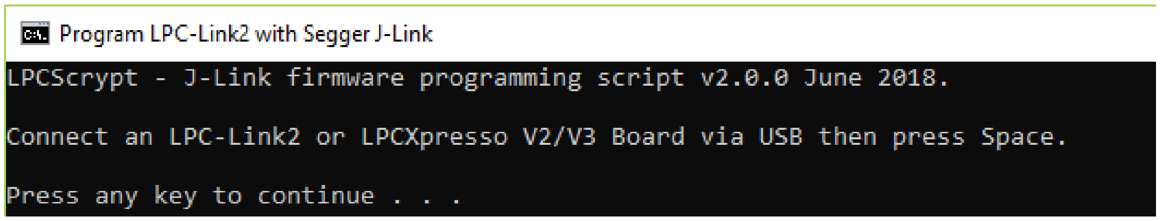
1.6 Program LPC-Link2 with SEGGER J-Link

In addition to standalone probes, the on-board LPC-Link2 debug probe can be used to debug HiFi4 DSP over SWD ports. All RT600 EVKs have a LPC4300 MCU (top right corner on EVK) and by default it has been pre-programmed as CMSIS-DAP probe. CMSIS-DAP is not compatible with HiFi4. We need a few extra steps to flash it with J-Link firmware and make it support both ARM CM33 core and HiFi4 DSP core.

First we need to install LPCScrypt, a command line tool for programming onboard LPC-Link2 debug probe. It could be downloaded from https://www.nxp.com/design/microcontrollers-developer-resources/lpc-microcontroller-utilities/lpcscrypt-v2.1.0:LPCSCRYPT

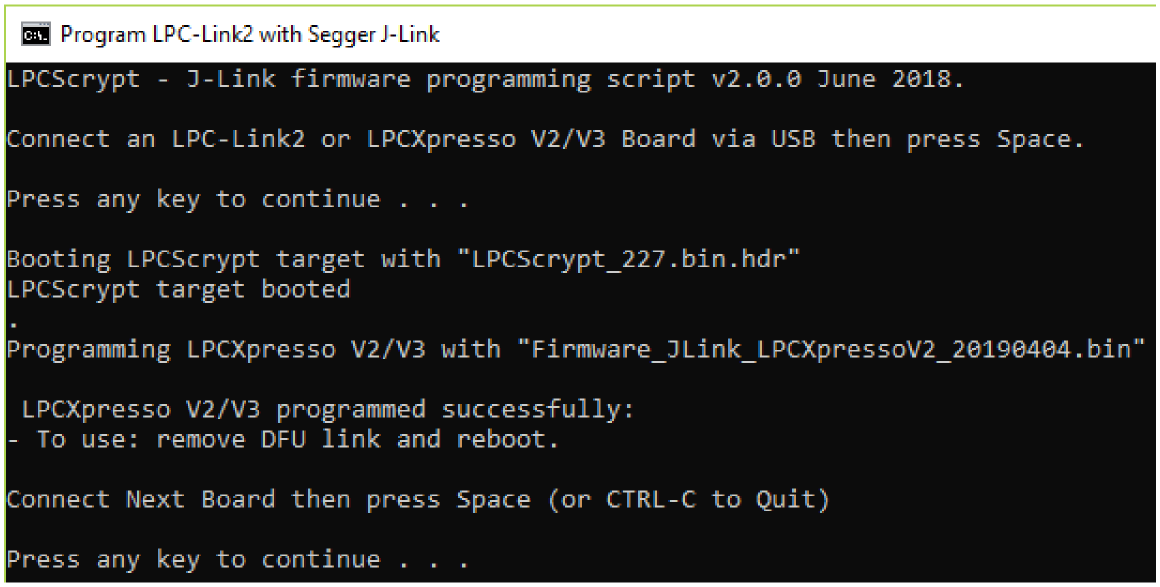
Make sure it is version 2.1.0 or newer. Check install_path\probe_firmware\LPCXpressoV2\Firmware_JLink_LPCXpressoV2_YYYYMMDD.bin make sure the file date is 20190404 or newer.

Then we can reprogram the LPC-Link2 debug probe. Using RT600 EVK as example, keep JP1 jumper (next to top right LPC4300 MCU) short/ connected, then power up EVK through top right USB port J5. Then Run LPCScrypt/ Program LPC-Link2 with Segger J-Link batch command from startup. You will see below message:



```
LPCScrypt - J-Link firmware programming script v2.0.0 June 2018.

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

Press any key to continue . . .
```

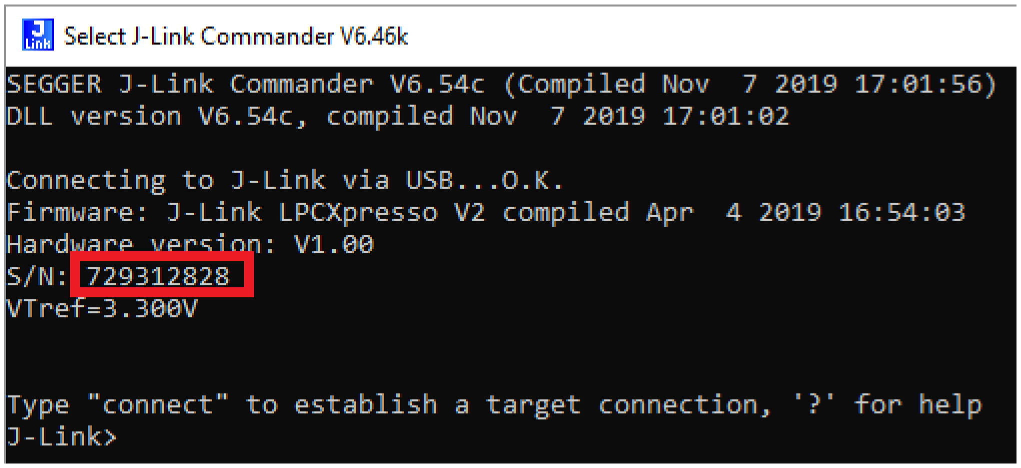
Then you can press any key to program the probe with SEGGER J-Link image



Program LPC-Link2 with Segger J-Link

```
LPCScrypt - J-Link firmware programming script v2.0.0 June 2018.

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

Press any key to continue . . .

Booting LPCScrypt target with "LPCScrypt_227.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 . . .
```

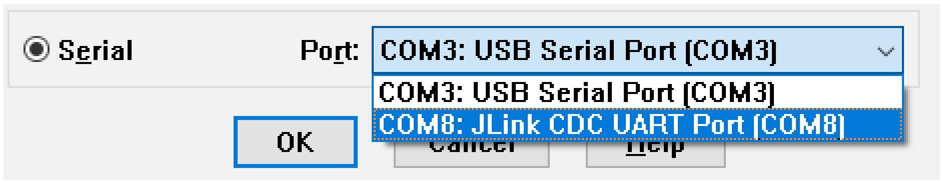
Open/ Disconnect JP1 and power cycle the board. The onboard LPC-Link2 is ready to be used as SEGGER J-Link probe. You can run J-Link Commander to check. Every EVK/ LPC-Link2 will have different J-Link S/N, so please make sure you write down the S/N for xt-ocd & topology.xml.



Select J-Link Commander V6.46k

```
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>
```

Another benefit is that LPC-Link2 debug probe creates a virtual serial port over USB, so you don't need extra UART2USB cable for debugging.


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

The above download link provides document, demo videos and more details about LPC-Link2. If you have any questions, or have difficulties to program the probe, please refer to above link.

1.7 Install Xtensa Software Tools without IDE

The Xtensa Software Tools optionally be installed without the use of the IDE, which may be desired for use 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. The IDE will need to be installed one time in your organization to gain access to the tools package, which is then available at:

```
~/xtensa/XtDevTools/downloads/RI2019.1/tools/XtensaTools_RI_2019_1_linux.tgz
```

With the tools package and the DSP Build Configuration package available from the Tensilica Tools download site (see section 1.1), the toolchain can be setup 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_2019_1_linux.tgz -C ~/xtensa/tools

# Set up the configuration-specific core files:
tar zxvf nxp_rt600_RI2019_newlib_linux_redist.tgz -C ~/xtensa/builds

# Install the Xtensa development toolchain:
cd ~/xtensa
./builds/RI-2019.1-linux/nxp_rt600_RI2019_newlib/install \
  --xtensa-tools./tools/RI-2019.1-linux/XtensaTools \
  --registry ./tools/RI-2019.1-linux/XtensaTools/config
```

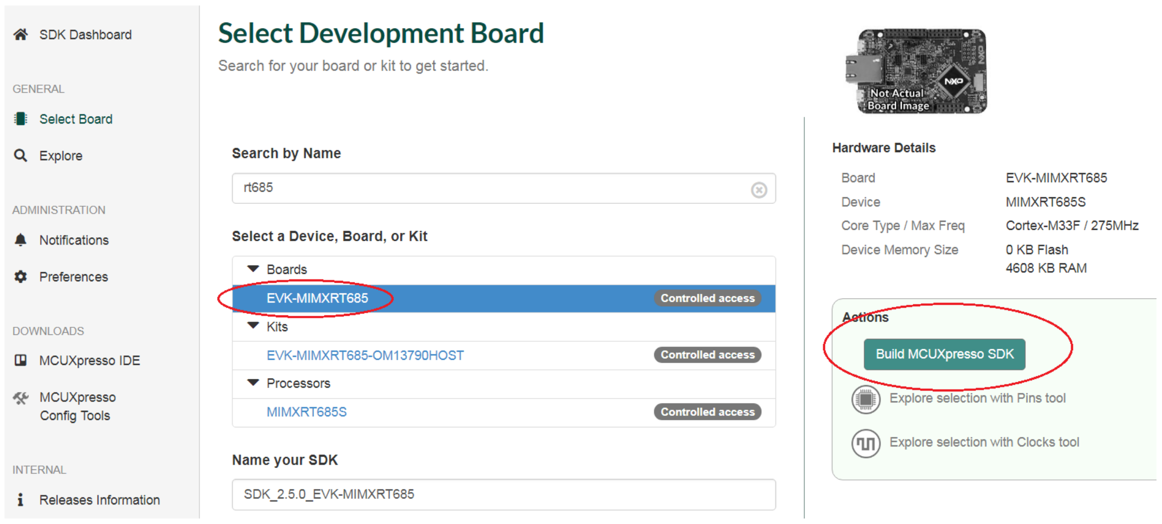
2 Install MCUXpresso SDK

2.1 Download MCUXpresso SDK for RT600

DSP enablement for RT600, including drivers, middleware libraries, and demo applications are included with the latest RT600 SDK which can be downloaded from https://mcuxpresso.nxp.com. If this is your fist time accessing the site, you will need to register first.

Once logged in you can use the SDK builder:

- Click 'Select Board'
- Search by name for board: 'RT685'
- Select 'EVK-MIMXRT685'
- Click 'Build MCUXpresso SDK'



Select Development Board

Search for your board or kit to get started.

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

2.2 MCUXpresso SDK DSP Enablement

Inside the MCUXpresso SDK release package for RT600 you will find the following DSP-specific enablement:

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

Unified device and peripheral driver source code that can be compiled for both ARM and DSP cores. NOTE that only a limited subset of peripheral drivers and components are 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

2.3 DSP Core Initialization

In order to minimize power consumption, the DSP core is NOT powered when RT600 boots up. To run or debug DSP applications, you will first need to execute some code on the ARM core to initialize the DSP.

A DSP management interface library is provided in the SDK, located at <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);
```

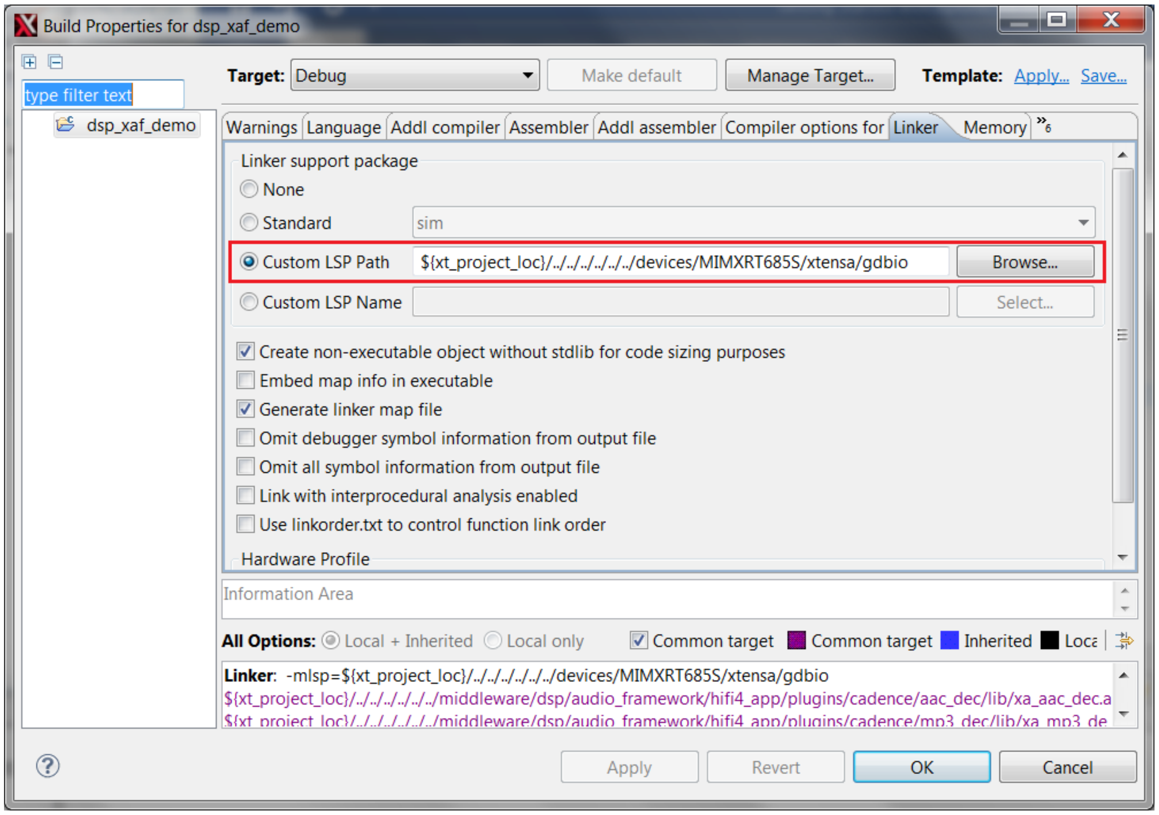
After executing this function during your ARM application startup, the DSP will be initialized and ready to run. From here, code can be loaded and debugged on the DSP with Xplorer IDE and tools.

2.4 DSP Linking 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. These LSPs are provided in the MCUXpresso SDK under <SDK_ROOT>/devices/MIMXRT685S/xtensa/.

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

You can see and change which LSP is being actively used by the project target in the Xplorer IDE in the Linker menu of the project Build Properties:


Build Properties for dsp_xaf_demo
type filter text
dsp_xaf_demo
Target: Debug
Make default
Manage Target...
Template: Apply... Save...
Warnings | Language | Addl compiler | Assembler | Addl assembler | Compiler options for | Linker | Memory
Information Area
Apply
Revert
OK
Cancel

Linker support package

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

- [x] Create non-executable object without stdlib for code sizing purposes
- [ ] Embed map info in executable
- [x] 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
```

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

3 Run and Debug DSP Demo using Xplorer IDE

3.1 Prepare ARM Core for ‘Hello World’

The DSP demos contained in the MCUXpresso SDK each consist 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 section 2.3 and executes other application-specific functionality.

In order to debug the ‘Hello World’ DSP application, you will first need to setup and execute the ARM application using an environment of your choosing:

- Build and execute the ‘Hello World’ ARM demo located at:

```
<SDK_ROOT>/boards/evkmimxrt685/dsp_examples/hello_world_usart/cm33/
```

Preparing an ARM core development environment is outside of the scope of this document. Please refer to the document ‘Getting Started with MCUXpresso SDK for MIMXRT600.pdf’ located under <SDK_ROOT>/docs/ for information on how to use the SDK for ARM core development.

> **NOTE:** IAR Embedded Workbench may require a patch to enable compatibility with RT600. Please contact NXP directly about this patch.

3.2 Start Xtensa Debugger Daemon

To debug DSP applications on RT600 you will need to have the xt-ocd daemon up running. This application runs a gdb server that the Xtensa core debugger will connect to.

Go to the command line window and cd to xt-ocd daemon installation path. By default, it is C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.0.1 on Windows.

Execute the daemon with your custom topology:

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

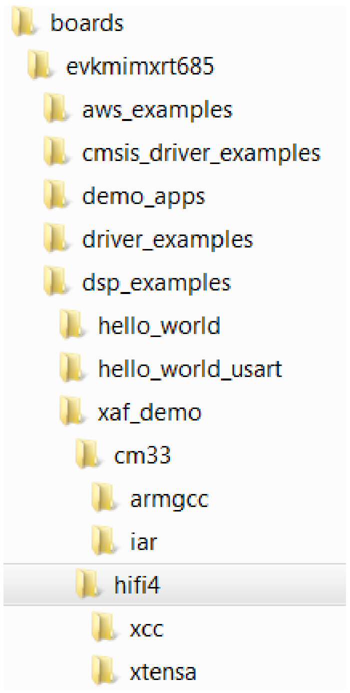
```
XOCD 14.01 2019-05-29 14:29:17
(c) 1999-2019 Cadence Design Systems Inc. All rights reserved.
[Debug Log 2019-07-29 14:25:30]
Loading module "gdbstub" v2.0.0.12
Loading module "jlink" v2.0.2.0
Using JLINK lib v.64607
Jlink USB Serial Number: 600112008
Connected to Jlink Device:
  Name:'SEGGER J-Link ARM'
  S/N:600112008
  Firmware: J-Link V10 compiled Jul 10 2019 16:31:42
  Requested/Set TCK: 1000kHz/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...
```

Please note that some warning messages are expected and can be ignored. If you receive an error initializing the XDM driver, it could be that the DSP core needs to be initialized and started before debugging – see section 2.3 / 2.4 of this document for more details.

Refer to chapter 7 of the Xtensa Debug Guide (available in Help -> PDF Documentation) for more information on xt-ocd runtime options and configuration.

3.3 Prepare DSP Core for 'Hello World'

The RT600 SDK provides a collection of DSP example applications located under 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):

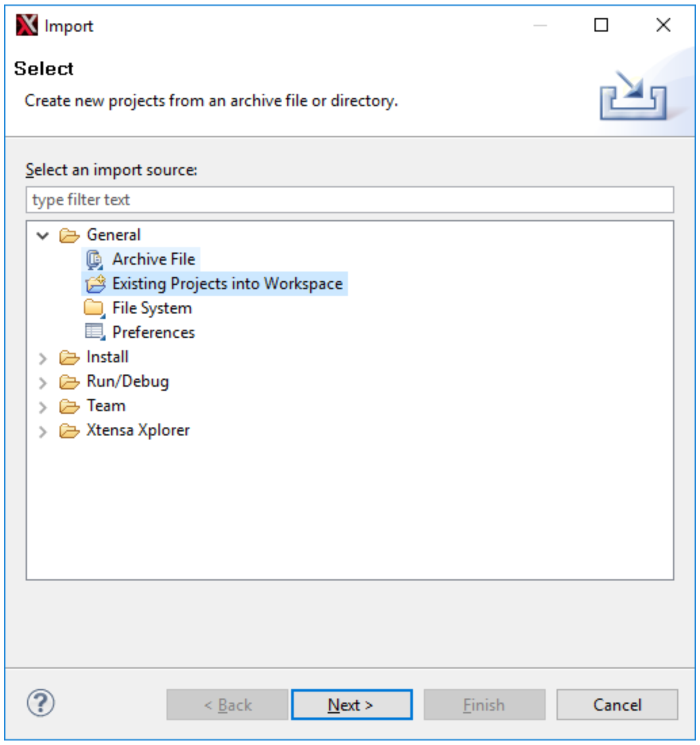


```
boards
  evkmimxrt685
    aws_examples
    cmsis_driver_examples
    demo_apps
    driver_examples
    dsp_examples
      hello_world
      hello_world_usart
      xaf_demo
        cm33
          armgcc
          iar
        hifi4
          xcc
          xtensa
```

Under these directories will be build projects for different supported toolchains. For the DSP example above, the 'xcc' project will allow to build on the command line and the 'xtensa' directory is an Xplorer IDE project.

To run the 'Hello World' demo, first you need to import SDK sources into Xplorer IDE.

- Use menu item File -> Import -> Existing Projects into Workspace
- Select SDK directory <SDK_ROOT>\boards\evkmimxrt685\dsp_examples\hello_world \dsp\xtensa as root directory and leave all other check boxes blank as default:



Import

Select

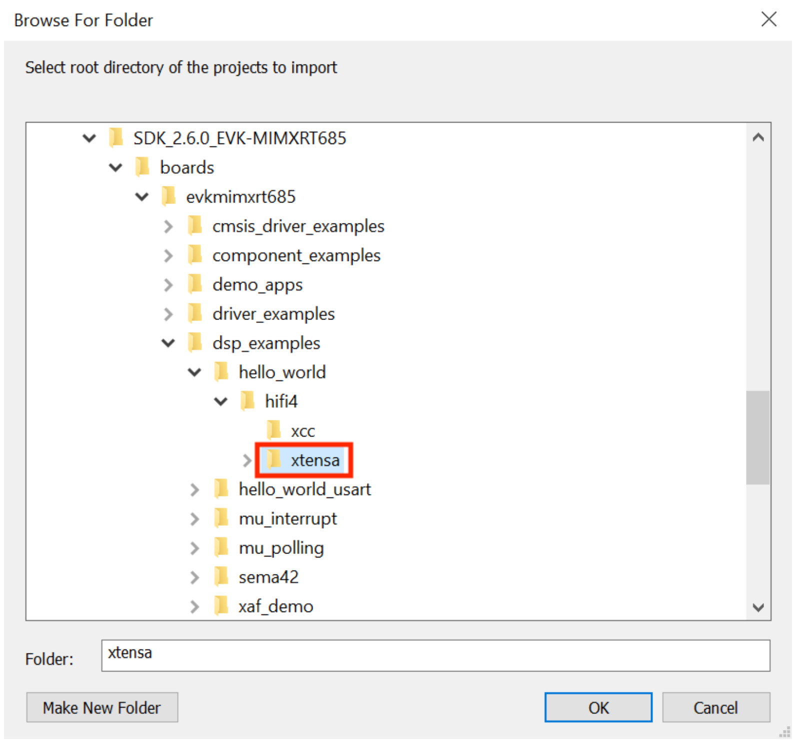
Create new projects from an archive file or directory.

Select an import source:

type filter text

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

< Back | Next > | Finish | Cancel



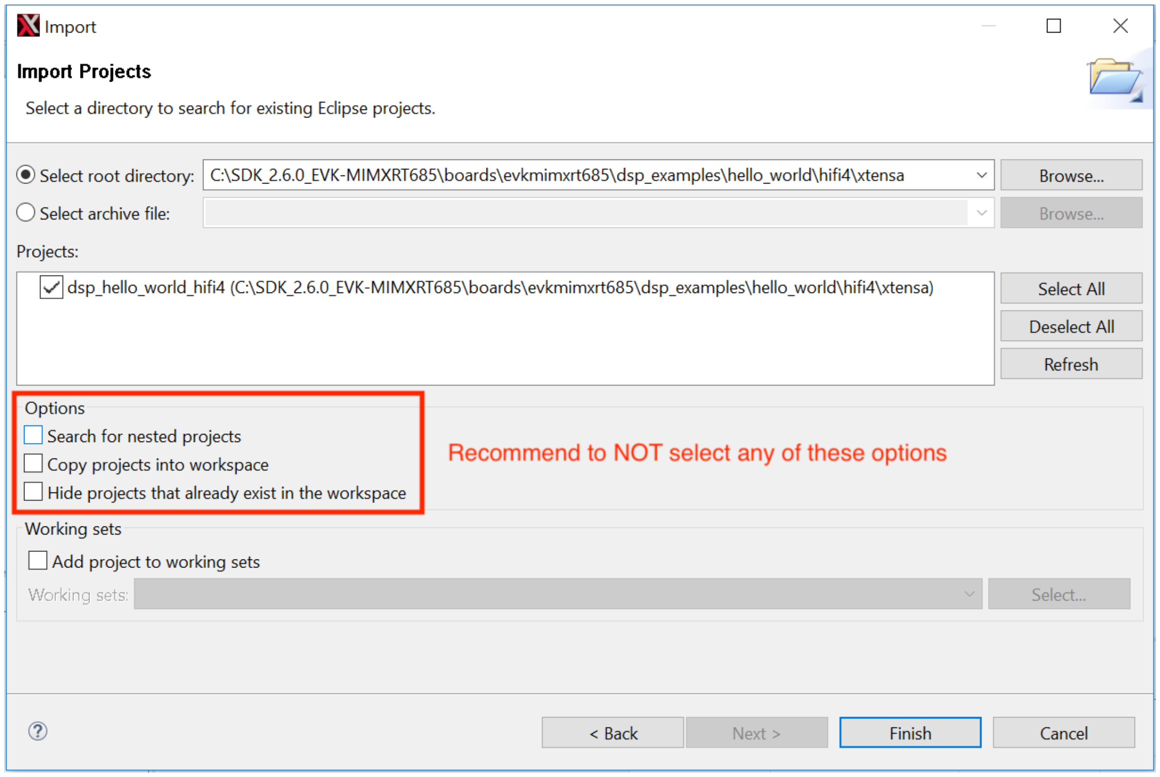
Browse For Folder

Select root directory of the projects to import

- SDK_2.6.0_EVK-MIMXRT685
  - boards
    - evkmimxrt685
      - cmsis_driver_examples
      - component_examples
      - demo_apps
      - driver_examples
      - dsp_examples
        - hello_world
          - hifi4
            - xcc
            - xtensa
        - hello_world_usart
        - mu_interrupt
        - mu_polling
        - sema42
        - xaf_demo

Folder: xtensa

Make New Folder | OK | Cancel


Import

Import Projects

Select a directory to search for existing Eclipse projects.

(● Select root directory:) C:\SDK_2.6.0_EVK-MIMXRT685\boards\evkmimxrt685\dsp_examples\hello_world\hifi4\xtensa

( ) Select archive file:

Projects:

☑ dsp_hello_world_hifi4 (C:\SDK_2.6.0_EVK-MIMXRT685\boards\evkmimxrt685\dsp_examples\hello_world\hifi4\xtensa)

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

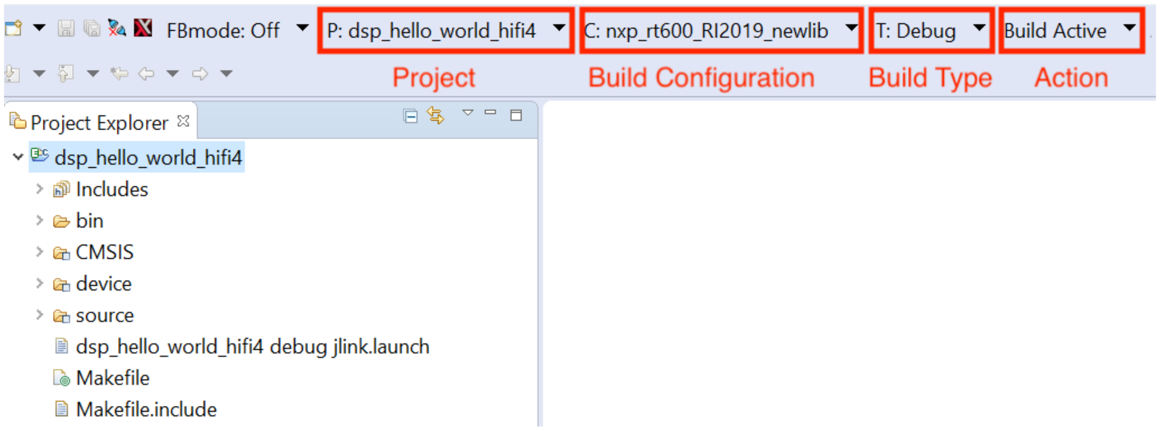
Recommend to NOT select any of these options

Working sets
- ☐ Add project to working sets

Working sets:

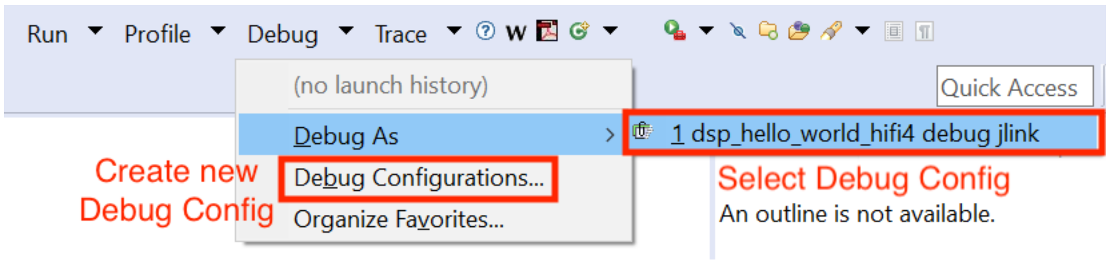
Once imported, you will see 'dsp_hello_world_hifi4' in the Project Explorer.

Use the drop down buttons on the menu bar to make a build selection for the project and hardware target configuration:

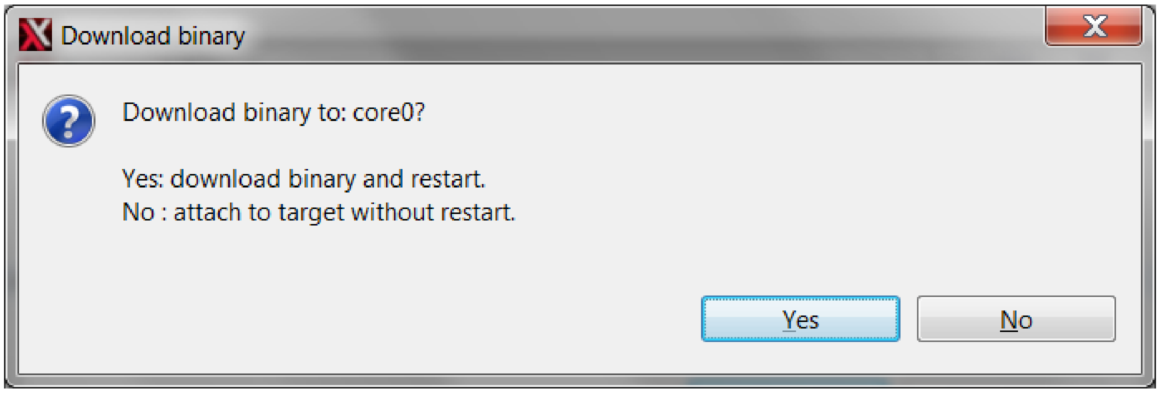

FBmode: Off | P: dsp_hello_world_hifi4 | C: nxp_rt600_RI2019_newlib | T: Debug | Build Active

Project Explorer
- dsp_hello_world_hifi4
  - Includes
  - bin
  - CMSIS
  - device
  - source
  - dsp_hello_world_hifi4 debug jlink.launch
  - Makefile
  - Makefile.include

Use the action buttons on the right side of the menu bar to debug / profile / trace. A default debug configuration is provided by the SDK project which will utilize the on-chip debugger:



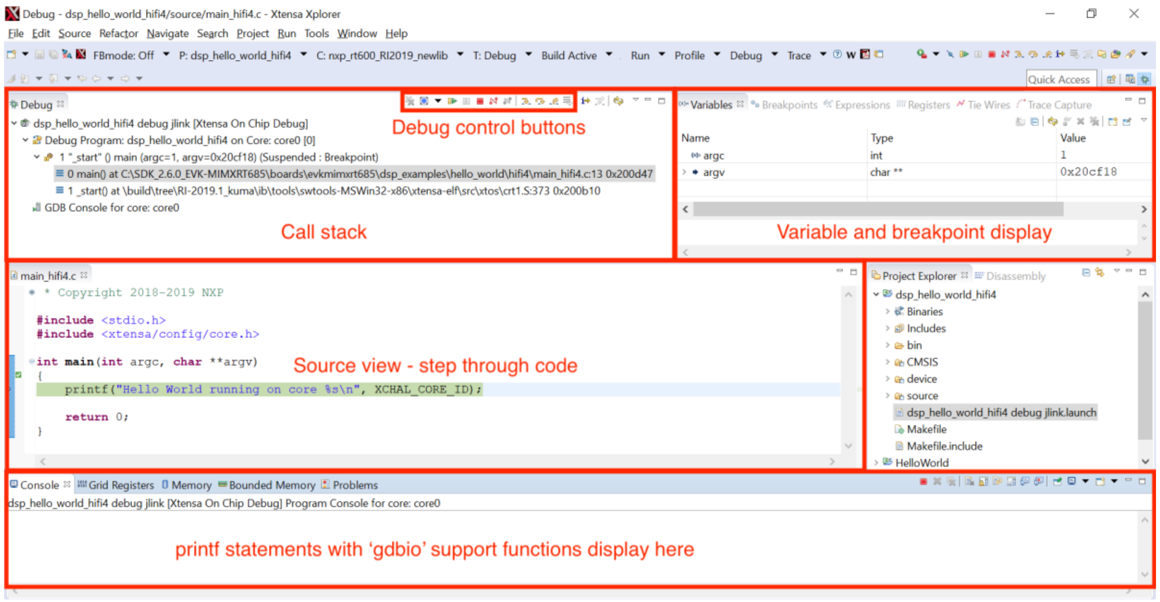
Once the 'Debug' button is selected, the actual debug on chip will be started. Xplorer will ask you if you like to download binaries to the hardware. Select Yes.


Download binary

Download binary to: core0?

Yes: download binary and restart.
No : attach to target without restart.

Xplorer IDE will transition to the 'Debug' perspective after binary download:


Debug control buttons
Call stack
Variable and breakpoint display
Source view - step through code
printf statements with ‘gdbio’ support functions display here

Debug - dsp_hello_world_hifi4/source/main_hifi4.c - Xtensa Xplorer

FBmode: Off P: dsp_hello_world_hifi4 C: nxp_rt600_RI2019_newlib T: Debug Build Active

```
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

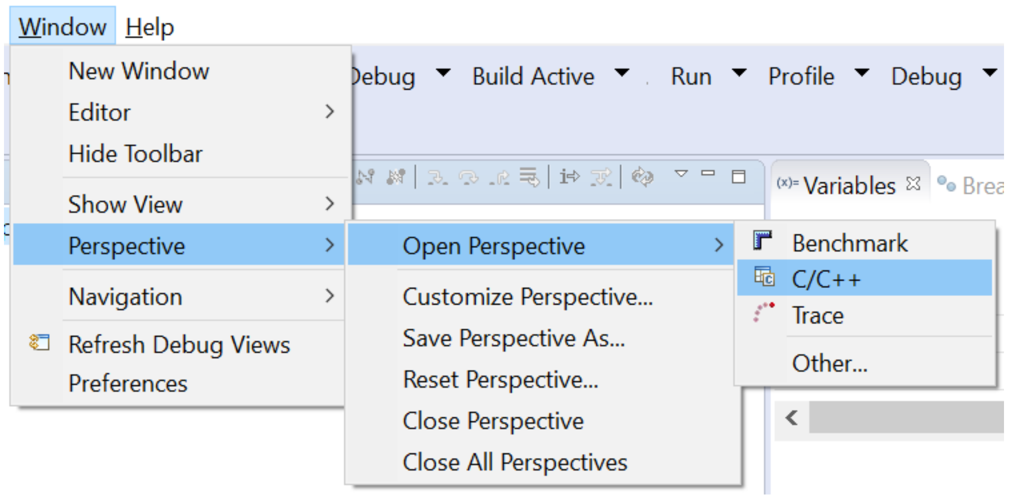
```c
 * Copyright 2018-2019 NXP

#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
```

After stepping through the ‘printf’ statement, you should see the output in the Console view of the IDE:

```
Hello World running on core nxp_rt600_RI2019_newlib
```

After debug is complete, select the previous code perspective to return to the default IDE layout:



3.4 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.

3.4.1 EVK Board Setup for Audio Demo

The DSP audio demo is tested against EVK-MIMXRT685 rev C and requires the use of the DMIC daughter board (attached at J31), the CODEC line out (J4), and UART for serial console.

In order for the CODEC to output audio properly, you will need to attach one jumper on the board as follows:

```
JP7-1  <--> JP8-2
```

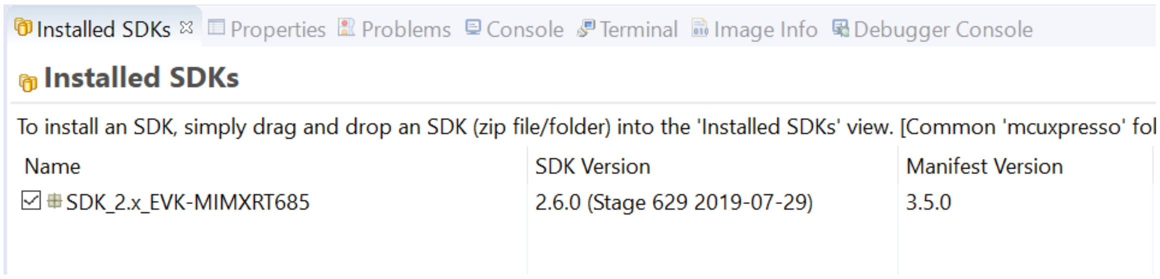
The demo uses the UART for console input and output. Connect the EVK board to a PC via the USB debug interface (J5) and open up a serial interface on your PC using a terminal tool such as PuTTY on Windows or screen on Linux.

3.4.2 Debug Audio Demo

In order to debug this DSP application, you will first need to setup and execute the ARM application using an environment of your choosing (see 'Getting Started with MCUXpresso SDK for EVK-MIMXRT685.pdf' for ARM development environment options).

The example that follows will use NXP MCUXpresso IDE for the ARM environment.

- Install the MCUXpresso SDK for RT600 into the MCUXpresso IDE using the 'Installed SDKs' panel at the bottom:



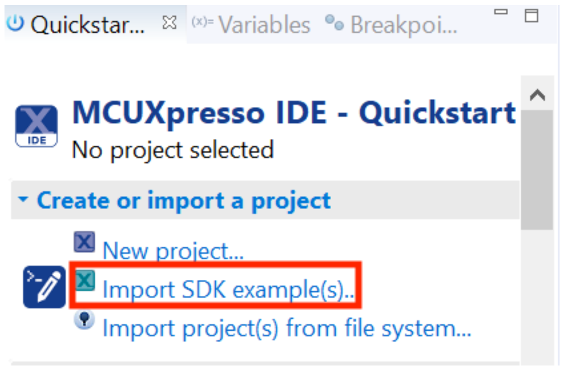
Installed SDKs ⊠ Properties Problems Console Terminal Image Info Debugger Console

Installed SDKs

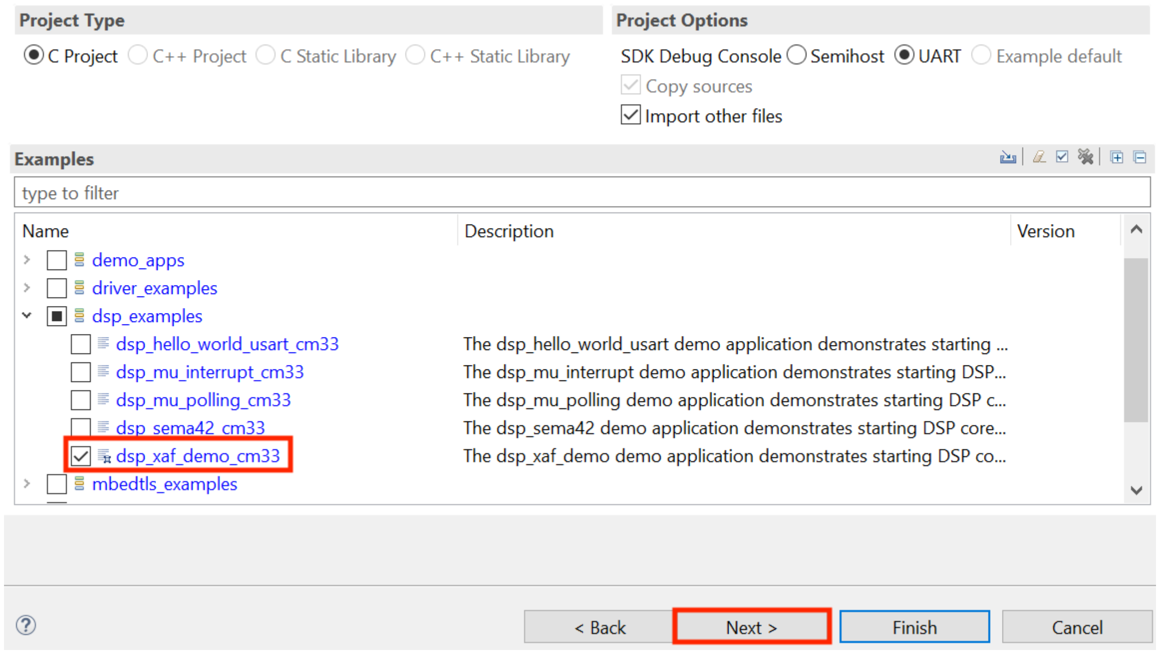
To install an SDK, simply drag and drop an SDK (zip file/folder) into the 'Installed SDKs' view. [Common 'mcuxpresso' fol

Name	SDK Version	Manifest Version
☑ SDK_2.x_EVK-MIMXRT685	2.6.0 (Stage 629 2019-07-29)	3.5.0

- Use the QuickStart menu on the lower left of the screen to import an example from the installed SDK:


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...

- 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]

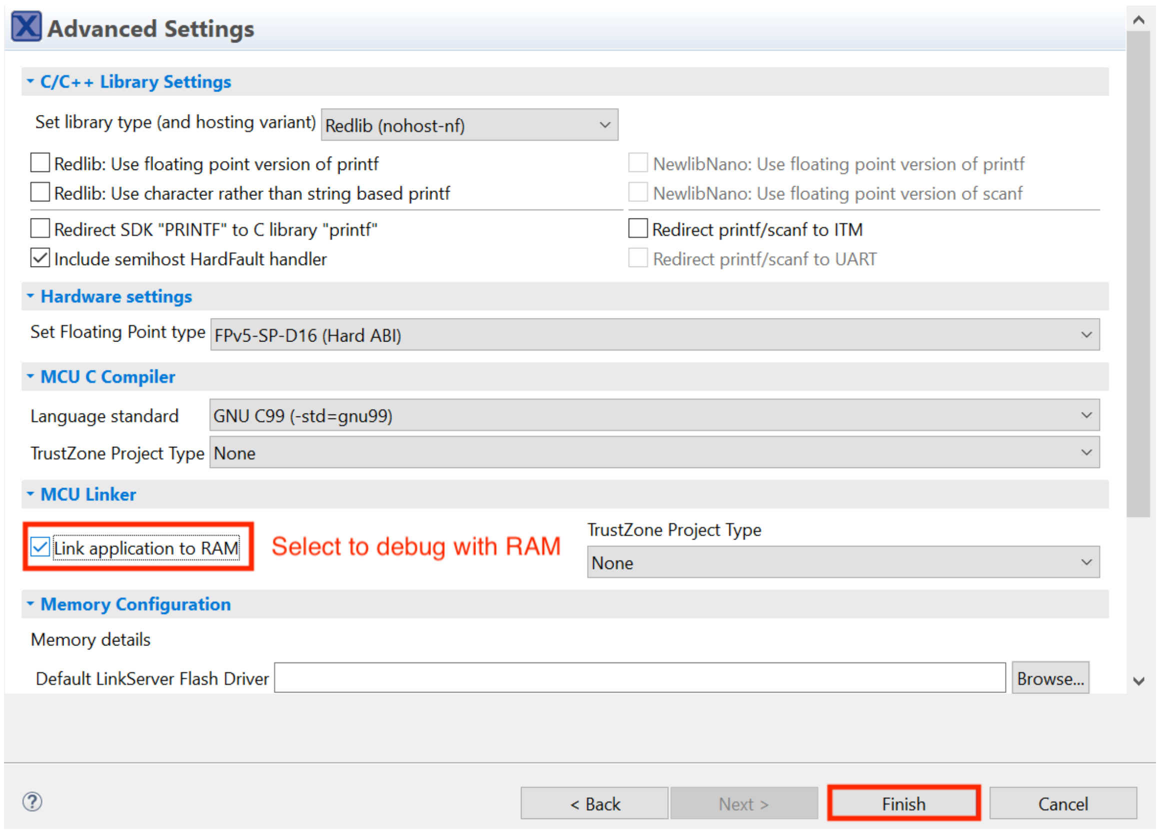
[✓ Import other files]

Examples

type to filter

Name	Description	Version
demo_apps		
driver_examples		
dsp_examples		
dsp_hello_world_usart_cm33	The dsp_hello_world_usart demo application demonstrates starting ...	
dsp_mu_interrupt_cm33	The dsp_mu_interrupt demo application demonstrates starting DSP...	
dsp_mu_polling_cm33	The dsp_mu_polling demo application demonstrates starting DSP c...	
dsp_sema42_cm33	The dsp_sema42 demo application demonstrates starting DSP core...	
✓ dsp_xaf_demo_cm33	The dsp_xaf_demo demo application demonstrates starting DSP co...	
mbedtls_examples		

- Configure project settings, notably choose to link the application to RAM for ease of initial debug. Select ‘Finish’ to complete the import:



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"
- [x] 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

- [x] Link application to RAM

Select to debug with RAM

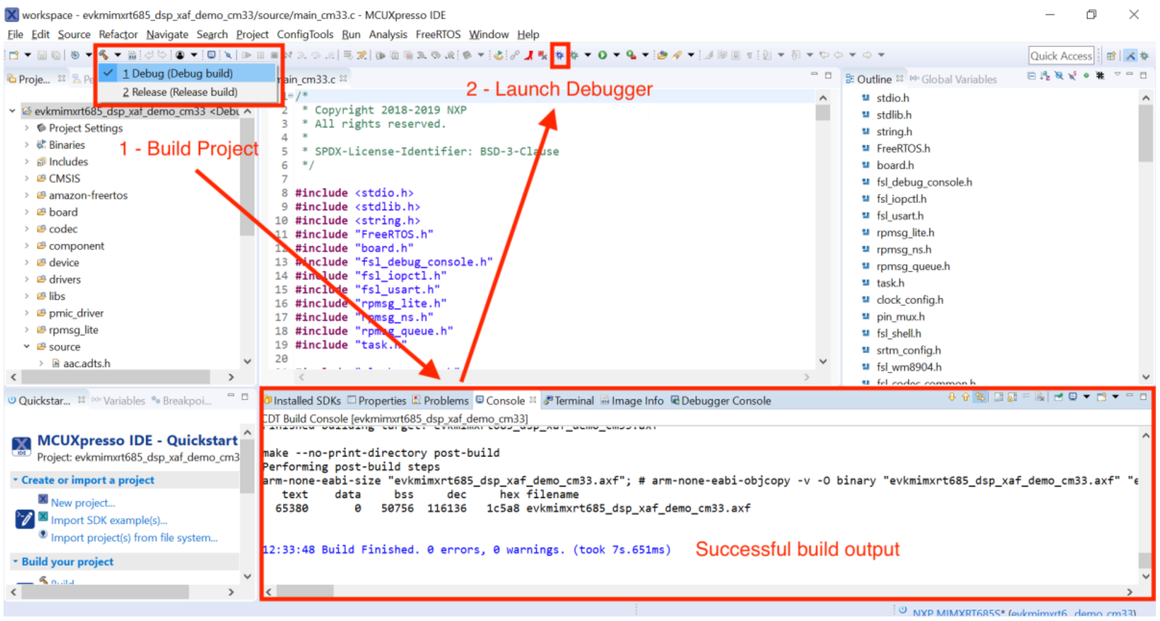
TrustZone Project Type: None

Memory Configuration

Memory details

Default LinkServer Flash Driver:

- Build the project and launch the debugger on success:

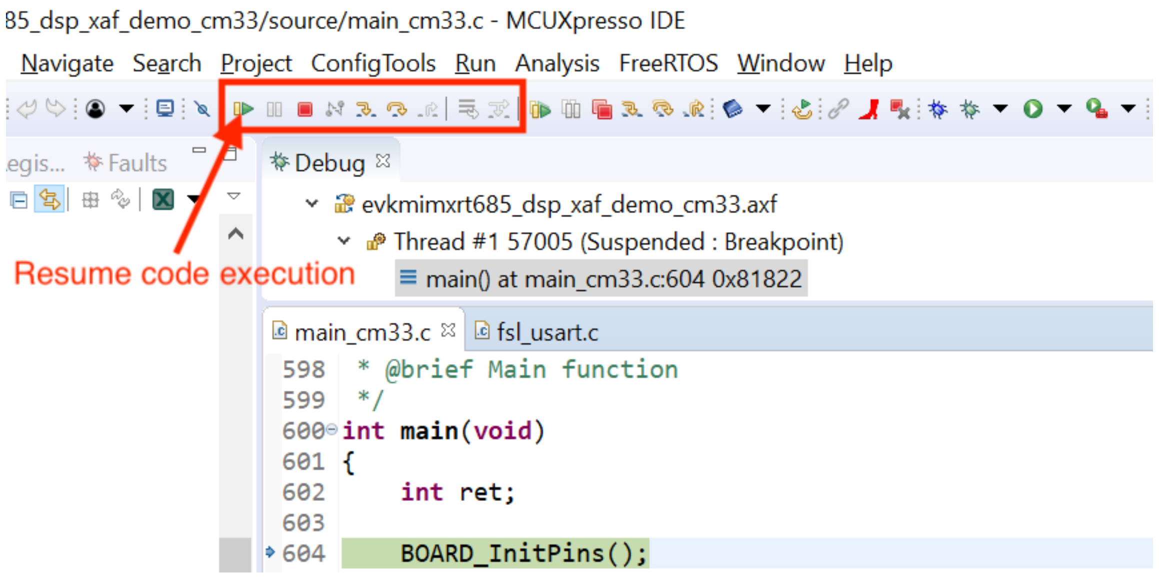

workspace - evkmimxrt685_dsp_xaf_demo_cm33/source/main_cm33.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run Analysis FreeRTOS Window Help
1 Debug (Debug build)
2 Release (Release build)
1 - Build Project
2 - Launch Debugger
Successful build output
MCUXpresso IDE -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

```
/*
 * 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)
```

- 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
Resume code execution
Debug
evkmimxrt685_dsp_xaf_demo_cm33.axf
Thread #1 57005 (Suspended : Breakpoint)
main() at main_cm33.c:604 0x81822
main_cm33.c
fsl_usart.c

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

    BOARD_InitPins();
```

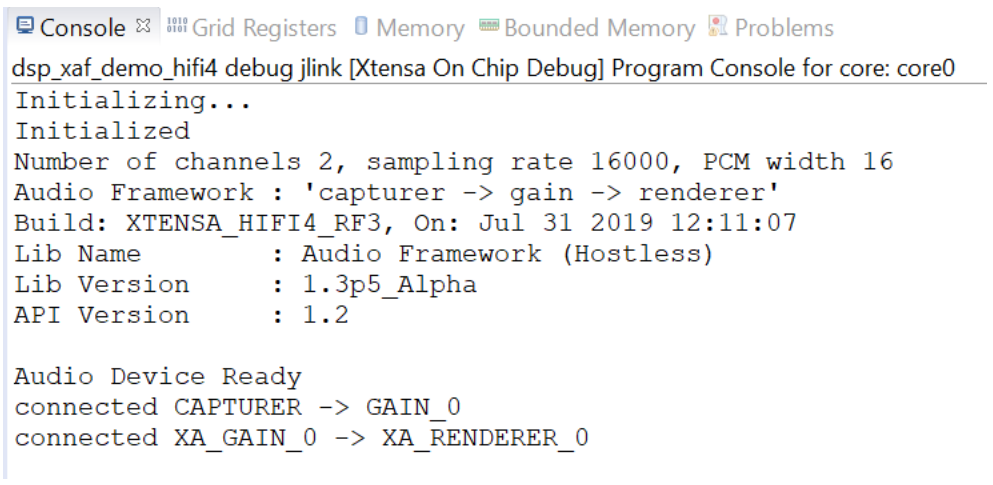
- Observe serial terminal output with shell prompt:

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

Configure WM8904 codec
[APP_SDCARD_Task] start
[APP_DSP_IPC_Task] start
[APP_Shell_Task] start

SHELL build: Feb 18 2020
Copyright  2018  NXP
>>
```

- Using the Xplorer IDE, load and execute xaf_demo using the procedure described in section 3.3 of this document.
- After the DSP application is running, use the serial shell to invoke the 'record_dmic' command – this will create an audio pipeline that captures microphone audio and plays it back 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": Record DMIC audio and playback on WM8904 codec
…
>> record_dmic
```

- The Xplorer IDE console will show output of the audio framework pipeline initializing:



dsp_xaf_demo_hifi4 debug jlink [Xtensa On Chip Debug] Program Console for core: core0

```
Initializing...
Initialized
Number of channels 2, sampling rate 16000, PCM width 16
Audio Framework : 'capturer -> gain -> renderer'
Build: XTENSA_HIFI4_RF3, On: Jul 31 2019 12:11:07
Lib Name        : Audio Framework (Hostless)
Lib Version     : 1.3p5_Alpha
API Version     : 1.2

Audio Device Ready
connected CAPTURER -> GAIN_0
connected XA_GAIN_0 -> XA_RENDERER_0
```

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

3.5 Launch DSP Application from ARM Core

In the previous example, the ARM application and DSP application were independently loaded and debugged. In this section, we will show how to produce one ARM application binary that includes and starts the DSP application without the use of a debugger/loader.

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

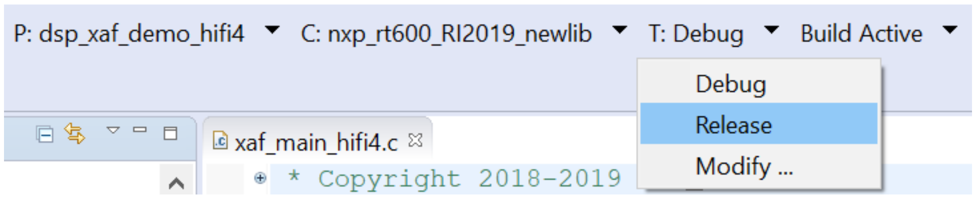
```
DSP_IMAGE_COPY_TO_RAM
```

This macro is set to 0 by default. When this macro is changed to '1'/TRUE, it will instruct 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: this macro must be supplied to both the C compiler and assembler. Please be aware of this when modifying your ARM project.

To build the DSP application image so it can be used by the ARM application, you must select the 'Release' target in Xplorer IDE (building with min-rt LSP – see section 2.4 for more information):



Three DSP binaries are generated, to be 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: you may need to manually copy these binary images into your ARM application workspace, depending on the environment used.

4 Run and Debug from Command Line Environment / LINUX

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

4.1 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. Please see the 'Getting Started with MCUXpresso SDK for EVK_MIMXRT685.pdf' in the <SDK_ROOT>/docs/ directory for more information on installation and configuration of required build tools for command line development.

- 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
```

- 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
```

- 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...
```

- 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();
```

4.2 Build and Debug DSP Application

The Xtensa command line toolchain is installed as part of the Xplorer IDE. The tools can optionally be installed on a new Windows or Linux system without the IDE using the redistributable compressed file found under <XTENSA_ROOT>/XtDevTools/downloads/RI-2019.1/tools/ (see section 1.5 for more information).

In order to use the command line tools, some environment variables need to be setup that are used by 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-2019.1-
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-2019.1-linux/XtensaTools
export XTENSA_SYSTEM=~/xtensa/XtDevTools/install/builds/RI-2019.1-
linux/nxp_rt600_RI2019_newlib/config
export XTENSA_CORE=nxp_rt600_RI2019_newlib
```

NOTE: on Windows, you can use the 'setx' command instead of 'export' to set environment variables.

- 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 – these are normal and can be ignored.

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

```
user@linux:/opt/Tensilica/xocd-14.01$ ./xt-ocd.exe -c topology.xml
```

NOTE: if the xt-ocd daemon fails to start, it may be because the DSP has not been initialized by the ARM core which must be done first.

- 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 14.01
...
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-2019.1/ib/tools/swtools-x86_64-linux/xtensa-
elf/src/xos/src/xos_vectors.S:216
216 /home/xpgcust/tree/RI-2019.1/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-2019.1/ib/tools/swtools-x86_64-linux/xtensa-
elf/src/xtos/xea2/reset-vector-xea2.S:71
71  /home/xpgcust/tree/RI-2019.1/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-2019.1/ib/tools/swtools-
  x86_64-linux ~/xtensa/tools/RI-2019.1-linux/XtensaTools
  ```

For more details about xt-gdb, please refer to Cadence GNU Debugger User's Guide and Cadence Xtensa Debug Guide. These are located at:

➔ ~/xtensa/XtDevTools/downloads/RI-2019.1/docs/gnu_gdb_ug.pdf
➔ ~/xtensa/XtDevTools/downloads/RI-2019.1/docs/xtensa_debug_guide.pdf

5 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, please refer to Xtensa IDE documents. You can find them under Xtensa Xplorer IDE menu Help-> PDF Documentation. These 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, a.k.a. the basic runtime and handlers has been depreciated and moved towards XOS. But 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 & GNU Linker User's Guide. Both documents are useful for understanding the HiFi program linker and the memory map.

Specifically for RT6xx, User Manual and Datasheet are the most important references. All product specifications can be found there.

5.1 HiFi4 Boot Loader and Memory Map

5.1.1 HiFi4 Boot Loader

For better power efficiency, by default HiFi4 DSP is powered off when RT6xx powers up. To boot up, Cortex M33 will act as master core to config DSP local memories, clocks and load DSP images etc. SDK has wrapped up this part as HiFi4 Boot Loader a.k.a. 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 e.g. 600MHz, SoC Vddcore has to be set to 1.1V. If you don't need full power e.g. to run DSP at the half speed at 300Mhz, then Vddcore only needs to be 0.8V. RT6xx EVK integrates NXP PCA9420 PMIC for power management and by default Vddcore has been set to 1.0V. So here we first initialize PMIC for better power management. For more Vddcore and DSP frequency operating conditions, please refer to any dsp example\cm33\pmic_support.c BOARD_SetPmicVoltageForFreq(), or refer to 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);
```

DSP can be clocked from various clock sources. Most commonly we set DSP PLL for full power. It can also be 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 300MHz, here we set DSP AHB bus clock divider as 2:

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

If DSP clock is running at 300MHz or lower, it is more efficient to use divider as 1. For divider as 1 please note that you also need to set an extra register SYSCTL0_PACKERENABLE. For more details please refer to User Manual 4.5.2.18 DSP main ram clock divider and 4.5.5.3 Packer Enable

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

Now we can power up TCM/ DSP local memories & cache, supply clock, and reset peripherals.

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

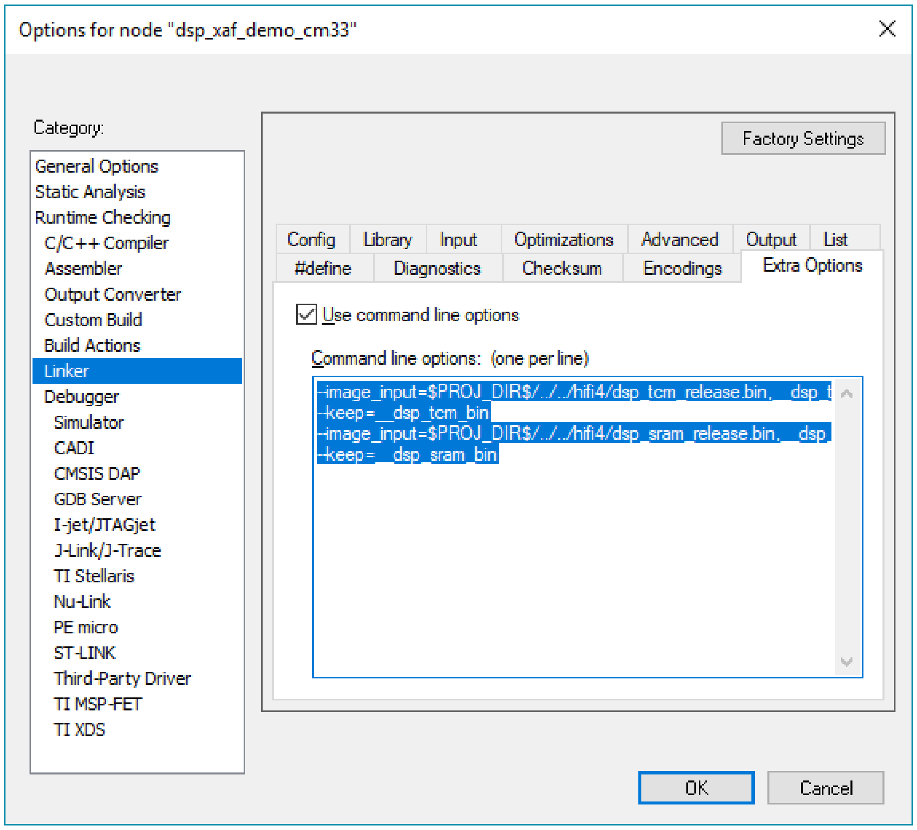
For SDK DSP examples, we split DSP images into three parts. One is for vectors and critical sections sitting on TCM/ DSP local memories, another one is for normal code and data sections sitting on SRAM, and the final is for non-cached DSP initialized data in SRAM. Here Cortex M33 load those binaries to its destination. When you debug the DSP program, you can also choose to load DSP binaries from Xtensa Xplorer IDE, as described in section 3.3 Prepare DSP core for 'Hello World'. For this purpose, you need to remove DSP_IMAGE_COPY_TO_RAM compilation flag or set it to 0. By default this flag is set to 1 to always load 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
```

HiFi4 operation is controlled by DSP stall register SYSCTL0_DSPSTALL. Now we can un-stall DSP and run it.

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

DSP images are created by post build scripts, please refer to 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 to be linked into Cortex M33 side. If you are using IAR, the linker could be set in Project Options -> Linker-> Extra Options. If you are using ARMGCC or Linux env., it is set in any DSP example\cm33\incbin_gcc.S


Options for node "dsp_xaf_demo_cm33"

Category: Linker

Extra Options

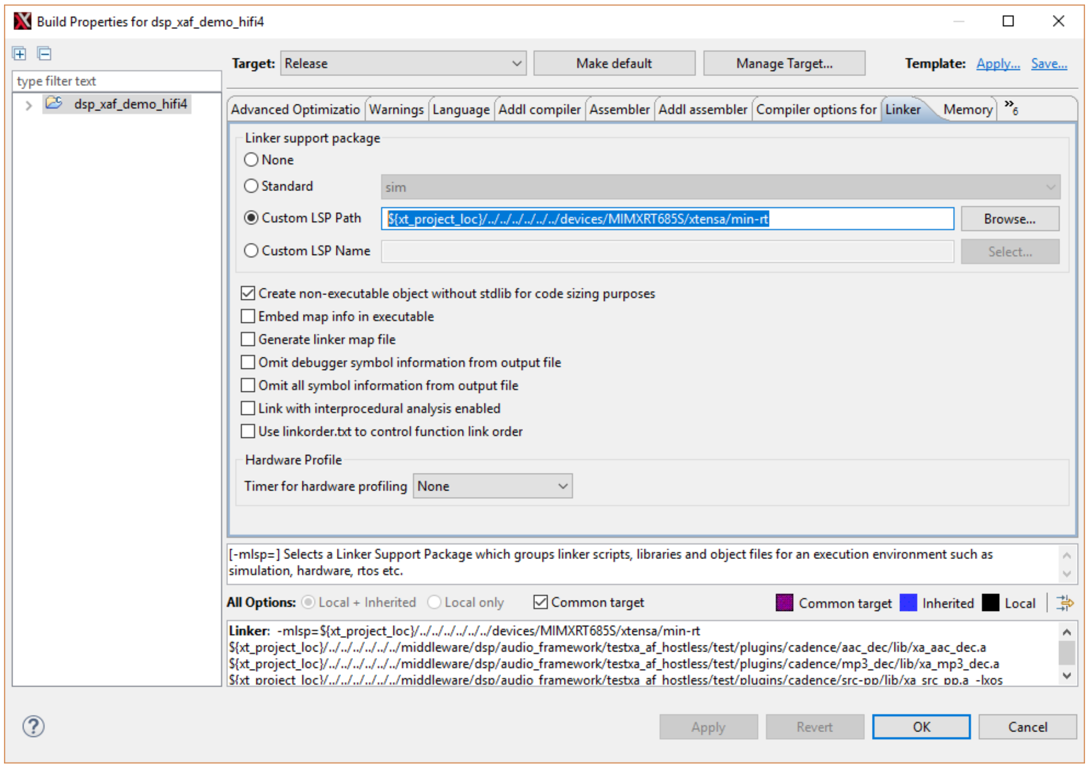
[✓] Use command line options

Command line options: (one per line)

```
--image_input=$PROJ_DIR$/../../hifi4/dsp_tcm_release.bin,__dsp_t
--keep=__dsp_tcm_bin
--image_input=$PROJ_DIR$/../../hifi4/dsp_sram_release.bin,__dsp_
--keep=__dsp_sram_bin
```

5.1.2 Linker and Memory Map

When you import SDK DSP examples, they are all set to Release mode by default. Which means those images need to be built in Release mode with 'min-rt' Linker Support Package/ LSP. You can double check this by opening any SDK DSP examples-> Build Properties-> Linker.


Build Properties for dsp_xaf_demo_hifi4

Target: Release

Template: Apply... Save...

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

```
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
```

SDK provides three different LSPs. ‘min-rt’ for Release mode, ‘gdbio’ for Debug mode, and ‘sim’ for simulations. ‘min-rt’ eliminates all unnecessary debug info and reduces image size; ‘gdbio’ support standard ‘printf’/ log output backto 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’ 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:

- 0x0020 0000 ~ 0x0048 0000, size 2.5M 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 64K bytes, for Data TCM. By default its empty;
- 0x2402 0000 ~ 0x2402 FFFF, size 64K bytes, for instruction TCM. By default it only contains essential vectors and left around 62K for applications;

- 0x2004 0000 ~ 0x2007 FFFF, size 256K bytes. This is non-cached area for Cortex M33 and HiFi4 DSP data exchange.

Please note that both Cortex M33 and HiFi4 DSP have access to all SRAM partitions. This means a unified memory map is necessary at system level, and both cores MUST NOT affect each other's memory map. For SDK examples, you could see that HiFi4 memory map starts from 0x0020 0000 and Cortex M33 side sits under this addresses. Using IAR env. 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 completely flexible, it could be adjusted by application needs. Please keep in mind that modifying one core's memory map MIGHT affect another. Please always change on BOTH cores accordingly. E.g. When you allocating more SRAM partitions to DSP you also need reduce the memory taken at Cortex M33 side, otherwise Cortex M33 might not work properly. Another note is: if you loading DSP image directly from Cortex M33 side, the image will still sit in Cortex M33 data section before booting up. So it raises the bar for application data section requirements. You might need to consider running the application from FLASH.

5.1.3 Cache and Data Exchange Memory Partitions

You might noticed that HiFi4 DSP has a small non-cached area that starts from 0x20040000. This is used for data exchange between two cores. As both M33 and HiFi DSP have shared access to all SRAM partitions, shared memory access will be the most effective way to exchange data between two cores. And a given physical addresses could be read/ write by both cores at same address, no memory mapping or address converting needed. For example, if a piece of data array needs to be passed from Cortex M33 to HiFi4 DSP, you only need to pass the start pointer and the size of the array, and vice versa. It provides the great convenience for system programming and simplify the inter-core communications.

We need to consider cache here. Cortex M33 has no cache, the entire SRAM to be considered as its local memory. So any memory write will be flush immediately. HiFi4 has 32K instruction cache and 64K data cache, and both cache is enabled by default. So memory write will NOT flush immediately. To make a tradeoff between performance and IPC convenience, we set this non-cached area for data exchange memory partitions.

The above memory map has specified the non-cached region, and in DSP code, we will call HAL functions to disable cache. Please refer to audio framework example in SDK path\boards\evkmimxrt685\dsp_examples\xaf_demo\dsp\xaf_main_dsp.c For more details about HAL cache function, please refer to 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);
```

Please note that this XHAL call sets cache attribute of the whole region/ 512M bytes even you pass the size to be set. That's also the reason that non-cached attribute is set on the overlapping SRAM address starts from 0x2000 0000. This is to distinguish with the physical SRAM addresses starting from 0x0000 0000, which is cacheable area for HiFi4.

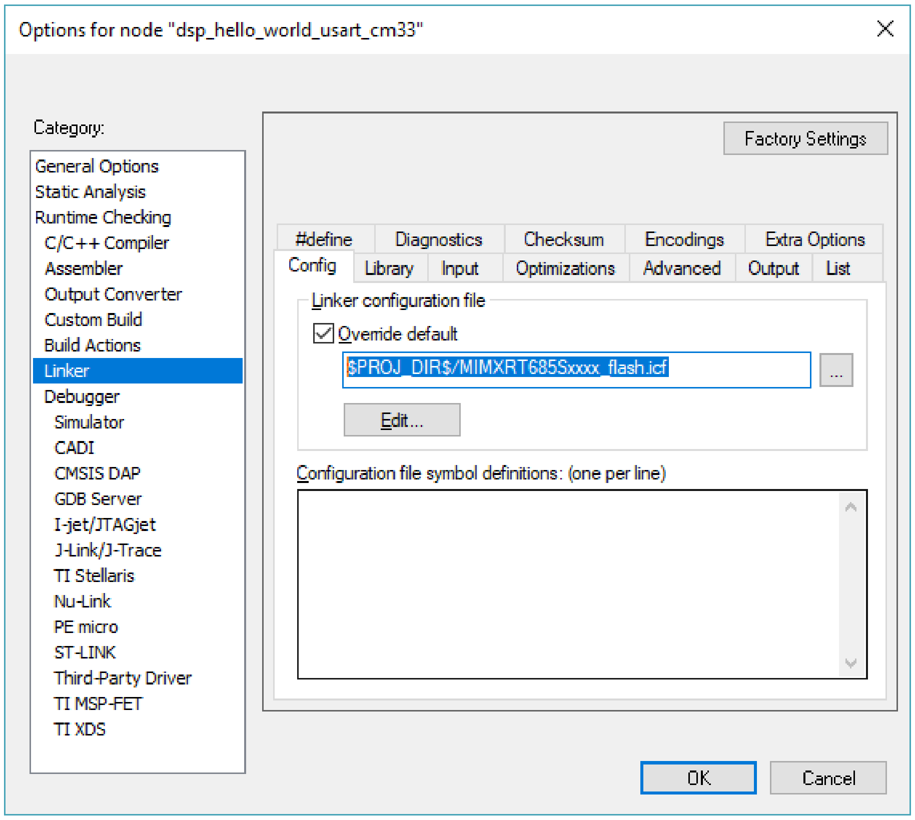
Data exchange memory partitions are completely flexible and could be configured per application needs. However to mitigate the possible AHB arbitration between the two cores, it is recommended to use the first eight 32K and following four 64K memory partitions. DSP Data TCM could also be used as data exchange area for those data have high demanding on timing performance. General speaking, it recommend to avoid accessing same partition at same time for frequent data exchange between two cores. You could keep one core in Sleep or Wait for Interrupt while another core operating, or setup a PING PONG data exchange/ DMA so that when one core fills one partition, another core fetches another partition, and vice versa.

For more details about the RT6xx memory map, please refer to User Manual section 2 Memory Map and section 2.1.11 HiFi4 memory map.

5.1.4 Boot or Run from Flash

Boot from Flash is pretty straight forward if you are using IAR env. SDK provide four different build configurations for each project: debug (from SRAM)/ release (from

SRAM)/ flash_debug/ flash_release. Flash configs use different memory map in project linker options, please refer to below setting:


Options for node "dsp_hello_world_usart_cm33"

Category: Linker

Config tab, Linker configuration file:

☑ Override default

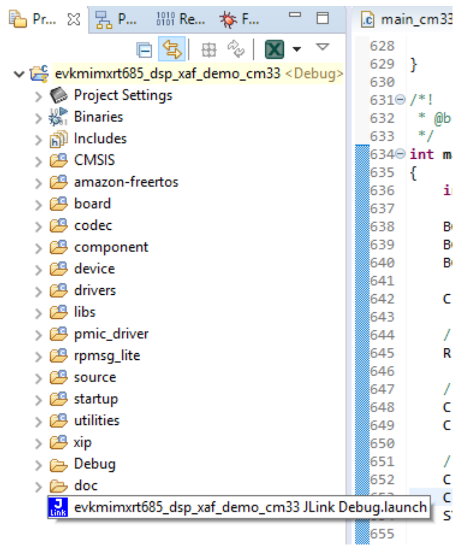
$PROJ_DIR$/MIMXRT685Sxxxx_flash.icf

Configuration file symbol definitions: (one per line)

Please don't forget to change ISP mode/ SW5 switches on the EVK to enable booting from flash.

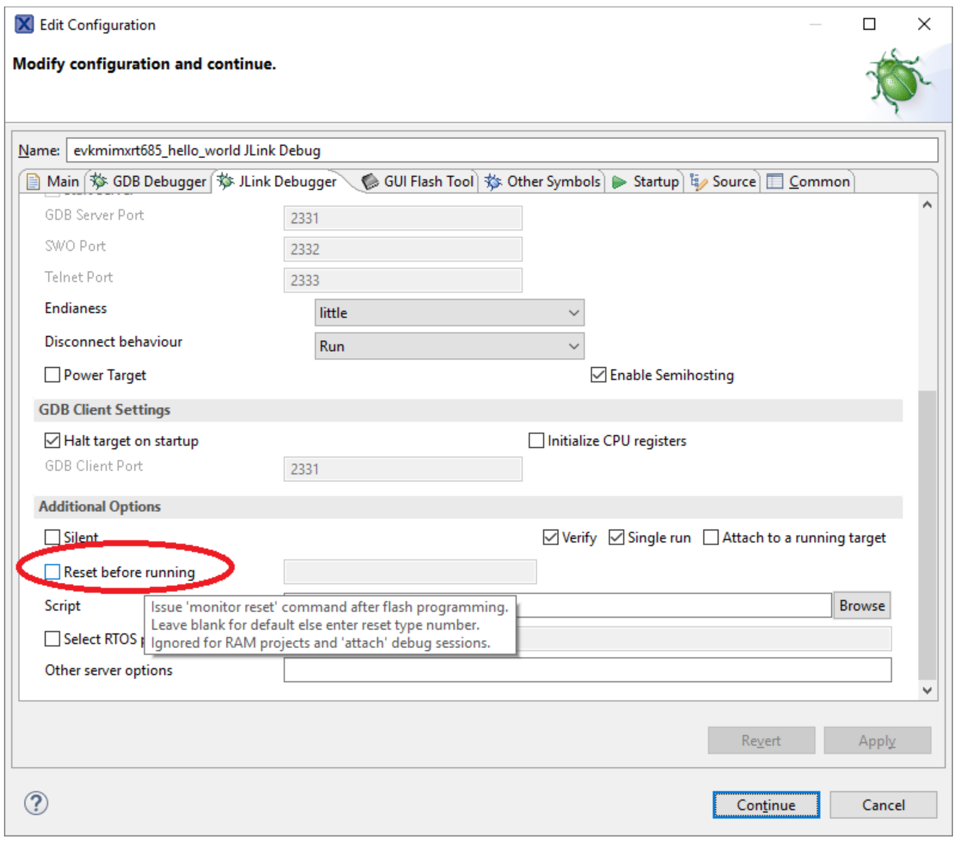
If you are using MCUXpresso/ armgcc build env., please note that by default build env. has been set to boot from flash. Using MCUXpresso as example:

- Please make sure you are using MCUXpresso version 11.1.1 or newer;
- Import the SDK examples. Once completed, double click the last file/ J-Link Debug. Launch to modify J-Link debugger setting. Make sure to uncheck 'Reset before running'. This will help the flash-based program get into main function.



- evkmimxrt685_dsp_xaf_demo_cm33 <Debug>
  - Project Settings
  - Binaries
  - Includes
  - CMSIS
  - amazon-freertos
  - board
  - codec
  - component
  - device
  - drivers
  - libs
  - pmic_driver
  - rpmsg_lite
  - source
  - startup
  - utilities
  - xip
  - Debug
  - doc
  - evkmimxrt685_dsp_xaf_demo_cm33 JLink Debug.launch

```
}

/*!
 * @b
 */
int m
{
    i

    B
    B
    B

    C

    /
    R

    /
    C
    C

    /
    C
    C
    S
```



Modify configuration and continue.

Name: evkmimxrt685_hello_world JLink Debug

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

Script

[ ] Select RTOS

Other server options

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

- Make sure to modify DSP_IMAGE_COPY_TO_RAM Define to 1 in project settings-> C/C++ General->Paths and Symbols-> Symbols.

- Furthermore, make sure we have right compilation flag as C/C++ compilation flags don't work on .S files. Make sure #define DSP_IMAGE_COPY_TO_RAM 1 As the first line of source/incbin.S to force it include DSP binaries.
- Please also make sure you are using correct DSP binaries at correct path, again must be release binaries. Make sure incbin.S gets the right image path.
- 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 , you can switch ISP0 off ISP2 ON to boot from flash.

5.2 Peripheral Drivers and Interrupts

5.2.1 The 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 an 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, you will see two major differences:

- Pin initialization is not necessary for DSP. It's doable but we highly recommend to manage 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's also doable but same as above we highly recommend to manage all clock sources all-in-one place;

5.2.2 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 non-secure DMA controller will be shared by other code and by the HiFi4 (if needed).

Normally HiFi4 DSP will always use DMA1 controller. Again, it's 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 please refer to 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 please refer to SDK path\boards\evkmimxrt685\driver_examples\dmic and SDK path\boards\evkmimxrt685\driver_examples\i2s

5.2.3 Interrupts

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

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 we need to include XOS system header files to include necessary XOS functions.

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

Then we need to setup UTick callback and delay functions. This part is identical with Cortex M33 side. You could refer to SDK path\\boards\evkmimxrt685\driver_examples\utick for Cortex M33 side implementation.

```
#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;
}
```

Then we can initialize XOS and start UTick timer. Those 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");
	}
```

Please pay attention to those instant numbers. Here we pick interrupt 15, which maps to DSP_INT0_SEL10 as a L1 interrupts, lowest priority level. Please refer to User Manual section 51.7 Interrupt for more details about RT6xx HiFi4 DSP interrupt configuration. Hence for Inputmux, we attach UTick interrupt to 10. 15-5=10 as first five interrupts are reserved, not user configurable. If you like to get it to highest priority level e.g. L3, we can config interrupt like this:

```
	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, please refer to Xtensa XOS Reference Manual Chapter 18 Interrupt and Exception Handling, and Chapter 26 Interrupt Handler Restrictions.

5.2.4 A 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 pinmux initializing or clock configurations. Please refer to above Cortex M33 example dmic_i2s_dma to setup Cortex M33 side. Once Cortex M33 side ready, this example could be 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.

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

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

#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#include "fsl_inputmux.h"
#include "fsl_dmic.h"
#include "fsl_dmic_dma.h"
#include "fsl_i2c.h"
#include "fsl_i2s.h"
#include "fsl_i2s_dma.h"

#include "pin_mux.h"
#include "board_hifi4.h"
/***********************************************************************
*********
```

```
 * Definitions

*************************************************************************
*******/
extern int NonCacheable_start, NonCacheable_end;
extern int NonCacheable_init_start, NonCacheable_init_end;

#define DMAREQ_DMIC0 16U
#define DEMO_I2S_MASTER_CLOCK_FREQUENCY CLOCK_GetMclkClkFreq()
#define DEMO_I2S_TX (I2S3)
#define DEMO_I2S_CLOCK_DIVIDER
\
    (24576000U / 48000U / 16U / 2) /* I2S source clock 24.576MHZ,
sample rate 48KHZ, bits width 16, 2 channel, \
                                  so bitclock should be 48KHZ * 16 =
768KHZ, divider should be 24.576MHZ / 768KHZ */

#define DEMO_DMA (DMA1)
#define DEMO_DMIC_RX_CHANNEL DMAREQ_DMIC0
#define DEMO_I2S_TX_CHANNEL (7)
#define DEMO_I2S_TX_MODE kI2S_MasterSlaveNormalSlave

#define DEMO_DMIC_CHANNEL kDMIC_Channel0
#define DEMO_DMIC_CHANNEL_ENABLE DMIC_CHANEN_EN_CH0(1)
#define DEMO_AUDIO_BIT_WIDTH (16)
#define DEMO_AUDIO_SAMPLE_RATE (48000)
#define DEMO_AUDIO_PROTOCOL kCODEC_BusI2S
#define FIFO_DEPTH (15U)
#define BUFFER_SIZE (128)
#define BUFFER_NUM (2U)


/************************************************************************
*********
 * Prototypes

*************************************************************************
*******/
static i2s_config_t tx_config;
static uint32_t volatile s_writeIndex = 0U;
static uint32_t volatile s_emptyBlock = BUFFER_NUM;
static dmic_dma_handle_t s_dmicDmaHandle;
static dma_handle_t s_dmicRxDmaHandle;
static dma_handle_t s_i2sTxDmaHandle;
static i2s_dma_handle_t s_i2sTxHandle;

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
);

static dmic_transfer_t s_receiveXfer[2U] = {
    /* transfer configurations for channel0 */
    {
        .data                    = s_buffer,
        .dataWidth               = sizeof(uint16_t),
        .dataSize                = BUFFER_SIZE,
        .dataAddrInterleaveSize = kDMA_AddressInterleave1xWidth,
        .linkTransfer            = &s_receiveXfer[1],
    },

    {
        .data                    = &s_buffer[BUFFER_SIZE],
        .dataWidth               = sizeof(uint16_t),
        .dataSize                = BUFFER_SIZE,
        .dataAddrInterleaveSize = kDMA_AddressInterleave1xWidth,
        .linkTransfer            = &s_receiveXfer[0],
    },
};

void dmic_Callback(DMIC_Type *base, dmic_dma_handle_t *handle, status_t
status, void *userData)
{
    if (s_emptyBlock)
    {
        s_emptyBlock--;
    }
}

void i2s_Callback(I2S_Type *base, i2s_dma_handle_t *handle, status_t
completionStatus, void *userData)
{
    if (s_emptyBlock < BUFFER_NUM)
    {
        s_emptyBlock++;
    }
}

/*********************************************************************
*********
 * Code

**********************************************************************
*******/
static void XOS_Init(void)
{
    xos_set_clock_freq(XOS_CLOCK_FREQ);
    xos_start_system_timer(-1, 0);
}

/*!
 * @brief Main function
 */
int main(void)
{
    dmic_channel_config_t dmic_channel_cfg;
    i2s_transfer_t i2sTxTransfer;

    xos_start_main("main", 7, 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);
    xthal_set_region_attribute((uint32_t *)&NonCacheable_init_start,
                               (uint32_t)&NonCacheable_init_end -
(uint32_t)&NonCacheable_init_start, XCHAL_CA_BYPASS,
                               0);

    /* Init board hardware. */
    /* As CM33 side take care of all pins config, no need to re-config
pins again on DSP */
    /* BOARD_InitPins(); */
    BOARD_InitDebugConsole();

    PRINTF("DSP starts on core '%s'\r\n", XCHAL_CORE_ID);
    /* attach AUDIO PLL clock to FLEXCOMM1 (I2S1) */
    CLOCK_AttachClk(kAUDIO_PLL_to_FLEXCOMM1);
    /* attach AUDIO PLL clock to FLEXCOMM3 (I2S3) */
    CLOCK_AttachClk(kAUDIO_PLL_to_FLEXCOMM3);

    /* attach AUDIO PLL clock to MCLK */
    CLOCK_AttachClk(kAUDIO_PLL_to_MCLK_CLK);
    CLOCK_SetClkDiv(kCLOCK_DivMclkClk, 1);
    SYSCTL1->MCLKPINDIR = SYSCTL1_MCLKPINDIR_MCLKPINDIR_MASK;
    /* DMIC source from audio pll, divider 8, 24.576M/8=3.072MHZ */
    CLOCK_AttachClk(kAUDIO_PLL_to_DMIC);
    CLOCK_SetClkDiv(kCLOCK_DivDmicClk, 8);

    /* Set shared signal set 0: SCK, WS from Flexcomm1 */
    SYSCTL1->SHAREDCTRLSET[0] = SYSCTL1_SHAREDCTRLSET_SHAREDSCKSEL(1) |
SYSCTL1_SHAREDCTRLSET_SHAREDWSSEL(1);
    /* Set flexcomm3 SCK, WS from shared signal set 0 */
    SYSCTL1->FCCTRLSEL[3] = SYSCTL1_FCCTRLSEL_SCKINSEL(1) |
SYSCTL1_FCCTRLSEL_WSINSEL(1);

    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);
    DMA_Init(DEMO_DMA);
    DMA_EnableChannel(DEMO_DMA, DEMO_I2S_TX_CHANNEL);
    DMA_EnableChannel(DEMO_DMA, DEMO_DMIC_RX_CHANNEL);
    DMA_SetChannelPriority(DEMO_DMA, DEMO_I2S_TX_CHANNEL,
kDMA_ChannelPriority3);
    DMA_SetChannelPriority(DEMO_DMA, DEMO_DMIC_RX_CHANNEL,
kDMA_ChannelPriority2);
    DMA_CreateHandle(&s_i2sTxDmaHandle, DEMO_DMA, DEMO_I2S_TX_CHANNEL);
    DMA_CreateHandle(&s_dmicRxDmaHandle, DEMO_DMA,
DEMO_DMIC_RX_CHANNEL);
    memset(&dmic_channel_cfg, 0U, sizeof(dmic_channel_config_t));

    PRINTF("Configure DMIC\r\n");
    /* Config DMIC */
    dmic_channel_cfg.divhfclk            = kDMIC_PdmDiv1;
    dmic_channel_cfg.osr                 = 32U;
    dmic_channel_cfg.gainshft            = 3U;
    dmic_channel_cfg.preac2coef          = kDMIC_CompValueZero;
    dmic_channel_cfg.preac4coef          = kDMIC_CompValueZero;
    dmic_channel_cfg.dc_cut_level        = kDMIC_DcCut155;
    dmic_channel_cfg.post_dc_gain_reduce = 1U;
    dmic_channel_cfg.saturate16bit       = 1U;
    dmic_channel_cfg.sample_rate         = kDMIC_PhyFullSpeed;
    DMIC_Init(DMIC0);
#if !(defined(FSL_FEATURE_DMIC_HAS_NO_IOCFG) &&
FSL_FEATURE_DMIC_HAS_NO_IOCFG)
    DMIC_SetIOCFG(DMIC0, kDMIC_PdmDual);
#endif
    DMIC_Use2fs(DMIC0, true);
    DMIC_EnableChannelDma(DMIC0, DEMO_DMIC_CHANNEL, true);
    DMIC_ConfigChannel(DMIC0, DEMO_DMIC_CHANNEL, kDMIC_Left,
&dmic_channel_cfg);
    DMIC_FifoChannel(DMIC0, DEMO_DMIC_CHANNEL, FIFO_DEPTH, true, true);
    DMIC_EnableChannnel(DMIC0, DEMO_DMIC_CHANNEL_ENABLE);

    PRINTF("Configure I2S\r\n");
    /*
     * masterSlave = kI2S_MasterSlaveNormalMaster;
     * mode = kI2S_ModeI2sClassic;
     * rightLow = false;
     * leftJust = false;
     * pdmData = false;
     * sckPol = false;
     * wsPol = false;
     * divider = 1;
     * oneChannel = false;
     * dataLength = 16;
     * frameLength = 32;
     * position = 0;
     * fifoLevel = 4;
     */
    I2S_TxGetDefaultConfig(&tx_config);
    tx_config.divider     = DEMO_I2S_CLOCK_DIVIDER;
    tx_config.masterSlave = DEMO_I2S_TX_MODE;
    tx_config.oneChannel  = true;
    I2S_TxInit(DEMO_I2S_TX, &tx_config);
    I2S_TxTransferCreateHandleDMA(DEMO_I2S_TX, &s_i2sTxHandle,
&s_i2sTxDmaHandle, i2s_Callback, NULL);
    DMIC_TransferCreateHandleDMA(DMIC0, &s_dmicDmaHandle,
dmic_Callback, NULL, &s_dmicRxDmaHandle);
    DMIC_InstallDMADescriptorMemory(&s_dmicDmaHandle,
s_dmaDescriptorPingpong, 2U);
    DMIC_TransferReceiveDMA(DMIC0, &s_dmicDmaHandle, s_receiveXfer,
DEMO_DMIC_CHANNEL);

    PRINTF("DMIC->DMA->I2S->CODEC running \r\n\r\n");
    while (1)
    {
        if (s_emptyBlock < BUFFER_NUM)
        {
            i2sTxTransfer.data     = s_buffer + s_writeIndex *
BUFFER_SIZE;
            i2sTxTransfer.dataSize = BUFFER_SIZE;
            if (I2S_TxTransferSendDMA(DEMO_I2S_TX, &s_i2sTxHandle,
i2sTxTransfer) == kStatus_Success)
            {
                if (++s_writeIndex >= BUFFER_NUM)
                {
                    s_writeIndex = 0U;
                }
            }
        }
    }
}
```

5.3 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 provide 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 more complete. It uses rpmsg_lite as IPC protocol, which based on MU to transfer messages between the two cores.

First Cortex M33 side initialize 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 try 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 structure to pass the commands/ messages. In SDK example, it shows how to pass a MP3 decoding message , a AAC decoding message, a DMIC recording message etc. 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 etc. 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 will listen 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 receive, will handle message and proceed the command. For MP3 decoding case, it will decode the MP3 data in the input buffer, and flush the output buffer. It will also fill 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 co-processor 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.

5.4 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's 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.

5.5 System Optimization

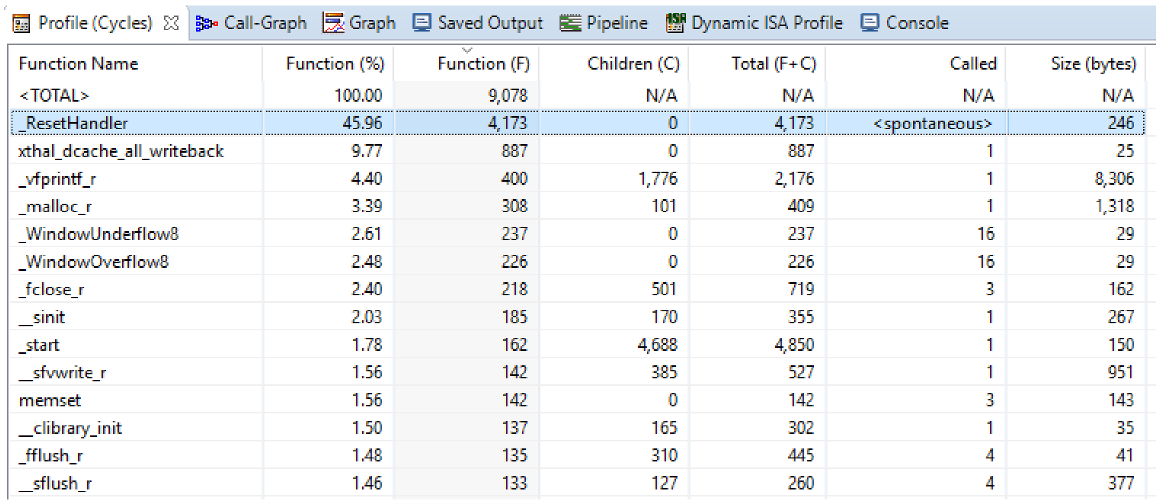
Performance and power efficiency are key for embedded systems. Here are some tips & best practices for system optimization from this perspective.

5.5.1 Profiling

Xtensa Xplorer IDE is a powerful tool. It can run software simulation and profile the application directly. Both simulation and profiling could be cycle accurate. This is really convenient for algorithm or heavy application developers.

Below showing the profiling result of helloworld program on simulation console as well as profiling chart, partially:

```
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 )
```



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

Please note that the generated license file only supports debug/run on the RT6xx device target. It does not support software simulation/Xplorer ISS. Please contact Cadence directly if you have special needs to run software simulations.

It is also common to simply measure exact cycle counts for specific processing/timing measurements. Below showing some example code how to do 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);
```

5.5.2 Using Local Memories

RT6xx HiFi4 DSP has 64K data TCM and 64K instruction TCM. The TCM is filled with less than 2K 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. Please consider to use TCM as much as possible.

5.5.3 Power Efficiency

RT6xx HiFi4 DSP has been equipped as a powerful processing core that can run at 600MHz, 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 ~130mW at Vddcore 1.1V / 452MHz, and ~4mW at Vddcore 0.7V / 29MHz, at room temperature. Please profile or do cycle counts for software workload. If it only requires 300MHz at peak, then you don't need to run at full power, you could run it with Vddcore 0.8V 300MHz. For more Vddcore and DSP frequency operating conditions, please refer to any dsp example\cm33\pmic_support.c BOARD_SetPmicVoltageForFreq(), or refer to 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;
- Current could be further reduced when you turn off clocks to un-used memory partitions. Please refer to 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.

@ END