Getting Started with Xplorer for EVK-MIMXRT595

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 MIMXRT595.

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/fusion-f1-dsp-sdk-for-rt500/
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.

➔ Download and install the **XTENSA Xplorer IDE V8.0.15** for your operating system (Windows or Linux).

➔ Download the **DSP Configuration V8.0.15** for your operating system – this will be installed later through the IDE (see section 1.4).

NXP recommends version **8.0.15** of the Xtensa Xplorer IDE and tools for use with the RT500 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 RT500 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
```

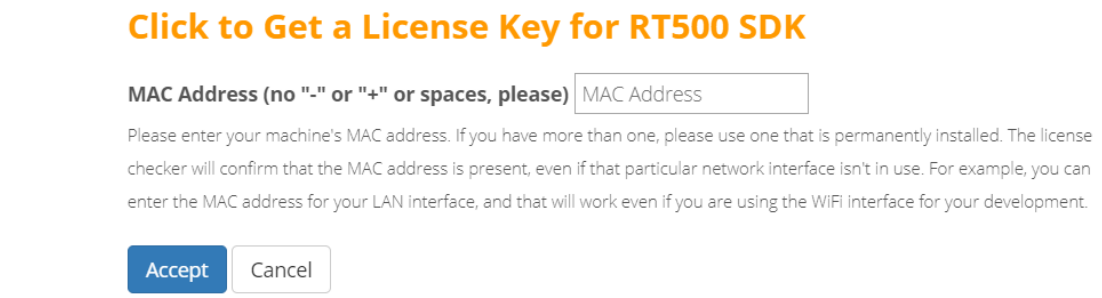
Linux NOTE: MAC address MUST be associated with eth0 interface. If not, Flexlm cannot perform the license checkout and you will not be able to compile or simulate you code. If your host has the MAC address associated with another interface, for example em1, you may use the following approach, or another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

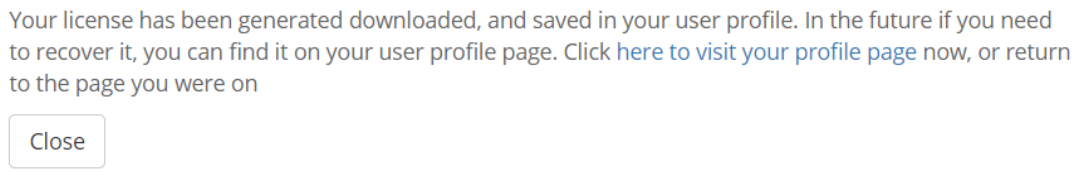
click Below button.

➔ CLICK TO GET A LICENSE KEY FOR RT500 SDK

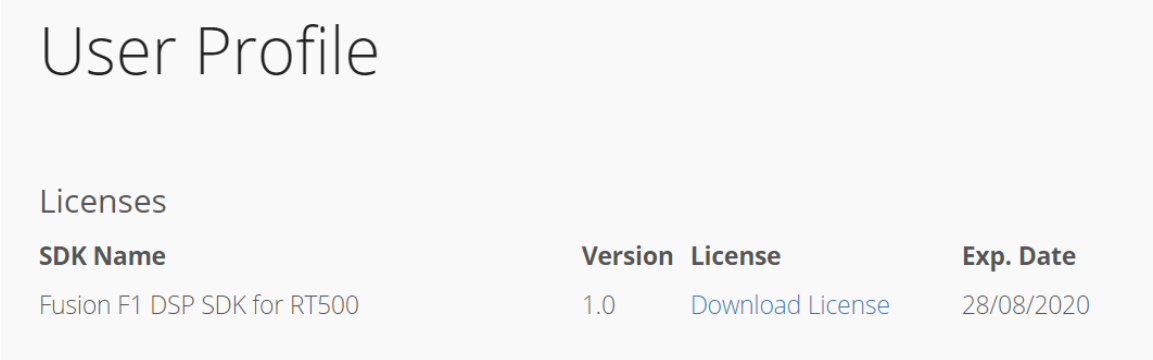


➔

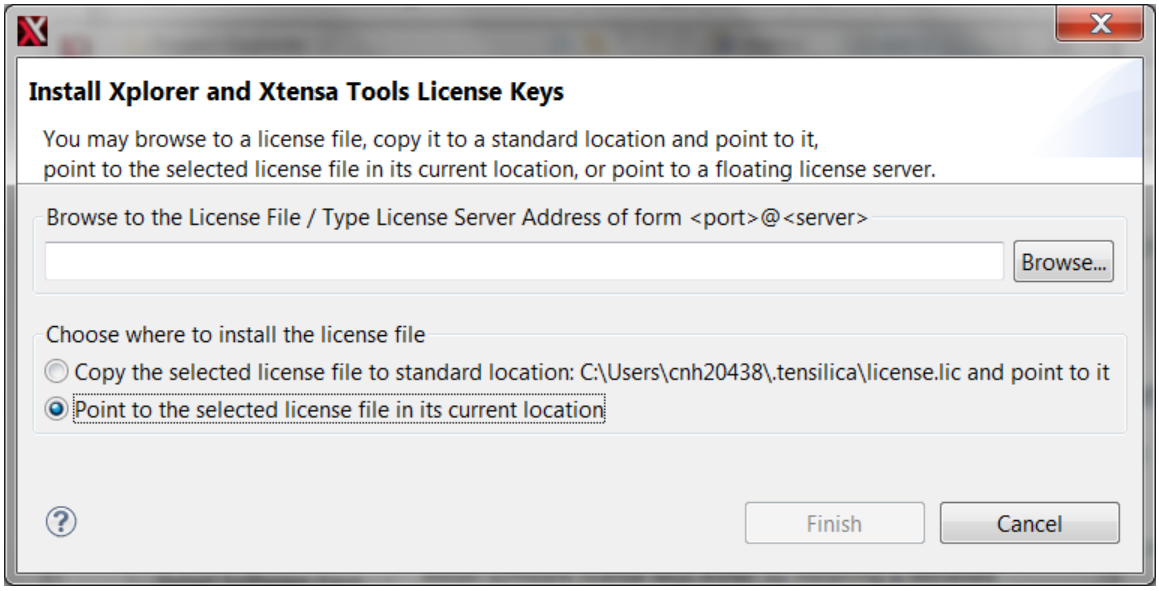
You will see below message :



You can check the license in the user-profile or email:



Once the license file has been generated and downloaded, open your recently installed Xplorer 8.0.15, select menu Help -> Xplorer License Keys -> Install Software Keys, select the license key file, and click 'Finish':



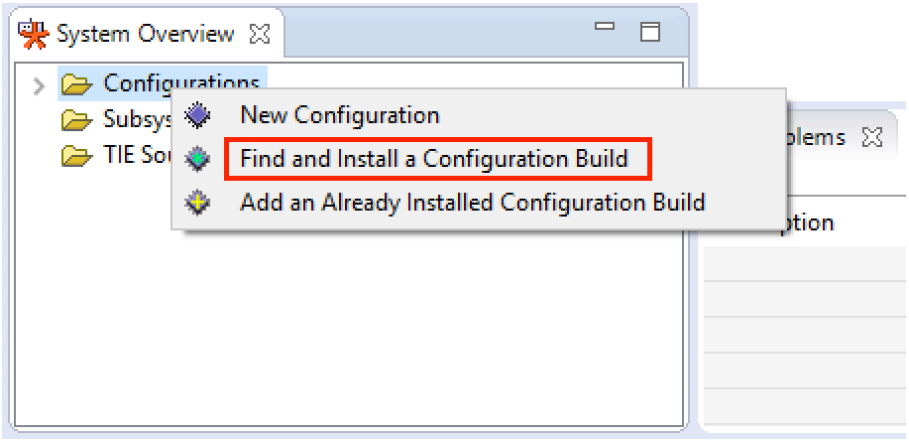
NOTE: The generated license file only supports debug/run on the RT500 device target. It does not support software simulation/Xplorer ISS. Please contact Cadence directly if you have special needs to run software simulations.

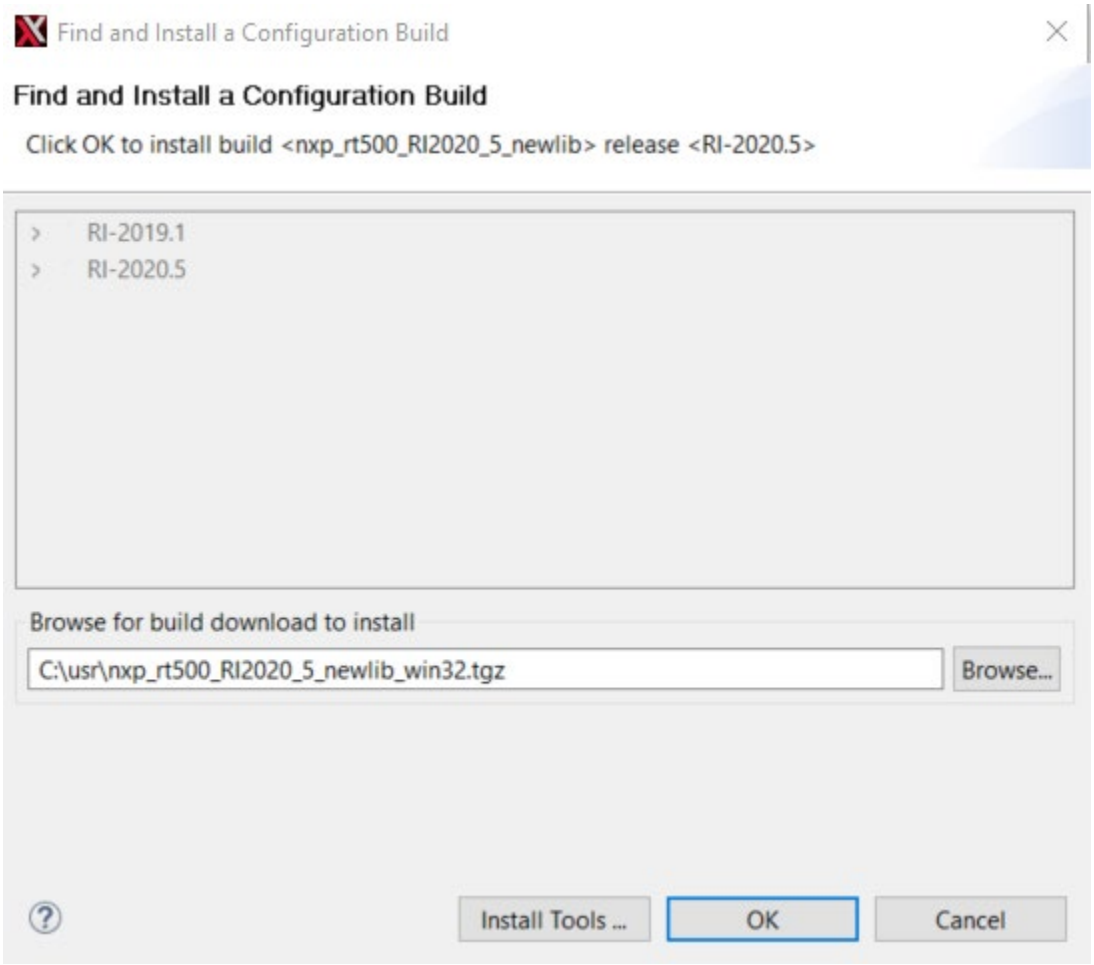
1.4 Install RT500 DSP Build Configuration

'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 RT500.

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.

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'.





Press OK after selecting the directory.

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\xt-ocd-14.05-windows64-installer.exe`

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

At this moment xt-ocd supports J-Link and ARM RVI/DSTREAM probes over Serial Wire Debug (SWD) for RT500. xt-ocd installs support for J-Link probes but does not install the required J-Link drivers which must be installed separately. The RT500 requires J-Link software version 6.46 or newer.

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 JINK 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.

1.5 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/RI2020.5/tools/ XtensaTools_RI_2020_5_linux.tgz
```

With the tools package and the DSP Build Configuration package available from the Tensilica Tools download site (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_2020_5_linux.tgz -C ~/xtensa/tools

# Set up the configuration-specific core files:
tar zxvf nxp_RT500_RI2020_5_newlib_linux_redist.tgz -C ~/xtensa/builds

# Install the Xtensa development toolchain:
cd ~/xtensa
./builds/RI-2020.5-linux/nxp_RT500_RI2020_5_newlib/install \
  --xtensa-tools./tools/RI-2020.5-linux/XtensaTools \
  --registry ./tools/RI-2020.5-linux/XtensaTools/config
```

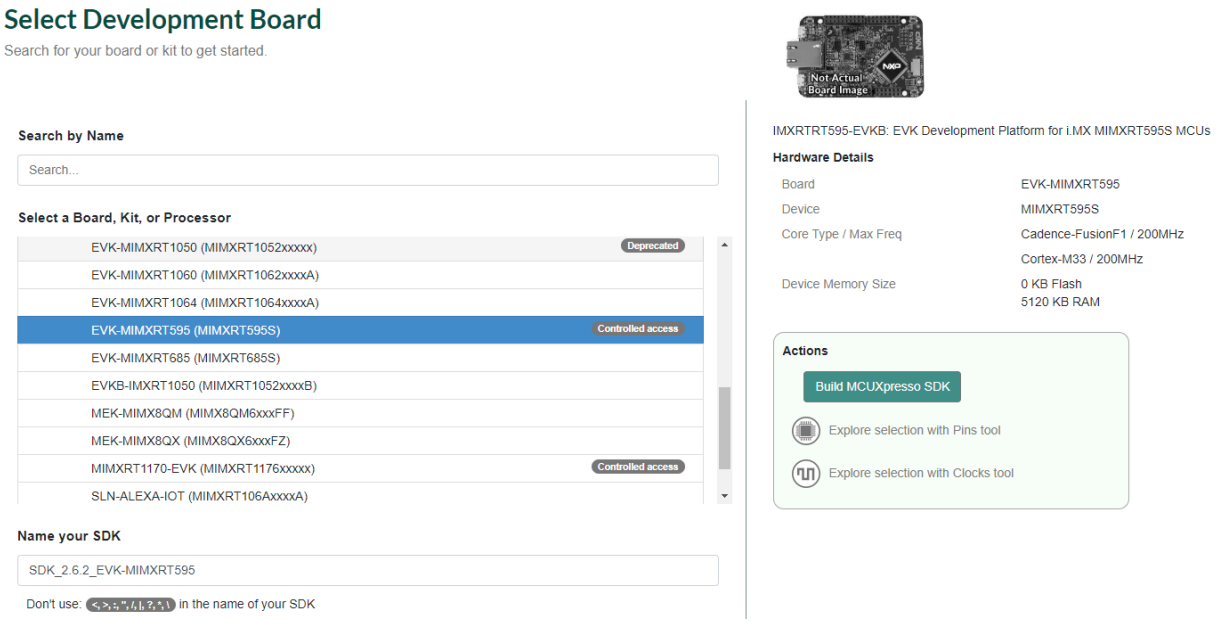
2 Install MCUXpresso SDK

2.1 Download MCUXpresso SDK for RT500

DSP enablement for RT500, including drivers, middleware libraries, and demo applications are included with the latest RT500 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: ’RT595’
- Select ‘EVK-MIMXRT595’
- Click ‘Build MCUXpresso SDK’



2.2 MCUXpresso SDK DSP Enablement

Inside the MCUXpresso SDK release package for RT500 you will find the following DSP-specific enablement:

`<SDK_ROOT>/devices/MIMXRT595S/`

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/evkmimxrt595/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

2.3 DSP Core Initialization

In order to minimize power consumption, the DSP core is NOT powered when RT500 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/MIMXRT595S/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);
```

<SDK_ROOT>/devices/MIMXRT595S/drivers/fsl_dsp.h:

```
/* 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/evkmimxrt595/dsp_examples/hello_world_usart /cm33/

```
/* 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.

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.

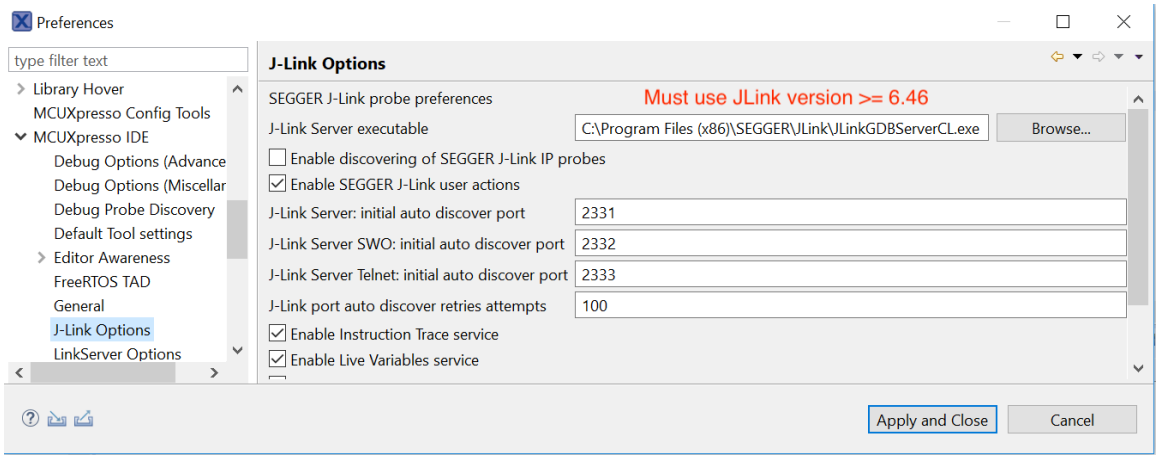
In order to debug the 'Hello World' DSP application, you will first need to setup and execute the ARM application using an environment of your choosing:

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

  `<SDK_ROOT>/boards/evkmimxrt595/dsp_examples/hello_world_usart/cm33/`

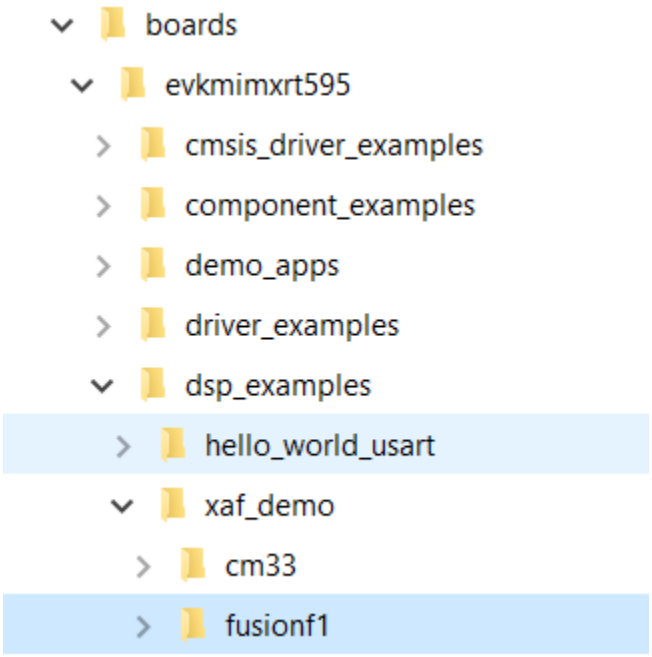
Preparing an ARM core development environment is outside of the scope of this document. Please refer to the document 'Getting Started with MCUXpresso SDK for MIMXRT500.pdf' located under <SDK_ROOT>/docs/ for information on how to use the SDK for ARM core development.

> **NOTE:** SEGGER J-Link software version >= 6.46 is required for compatibility with RT500. MCUXpresso IDE may ship with an older version, which can be customized as described as follows:



3.2 Prepare DSP Core for 'Hello World'

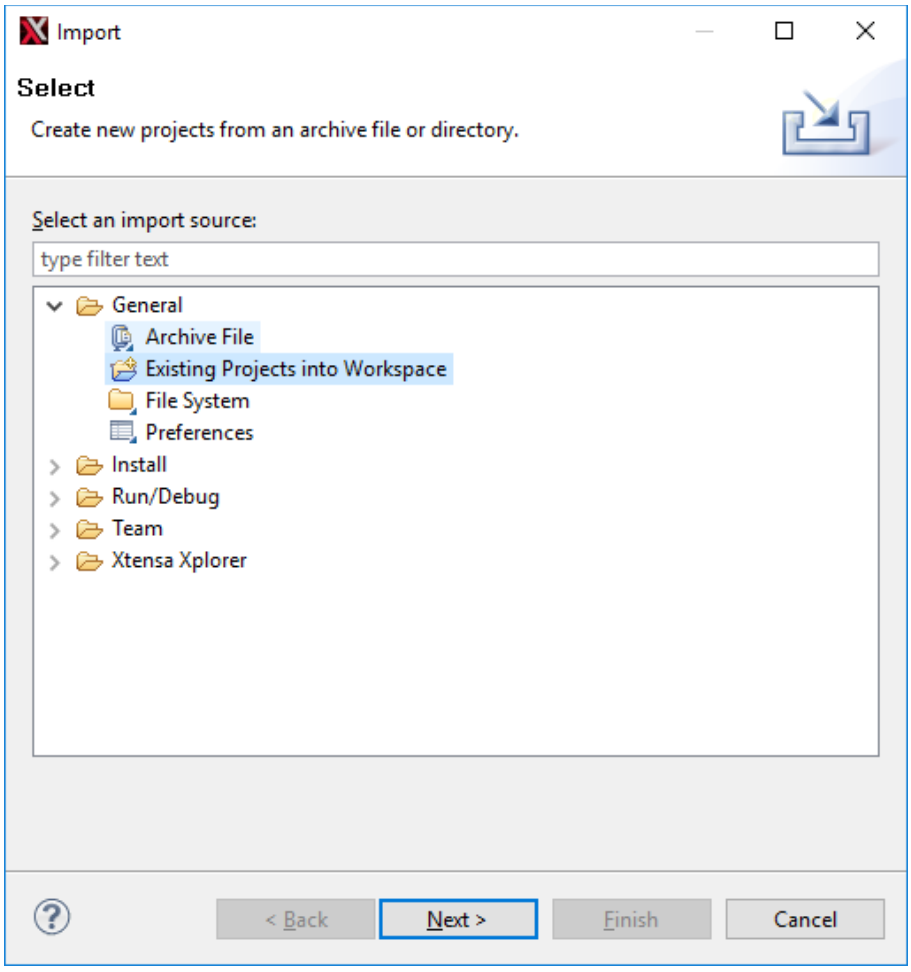
The RT500 SDK provides a collection of DSP example applications located under boards/evkmimxrt595/dsp_examples/. Each DSP example has two source directories, one for the ARM Cortex-M33 core ('cm33') and one for the DSP fusion f1 core ('fusionf1'):

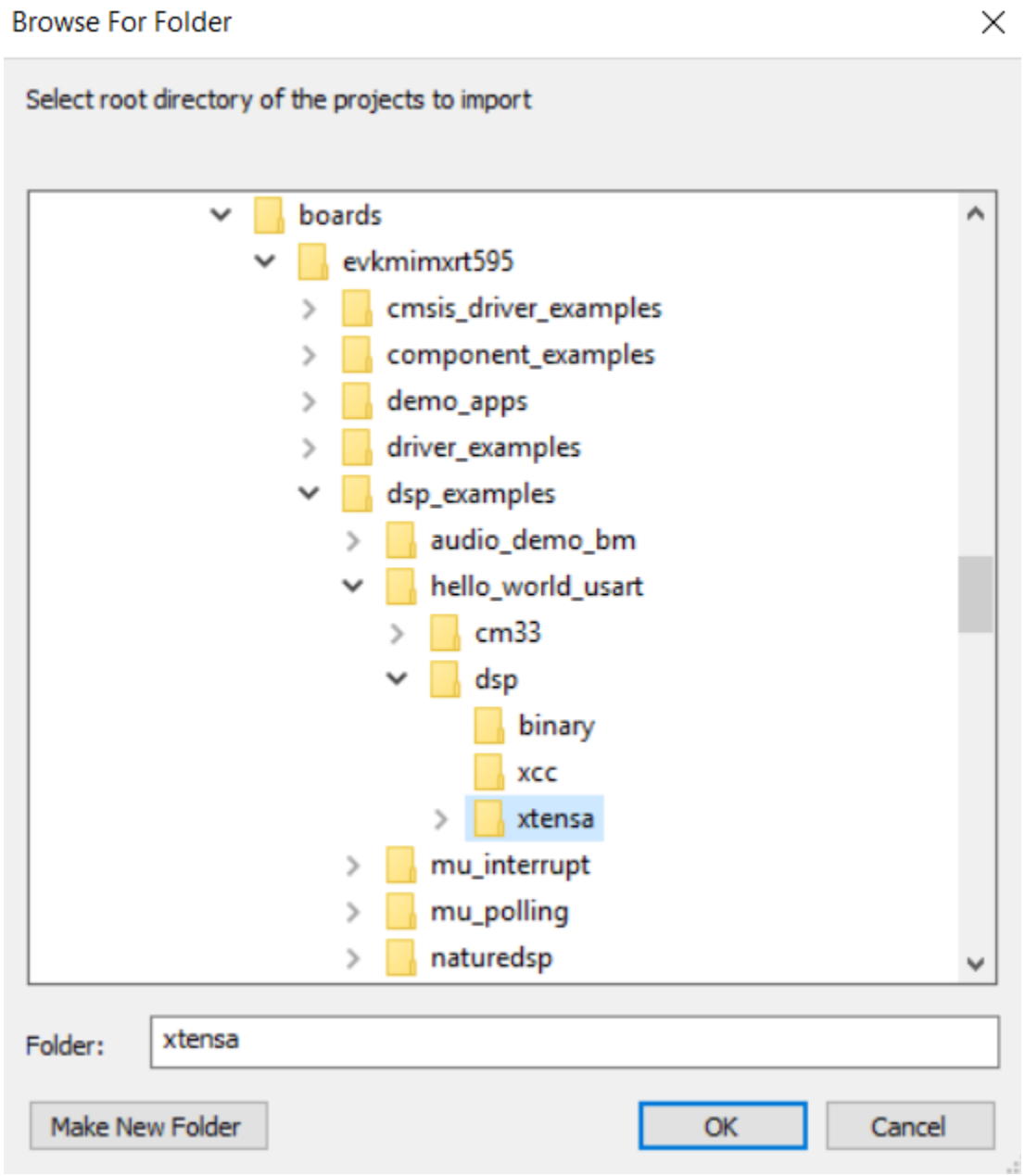


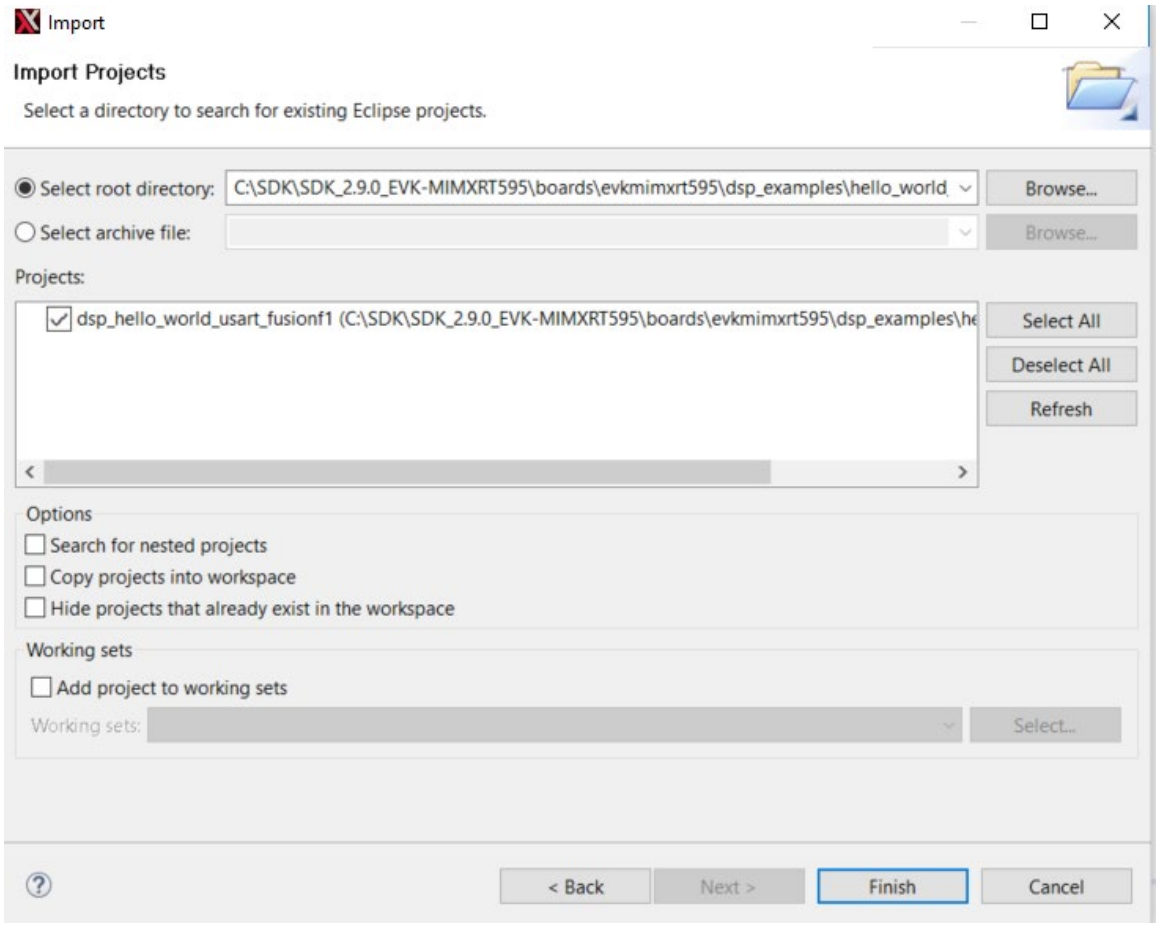
Under these directories will be build projects for different supported toolchains. For the DSP example above, the 'xcc' project will allow to build on the command line and the 'xtensa' directory is an Xplorer IDE project.

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\evkmimxrt595\dsp_examples\hello_world_usart \fusionf1\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
boards
evkmimxrt595
cmsis_driver_examples
component_examples
demo_apps
driver_examples
dsp_examples
audio_demo_bm
hello_world_usart
cm33
dsp
binary
xcc
xtensa
mu_interrupt
mu_polling
naturedsp
Folder:
xtensa
Make New Folder
OK
Cancel



Click Finish button.

Once imported, you will see 'dsp_hello_world_usart_fusionf1' in the Project Explorer.

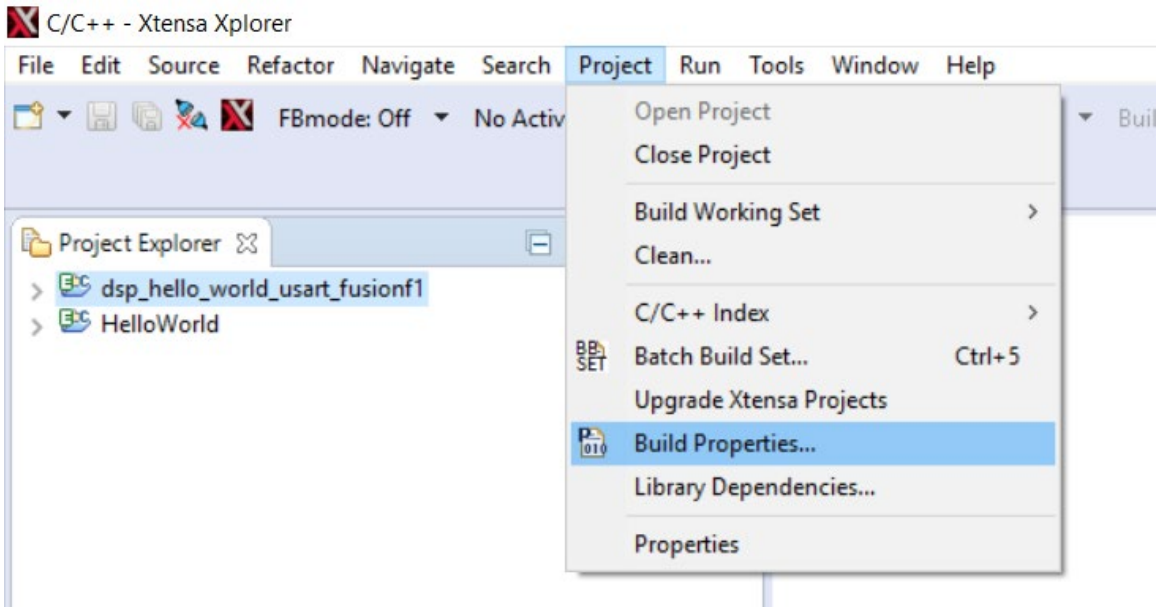
3.3 DSP Linking Profiles

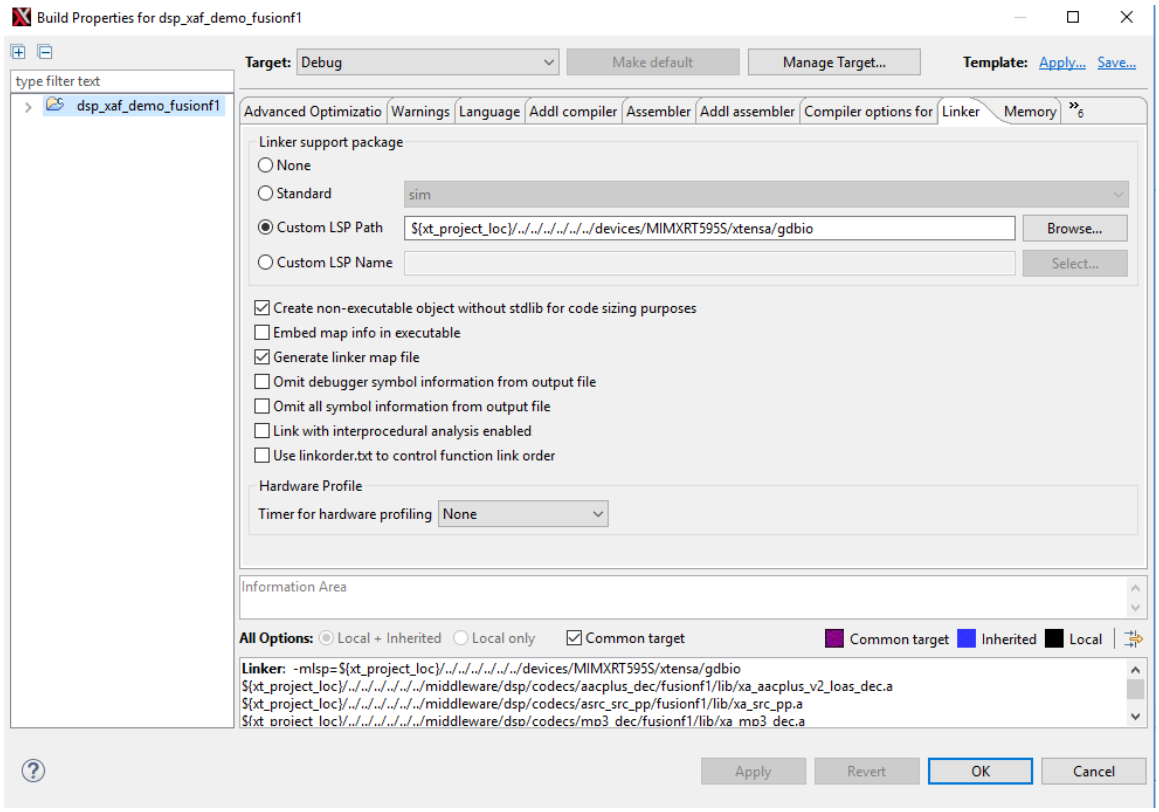
The Xtensa Software Tools use linker support packages (LSPs) to link a Fusion DSP application for the RT500. An LSP includes both a system memory map and a collection of libraries to include into the final binary. These LSPs are provided in the MCUXpresso SDK under <SDK_ROOT>/devices/MIMXRT595S/xtensa/.

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:

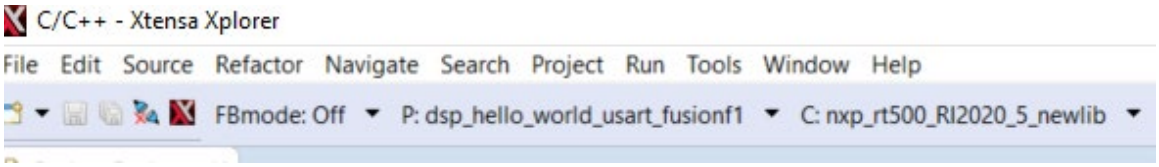




The MCUXpresso SDK ships with other standard LSPs for RT500. Please see the Cadence Linker Support Packages (LSPs) Reference Manual for more information on using LSPs and how to create a custom memory map using Xtensa software tools.

3.4 Build the Xplorer project.

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



3.5 Start Xtensa Debugger Daemon

Connect the EVK board to a PC via the USB debug interface (J40) and open up a serial interface on your PC using a terminal tool such as Tera term or PuTTY on Windows or screen on Linux.

Remove Jumpers JP17, JP18, JP19 for SWD to connect to the chip and Serial interface on J40.

I. Load CM33 hello_world_usart you build from step 3.1 using J-link

II. To debug DSP applications on RT500 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 14.05 on Windows.

Please update topology.xml file with your J-link serial number.

Topology.xml file : Edited J-link serial number in usbser field and please see below picture for any reference. Make sure dap = 1 in order for it to work.

```
<configuration>
        <controller id='Controller0' module='jlink' usbser='600112004' 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>
```

Execute the daemon with your custom topology:

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

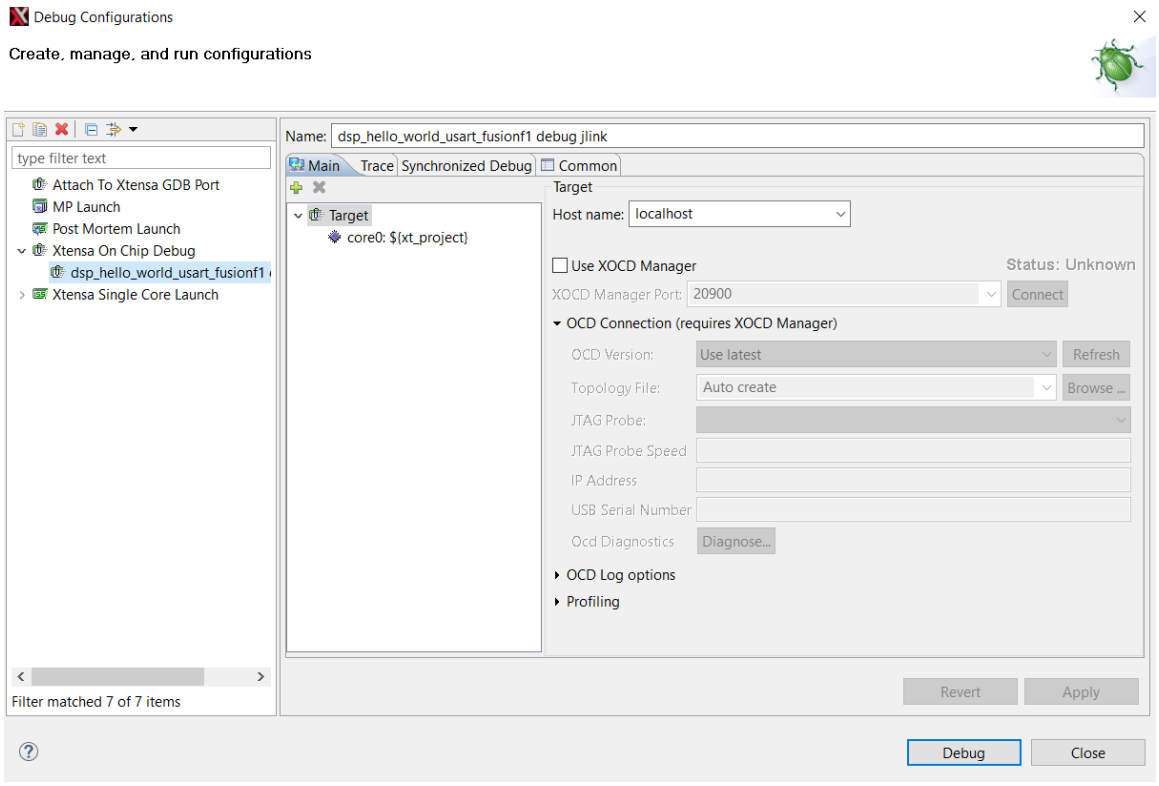
C:\WINDOWS\system32\cmd.exe - xt-ocd.exe -c topology.xml

```
C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.05>
C:\Program Files (x86)\Tensilica\Xtensa OCD Daemon 14.05>xt-ocd.exe -c topology.xml
XOCD 14.05 2020-10-12 17:38:02
(c) 1999-2021 Cadence Design Systems Inc. All rights reserved.
[Debug Log 2021-02-02 22:06:58]
Loading module "gdbstub" v2.0.0.12
Loading module "jlink" v2.0.2.0
Using JLINK lib v.68803
Jlink USB Serial Number: 600112004
Connected to Jlink Device:
  Name:'SEGGER J-Link ARM'
  S/N:600112004
  Firmware: J-Link V10 compiled Nov 12 2020 10:06:35
  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...
```

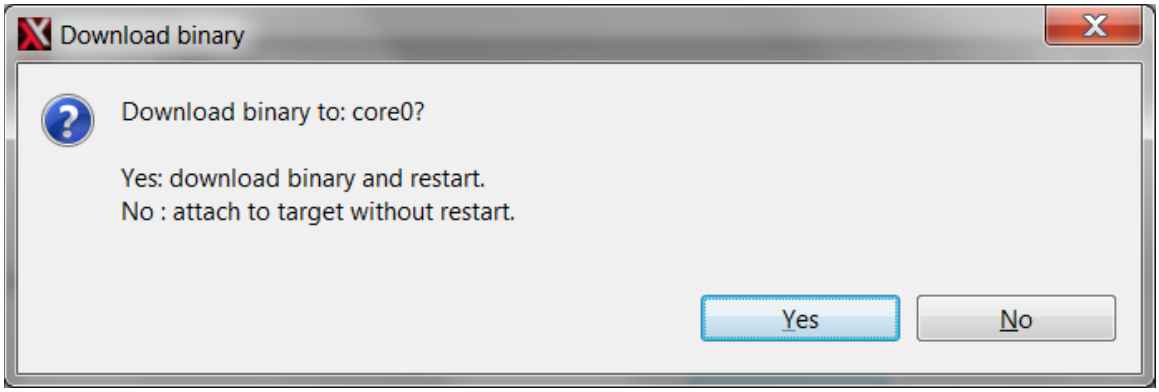
Please note that some warning messages are expected and can be ignored. If you receive an error initializing the XDM driver, it could be that the DSP core needs to be initialized and started before debugging – see section 2.3 of this document for more details.

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

III. 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:

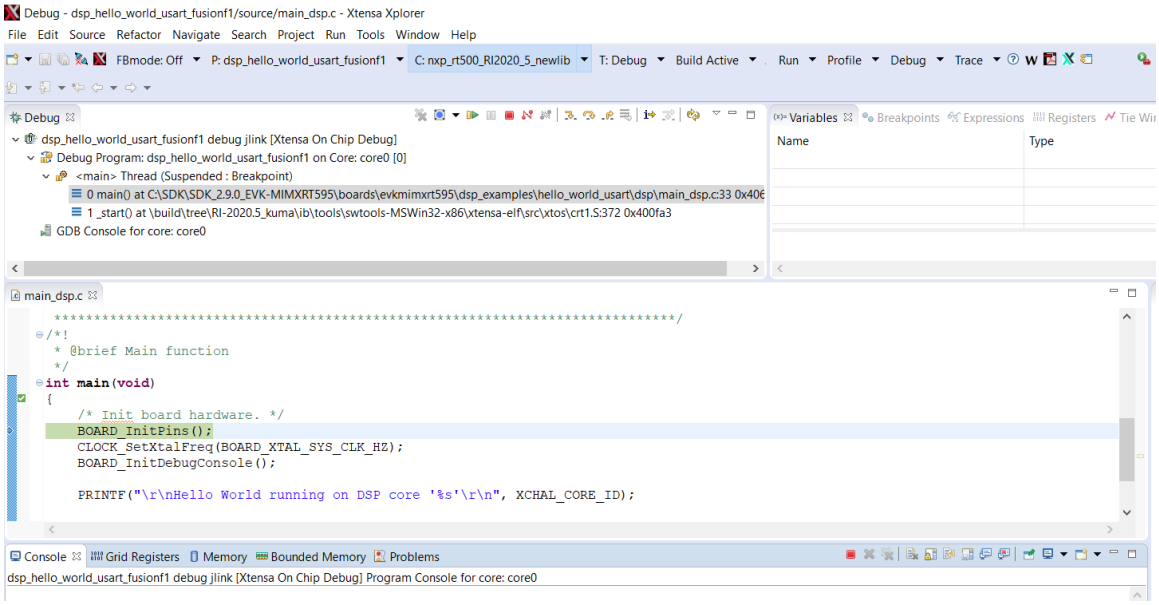


Once the 'Debug' button is selected, the actual debug on chip will be started. Xplorer will ask you if you like to download binaries to the hardware. Select Yes.



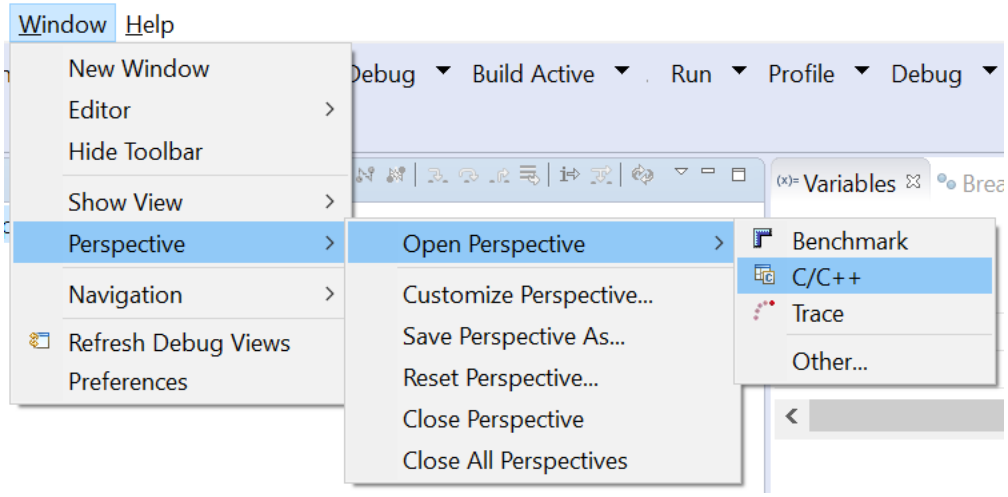
Xplorer IDE will transition to the 'Debug' perspective after binary download:

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




```
Hello World running on core nxp_RT500_RI2020_5_newlib
```

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



3.6 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.

3.6.1 EVK Board Setup for Audio Demo

The DSP audio demo is tested against EVK-MIMXRT595 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 two jumpers on the board as follows:

- The 1-2 connected for JP7, JP8, JP9, JP28,JP29.

The demo uses the UART for console input and output. Connect the EVK board to a PC via the USB debug interface (J40) and open up a serial interface on your PC using a terminal tool such as Tera term or PuTTY on Windows or screen on Linux.

- Remove Jumpers JP17, JP18, JP19 for SWD to connect to the chip and Serial interface on J40.

- DSP debugging is through SWD only, so please connect J-Link to SWD interface.

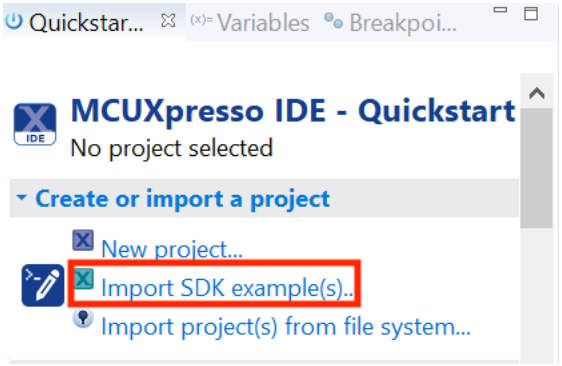
3.4.2 Debug Audio Demo

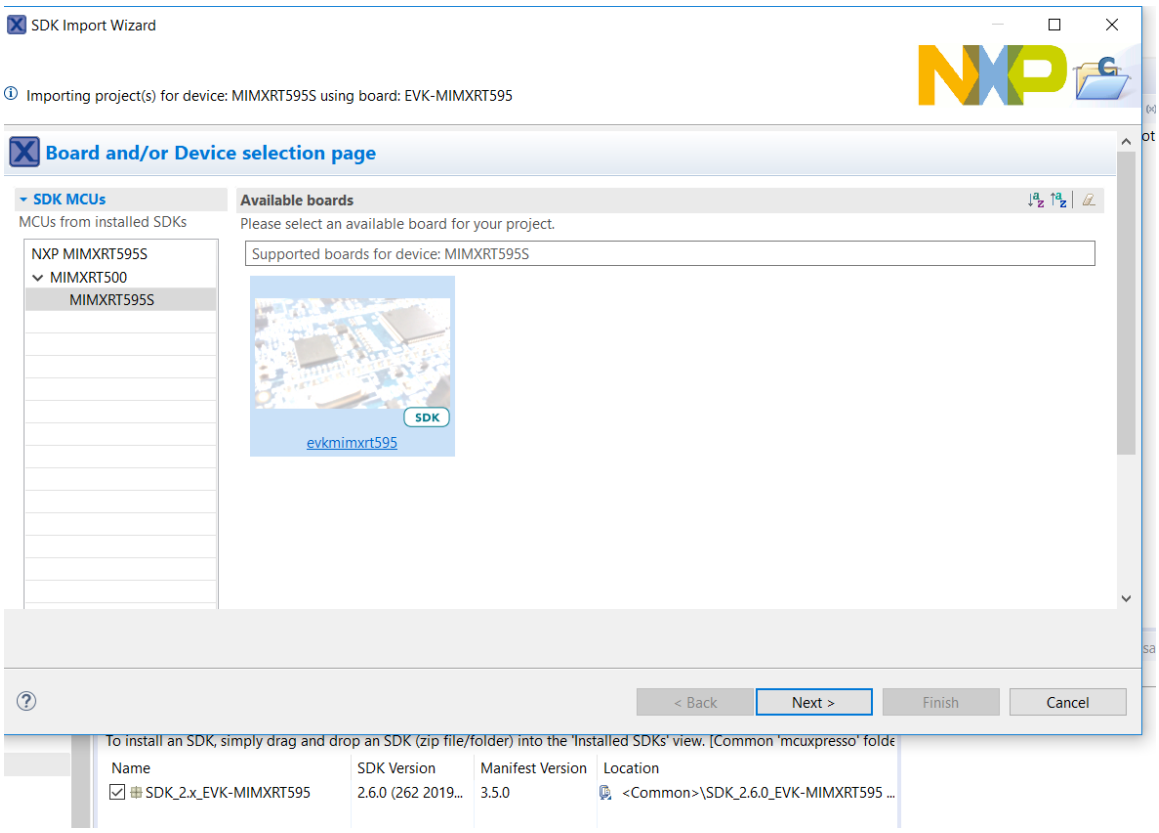
In order to debug this DSP application, you will first need to setup and execute the ARM application using an environment of your choosing (see 'Getting Started with MCUXpresso SDK for EVK-MIMXRT595.pdf' for ARM development environment options).

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

- Install the MCUXpresso SDK for RT500 into the MCUXpresso IDE using the 'Installed SDKs' panel at the bottom:

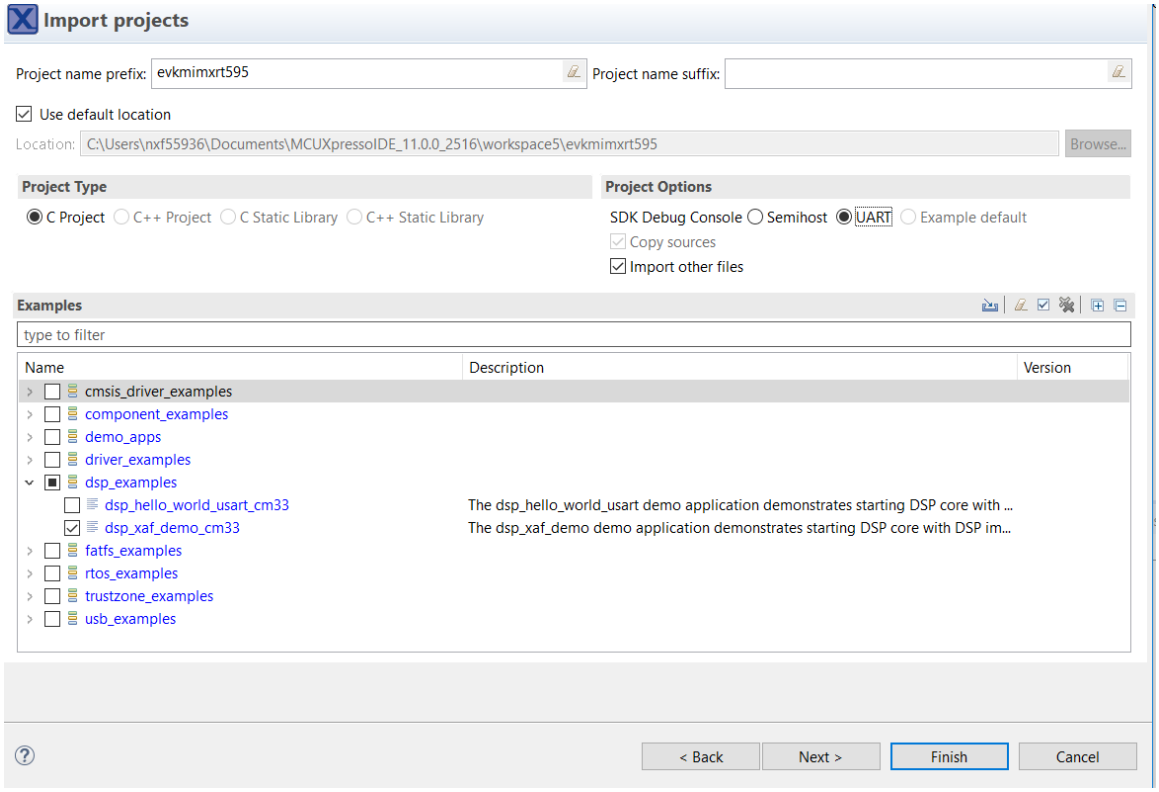
- Use the QuickStart menu on the lower left of the screen to import an example from the installed SDK:



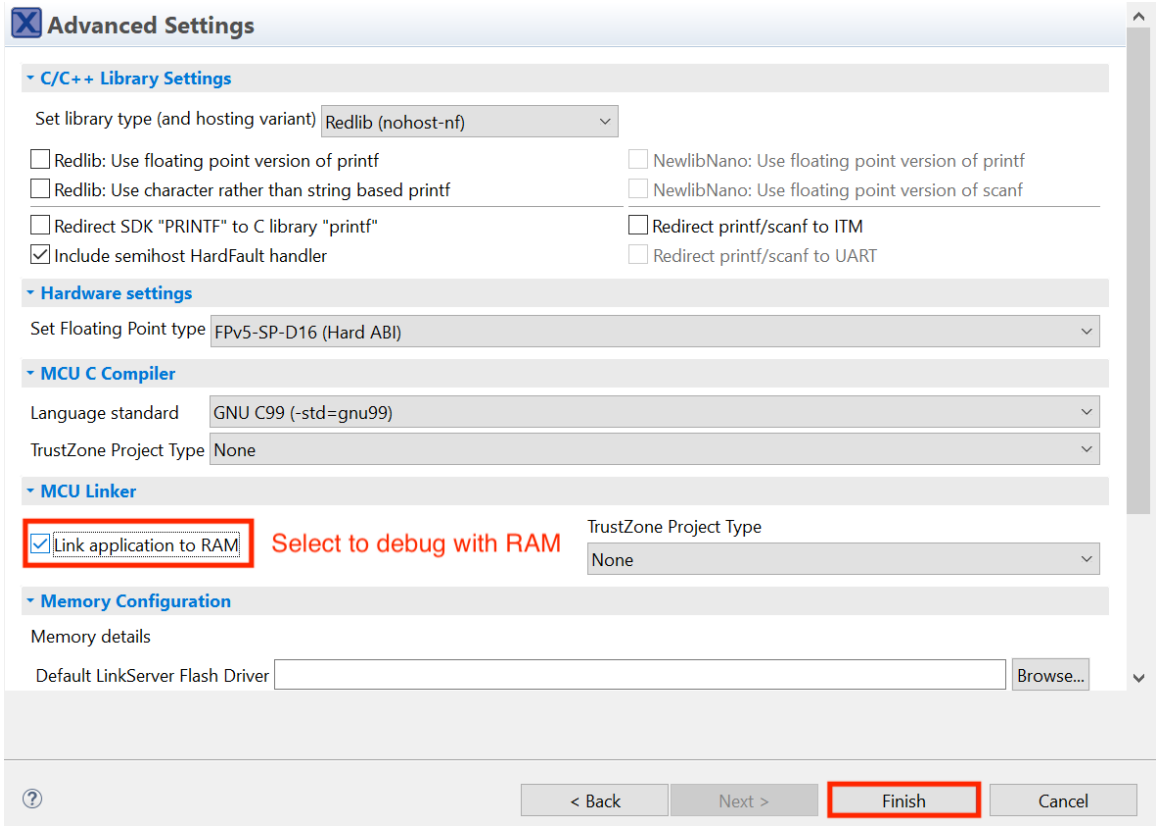


- Select the ‘dsp_xaf_demo_cm33’ example for Cortex-M33 core:

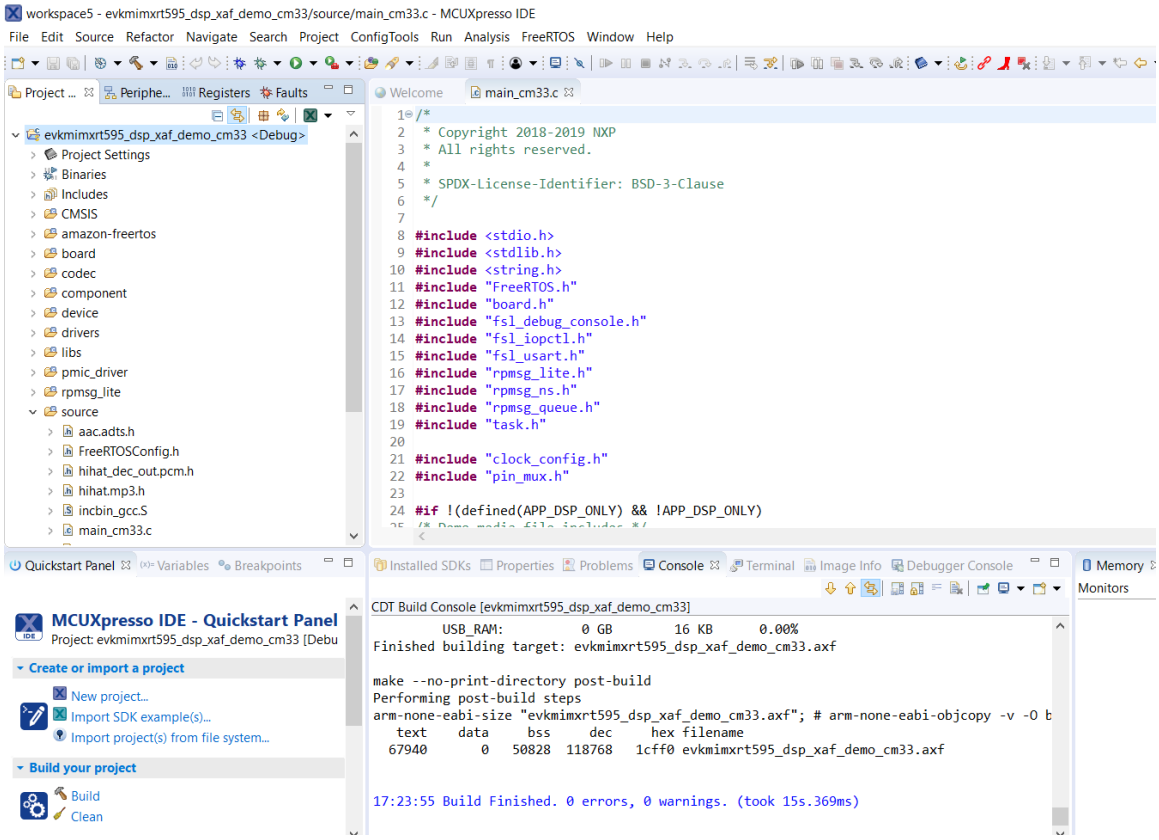
Please select UART in project options after clicking the dsp_xaf_demo_cm33 example project.



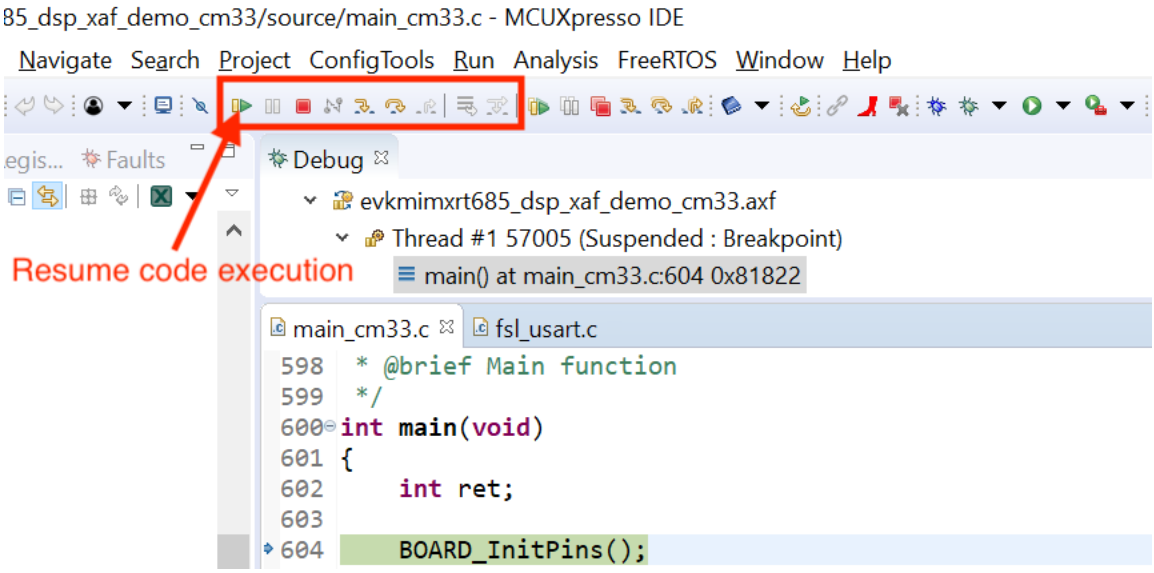
- Configure project settings, notably choose to link the application to RAM for ease of initial debug. Select ‘Finish’ to complete the import:



- Build the project and launch the debugger on success:



- Use the debug toolbar to resume the code execution:



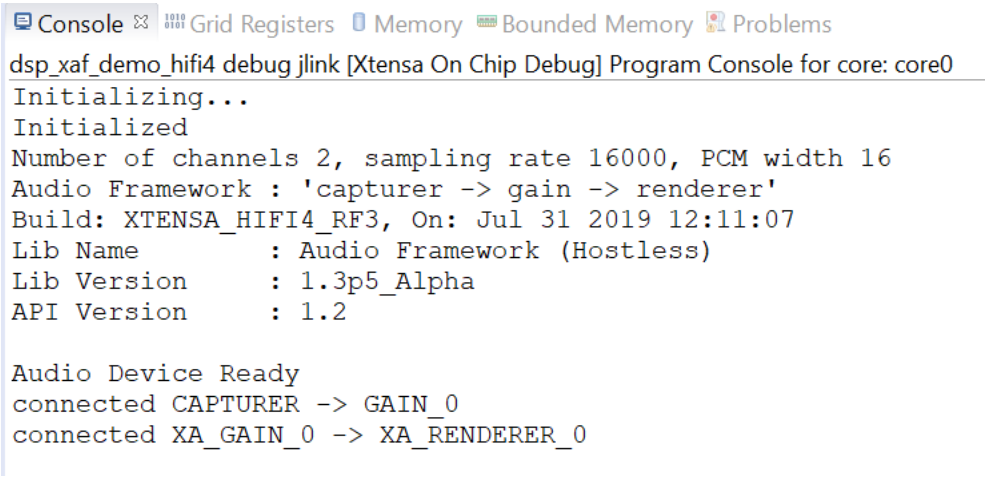
- Observe serial terminal output with shell prompt:

```
Configure WM8904 codec

XAF DSP demo start

SHELL build: Nov 30 2020
Copyright  2021  NXP
SHELL>>
```

- Using the Xplorer IDE, load and execute xaf_demo using the procedure described in sections 3.4 and 3.5 of this document.
- After the DSP application is running, use the serial shell to invoke the ‘rec’ 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):

```
SHELL>> help
"help": List all the registered commands
"exit": Exit program
"echo": Query DSP for component versions
"rec": Record DMIC audio and playback on WM8904 codec
SHELL>>
SHELL>> rec
```

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



- Please make sure jumper settings on the board as follows to hear the sound output.

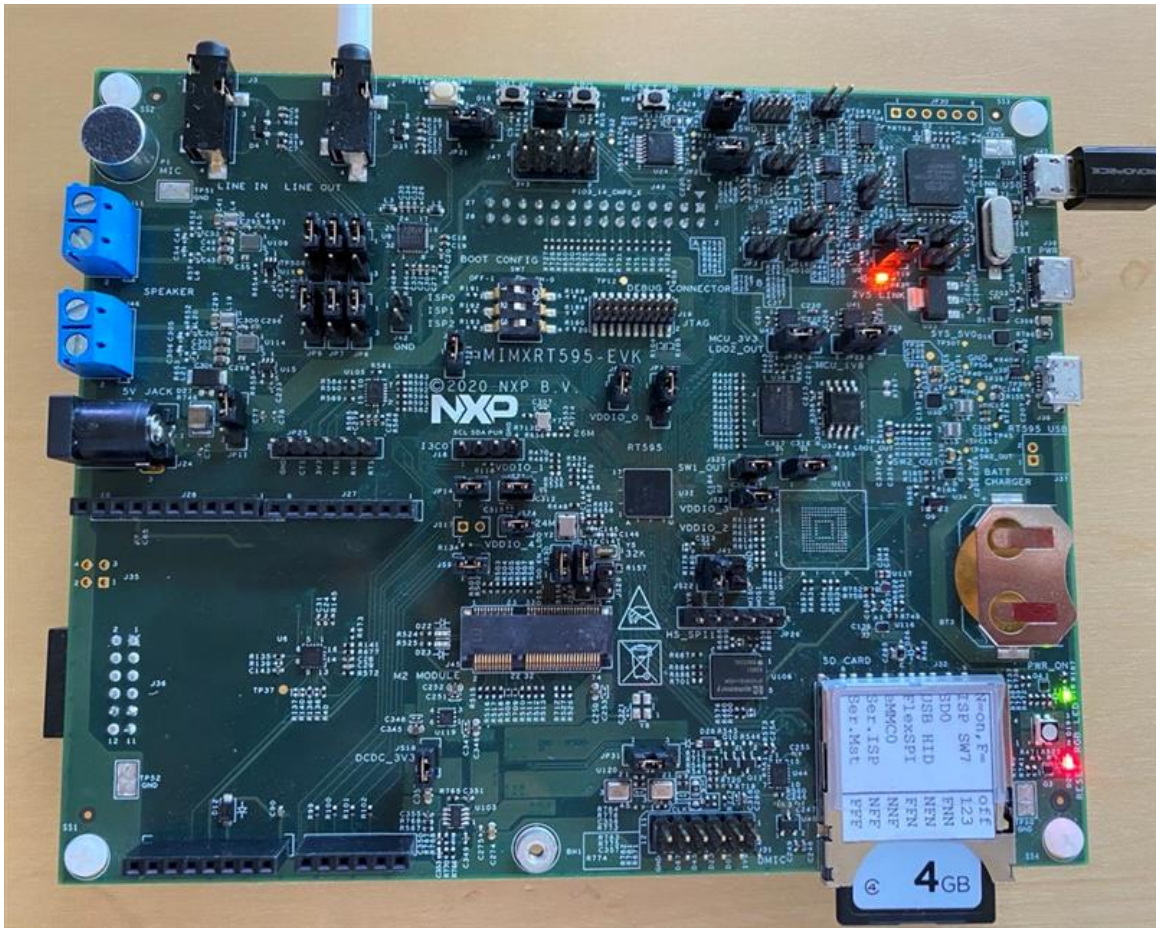


Figure : MIMXRT595-EVK Board.

3.7 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:

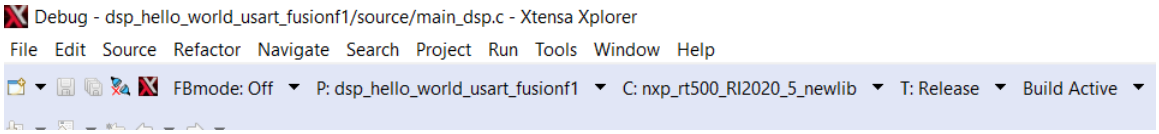
```
DSP_IMAGE_COPY_TO_RAM
```

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):



Two DSP binaries are generated

`<SDK_ROOT>\boards\evkmimxrt595\dsp_examples\xaf_demo\dsp\binary\`**`dsp_text_release.bin`**
`<SDK_ROOT>\boards\evkmimxrt595\dsp_examples\xaf_demo\dsp\binary\`**`dsp_data_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

RT500 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 RT500.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/evkmimxrt595/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/evkmimxrt595/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 (NOTE: J-Link requires version >= 6.46):

```
user@linux:/opt/JLink$ ./JLinkGDBServerCLExe -device MIMXRT595_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/evkmimxrt595/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-2020.5/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-2020.5-
linux/XtensaTools/bin
# (Optional) Use environment variable to control license file
# NOTE: ~/.flexlmrc will override this selection.  Please delete that file
before proceeding.
export LM_LICENSE_FILE=~/RT500.lic

# Setup env vars needed for compile and linking
export XCC_DIR=~/xtensa/XtDevTools/install/tools/RI-2020.5-linux/XtensaTools
export XTENSA_SYSTEM=~/xtensa/XtDevTools/install/builds/RI-2020.5-
linux/nxp_RT500_RI2020_5_newlib/config
export XTENSA_CORE=nxp_RT500_RI2020_5_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/evkmimxrt595/dsp_examples/xaf_demo/dsp/xcc$
./build_debug.sh
...
[100%] Built target dsp_xaf_demo_fusion1.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-14.05$ ./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/evkmimxrt595/dsp_examples/xaf_demo/dsp/xcc$ xt-gdb
debug/dsp_xaf_demo_fusion1.elf
GNU gdb (GDB) 7.11.1 Xtensa Tools 14.05
...
Reading symbols from debug/dsp_xaf_demo_fusion1.elf...done.
(xt-gdb)
(xt-gdb) target remote localhost:20000
Remote debugging using localhost:20000
_DoubleExceptionVector ()
    at /home/xpgcust/tree/RI-2020.5/ib/tools/swtools-x86_64-linux/xtensa-
elf/src/xos/src/xos_vectors.S:216
216 /home/xpgcust/tree/RI-2020.5/ib/tools/swtools-x86_64-linux/xtensa-
elf/src/xos/src/xos_vectors.S: No such file or directory.
(xt-gdb) reset
_ResetVector ()
    at /home/xpgcust/tree/RI-2020.5/ib/tools/swtools-x86_64-linux/xtensa-
elf/src/xtos/xea2/reset-vector-xea2.S:71
71  /home/xpgcust/tree/RI-2020.5/ib/tools/swtools-x86_64-linux/xtensa-
elf/src/xtos/xea2/reset-vector-xea2.S: No such file or directory.
(xt-gdb) load
Loading section .rtos.rodata, size 0x80 lma 0x200000
Loading section .rodata, size 0x17d50 lma 0x200080
Loading section .text, size 0x633f0 lma 0x217dd0
Loading section .rtos.percpu.data, size 0x4 lma 0x27b1c0
Loading section .data, size 0x110c lma 0x27b1d0
Loading section NonCacheable, size 0x2960 lma 0x20040000
Loading section .Level3InterruptVector.literal, size 0x4 lma 0x24000000
Loading section .DebugExceptionVector.literal, size 0x4 lma 0x24000004
Loading section .NMIExceptionVector.literal, size 0x4 lma 0x24000008
Loading section .ResetVector.text, size 0x13c lma 0x24020000
Loading section .WindowVectors.text, size 0x16c lma 0x24020400
Loading section .Level2InterruptVector.text, size 0x1c lma 0x2402057c
Loading section .Level3InterruptVector.text, size 0xc lma 0x2402059c
Loading section .DebugExceptionVector.text, size 0xc lma 0x240205bc
Loading section .NMIExceptionVector.text, size 0xc lma 0x240205dc
```

```
Loading section .KernelExceptionVector.text, size 0xc lma 0x240205fc
Loading section .UserExceptionVector.text, size 0x18 lma 0x2402061c
Loading section .DoubleExceptionVector.text, size 0x8 lma 0x2402063c
Start address 0x24020000, load size 520016
Transfer rate: 8 KB/sec, 10612 bytes/write.
(xt-gdb) b main
Breakpoint 1 at 0x21ab5b: file /home/jlydick/mcu-sdk-
2.0/boards/src/dsp_examples/xaf_demo/dsp/xaf_main_dsp.c, line 366.
(xt-gdb) c
Continuing.

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

- **NOTE:** you can use the gdb command 'set substitute-path' to map the missing symbols from the toolchain libraries, for example:

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

For more 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-2020.5/docs/gnu_gdb_ug.pdf
➔ ~/xtensa/XtDevTools/downloads/RI-2020.5/docs/xtensa_debug_guide.pdf