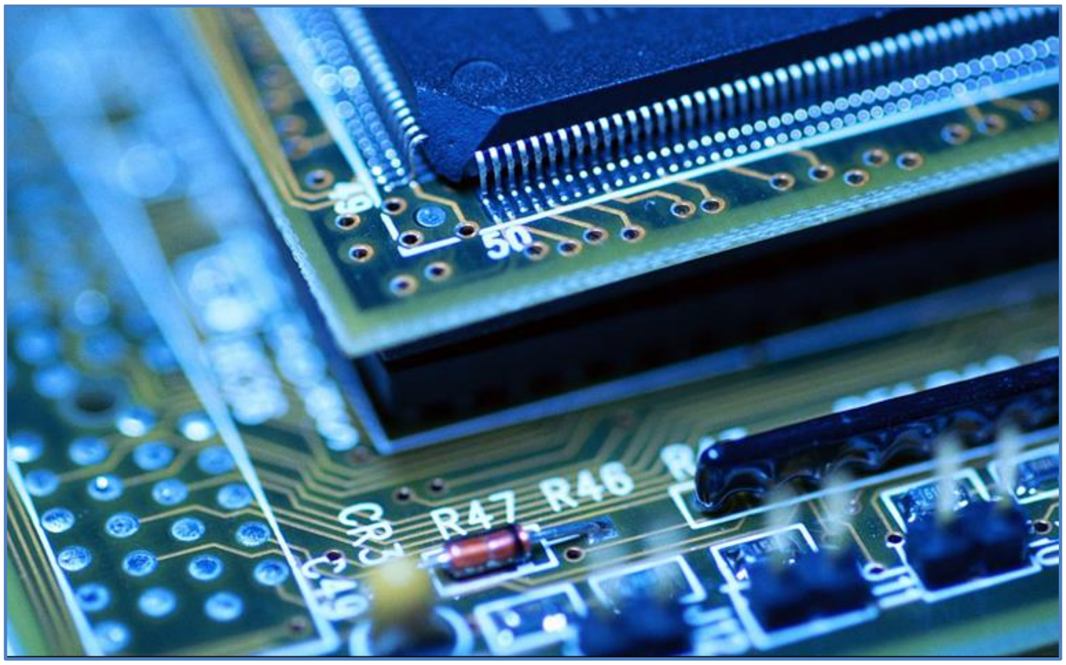
EMBEDDED ADVANTAGE

IN ASSOCIATION WITH TEXAS INSTRUMENTS

C6000/Keystone Architecture & Optimization

LAB 01: Audio Application Walkthrough



Version 1.0
2022.02.02
www.EmbeddedAdvantage.com
TMDSLCDKC6748
TMDSLCDKOMAP-L138

LAB 01: Audio Application Walkthrough

Lab Project Name: lab_01_workshopIntro_tirtos

Introduction

Welcome to the C6000 architecture workshop. This lab will introduce you to the project you will be using throughout ALL labs in this 2-day workshop.

In this lab, you will simply look at 4-5 files to see how the code works from a high level and then we will spend the next few labs optimizing the performance of the system using the TI C compiler/optimizer and then later on, putting our audio buffers in external DRAM and turning on the cache - all the time benchmarking our results of the FIR filter that is processing the audio. We have found over many years that this is the best way to compare/contrast performance to literally SEE the effects of your choices with the compiler and cache.

This project is based on the standard TI McASP example but has been re-written to optimize the McASP performance (now 93% FASTER than the TI original) and we have added some benchmarking capability and cache coherency code to support our learning during the workshop labs. If you start with the optimized CACHE (the cache lab solution) version of this project, and you are actually doing an audio application on the C6748 or OMAP-L138, just HAVING this code example is worth the price of the workshop alone and will save you MONTHS of develop time.

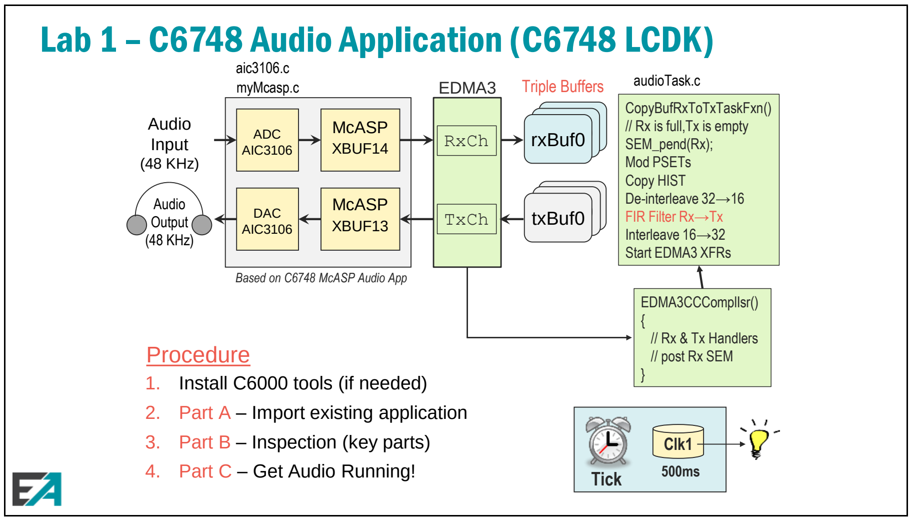
In this lab, we will NOT turn on any compiler optimizations. It is simply to walk through the code and get the audio running successfully into the board and out through your speakers or headphones. Then, in future labs, you will begin to optimize the performance.

Prerequisites

It is very important that you understand the concepts covered in this chapter's video AND you have taken the "*Getting Started With TI-RTOS*" workshop for the C6748.

Learning Objectives

- Import the McASP/EDMA3 audio application and peruse its contents



Procedure
1. Install C6000 tools (if needed)
2. Part A – Import existing application
3. Part B – Inspection (key parts)
4. Part C – Get Audio Running!

Table of Contents

Audio Application Introduction

Reviewing TI's Original McASP Audio Example

If you would like to see/use/view the ORIGINAL TI source code for their McASP Audio project, you can find it here (assuming you installed the PDK in the same location we did):

```
C:\ti\pdk_omapl138_1_0_11\packages\MyExampleProjects\MCASP_Audio_lcdkOMAPL138_c674xExampleProject
```

Note: Please note that while this project should cleanly build and run, it is very inefficient. We suggest you only peruse it as an educational exercise. If you plan to implement an audio application using the C6748/OMAP-L138, please use the McASP audio application we designed for this workshop – in fact, use the final solution for the Cache lab (chapter 4) assuming you want your audio buffers in external DRAM with cache turned on. Walking through ALL the videos and labs in this workshop will provide you the context you need to easily adapt our final solution to your own application needs.

Lab Introduction/Overview

Note: If you are new to Code Composer Studio, TI embedded processing, C6000 peripherals and the PDK, TI SYS/BIOS or TI-RTOS, etc….we STRONGLY recommend taking the "Getting Started With TI-RTOS Workshop" for C6000 before attempting to start this C6000 architecture workshop. We don't have any time here to walk through ALL of the basics in terms of the IDE and how the RTOS works. We will just assume you know all the content from that workshop. If you find yourself getting confused about simple topics like build/load/run, Tasks, BIOS_start(), semaphores, Task priorities, BIOS timers and clock functions, using the RTOS Analyzer and ROV tools, target configuration files, etc, you REALLY need to take this "getting started" workshop first to provide the context you need to be successful in THIS workshop.

We have THREE basic labs and MANY EDMA3 examples in this two-day workshop. Walking through ALL of the chapter videos PRIOR to doing the lab is really important to understand WHY we are doing specific actions, setting specific compiler switches and turning on specific cache mechanisms. Without the context of the videos…well, you will just be doing what we call "monkey see, monkey do" with no understanding of WHY. Without the WHY, it will be nearly impossible to adapt this learning to your own application.

The three basic labs (plus EDMA3 examples) are:

- Lab 01 – THIS lab where you are introduced to the contents and basic operation of our revised McASP Audio Application
- Lab 03 – In this lab, you will add/use most of the C Compiler optimizations we discuss in the Chapter 3 video. You will see a 24X improvement in performance. WOW!

- Lab 04 – In this lab, you will take the optimized audio application from the end of lab 03 and then put the buffers in external DRAM and turn on the cache and compare this performance to running the buffers in internal IRAM alone. You will be pleasantly surprised at the performance levels you can achieve.
- Chapter 05 – after introducing the basics of the EDMA3 peripheral, we provide you 12 examples that show off different capabilities of EDMA3. No where else on the PLANET will you find these simple and easy to use examples to learn how the EDMA3 works. Once again, these examples alone are worth the price of admission.

AN IMPORTANT NOTE ABOUT INSTALLING THE TOOLS AND SOFTWARE…

Note: Before starting step 1 below, please make sure you have followed ALL installation instructions mentioned in Chapter 0. These are the same installation instructions for the "C6000 Getting Started with TI-RTOS" workshop. If you already took that workshop, there is no need to install anything else – you are ready to go. If you have NOT taken that workshop, we copied the installation instructions into chapter 0 of this course for your convenience. If the tools and software aren't installed properly…you will just run into a bunch of problems right at the beginning of this lab. So please double-check your installation of all C6000 tools, SDK, PDK and CCS are all done correctly before moving on.

The reason we make a big deal about the proper installation of the tools/software is that many students say "I already did that" and then move on and run into troubles because they didn't perform ONE step we mentioned which is the source of all the build problems. So this is why we warn and moan and make this quite dramatic…so that students LISTEN and are aware of the importance of getting everything in the right place. Again, this is the main reason we suggest you take the "Getting Started with TI-RTOS Workshop for C6000" FIRST.

Now, finally, it is time to EXPLORE the audio application…

1. **Please double-check you have all of the C6000 tools, software and CCS installed properly (hah, we have now mentioned this three times)**

2. **Open CCS and connect your hardware target board to your computer via a JTAG emulator.**

   You can use whatever emulator you choose…xds100v2, xds200, xds560v2, etc. However…

Hint: The project you are about to import DEFAULTS to using the XDS200 emulator connection in the target configuration file. In a step later in this lab, we will ask you to open up the target config file and make sure your emulator is specified and connected properly! (or else, NOTHING will work)

Prepare LCDK Hardware & Audio Cables

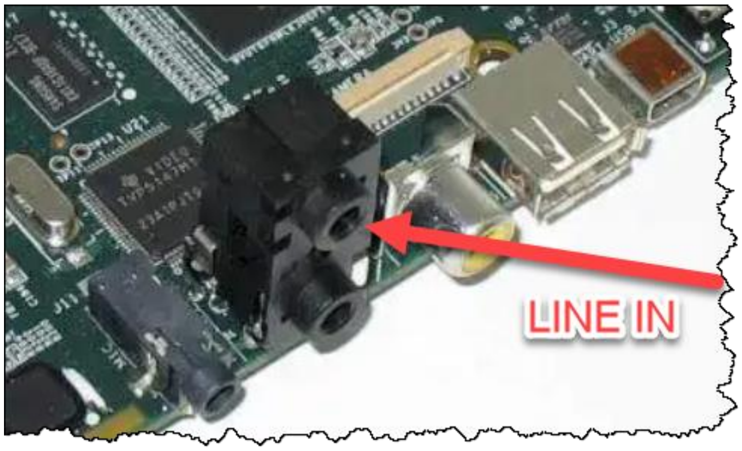
3. **Prepare hardware, audio patch cable/headphones and music.**

   In order for us to process and HEAR the audio, you will need the following:

   - Source of music (MP3, streamed content, etc)
   - 1/8" stereo patch cable
   - 1/8" stereo connection to speakers or headphones

   ▶ Set up your source of music ready to play on your PC such that it is streaming out of your PC's 1/8" stereo headphone/speaker OUT connection.

   ▶ Install a 1/8" stereo patch cable from your PC (speaker/headphone OUT) to the TOP side (Audio LINE IN) of the stacked audio connector:

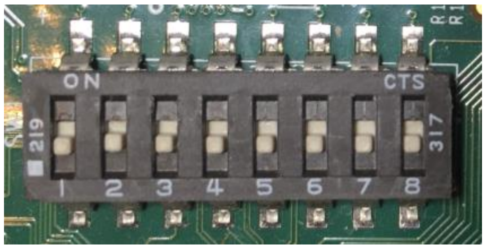


   ▶ Connect your headphones or speakers to the BOTTOM side (Audio LINE OUT) of the audio connector.

4. **Ensure boot switches are set for EMULATION/DEBUG mode.**

   In the installation instructions, we covered this topic, but again, this is one of those go/no-go items.

   ▶ Check SW1 (Switch 1) and make sure ALL switches are OFF (down, closest to the numbers):



   The first four swtiches (1-4) affect the boot modes. You want "no boot", which means all OFF.

Import Audio Project and View Key Contents

5. **Ensure lab solution files are unzipped to the proper folder.**

   The installation instructions (in Chapter 0) asked you to create the following folder and unzip the lab solution files into it. We assume you have already done this. If not, please create the following folder and unzip the lab files into it:

   ```
   c:\ea\c6000_arch
   ```

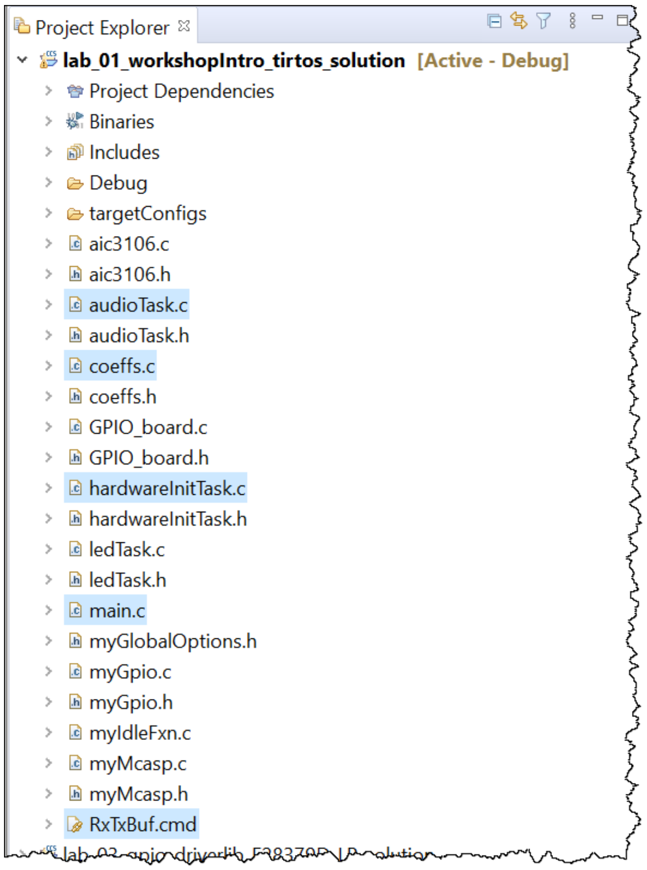
6. **Import the workshop's McASP Audio Application.**

   In CCS, from the C6000 workshop folder (shown just above)…

   ▶ Import: `lab_01_workshopIntro_tirtos_solution`

Note: You just imported TWO projects actually – the application and the dependent TI-RTOS (SYSBIOS) configuration project. We will look at the application FIRST and then the BIOS configuration project.

▶ Expand the project…it should look something like this:



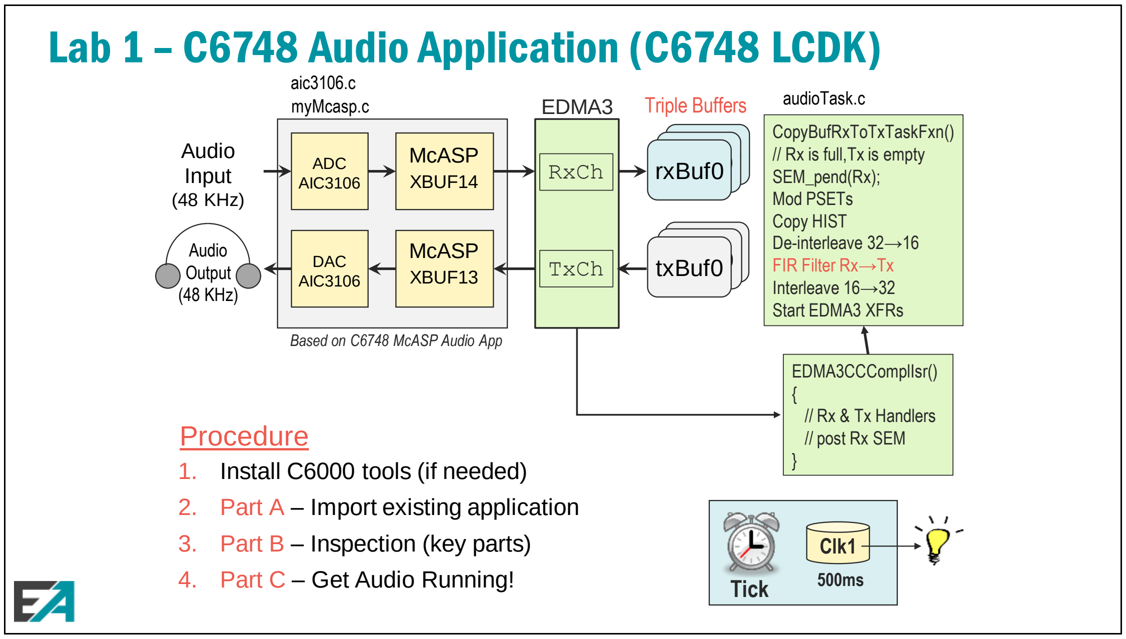
```
lab_01_workshopIntro_tirtos_solution  [Active - Debug]
  Project Dependencies
  Binaries
  Includes
  Debug
  targetConfigs
  aic3106.c
  aic3106.h
  audioTask.c
  audioTask.h
  coeffs.c
  coeffs.h
  GPIO_board.c
  GPIO_board.h
  hardwareInitTask.c
  hardwareInitTask.h
  ledTask.c
  ledTask.h
  main.c
  myGlobalOptions.h
  myGpio.c
  myGpio.h
  myIdleFxn.c
  myMcasp.c
  myMcasp.h
  RxTxBuf.cmd
```

If your project looks *wildly* different, a mistake was made.

Highlighted in BLUE are the key files you will open and inspect in the following steps.

7. **Audio Application Overview**

So, let’s start by looking at the lab slide again and then following the audio samples from INPUT, to PROCESSING and then to OUTPUT…



Lab 1 – C6748 Audio Application (C6748 LCDK)

Procedure
1. Install C6000 tools (if needed)
2. Part A – Import existing application
3. Part B – Inspection (key parts)
4. Part C – Get Audio Running!

- *Input (RCV)* – each analog audio sample from the audio INPUT port of the LCDK (top stereo audio jack) is converted by the A/D and sent to the McASP port on the C6748. For each sample, the McASP generates an EDMA3 event which fills up the rxBuf[0-2] buffer. When the rxBuf is full, the EDMA generates an interrupt to the CPU. In the ISR, Rx and Tx are handled separately. Assuming the Rx and Tx are running at the same frequency, only Rx unblocks the processing Task to filter the audio data…

- *Process* – Assuming at this point that the next Rx buffer is full and the next Tx buffer is empty, it is time to process the data. A simple pass-thru would just copy Rx to Tx which is what the original PDK application did. However, we modified that code to perform an FIR filter. The original code performed 32-bit reads and writes with the upper 16 bits of each “sample” padded with zeroes. An off the shelf FIR filter can’t process this type of data. It needs to be channelized (L and R), converted to 16 bits and remove the zeroes. So, a de-interleave routine is done first to create a new local Rx buffer. That buffer is then filtered to create a new local Tx buffer. When that process is complete, an interleave routine is done to put the data back into a zero-padded 32-bit format which the EDMA copies to the McASP (Tx) serializer…

- *Output (XMT)* – When the EDMA Tx transfer starts, it copies one sample at a time to the McASP Tx serializer. When a buffer full of samples have been transferred, the Tx EDMA channel interrupts the CPU to say “done”. Again, assuming that the Rx and Tx transfers are synced, the Rx interrupt would then unblock the processing Task above.

Several source files are needed to create this application. Let’s explore those briefly…

8. Explore the Audio Application Project Files

Feel free to open ANY file as we walk through all of the files and what they are for. Just don't make any modifications yet to the code unless otherwise asked to do so. We don't want to introduce more errors into the code than will happen naturally.

For each file listed below, open and briefly inspect its contents:

audioTask.c – this is the MAIN task code that we will be modifying in each lab, so this one is really crucial. Note the following about this file (from top to bottom):

- All #include files specified and what each one is for
- Note the "`myGlobalOptions.h`" include…open that to find common `#defines` for the entire project
- Next up is the audio data buffer structure. Each are aligned on 128-byte boundaries…which matches a cache LINE and the size of each buffer is a multiple of a cache-line size (256). This is explained more in the cache chapter. Also notice the SECTION name that these buffers are placed in…"`.far:txBufs`" and "`.far:rxBufs`". This becomes significant when we want to place these buffers EXTERNAL in DDR2 memory.
- Then move down to the function `CopyBufRxToTxTaskFxn()`. Near line 383, notice the call to `Cache_inv()` that is commented out. We will uncomment this in the cache lab. Below that, near line 398-399, you can see the two calls to `cfir()` to filter the audio buffers. Note just above and below these calls, we are grabbing a "timestamp" so that we can subtract them and benchmark the time it takes to filter the L (left) and R (right) audio samples. You can also see the code that de-interleaves the data just before the calls to `cfir()` and the code that interleaves the data back for the transmit buffers. Note that this could have also been AUTOMATICALLY done by the EDMA3 receive and transmit channels…an optimization not covered in this workshop. Then, see near line 415 where the `Cache_wb()` command is used to write back the TX buffers – more on this in the cache chapter. And then, near line 431, we "publish" the FIR benchmark using `Log_info()` to the RTOS Analysis tools so we can see the runtime benchmarks. You will actually SEE this later in this lab.

coeffs.c – this is where the FIR coefficients are specified – for low pass, high pass and ALL pass. Leave the coeffs as ALL PASS for now…but you can change them later whenever you want to. Then, just below the different coeff tables, you will see the STAR of the show: `cfir()`. It is a simple nested `for()` loop that processes all 256 samples against 64 coefficients. Note that there are two invocations of `cfir()` – the one currently uncommented that we will use in this lab…and the one above it (near line 67) that we will use in the next optimization lab. Also note we have two pragmas that are commented out for now – those will be used in the optimization lab as well. More on this function later.

hardwareInitTask.c – in this file are all the initialization calls for the hardware peripherals. We could have placed these in main(), but a good practice when using TI-RTOS (SYS/BIOS) is to put them in a Task that is the highest priority thread and NOT use a `while(1)` loop – therefore it runs FIRST and only ONCE. Hey – just like INIT could SHOULD run. The difference is that BIOS is active and running…unlike in main() before BIOS_start().

main.c – this is where all of the SYS/BIOS objects are created – HWIs, Tasks and Semaphores. You can see the `MOD_create()` calls and the error handling that follows. And near the end…you see the START of the RTOS: `BIOS_start()`. This is a call that is never returned from. As we have said before, if you want to FULLY understand TI-RTOS (SYS/BIOS), please go take the "getting started with TI-RTOS" workshop for C6000. All of the labs run on the same hardware you are using.

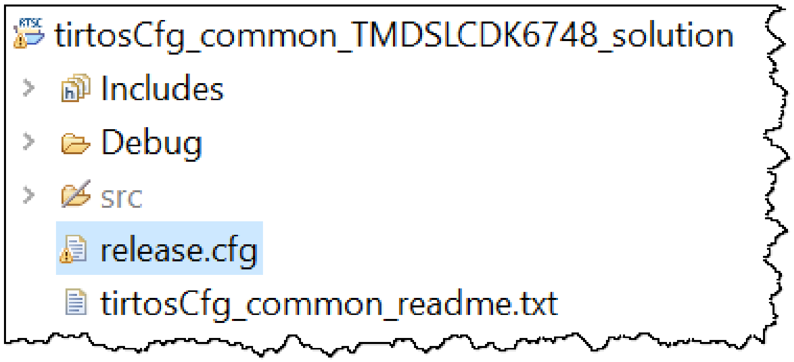
RxTxBuf.cmd – When the compiler builds C code, it creates sections for code and data and initialized values, etc. The `linker.cmd` file specifies WHERE in memory these sections GO (or our bound). In the Debug/ folder, open up the .map file and you can see the RESULTS of the linking process…where all the code and data went in the memory map. Right now, looking at RxTxBuf.cmd, our Rx and Tx buffers are being bound into IRAM or internal L2 RAM. This is why this code works ok…everything is running out of internal RAM. In the cache lab, later, we will modify this command file to bind the Rx and Tx buffers into DDR2 memory area. Question…so where are these memory areas defined? Hah! In the platform file. But where is that? We will show you soon in this lab.

View BIOS Configuration & Build Properties

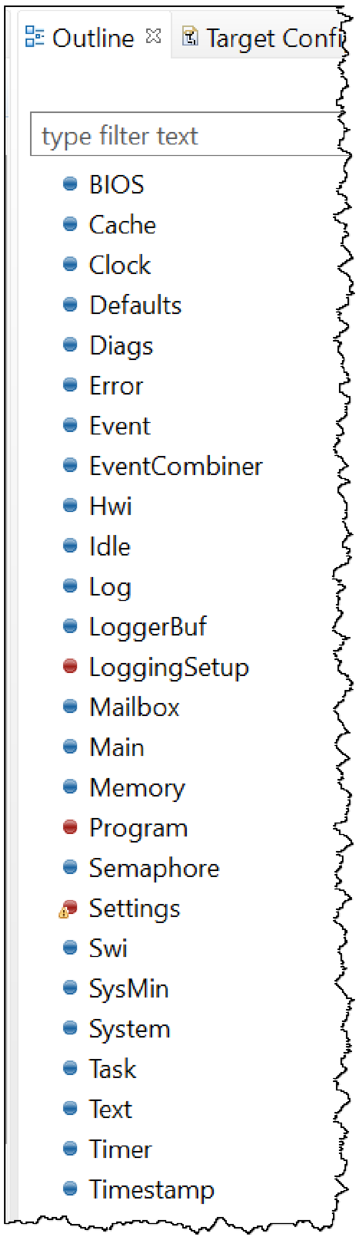
9. **Inspect the BIOS Configuration File**

In all of our TI-RTOS/SYSBIOS labs, we create a DEPENDENT (separate) BIOS configuration PROJECT which contains all of the settings/configuration for the RTOS.

▶ Expand the BIOS configuration project:


tirtosCfg_common_TMDSLCDK6748_solution
Includes
Debug
src
release.cfg
tirtosCfg_common_readme.txt

▶ Right-click on the `release.cfg` file and choose "Open with XGCONF" to see all of the services that are included in this configuration file. You will see a tab listing all of the "Available Products" for the RTOS as well as an outline view of all of the services:



- BIOS
- Cache
- Clock
- Defaults
- Diags
- Error
- Event
- EventCombiner
- Hwi
- Idle
- Log
- LoggerBuf
- LoggingSetup
- Mailbox
- Main
- Memory
- Program
- Semaphore
- Settings
- Swi
- SysMin
- System
- Task
- Text
- Timer
- Timestamp

If you want to know more about these services, what they mean, how to configure and use them effectively in your application…uh…what am I going to say? Take the "Getting Started…" workshop. Ok, enough upselling…eh? Well, in just two days you can know EVERYTHING you need to get TI-RTOS up and running WELL in any system…so….STOP! Ok. 😜

How are these projects "bound" together? Let's go check out the properties of our main application project…

10. Inspect Audio Application Build Properties

Each project has a set of properties that change the behavior of the build process – compiling and linking your code. Let's go look at just a few…

- ▶ Right-click on the audio application project and select "Properties".
- ▶ Click on General and view the Project tab – this is where the CPU, connection (XDS emulator), compiler version, etc are all specified. Under the Products tab…you will see SYS/BIOS, System Analyzer and the PDK.
- ▶ Click on Build and then the Dependencies tab. This is where the dependent BIOS configuration project is "linked" to this audio application project.
- ▶ Under the Build menu, click on C6000 Compiler -> Optimization. Note that the optimizer is OFF and the code is NOT optimized for speed. We will be messing with some of these settings in the optimization lab. View the Project tab – this is where the CPU, connection (XDS emulator), compiler version, etc are all specified. Under the Products tab…you will see SYS/BIOS, System Analyzer and the PDK.

Note: These tools listed under the Products tab are INHERITED from a different project…which we will look at shortly

---

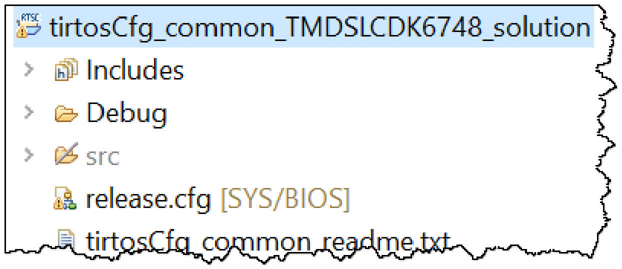
Feel free to inspect any other properties that interest you…just don't change any of them yet…

11. Modify TI-RTOS & PDK Build Properties

When this lab was created, it was at time "x" with a given set of tools for SYS/BIOS, PDK, UIA, XDC, etc. You may be using this lab at time "x+1" and installed LATER tools than what this lab is using. SO it is important to update the tools before building your project the first time.

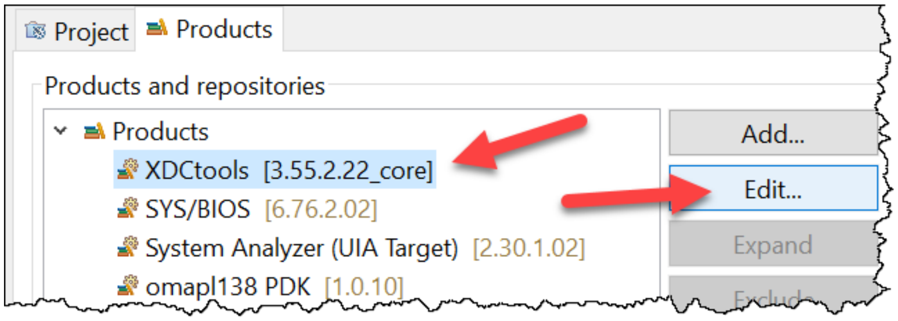
The tools are actually specified in the DEPENDENT RTOS project and then INHERITED by the audio application. So let's go look at the dependent RTOS project…

- ▶ Expand the RTOS configuration project below:



tirtosCfg_common_TMDSLCDK6748_solution
- Includes
- Debug
- src
- release.cfg [SYS/BIOS]
- tirtosCfg_common_readme.txt

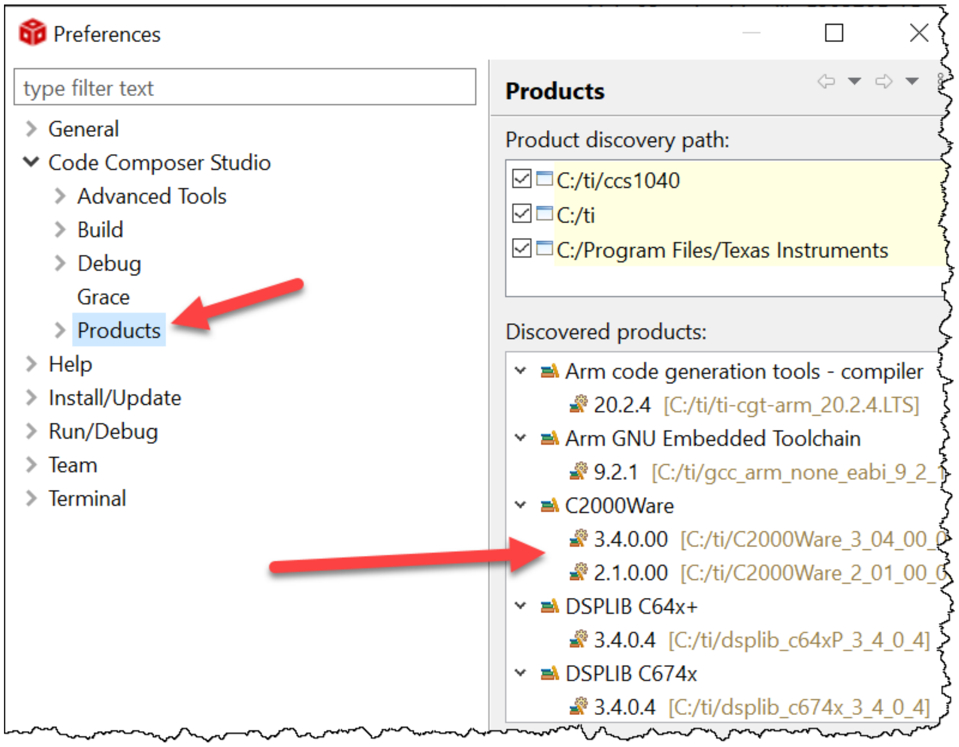
- Right-click on the project name and select Properties. Then go back to “General” and the Products tab:



Products and repositories

- Products
  - XDCtools [3.55.2.22_core]
  - SYS/BIOS [6.76.2.02]
  - System Analyzer (UIA Target) [2.30.1.02]
  - omapl138 PDK [1.0.10]

Your tools may look a little different…and that’s ok. The idea here is that we can edit them and update them to the latest tools. First, you need to have INSTALLED the later tools AND CCS has to recognize them. If you are curious about which tools CCS has recognized…select “Window -> Preferences” and click on the Products tab:



Products

Product discovery path:

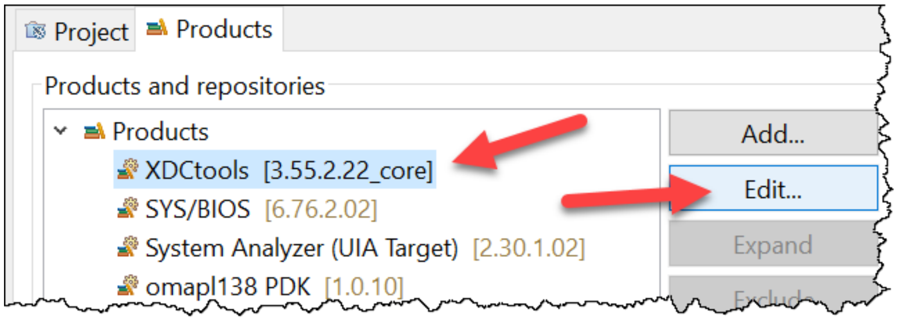
- ☑ C:/ti/ccs1040
- ☑ C:/ti
- ☑ C:/Program Files/Texas Instruments

Discovered products:

- Arm code generation tools - compiler
  - 20.2.4 [C:/ti/ti-cgt-arm_20.2.4.LTS]
- Arm GNU Embedded Toolchain
  - 9.2.1 [C:/ti/gcc_arm_none_eabi_9_2_
- C2000Ware
  - 3.4.0.00 [C:/ti/C2000Ware_3_04_00_
  - 2.1.0.00 [C:/ti/C2000Ware_2_01_00_
- DSPLIB C64x+
  - 3.4.0.4 [C:/ti/dsplib_c64xP_3_4_0_4]
- DSPLIB C674x
  - 3.4.0.4 [C:/ti/dsplib_c674x_3_4_0_4]

If the latest tools are listed in “Discovered products”, then they will populate when you edit them in the “Products” tab in Build Properties. If they are NOT listed under discovered products, you must have placed them NOT in the c:/ti folder (which is a no-no) and you will need to add the proper path to “product discovery path” and force CCS to re-discover them. THEN they will populate properly.

Now that we have THAT discussed and correctly done…let's go back to updating our tools in the BIOS Configuration dependent project:

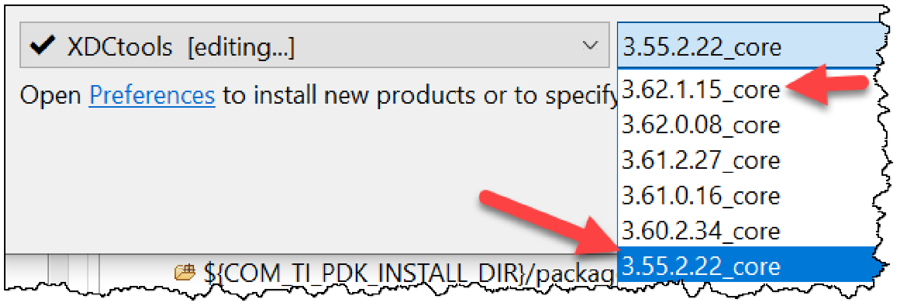


Products and repositories

- Products
  - XDCtools [3.55.2.22_core]
  - SYS/BIOS [6.76.2.02]
  - System Analyzer (UIA Target) [2.30.1.02]
  - omapl138 PDK [1.0.10]

When you CLICK on any of these tools and they highlight in BLUE, the Edit button will un-grey and light up in blue also.

So pick a tool, click on it and then click the Edit button.



Open Preferences to install new products or to specify

${COM_TI_PDK_INSTALL_DIR}/packag

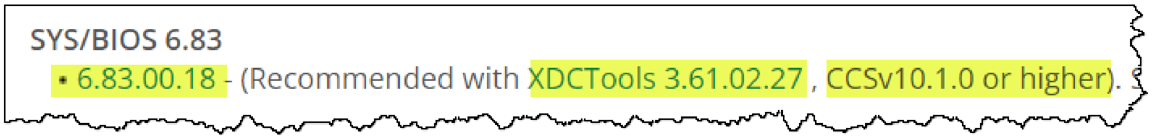
You may have WAY fewer XDC tools versions installed vs us who develop/teach these workshops…but the choices are similar. So originally, this project was set up using XDC version 3.55.2.22 and I would like to update it to the latest version. I simply click on the newest version and then click OK.

BUT, which one should you choose? Is it always the latest? We have found that these products are BOUND together and recommended with specific releases of CCS. SO, it is REALLY important that YOU TOO find a SET of these tools that work together. But how do you do that?

- Click on the link below to go to the Embedded Software site at TI:

  http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/index.html

- If you click on the TI-RTOS Kernel (SYS/BIOS) link, you will see something like this:



SYS/BIOS 6.83

- 6.83.00.18 - (Recommended with XDCTools 3.61.02.27 , CCSv10.1.0 or higher).

Other combos are listed above and below this. So, it is REALLY important, when you pick ONE of these (like CCSv10.1.0 or higher), that you MATCH that choice with the matched BIOS/XDC versions.

Note: FYI – the ORIGINAL tools SET that was used to create the audio application used the following versions of tools:

- CCS 9.3.0
- XDCTools 3.55.2.22_core
- SYS/BIOS 6.76.2.02
- System Analyzer (UIA Target) 2.30.1.02
- Omapl138 PDK 1.0.10
- Platform: ti.platforms.evmOMAPL138

Note: So, if something continues to be REALLY wrong with your build/load/run sequences and you are getting REALLY strange errors, one way to debug the problem is switching back to THIS specific set of tools and seeing if your problems go away. TI normally does a GREAT job of creating upward compatibility…and they test the "matched sets" often to guarantee safe operation…but if you have a mixed bag of XDC and BIOS and UIA…well, maybe that is what is causing your problems. We can't guarantee anything – but this may help your debug process.

It is usually VERY safe to update the PDK and UIA to the latest. However, BIOS/XDC are matched. But if you want all of the real dependencies, always check the release notes for each tool.

▶ Update your tools (installed) to the latest, matched sets and then choose these in the Properties->General->Products tab in the BIOS Configuration dependent project.

FYI, the updated set of tools used to TEST and VERIFY all the labs in this workshop (as of Nov 2021) were the following:

- CCS 10.4
- XDCTools 3.61.2.27_core
- SYS/BIOS 6.83.0.18
- System Analyzer (UIA Target) 2.30.1.02
- Omapl138 PDK 1.0.11
- Platform: ti.platforms.evmOMAPL138

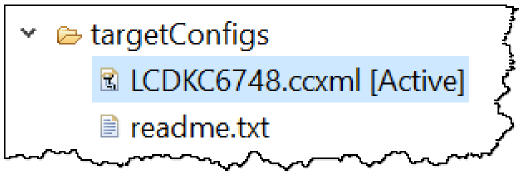
Build, Load and Run!!

Verify Target Config File is Correct

12. Check to make sure your target config file is correct for your emulator.

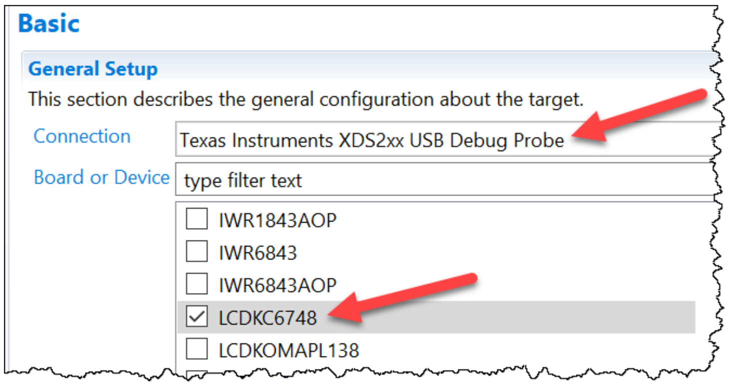
Remember the target configuration files? They specify your target (LCDK6748), connection (XDS200, XDS100v2, etc) and GEL file (which is run to set up DDR2, PLLs, etc. when you “connect to target”). Let’s go explore each one of these…

▶ Expand the targetConfigs folder:



▶ Double-click on the .ccxml file to open it.

The XDS200 USB Debug Probe and LCDK6748 targets are selected:


Basic

General Setup

This section describes the general configuration about the target.

Connection: Texas Instruments XDS2xx USB Debug Probe

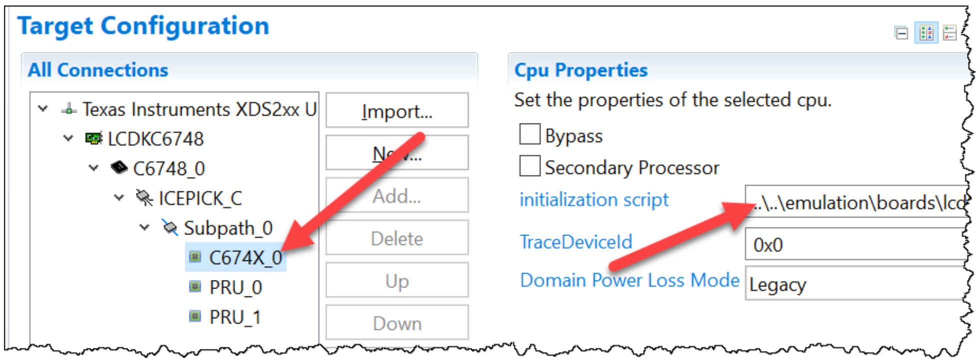
Board or Device: type filter text

- [ ] IWR1843AOP
- [ ] IWR6843
- [ ] IWR6843AOP
- [x] LCDKC6748
- [ ] LCDKOMAPL138

Make sure these both match your setup (OMAP-L138 users should change the target to LCDKOMAPL138).

▶ If you are using a different emulator, please change it here and save the target config file.

Click on the Advanced tab…and click on the C674X_0 CPU…



Cpu Properties

Set the properties of the selected cpu.

- [ ] Bypass
- [ ] Secondary Processor

initialization script	..\..\emulation\boards\lcd
TraceDeviceId	0x0
Domain Power Loss Mode	Legacy

When you click on the CPU, the CPU Properties dialogue opens on the right. This is path that is used for the GEL file that runs when you connect to the target. This GEL (General Extension Language – which means nothing) is a script that sets up PLL and DDR2 registers. These types of actions will eventually need to be performed by YOU (the user) in main() as part of your standard init code. For now, we are using the convenience of the GEL file to bypass having to write that code before we can start playing around with our audio (or any application for that matter).

Bottom line – just make sure these three items – target, connection, GEL file – match your specific hardware and environment.

Build and Run the Audio Application

13. Build the audio application.

▶ Build the application and fix any errors that occur.

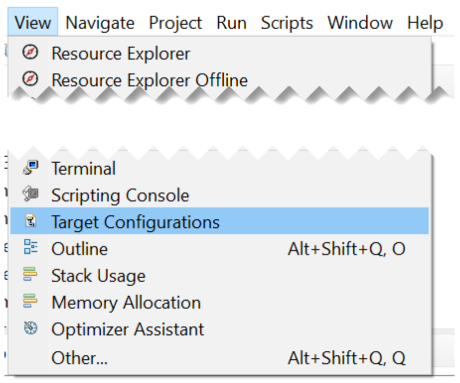
According to TI, C6748 full support requires CCS 9.3.0. However, the authors are using CCS10.4.1 and above to test the code and everything seems to work fine.

You should have ZERO build errors and one small warning (that you can ignore). If so…move onward. If not, try to figure out what is missing that gave you the build errors and fix them. They are probably introduced by something missing in your environment. You may want to check the installation procedure to make sure you didn't skip a step.

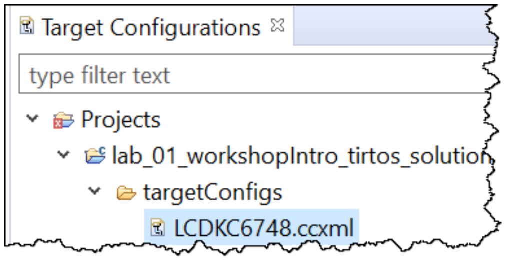
14. Launch the target configuration file and connect to target.

Normally, users will just hit the "bug" button to build, launch, connect, load the .out file. The first time I work with a new board/emulator, I always use the 4-step approach carefully so I can see WHICH step an error occurs if one does occur. So that's what we will try here…

▶ Select View -> Target Configurations



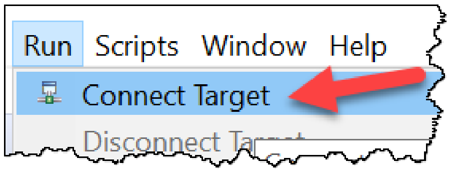
▶ Expand Projects, the audio project and the targetConfigs folder:



▶ Right-click on the .ccxml file and select "Launch Selected Configuration".

This will change the perspective to Debug and launch the target configuration file.

▶ Next, click on the "Connect" button or choose Run -> Connect Target:



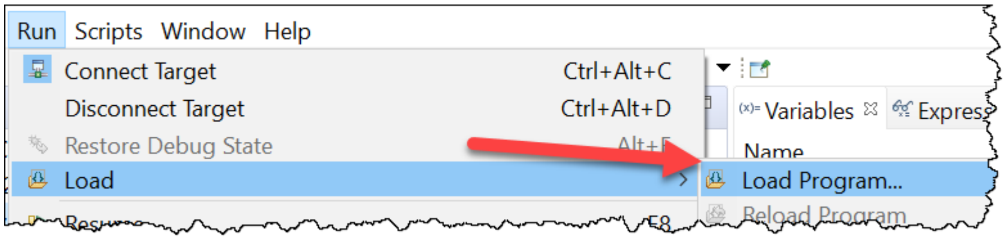
WATCH THE GEL FILE RUNNING! (and the comments it spits out to the console window). You will notice that it is setting up PLL, DDR, PSC, memory map, etc.

If you have gotten to this point with no errors, you are home free…your emulator is working properly. If not, try to debug the problem (most likely a wrong emulator choice in the target configuration file…or no power to your board…or bad usb cable).

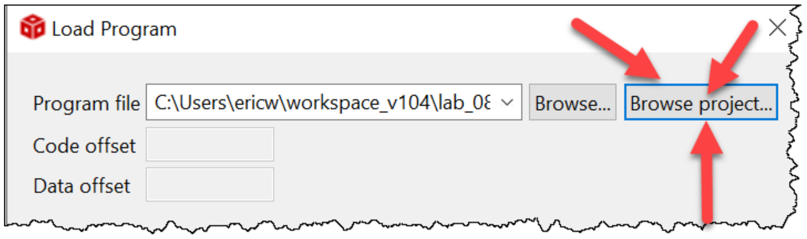
15. Load your .OUT file into the C6748 device.

The last of the 4 steps (build, launch, connect, load) is LOAD.

▶ Click the Load Program button or choose Run -> Load Program:


Run
Scripts
Window
Help
Connect Target Ctrl+Alt+C
Disconnect Target Ctrl+Alt+D
Restore Debug State Alt+F
Load
Load Program...
Reload Program
Variables
Express
Name

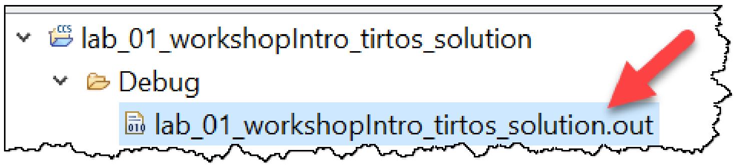
Next, a dialogue box appears:



Program file: C:\Users\ericw\workspace_v104\lab_0&
Code offset
Data offset

- Always click on "Browse Project" because it will bring up a list of projects in your Project Explorer and allow you to navigate to the .out file associated with the project you just built.

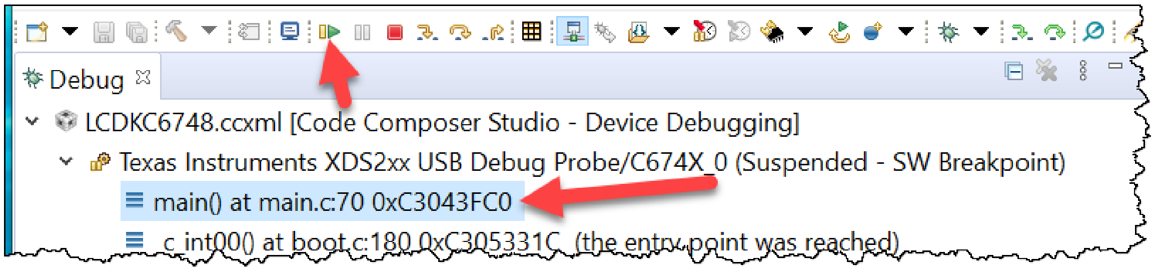
- Expand your project and Debug folder to find the .out file that was just built:



lab_01_workshopIntro_tirtos_solution
Debug
lab_01_workshopIntro_tirtos_solution.out

- Click OKs to load the program…

The call stack should automatically STOP at main() and the green RESUME (play) button is green…ready to play/run/resume your code.



Debug
LCDKC6748.ccxml [Code Composer Studio - Device Debugging]
Texas Instruments XDS2xx USB Debug Probe/C674X_0 (Suspended - SW Breakpoint)
main() at main.c:70 0xC3043FC0
c_int00() at boot.c:180 0xC305331C (the entry point was reached)

If your code is already running…or the call stack looks different, try reloading your program again. If that doesn't work, power cycle your board and RE launch, connect, load again.

Your code is now READY TO RUN! Are you read to hear music?

16. Get Audio Running

Assuming you followed the instructions regarding the audio source, patch cable and setting up your headphones/speakers, you are ready to see if the audio code works for you.

- Play your music on the PC (and make sure it is set to play forever…some students don't realize the music has STOPPED and they think their code isn't working. Don't do that.) Make sure it is playing from your speaker OUT of the PC.

- Put on your headphones (or turn on your speakers)

- HIT PLAY! (Resume…green button)

You should hear music playing through your speaker/headphones and see one LED blinking every second. Yes??? Success!!

- HIT PAUSE button to pause your code…

Now that the code is stopped…let's go explore a few cool items in ROV and RTOS Analyzer…

View Application Results in ROV and RTOS Analyzer

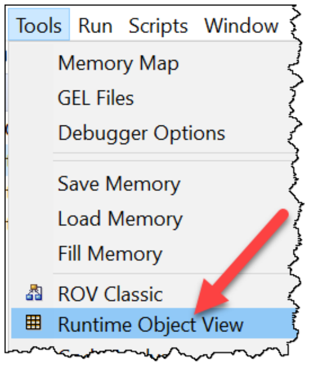
All of the details of how to set up and use ROV and RTOS Analyzer are covered in the 2-day "Getting Started with TI-RTOS" workshop. We assume you have this knowledge already…and won't repeat it here…we will just look at the RESULTS that these tools provide but not HOW it gets set up in the tools (which is non trivial).

Explore ROV

17. Explore ROV

BIOS supports a tool called "Runtime Object View"…and we like to call it "RTOS Object Viewer"…

▶ Select Tools -> Runtime Object View:



▶ When the dialogue box appears, select "CONNECT".

Note: If ROV starts to act odd…just reload your program, run it and try again.

ROV is simply a list of ALL of the BIOS objects and their current state when you hit PAUSE. So you can see the Hwi's, Semaphores and Tasks and their states when you halt the processor. This is an incredible debug tool. So let's go look at just a few items…

▶ Click on the following modules and explore the contents of each:

- Hwi
- Semaphore
- Task

Before we go look at the benchmark for the cfir() (left and write) calls, write down your ESTIMATE of how long (# CPU cycles) you think it takes a processor that can do 8 16x16 MACs per CYCLE to filter 256 audio samples with 64 coeffs. The math dictates that 64x256 = 16,384 multiply and accumulates to solve the problem. So you think it is more or less than 16K CPU cycles?

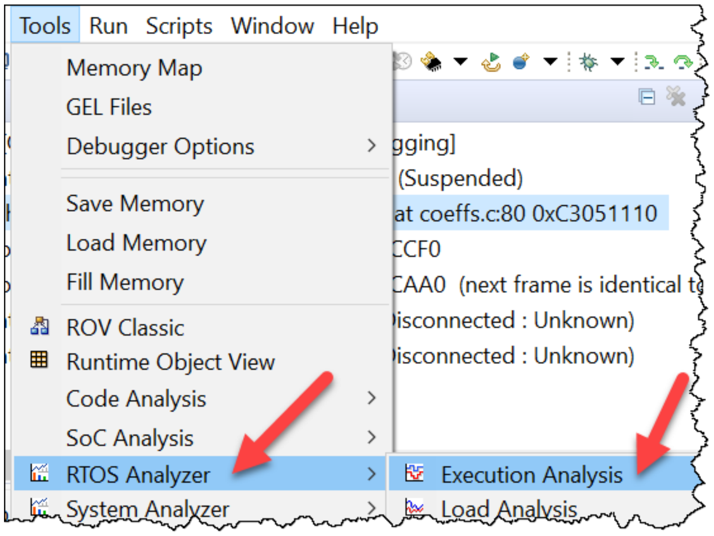
Your estimate _______________ CPU cycles

Explore RTOS Analyzer (CFIR benchmark results)

18. Explore RTOS Analyzer

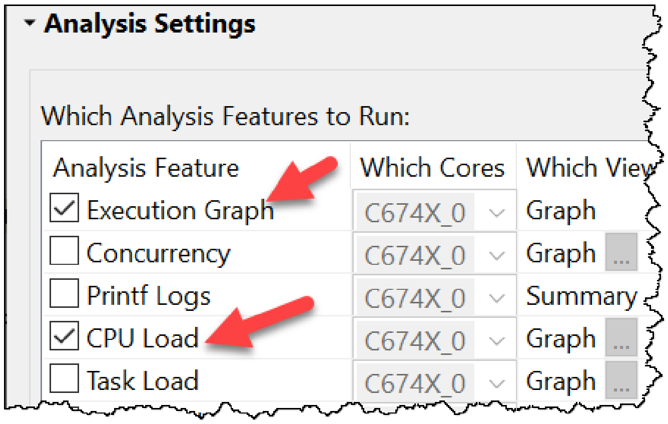
The RTOS Analyzer is a fantastic debug tool when using TI-RTOS/SYSBIOS in your application. Let's go turn it on and view some of the results we can see. We will hang out in this tool in EVERY lab to check the benchmarks of our code over and over again…so you will get quite skilled and bringing up the Analyzer and filtering the system log to see your results quickly.

- Once your code is halted, select Tools -> RTOS Analyzer -> Execution Analysis:



gging]
(Suspended)
at coeffs.c:80 0xC3051110
CCF0
CAA0 (next frame is identical t
isconnected : Unknown)
isconnected : Unknown)

- When the dialogue appears, select CPU Load and Execution Graph and click Start:



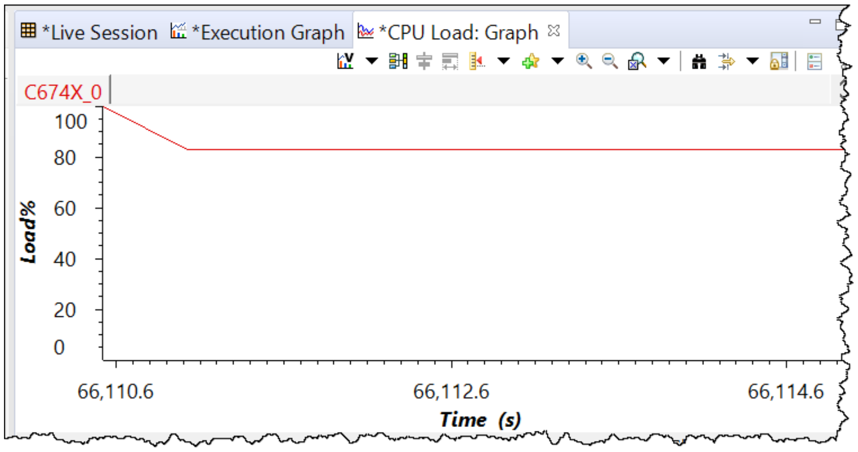
Analysis Settings

Which Analysis Features to Run:

Analysis Feature	Which Cores	Which View
✓ Execution Graph	C674X_0	Graph
☐ Concurrency	C674X_0	Graph
☐ Printf Logs	C674X_0	Summary
✓ CPU Load	C674X_0	Graph
☐ Task Load	C674X_0	Graph

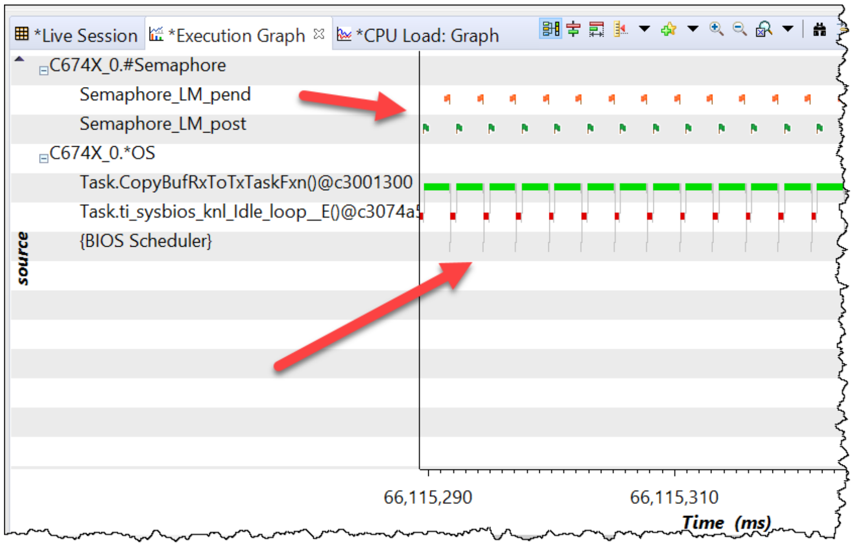
View each window one at a time…

▶ Select the CPU Load Graph tab and check out the CPU load…



Looks like about 83% CPU load which means the processor is DOING SOMETHING (not in BIOS Idle loop) 83% of the time. This is a pretty heavy CPU load…when all the CPU is doing is filtering audio. DSPs are really fast at math…so this is an indication that something is going REALLY slow. Of course we will investigate this.

▶ Select the Execution Graph and expand the Semaphore and OS items…and zoom in a little…

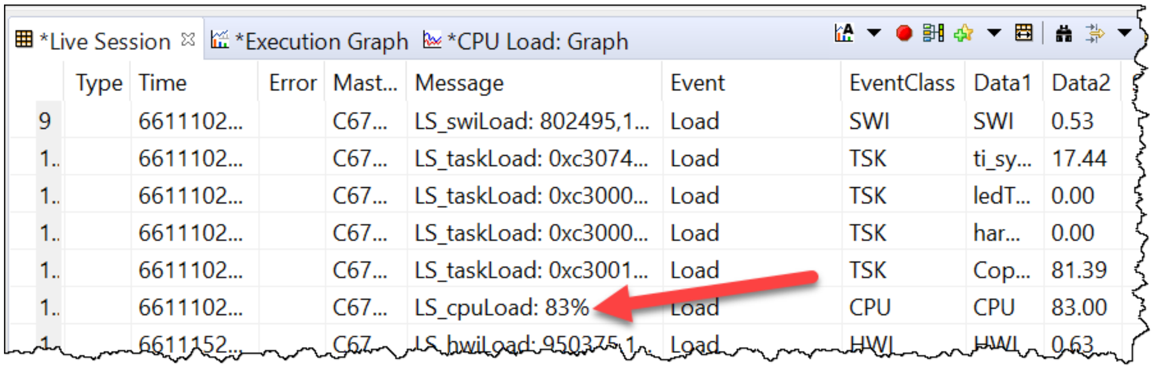


This helps you see when semaphores are posted/pending, when the CopyBufRxtToTXTaskFxn runs, when the processor is in the Idle thread and when the BIOS scheduler runs…all timestamped. Wow – tons of information here to look at an explore. You can even do benchmarks RIGHT HERE on this diagram.

Lastly, let's look at the SYSTEM LOG….

▶ Click on the tab that says “Live Session”.

This is the entire system log…i.e. EVERY event that is happening in the system that BIOS is aware of. So you will see all kinds of activities listed and the timestamp that occurred at:



	Type	Time	Error	Mast…	Message	Event	EventClass	Data1	Data2
9		6611102…		C67…	LS_swiLoad: 802495,1…	Load	SWI	SWI	0.53
1..		6611102…		C67…	LS_taskLoad: 0xc3074…	Load	TSK	ti_sy…	17.44
1..		6611102…		C67…	LS_taskLoad: 0xc3000…	Load	TSK	ledT…	0.00
1..		6611102…		C67…	LS_taskLoad: 0xc3000…	Load	TSK	har…	0.00
1..		6611102…		C67…	LS_taskLoad: 0xc3001…	Load	TSK	Cop…	81.39
1..		6611102…		C67…	LS_cpuLoad: 83%	Load	CPU	CPU	83.00
1..		6611152…		C67…	LS_hwiLoad: 950375,1…	Load	HWI	HWI	0.63

Here we can even see the CPU load reported as 83%…this is the data (and timestamp) the CPU Load graph was actually graphing. This is also the log that is used by the Execution Graph to GRAPH all events that you saw earlier – a software logic analyzer. Very cool.

But what about the benchmark data? We saw in the code earlier that we were using Log_info() to “print” the results of the cfir() algorithm to “something”…well, it appears in THIS system log. But there are too much stuff to wade through to see it.

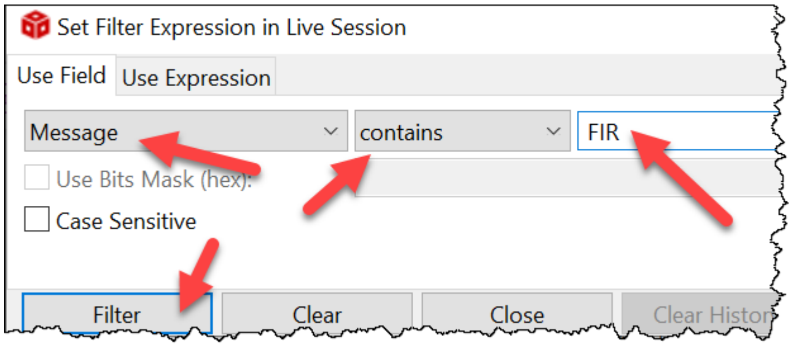
Well, you can either scroll down and look for the words “FIR BENCHMARK” or you can filter them easily.

▶ First, click the Auto Fit Columns button:

This will make all the data easier to read inside the columns (we wish this was a default behavior…but oh well)

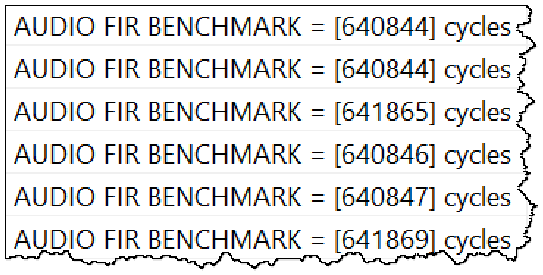
Next, click the Filter button:

A dialogue box appears…and then fill in the items like this and when done, click Filter:



What do you see?

You see timestamps along with ONLY the FIR BENCHMARK results:



```
AUDIO FIR BENCHMARK = [640844] cycles
AUDIO FIR BENCHMARK = [640844] cycles
AUDIO FIR BENCHMARK = [641865] cycles
AUDIO FIR BENCHMARK = [640846] cycles
AUDIO FIR BENCHMARK = [640847] cycles
AUDIO FIR BENCHMARK = [641869] cycles
```

YEEEEOOOWWW – 641K cycles to do the cfir() filtering. Awful. But hey, the music sounds ok…it works. SHIP IT! Heck no…you paid money to be in this workshop to learn WHY this happens and how to fix it. Patience young padawan…we will get there.

SO, that's the end of the lab. We will return to the RTOS Analyzer tool OFTEN in the upcoming labs.

Clean Up

19. That's it, You're Done!

- Save and close all files and close the project
- Close CCS and power cycle your board…

Congratulations – you now have the STARTER code for this workshop running and working just fine…other than the benchmark SUCKS. We will fix that soon.

Congrats, you are done with this lab. You are now ready to proceed to the next C6000/Keystone Architecture & Optimization Chapter….Chapter 2 video – introducing you to the guts of the CPU. Then you will proceed to Chapter 3 and learn how to use the proper C Compiler switches and some other tricks to optimize your application code in Lab 03.

Page left intentionally blank