Azcom Stack - Code Organization and Build Instructions

Customer Guide

The document describes Code organization and build Instructions for Layer-2 and Layer-3 software of Azcom's eNodeB on K2K platform

Document Revision: 1.0

Azcom Technology srl
Headquarter:
Centro Direzionale Milanofiori
Strada 6, Palazzo N/2
20089 Rozzano (MI) – Italy
Tel.: +39 02 82450311

Development Center (Gurugram, India)
Azcom Infosolutions (India) Pvt. Ltd.
3rd Floor, Tower B, UM House
Plot #35P, Sector 44
GURUGRAM-122002 (Haryana) - INDIA.
Tel. +91-124-4937650

Website: www.azcomtech.com
E-mail: info@azcom.in

Acronyms

API	Application Programming Interface
ARM	Advanced RISC Machine
ASN1	Abstract Syntax Notation 1
BSL	Basic Service Layer
DSP	Digital Signal Processor
EDMA	Enhanced DMA (HW Device)
eNB	Enhanced-NodeB
ETB	Embedded Trace Buffer
FAPI	Femto API
FDD	Frequency Division Duplex
GTPU	Gateway Tunneling Protocol - User
HW	Hardware
IP	Internet Protocol
I/F	Interface
MAC	Media Access Control
OAM	Operation, Administration and Maintenance
OSAL	Operating System Application Layer
PDCP	Packet Data Convergence Protocol
PHY	Physical Layer
RAM	Random Access Memory
RLC	Radio Link Control
ROM	Read Only Memory
RRC	Radio Resource Control
RRM	Radio Resource Management
S1AP	S1 Application
S1U	S1 User
SW	Software
TDD	Time Division Duplex
UDP	User Datagram Protocol
UE	User Equipment
X2AP	X2 Application

CONTENTS

LIST OF FIGURES

1 Scope

This document explains the code organization of layer-2 and layer-3 software. It also explains tools installation procedure and steps required to build layer-2 and layer-3 stack.

2 Code Organization

This section explains the organization of AZCOM eNB layer 2 and layer 3 code. *Figure 1* depicts the code orgainzation.

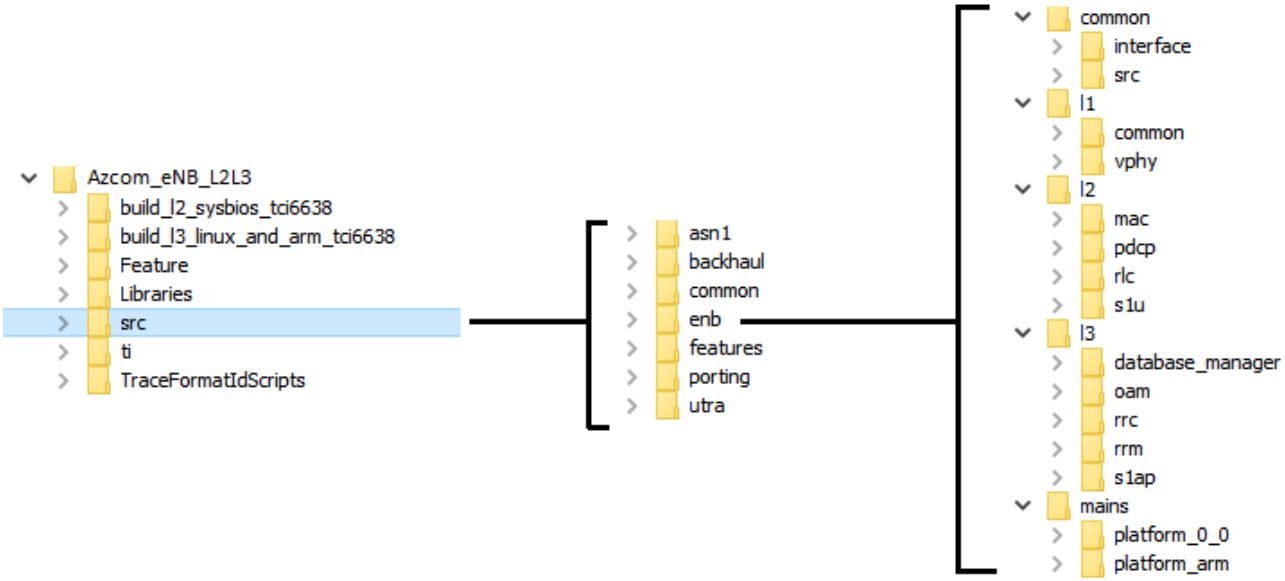

Figure 1: Code organization in separate folders for Layer-2 and Layer-3

- **Build_l2_sysbios_tci6638:** Folder contains the build projects for Layer-2 (tci6638 - Kepler).
- **Build_l3_linux_and_arm_tci6638:** This folder contains the build projects for Layer-3 (tci6638 - Kepler).
- **Feature:** This folder contains the code for features to be enabled. This code needs to be copied to '*features*' folder in '*src*' folder
- **Libraries:** It contains the DSP and ARM libraries providing the necessary APIs for interface modules and other build support.
- **Src:** It contains the complete code for Layer-2 and Layer-3. As shown in Figure 1, *src* folder further contains
  - **asn1:** It contains the code for ASN1 encoder/decoder used at Layer-3.
  - **backhaul:** It contains the code for GTPU, S1AP and X2AP interfaces.
  - **common:** It contains files used by all the modules.
  - **enb:** It contains code for all the protocol layers and management modules of Layer-2 and Layer-3 software. This is further arranged in subfolders and are explained in next sections.
  - **features:** It contains the dummy code for features which is currently compiled along with the source code.
  - **porting:** It contains the code for BSL and OSAL. These layers have APIs to communicate with the processor. The OSAL folder has different code source for DSP and ARM due to different processor types.
  - **utra:** It contains the code files for rrc_utra project of Layer-3.
- **ti:** It contains platform related sample debug projects.

- **TraceFormatIdScripts:** It contains the scripts and .dat files for UDP logging.

2.1 Layer-2 and Layer-3 Projects

This section describes code organization of layer 2 and layer 3.

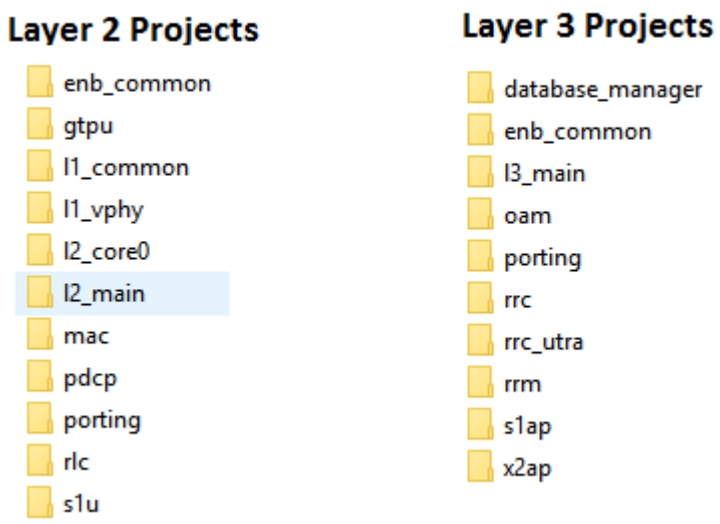

Figure 2: Build Projects for Layer-2 and Layer-3 software

2.1.1 Layer 2

Layer-2 software comprises of protocol layers of PDCP, RLC, MAC, S1_U interface to communicate with gateway and FAPI interface to communicate with Physical layer.

Layer 2 code is organized in following projects:

1. ***enb_common***: It contains commonly used files for all the projects of layer-2 and layer-3. It is built as a library.
2. ***gtpu***: It is the GTP project to communicate with gateway and is built as a library.
3. ***l1_common*** & ***l1_vphy***: They are the projects for FAPI interface and are built as libraries.
4. ***l2_core0***: This project contains DSP Core 0 binary that is built using libraries from enb_common, gtpu, pdcp, porting, rlc and s1u.
5. ***l2_main***: This project contains DSP Core 1 binary that is built using libraries from enb_common, l1_common, l1_vphy, mac and porting projects.
6. ***mac***: It is the project for MAC layer and is built as a library.
7. ***pdcp***: It is the project for PDCP layer and is built as a library.
8. ***porting***: It is the project for BSL and OSAL and is built as a library. It is used for both layer-2 and layer-3 projects.
9. ***rlc***: It is the project for RLC layer and is built as a library.
10. ***s1u***: It is the project for S1U interface and is built as a library.

2.1.2 Layer 3

Layer-3 software comprises of protocol layer RRC, management modules RRM and OAM. It uses X2AP and S1AP interfaces to communicate with neighboring eNodeB and MME respectively.

Layer 3 code is organized in following projects:

1. ***database_manager***: It deals with database for all the values needed on layer-3 and is built as a library.
2. ***enb_common***: It contains commonly used files for all the projects of layer-2 and layer-3. It is built as a library.
3. ***l3_main***: This project contains the Layer-3 ARM binary that is built using libraries from all the other projects.
4. ***oam***: It contains the precompiled library for OAM module.
5. ***porting***: It is the project for BSL and OSAL and is built as a library. It is used for both layer-2 and layer-3 projects.
6. ***rrc***: It is the project for RRC layer and is built as a library.
7. ***rrc_utra***: It is the project for RRC UTRA and is built as a library.
8. ***rrm***: It is the project for RRM module and is built as a library.
9. ***s1ap***: It is the project for S1AP interface and is built as a library.
10. ***x2ap***: It is the project for X2AP interface and is built as a library.

3 Build Environment Prerequisites

Layer-2 runs on two DSP cores of tci6638 and Layer-3 runs on ARM processor. Therfore, separate tool set is required for building binary images for Layer-2 (DSP) and Layer-3 (ARM).

3.1 Tools required for Layer 2

3.1.1 Hardware Requirements

PC with Windows operating system, having RAM > 4GB.

3.1.2 Software Requirements

The following software tools are required for building Layer 2 software

- Code Composer Studio version 6.1.0.00104
- BIOS 6.41.00.26
- MCSDK 3.01.02.05
- Python version 2.7

3.2 Tools required for Layer 3

3.2.1 Hardware Requirements

PC with Ubuntu version 14.04 (32 bit).

3.2.2 Software Requirements

- CodeSourcery ARM tool chain.
- Eclipse IDE: Latest version.

4 Build Tools Installation

This section describes the build tools installation process for layer 2 and layer 3.

4.1 Layer 2

All compilation tools must be installed at the same location. To ensure this, create a folder in **C:** drive and name it "**ti**". Provide the path of this folder during tools installation.

4.1.1 CCS Installation

Texas Instrument's CCS (code composer studio) is used for building layer-2 source. Install following version of CCS on the windows PC.

- **Code Composer Studio 6.1.0.00104**

Following steps must be followed to install CCS.

a) CCS setup is provided in `*ccs_setup_6.1.0.00104.exe*` file. Install the setup by double clicking on the exe file.

b) Accept the license agreement and provide the installation path as C:\ti and click next.
c) Select the options of "Single Core DSPs" & "Multi Core processors" and click next.
d) Click next without selecting any option.
e) Without selecting any option, Click Finish.
f) It may take some time to install. Make sure that the setup runs without errors.

4.1.2 BIOS Installation

BIOS version 6.41.00.26 is used with CCSv6 for building Layer-2 code. Its setup is provided as `*bios_setupwin32_6_41_00_26.exe*` file. Install the given setup by double clicking on the .exe file and make sure that it runs without errors. Provide the path for installation as **C:\ti** (created earlier) and use all the default options.

4.1.3 MCSDK Installation

SC-MCSDK version 3.01.02.05 is provided as `*mcsdk_3_01_02_05_setupwin32.exe*`file. Run the .exe file by double clicking on it and install the setup. Provide the installation path as **C:\ti** and make sure the setup runs without any error. Use all the default options.

4.1.4 Python Installation

Install Python using the "*python-2.7.amd64.msi*" file at the default path.

After installing above mentioned tools verify that the following packages have been installed:

- pdk_keystone2_3_01_02_05
- ndk_2_24_00_11
- edma3_lld_02_11_13_17
- ipc_3_30_01_12
- bios_6_41_00_26
- xdctools_3_30_04_52
- uia_2_00_01_34

These packages will be installed in the installation path provided (**C:\ti**). Note the following points before building any DSP image:

- Copy and replace the file **cslr_spi.h** as provided along with the installers to the location **C:\ti\pdk_keystone2_3_01_02_05\packages\ti\csl**

After installing all the aforementioned software tools, launch CCS and install all the discovered projects.

4.2 Layer 3

4.2.1 Configuring Ubuntu Machine

Following steps should be followed for configuring the Ubuntu machine, ububtu 14.04 is recommended, which would be used for compiling Layer 3.

4.2.1.1 Install Java runtime environment:

Java installation is subject to the version of eclipse. Along with this document, eclipse neon is provided, which require JAVA 8.

To install java8 on ubuntu 14.04 the fillowing steps should be followed:

Copy the tar file: `*jdk-8u201-linux-x64.tar.gz*` having jdk_1.8 package on any path and untar the file using command:

- tar -xvzf jdk-8u201-linux-x64.tar.gz

It will create a folder with name "jdk1.8.0_201" on the current directory. Now set the java installation environment variables:

- sudo update-alternatives --install "/usr/bin/java" "java" "<path_to_ jdk1.8.0_201_folder>/bin/java" 1
- sudo update-alternatives --install "/usr/bin/javac" "javac" <path_to_ jdk1.8.0_201_folder >/bin/javac" 1
- sudo update-alternatives --install "/usr/bin/javaws" "javaws" "<path_to_ jdk1.8.0_201_folder >/bin/javaws" 1

- sudo update-alternatives --set java <path_to_ jdk1.8.0_201_folder>/bin/java
- sudo update-alternatives --set javac <path_to_ jdk1.8.0_201_folder >/bin/javac
- sudo update-alternatives --set javaws <path_to_ jdk1.8.0_201_folder >/bin/javaws

After these commands, verify that java is installed on the system with this command:

- java -version

It should give the following results:

```
islam@AGIT-VM-STACK:~$ java -version
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
islam@AGIT-VM-STACK:~$
```

4.2.1.2 Install CodeSourcery (Linux –ARM Cross Compiler).

It can be installed in two ways

1. Extract from tar file
    a. Copy the tar file any where in the system
    b. Untar using command
        i. tar -xvzf codeSourcery.tar.gz
        ii. It will create a folder "CodeSourcery" in the current directory.
        iii. If the ubuntu is installed in a 64 bit system, gcc multilib package will be needed to use the 32-bit CodeSourcery compiler.
            1. to install gcc multilib, run this command:
                a. sudo apt-get install gcc-multilib

2. Install from BIN file
    a. At the terminal go to the directory that contains the setup of ARM tool chain.
        i. Invoke the ./arm-2010q1-198-arm-none-linux-gnueabi.bin command at command prompt.

After installing the CodeSourcery from any of these two ways, the compiler flags are required to be set, these flags remain set for a particular shell period, if the shell is closed then re export these flags:

- export ARCH=arm
- export CROSS_COMPILE=arm-none-linux-gnueabi-
- export PATH=$PATH: '<path_to_CodeSourcery_folder>/Sourcery_G++_Lite/bin/'

4.2.1.3 Install Eclipse

Eclipse is used to view the layer-3 code in an ordered manner and build the binaries for ARM. Its setup is present as .exe file. Install the setup from tar file provided with this document.

- *'tar -xvzf eclipse.tar.gz'*

Now the eclipse can be run using this command on the directory from where previous command was run:

- eclipse/eclipse &

5 Build Process for Layer 2

The source code contains Layer-2 projects that can be imported into CCS and compiled for tci6638 (Kepler).

5.1 Importing Projects into CCS

Follow the below mentioned steps to import the build projects for Layer-2 in CCS.

- Start CCS. From the Workspace launcher, click Browse and select the path where the workspace is to be created.

Figure 3: Creating a new workspace

- When the CCS window opens, go to the *File* option in top left of the window. Click on the *Import* option.

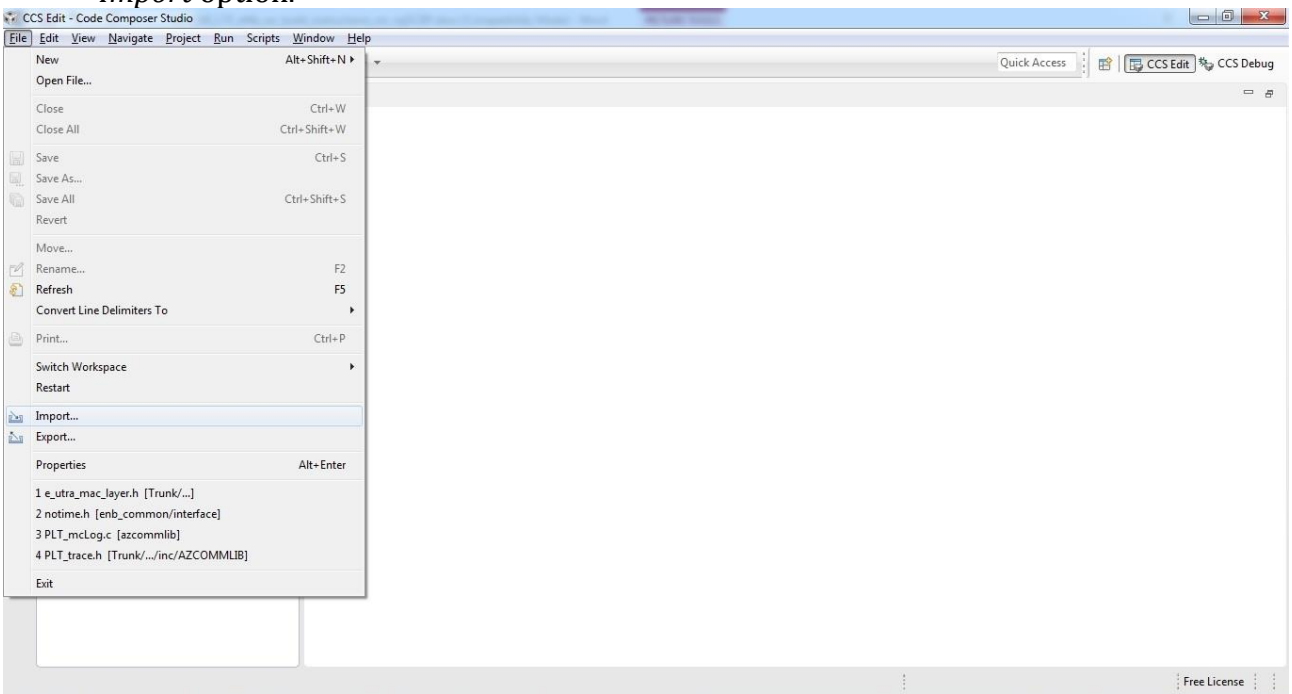

Figure 4: Importing L2 project into Workspace - step1

- Import dialogue box will appear. Then go to the *Code Composer Studio* drop down menu and select *CCS Projects*. Then click *next*.

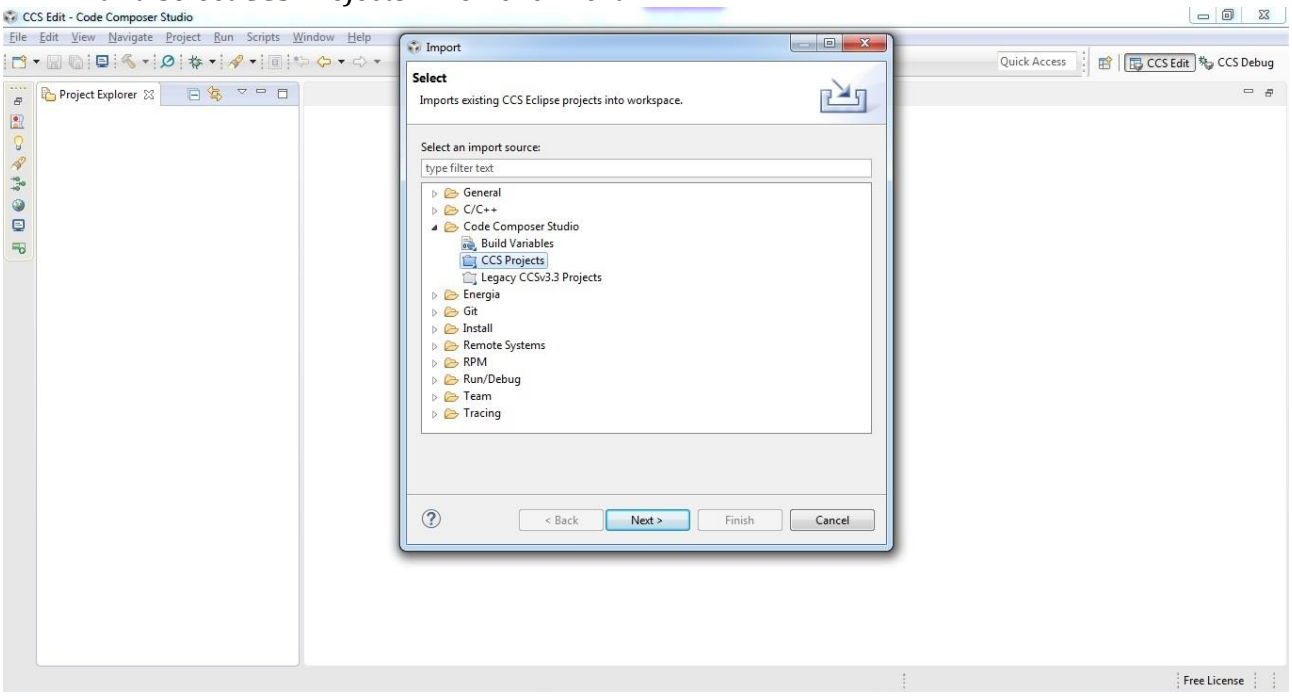

Figure 5: Importing L2 project into Workspace - step2

- An *Import CCS Eclipse Projects* window will appear. Click on *Browse* and select the folder *build_l2_sysbios_tci6638* in the parent directory from where the projects have to be imported for Layer-2 kepler (K2K) build projects.

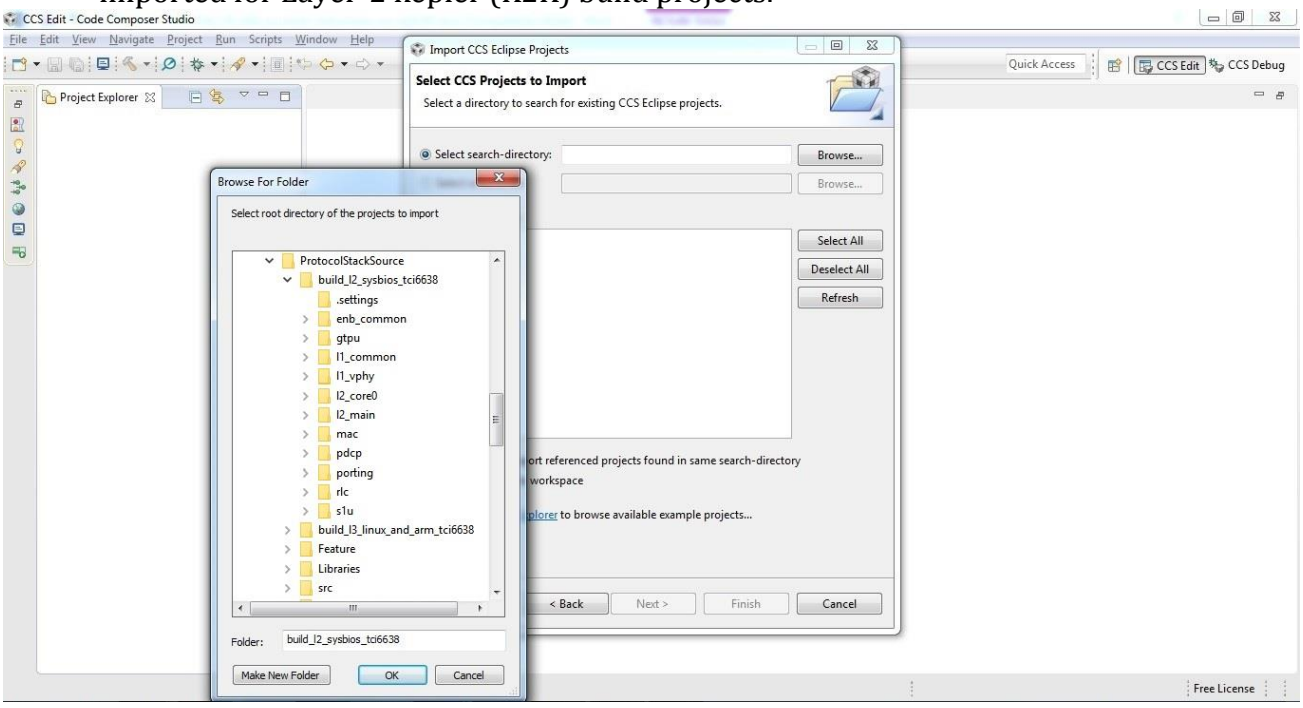

Figure 6: Importing L2 project into Workspace - step3

- This would list all the projects under the *Discovered Projects* area. Click on *Select All* option on the right and the click the *Finish* tab in the end.

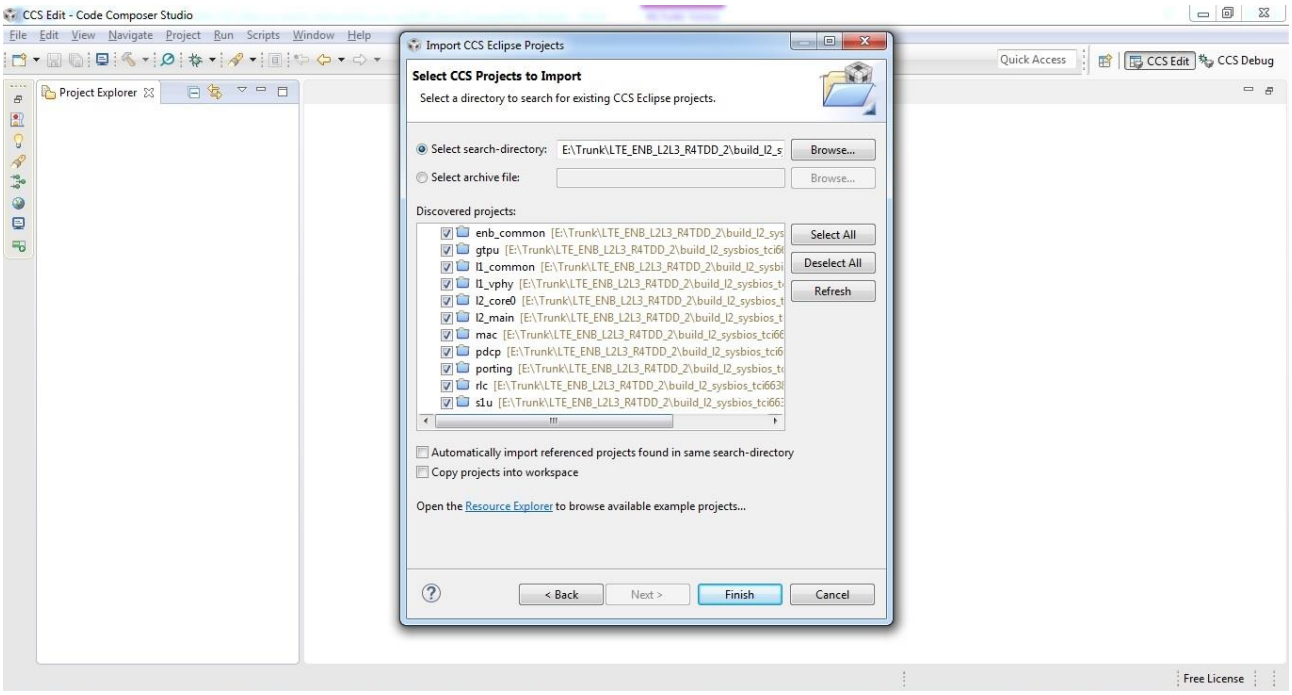

Figure 7: Importing L2 project into Workspace - step4

- All the selected projects will now be imported in the workspace and appear in the CCS window.
  Projects: l2_core0, l2_main, enb_common, gtpu, l1_common, l1_vphy, mac, pdcp, porting, rlc, s1u
  Select the build configurations for the projects. The build process is described in the next section.

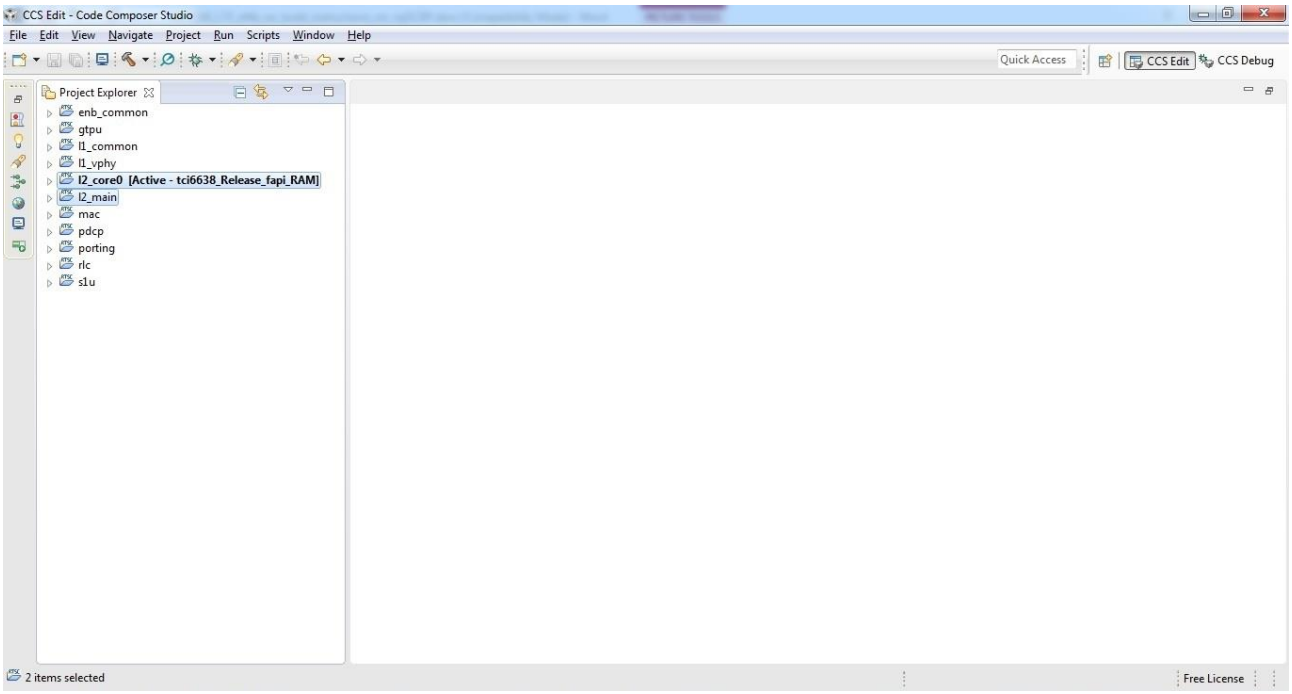

Figure 8: Importing L2 project into Workspace - step5

5.2 Building Layer 2

5.2.1 Build Configurations for Layer 2

The following build configurations are supported for Core-1:

- **tci6638_Release_fapi_RAM:** Build configuration for FDD duplex mode for RAM model (boot loadable via ARM).
- **tci6638_Release_fapi_TDD_RAM:** Build configuration for TDD duplex mode for RAM Model (boot loadable via ARM).

The following configurations are supported for Core-0:

- **tci6638_Release_fapi_RAM:** Build configuration for FDD duplex mode for RAM model (boot loadable via ARM).
- **tci6638_Release_fapi_TDD_RAM:** Build configuration for TDD duplex mode for RAM Model (boot loadable via ARM).

5.2.2 Core0 Compilation

Follow the instructions below for compiling these projects:

- Change the build configuration of project *l2_core0* project. Right click on l2_core0 -> Build Configurations -> Set Active ->Choose the Configuration as
  - *tci6638_Release_fapi_RAM* configuration must be used for FDD Core0 binary, loadable from ARM
  - *tci6638_Release_fapi_TDD_RAM* configuration must be used for TDD Core0 binary, loadable from ARM.
- Do clean project.
- Right click on l2_core0 project and select Build Project.

- Verify in console that all layer 2 projects are built without errors. Console can be seen by going to View->Console.
- The binary *dsp_core0.out* for project l2_core0 for configurations
  - *tci6638_Release_fapi_RAM* is built at the location '*build_l2_sysbios_tci6638/l2_core0/tci6638_Release_fapi_RAM*'
  - *tci6638_Release_fapi_TDD_RAM* is built at the location '*build_l2_sysbios_tci6638/l2_core0/ tci6638_Release_fapi_TDD_RAM'*

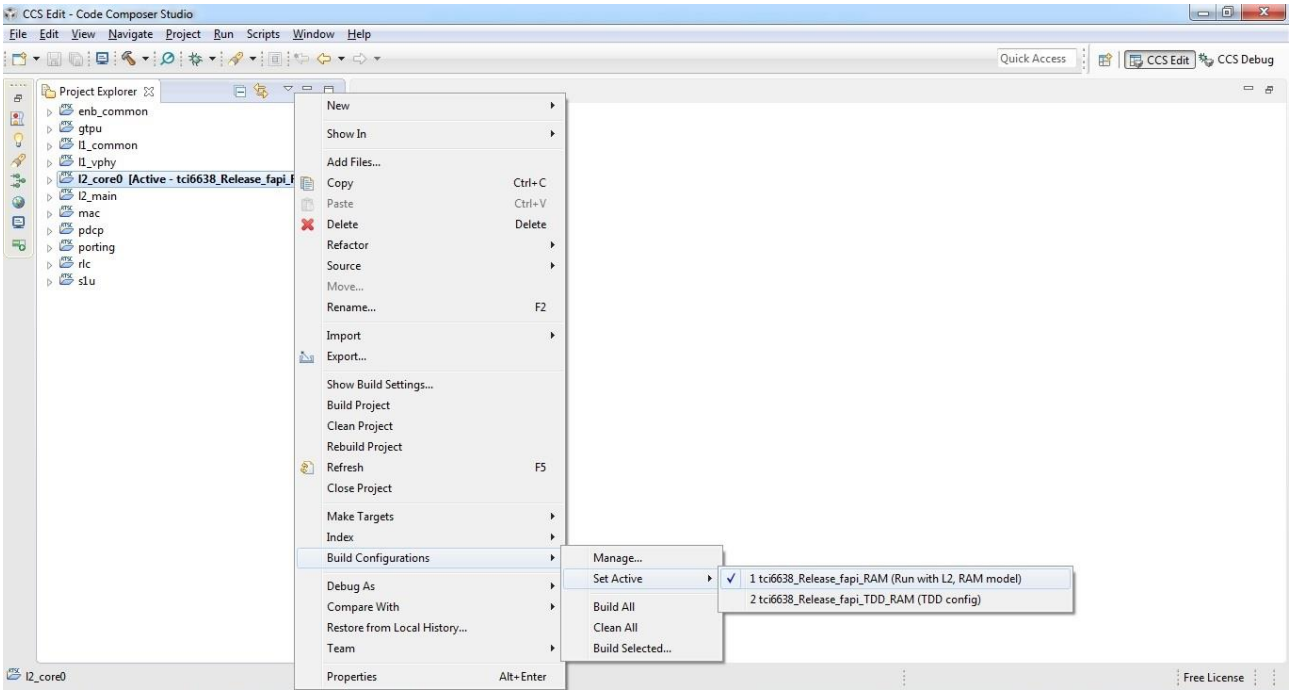

Figure 9: Changing the Build Configuration for l2_core0 for FDD

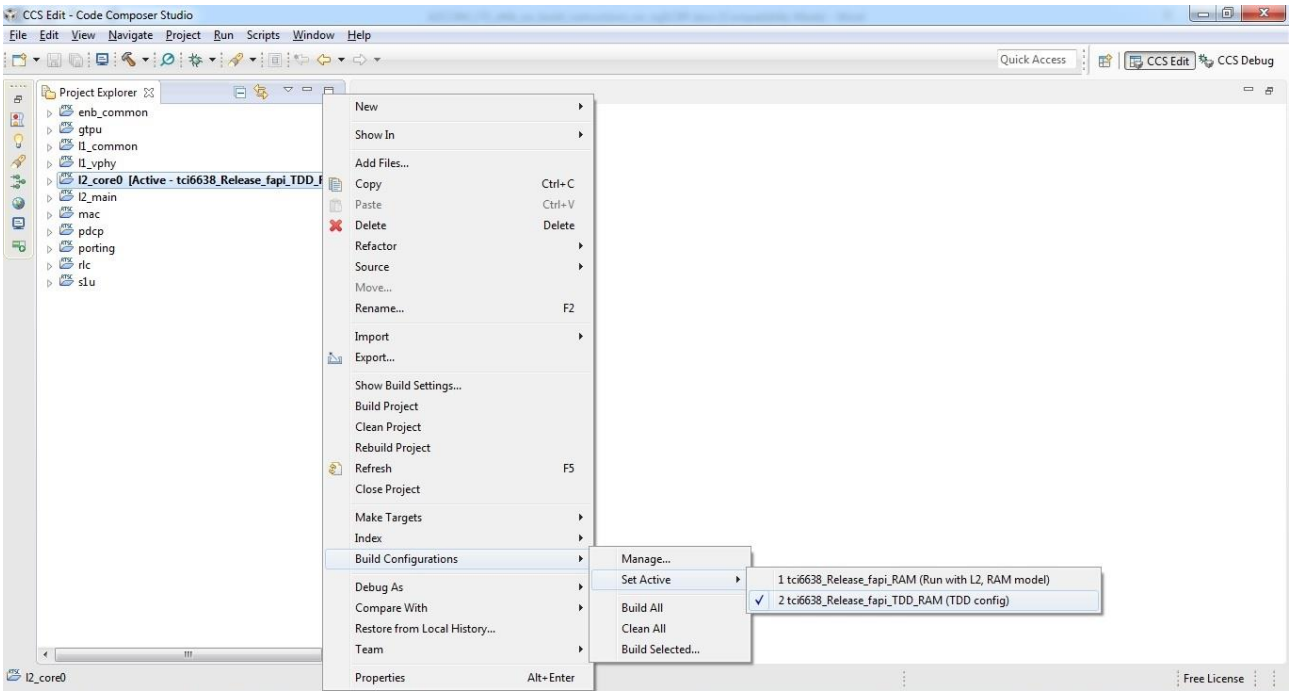

Figure 10: Changing the Build Configuration for l2_core0 for TDD

5.2.3 Core1 Compilation

Follow the instructions below for compiling these projects:

- Change build configuration of project l2_main according to the desired outcome. Right click on l2_main -> Build Configurations -> Set Active ->Choose the Configuration as
  - *tci6638_Release_fapi_RAM* configuration must be used for FDD Core1 binary, loadable from ARM.
  - *tci6638_Release_fapi_TDD_RAM* configuration must be used for TDD Core1 binary, loadable from ARM.
- Do clean project.
- Right click on l2_main project and select Build Project.
- Verify in console that all the layer 2 projects are built without errors. Console can be seen by going to View->Console.
- The binary dsp-core1.out for peoject l2_main for configurations
  - *tci6638_Release_fapi_RAM* configuration is built at the location '*build_l2_sysbios_tci6638/l2_main/tci6638_Release_fapi_RAM*'
  - *tci6638_Release_fapi_TDD_RAM* configuration is built at the location '*build_l2_sysbios_tci6638/l2_main/ tci6638_Release_fapi_TDD_RAM*'

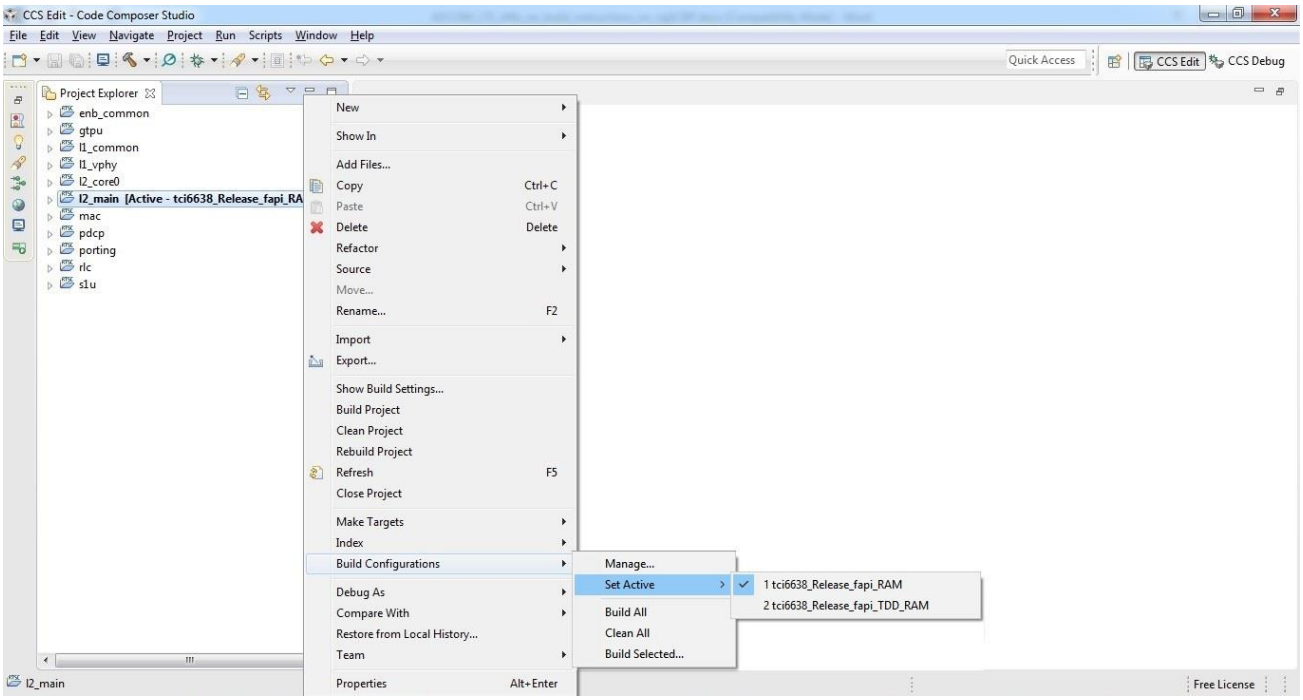

Figure 11: Changing the Build Configuration of l2_main for FDD

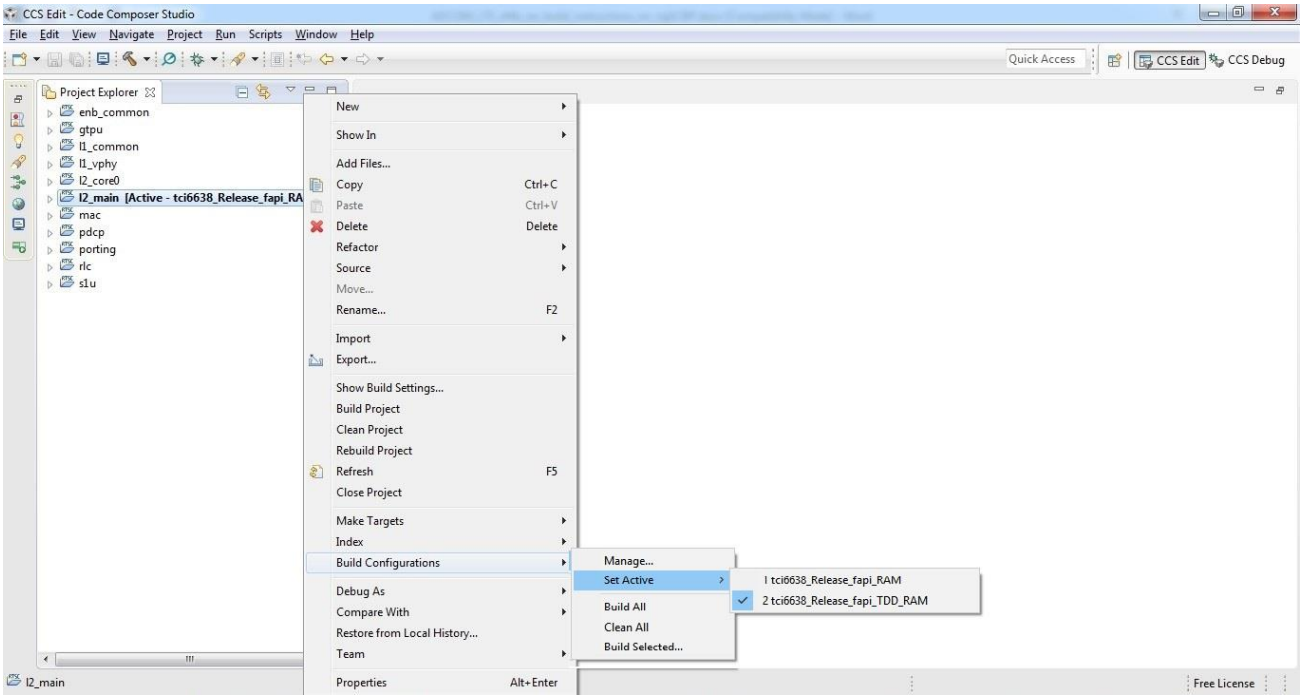

Figure 12: Changing the Build Configuration of l2_main for TDD

6 Build Process for Layer-3

The source code contains Layer-3 projects that should be imported into Eclipse and compiled for tci6638 (Kepler).

6.1 Importing Projects into Eclipse

- Start eclipse on Ubuntu.
- Click on *Browse* from the Eclipse launcher and select the path for the workspace. Then click *OK*.

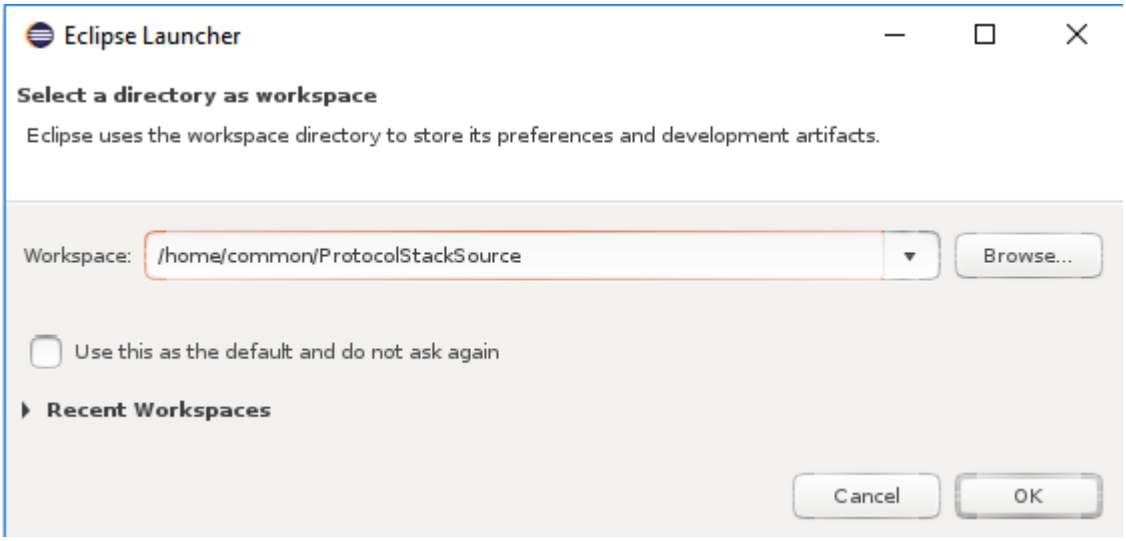

Figure 13: Creating Workspace for layer 3

- An Eclipse editing window will open. Go to the *File* option in the top left and select the *Import* option.

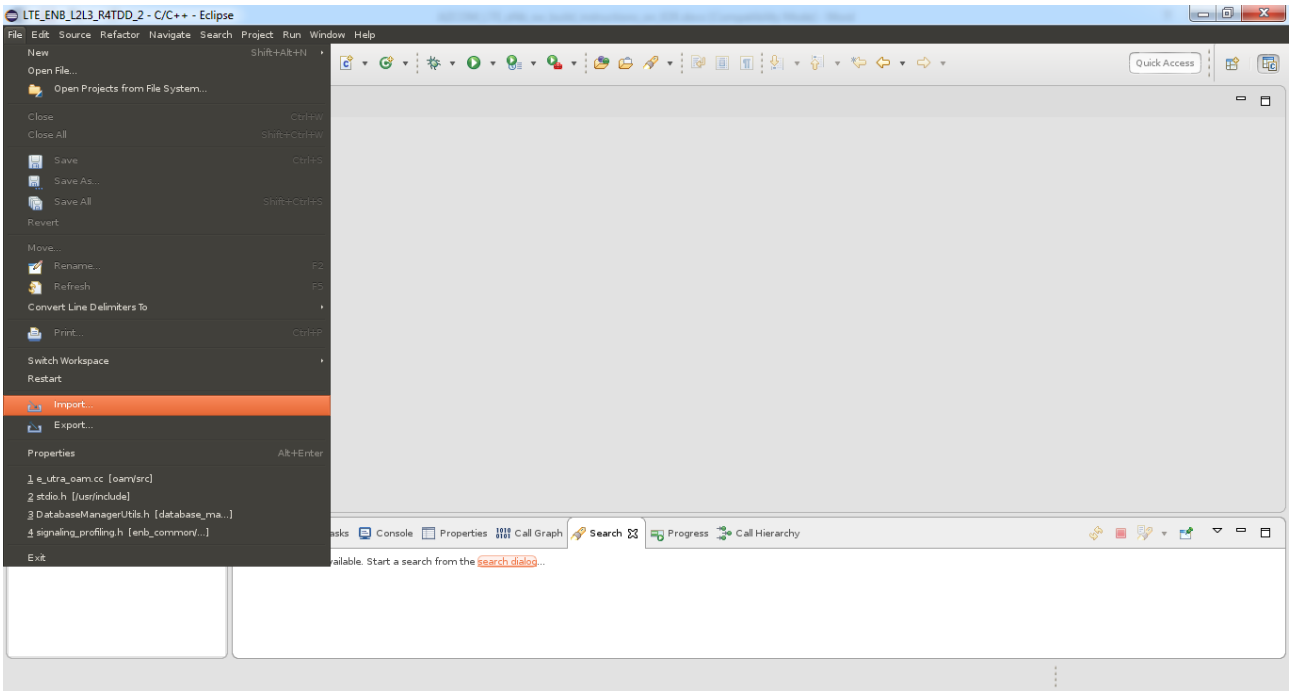

Figure 14: Importing L3 projects into the Workspace - step1

- An *Import* window will pop up. Go to the *General* drop down menu and select *Existing Projects into Workspace.* Then click *next.*

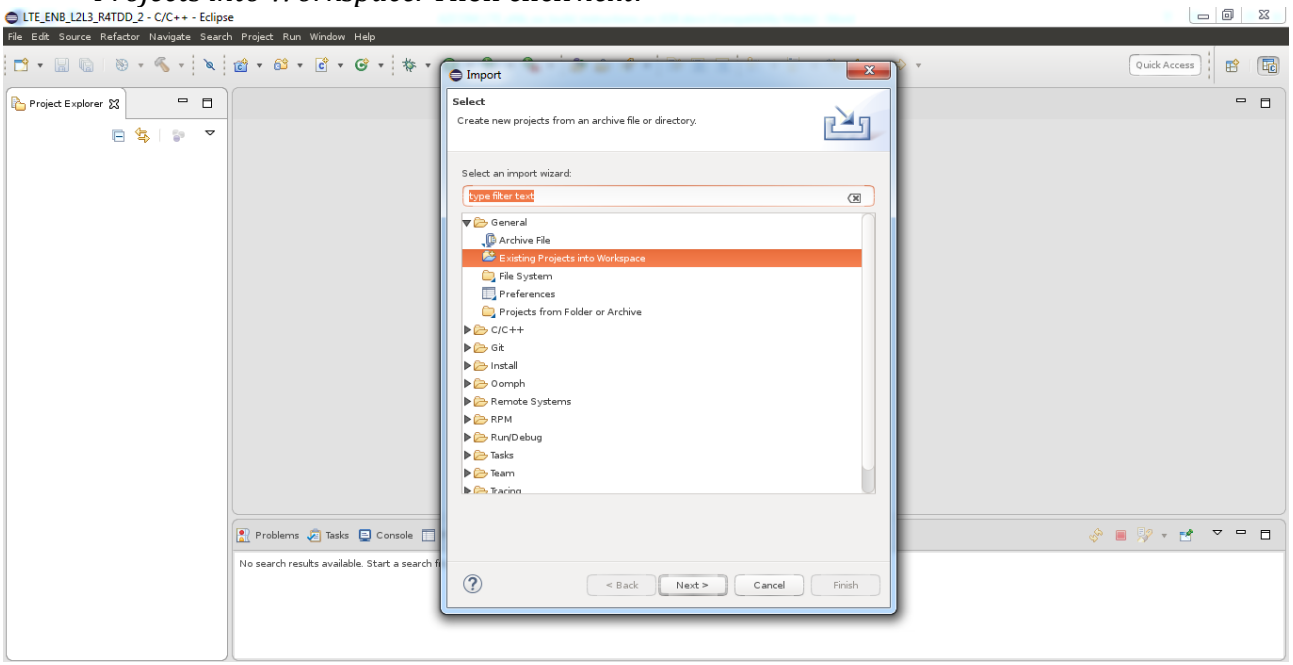

Figure 15: Importing L3 Projects into Workspace - step2

- A new window for importing the build projects pops up. Click on the *Browse* option and select the Folder *build_l3_linux_and_arm_tci6638* for the Layer-3 kepler (K2K) build projects to be imported.
- All the projects for Layer-3 will be listed in the *Projects* area. Click on *Select All* option on the right and then click *Finish* in the end.

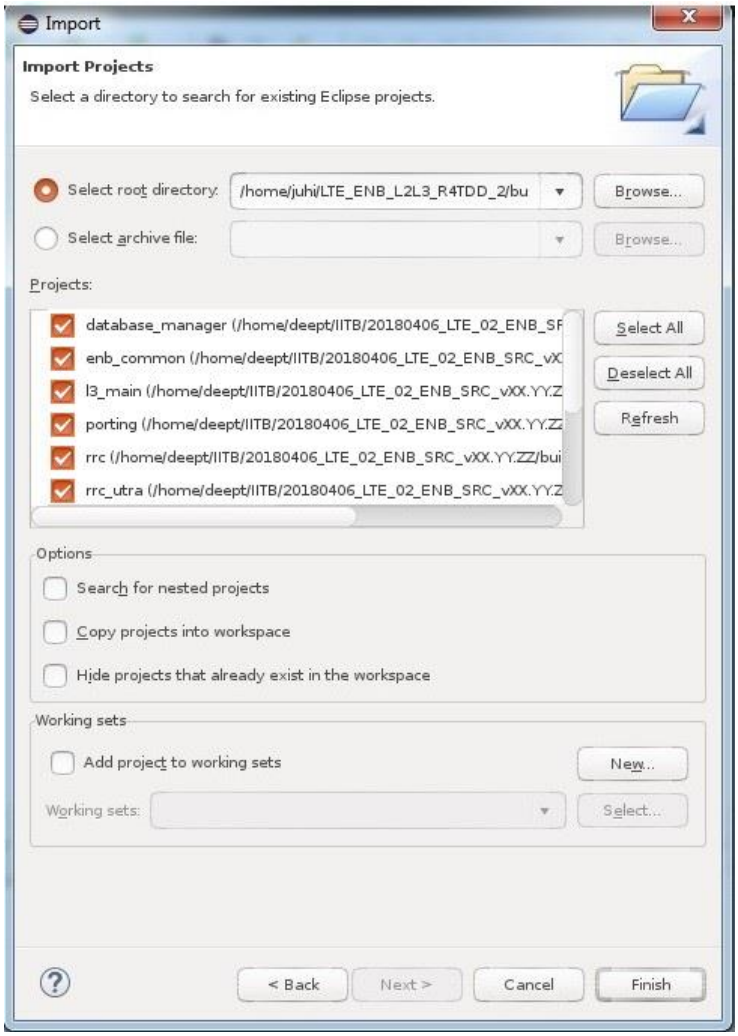

Figure 16: Importing L3 projects into Workspace - step3

- This will import all the Layer-3 projects into the Eclipse workspace.
  - *database_manager*
  - *enb_common*
  - *l3_main*
  - *porting*
  - *rrc*
  - *rrc_utra*
  - *rrm*
  - *s1ap*
  - *x2ap*

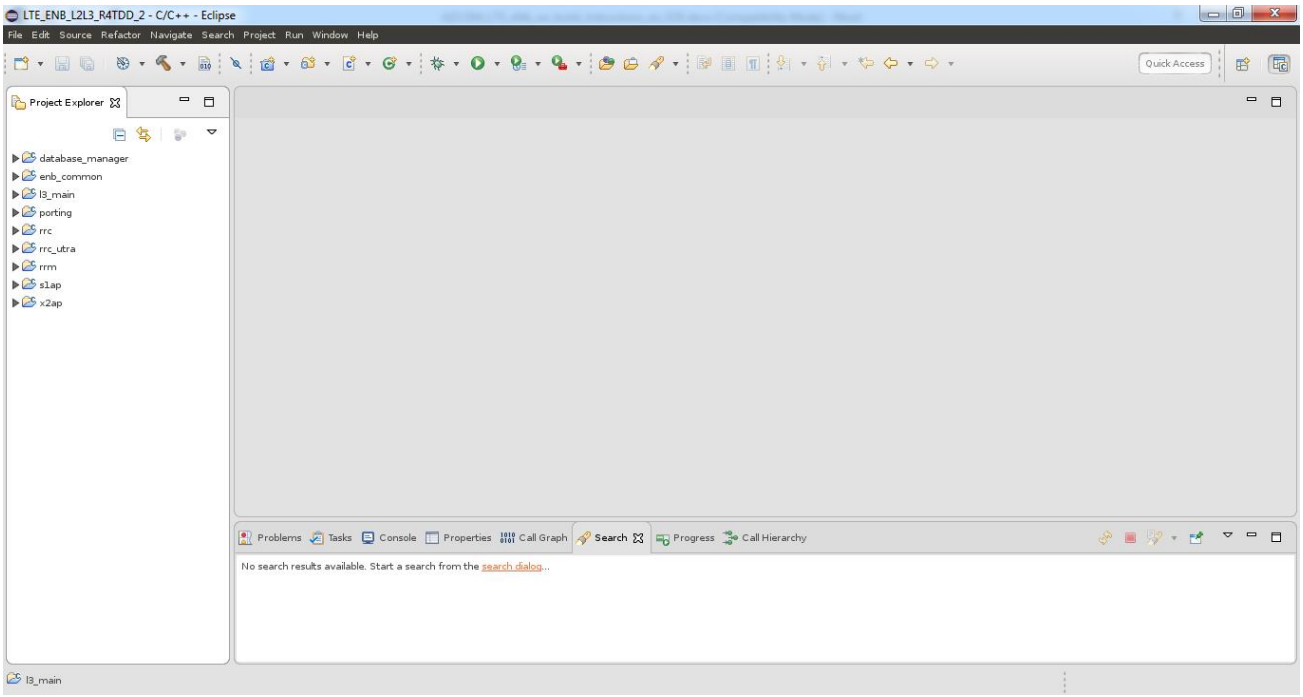

Figure 17: L3 Projects imported into the workspace

6.2 Building Layer-3

6.2.1 Available Build Configurations for Layer-3

The following build configuration is available:

- **Release_ARM**
  - Layer 3 binary for Azcom baseband board. Two binaries are created at build_l3_linux_and_arm/l3_main/Release_Arm
    - "enb_l3" for local debugging
    - "eNB.bin" to be used on the Azcom baseband board.

6.2.2 Layer 3 Compilation

Follow the instructions below for compiling these projects:

- Select build configuration.

- **Layer 3 binaries for Azcom baseband board**
  - Set the active build configuration as Release_ARM for project *l3_main*. Right click on the project and select Build Configurations->Set Active->Release_ARM.

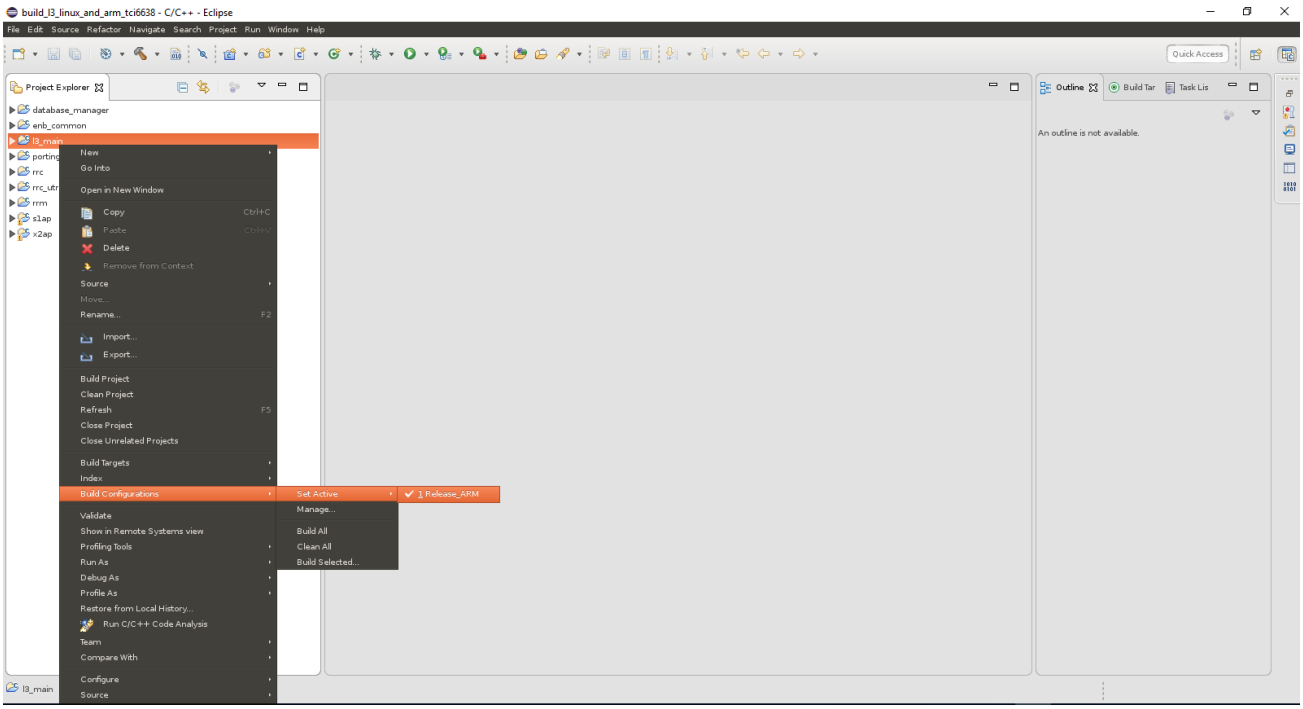

Figure 18: Setting Build Configuration as Release_ARM

- Do clean project.
- Ensure active build configuration has been set to Release_ARM
- Build the *l3_main project*. Right click on *l3_main* and click *Build Project*.
- Verify in console that all the layer 3 projects are built without errors. Console can be seen by clicking on Window->Show View->Console.
- This should build both the debug and stripped binaries at the location *build_l3_linux_and_arm_tci6638/l3_main/ Release_ARM* of the code source.