Getting Started with 4G LTE using Eurecom OpenAirInterface (OAI) on the USRP 2974

Contents

Neel Pandeya

AN-888

This Application Note explains how to install and configure the OpenAirInterface (OAI) software on the USRP 2974 hardware to implement a 4G LTE cellular basestation (eNodeB). Specifically, this document discusses the installation and configuration of the OAI eNodeB physical layer software. Future application notes will discuss the installation and configuration of the other layers of the stack, other network components such as the EPC and MME, as well as the integration of the handset side (the UE side) with connectivity to the public internet.

The USRP 2974 is a high-performance, stand-alone software defined radio (SDR) device. It is equivalent to a USRP X310, with two UBX-160 daughterboards and a GPSDO module, and an on-board Intel-based System-on-Module (SoM) computer.

The X310 and UBX radio provide a tuning range between 10 MHz and 6 GHz, with up to 160 MHz of baseband bandwidth per channel. The radio has two transmit channels and two receive channels, which can each be tuned independently. The X310 contains a Xilinx Kintex-7 XC7K410T FPGA, 14-bit ADCs, 16-bit DACs, and supports multiple interfaces to the host computer (1 and 10 Gbps Ethernet, and PCIe).

The internal Intel-based SoM features an Intel Core i7-6822EQ quad-core CPU, running at 2.0 GHz, and 16 GB DD4 memory, with a 512 GB SSD disk, and with both 1 and 10 Gbps Ethernet interfaces. The front panel of the USRP 2974 provides USB, video, and Ethernet ports for connecting to external devices. The internal SoM can fully control the X310 without the need for any external computer or device.

The Knowledge Base contains detailed technical information, schematics, photos, and physical dimensions and other mechanical information for the USRP 2974. The code for the UHD software driver and the internal FPGA is open-source and is hosted on GitHub.

The OpenAirInterface (OAI) software provides an open-source, standards-compliant implementation of a 3GPP 4G LTE stack that runs on a commodity x86 CPU and a USRP radio device. OAI was initially developed by Eurecom, but is now managed by the OpenAirInterface Software Alliance (OSA), which is a French non-profit organization that provides open-source software and tools for 4G and 5G wireless research. The OAI software provides a full experimental LTE implementation (3GPP Releases 8 and 9, and partially 10 and 11) that runs in real-time and is capable of operating with commercial LTE handsets (UEs). The OAI software spans the full protocol stack of the 3GPP LTE standards, and includes implementations of EUTRAN (both eNB and UE) and EPC (MME, S+P-GW, and HSS). There is a Wiki on the OAI repository with further information here.

The availability of OAI source code is free for non-commercial and academic research purposes. The eNB and UE implementations are licensed under OAI Public License v1.1, which is a modified version of the Apache v2.0 License, with a modified patent clause that allows contributing parties to make patent licenses available to third parties under fair, reasonable and non-discriminatory (FRAND) terms. The EPC (MME, S+P-GW, and HSS) implementation is licensed under the Apache v2.0 License.

The eNB physical layer software implements 3GPP 36.211, 36.212, 36.213, and provides the following features:

- FDD and TDD configurations: 1 and 3
- Carrier bandwidths: 5, 10, and 20 MHz
- Transmission modes: 1, 2 (stable); 3, 4, 5, 6, 7 (experimental)
- Max number of antennas: 2
- PRACH preamble format 0
- All downlink (DL) channels supported: PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH, PMCH
- All uplink (UL) channels supported: PRACH, PUSCH, PUCCH (format 1/1a/1b), SRS, DRS
- HARQ support for UL and DL
- Highly optimized baseband processing (including Turbo decoder)
- Possible DL throughputs (measured by OSA with COTS UE Cat 3/4):
    - 5 MHz, 25 PRBs / MCS 28 = 16-17 Mbps
    - 10 MHz, 50 PRBs / MCS 28 = 34-35 Mbps
    - 20 MHz, 100 PRBs / MCS 28 = ~70 Mbps
- Possible UL throughputs (measured by OSA with COTS UE Cat 3/4):
    - 5 MHz, 20 PRBs / MCS 20 = 9 Mbps
    - 10 MHz, 45 PRBs / MCS 20 = 17 Mbps

    - 20 MHz, 96 PRBs / MCS 20 = ~35 Mbps

Support for Carrier Aggregation (CA) and measurement gap handling is currently incomplete.

New and upcoming eNB features include:

- DRX/eDRX and CDRX handling
- Multi RRU handling and synchronization
- X2 interface and handover
- Release 12 Dual Connectivity (DC)
- Release 13 LTE-M
- Release 14 NB-IoT

The eNB MAC layer implements a subset of the 3GPP 36.321 v8.6 in support of BCH, DLSCH, RACH, and ULSCH channels.

The eNB MAC implementation includes:

- RRC interface for CCCH, DCCH, and DTCH
- Proportional fair scheduler (round robin scheduler soon)
- DCI generation
- Support for HARQ
- RA procedures and RNTI management
- RLC interface
    - Acknowledged Mode (AM)
    - Unacknowledged Mode (UM)
- UL power control
- Link adaptation

The current PDCP is header-compliant with 3GPP 36.323 v10.1.0, and implements the following functions:

- User and control data transfer
- Sequence number management
- RB association with PDCP entity
- PDCP entity association with one or two RLC entities
- Integrity check and encryption using the AES and Snow3G algorithms

The RLC layer implements the full specification of the 3GPP 36.322 v9.3.

The RRC layer is based on 3GPP 36.331 v14.3.0 and implements the following features:

- System Information Broadcast (SIB) formats 1, 2, 3, and 13
- RRC connection establishment, reconfiguration, release, re-establishment
- RRC inactivity timer
- Inter-frequency measurement collection and reporting
- eMBMS for multicast and broadcast
- X2 Handover
- Paging

The OAI eNB software uses FAPI for the interface between the Layer 1 physical layer and the Layer 2 MAC. The FAPI is essentially a standardized interface between MAC and PHY layers, where both layers run on the same physical system. The nFAPI extends the FAPI for networked functions. This enables the PHY and MAC layers to run two different physical systems. They can be separated into Virtual Network Functions (VNF) and Physical Network Functions (PNF). FAPI is specified by Small Cell Forum (version SCF082.09.05 on 2017-05-18), and the OAI eNB software uses the open-source implementation Open-nFAPI, provided by Cisco Systems.

The OSA recommends the use of Ubuntu 18.04 or RHEL/CentOS 7.4 with the OAI eNB software. Newer versions of these Linux distributions, specifically Ubuntu 19.04 and RHEL/CentOS 8, should also work, as they have newer kernels. For real-time operation, the use of a low-latency kernel is strongly recommended. Ettus Research recommends using Ubuntu instead of RHEL/CentOS for better work-flow and ease-of-use with the UHD driver. The USRP 2974 requires UHD version 3.14.1.0 or higher, and version 1.0.3 of the OAI software will be used.

The code for the OAI eNB and UE software is located in a GitLab repository here. There is an OAI Wiki here. The OAI EPC code is located in a separate GitHub repository.

The OAI eNB and UE codebase uses the CMake build system and the GCC compiler. The GitLab repository contains the following top-level folders listed below.

- `ci-scripts`: Contains scripts and files for automated testing.
    - `ci-scripts/conf_files`: Contains example OAI configuration files.

- `cmake_targets`: Contains build utilities for building all the targets (components). Contains the top-level `build_oai` script.

- `common`: Contains various OAI tools and utilities.
    - `common/utils/T`: Contains the T tracer tool.

- `doc`: Contains the Doxygen-based OAI documentation. Generated at build-time.

- `nfapi`: Contains nFAPI code, including a clone of the Cisco Open-nFAPI GitHub repository, as well as code from Cisco to integrate Open-nFAPI into OAI.

- `oaienv`: The script that sets the environment of the shell appropriately to include the OAI installation.

- `openair1`: Contains the code for the eNB and UE Layer 1 physical layer implementation (3GPP LTE Release 10/12 PHY layer and PHY RF simulation, TS 36.211, 36.212, 36.213).

- `openair2`: Contains the implementation for Layer 2 (3GPP LTE Release 10 for RLC/MAC/PDCP/RRC/X2AP implementation).

- `openair3`: Contains the implementation for the 3GPP LTE Release 10 interfaces for S1-C, S1-U (GTP, SCTP, S1AP) and NAS UE.

- `targets`: Top-level wrappers for unitary simulation for PHY channels, system-level emulation (eNB-UE with and without S1), and real-time eNB and UE and RRH GW.

This document uses version 1.0.3 of the OAI software, which is defined by the Git tag `v1.0.3`.

Before building or running OAI, it is necessary to run `source openairinterface5g/oaienv` to include the OAI installation into your shell's environment. The primary script for building the OAI software is `openairinterface5g/cmake_targets/build_oai`. This script has command-line options for building each of the components. Once successfully built, the binaries will be placed in the `openairinterface5g/targets/bin` folder. The binary for the OAI eNB physical layer is `lte-softmodem-nos1.Rel14`.

In this document, we will run the OAI software without the S1 interface to the MME and HSS in the EPC. For this configuration, we will need to invoke the `build_oai` script with the `--noS1` option. A kernel module `nasmesh.ko` will be built, and we will load it using the `openairinterface5g/cmake_targets/tools/init_nas_nos1` tool.

Note that before starting the eNB, the `nasmesh.ko` kernel module has to be loaded to set up the radio bearer and provide the IP connectivity between eNB and attached UE. The system should have the `oai0` network interface created, with IP address of 10.0.1.1, and with netmask of 255.255.255.0. Use `ifconfig` to verify this. In this exercise, we are not running the eNB with any real UE or UE emulator, so this kernel module and network interface do not get used for any data transfer, but the eNB software still requires that they be created and exist.

The installation procedure starts from the point of having a brand-new, out-of-the-box USRP 2974. The user will need to perform the following steps: install and configure Ubuntu 18.04.3; install and configure UHD version 3.14.1.0; install and configure OAI version 1.0.3; and finally run the OAI eNB software.

There is an Application Note on the Ettus Research Knowledge Base that provides a comprehensive installation guide for UHD on Linux. That Application Note is a general-purpose reference, and the complete procedure specific to the USRP 2974 and the OAI software is documented here, and it is recommended that readers follow this document.

The user can install Ubuntu directly onto the USRP 2974. Be sure to specifically download version 18.04.3. Download the ISO file, which should be about 1.9 GB in size, and then write this ISO file to a USB 3.0 flash drive. There are many free tools available for Windows, OS X, and Linux for doing this, and there are many tutorials posted online, such as here and here and here. Be sure to use a USB flash drive with at least 8 GB capacity, and use a USB 3.0 flash drive, not a USB 2.0 flash drive. If you use a slower USB 2.0 flash drive, then the install process will take significantly longer.

Once Ubuntu 18.04.3 is installed, you should first make sure that all the packages that are already installed on your system are up-to-date. To do this, run the command listed below.

```
sudo apt-get update
sudo apt-get upgrade
```

For real-time operation, the use of a low-latency kernel is strongly recommended. To do this, run the command listed below.

```
sudo apt-get install linux-lowlatency-hwe-18.04
```

Once the installation completes without any errors, reboot the system. Once the system has rebooted, verify that the low-latency kernel is running. To do this, run the command listed below.

```
uname -a
```

The system should display something similar to the output listed below. Note the `-lowlatency` suffix added to the kernel version.

```
Linux hostname 5.0.0-25-lowlatency #26~18.04.1-Ubuntu SMP PREEMPT Thu Aug 1 14:35:26 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

Next, we must configure the Ethernet interfaces. The USRP 2974 should have three Ethernet interfaces: `enp0s31f6` (for the 1 Gbps RJ-45 Ethernet interface), `enp1s0f0` (for Port 0 of the 1/10 Gbps SFP+ Ethernet interface), `enp1s0f1` (for Port 1 of the 1/10 Gbps SFP+ Ethernet interface).

The 1 Gbps RJ-45 Ethernet interface (`enp0s31f6`) will usually be connected to a local Ethernet network to provide public internet access to the system, and will have a dynamic address given by DHCP, or a static address prescribed by your company, university, or organization.

The Port 0 SFP+ interface (`enp1s0f0`) should have a static IP address of 192.168.40.1, with a netmask of 255.255.255.0, and with an MTU value of 9000. To do this, run the command listed below.

```
sudo ifconfig enp1s0f0 192.168.40.1 netmask 255.255.255.0 mtu 9000
```

You can also make these settings for the `enp1s0f0` interface using the graphical Network Manager. If you set the IP address from the command line with `ifconfig`, note that Network Manager might overwrite these settings. The command below will use the Network Manager to create a new connection with the appropriate settings.

```
sudo nmcli con add con-name "USRP 2974 10Gbps" ifname enp1s0f0 type ethernet ip4 192.168.40.1/24 mtu 9000
```

The Port 1 SFP+ interface (`enp1s0f1`) is not directly connected to the radio, and does not need to be configured.

Next, ensure that your CPU governor is set to `performance` mode. This can be done with the Linux utility `cpufreq-set`, which is provided by the `cpufrequtils` package, and can be installed with the command listed below.

```
sudo apt install cpufrequtils
```

Then, set the CPU governor to `performance` for each CPU core with the command listed below.

```
sudo cpufreq-set -c $core_number -g performance
```

To set the CPU governor to `performance` for all cores on the system, use the command listed below.

```
for ((i=0;i<$(nproc);i++)); do sudo cpufreq-set -c $i -r -g performance; done
```

Then, verify that the CPU governor has been set correctly by running the command listed below.

```
cpufreq-info
```

You can also verify this using the `i7z` utility, as shown below.

```
sudo apt-get install i7z
sudo i7z
```

Additional information about this is posted on the Knowledge Base here and here.

Disable the C-states of the CPU with the commands listed below.

```
sudo apt install linux-tools-common linux-tools-lowlatency linux-tools-5.0.0-27-lowlatency
sudo cpupower idle-set -D 2
```

You can verify that the new settings have been applied by using the command below.

```
sudo cpupower idle-info
```

UHD is open-source, and is hosted on GitHub. The UHD driver may be installed either manually from source code, or by the OAI build scripts. We recommend building manually from source code, and that procedure will be explained in this section.

First, it is necessary to install the required dependencies for UHD. To do this, run the command listed below. Some of the packages shown in the command are not strictly necessary but provide utilities that greatly help with workflow, system usage, and debugging system and software issues. The dependencies for the OAI software will be installed by the OAI build script.

```
sudo apt-get -y install tree htop glances ethtool net-tools git qgit swig cmake doxygen build-essential gedit evince i7z lshw inxi smartmo
```

After installing the dependencies, reboot the system.

Next, we will clone the UHD repository on GitHub, check out a specific tagged release of the repository, and then build and install from source code.

First, make a folder in the home directory to hold the repository.

```
cd $HOME
mkdir git-repositories
cd git-repositories
```

Next, clone the repository, and then go into the cloned repository.

```
git clone https://github.com/EttusResearch/uhd
cd uhd
```

Next, checkout UHD version 3.14.1.0.

```
git checkout v3.14.0.0
```

Next, create a `build` folder within the `host` folder of the repository.

```
cd host
mkdir build
cd build
```

Next, invoke CMake to create the Makefiles.

```
cmake ../
```

Next, run Make to build UHD.

```
make
```

Next, you can optionally run some basic tests to verify that the build process completed properly.

```
make test
```

Next, install UHD, using the default install prefix, which will install UHD under the `/usr/local/lib` folder. You need to run this as root due to the permissions on that folder.

```
sudo make install
```

Next, update the system's shared library cache.

```
sudo ldconfig
```

Finally, make sure that the `LD_LIBRARY_PATH` environment variable is defined, and includes the folder in which UHD was installed. Usually you can add the line below to the end of your `$HOME/.bashrc` file:

```
export LD_LIBRARY_PATH=/usr/local/lib
```

If the `LD_LIBRARY_PATH` environment variable is already defined with other folders in your `$HOME/.bashrc` file, then add the line below to the end of your `$HOME/.bashrc` file to preserve your current settings.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

For this change to take effect, you will need to close the current terminal window, and open a new terminal window.

At this point, UHD should be installed and ready to use. Verify this by running `uhd_find_devices`. You should see something similar to the following.

```
[INFO] [UHD] linux; GNU C++ version 7.4.0; Boost_106501; UHD_3.14.1.HEAD-0-gbfb9c1c7
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    serial: 3333333
    addr: 192.168.40.2
    fpga: HG
    name:
```

```
    product: X310
    type: x300
```

Now that UHD has been installed, we need to download the USRP FPGA images from the Ettus Research website. To do this, run the command listed below. The FPGA image files will be stored in the `/usr/local/share/uhd/images` folder.

```
sudo uhd_images_downloader
```

Next, we need to write the "XG" FPGA image to the USRP 2974.

```
uhd_image_loader --args "type=x300,fpga=XG"
```

Do not interrupt the writing program. If there is an interruption, an incomplete FPGA image will be written to flash memory, and the USRP 2974 will be bricked. To unbrick the device, you will need to write the FPGA via JTAG, and then re-write it with the `uhd_image_loader` utility.

Once the writing program successfully completes, reboot the USRP 2974.

Verify that the FPGA image has been correctly written by running the `uhd_usrp_probe` utility.

UHD requires larger read and write socket buffer sizes than the default values to avoid potential overflows and underruns at high sample rates. A size of 32 MB should be sufficient. To do this, run the command listed below.

```
sudo sysctl -w net.core.rmem_max=33554432
sudo sysctl -w net.core.wmem_max=33554432
```

These settings will not persist across a reboot of the system. In order to make these settings permanent, add the following lines into the `/etc/sysctl.conf` file, as shown below.

```
net.core.rmem_max=33554432
net.core.wmem_max=33554432
```

Next, increase the system stack size limit, with the command below.

```
sudo ulimit -s 8192
```

When UHD spawns a new thread, it may try to boost the thread's scheduling priority. If setting the new priority fails, then the UHD software prints a warning to the console, as shown below. This warning is harmless; it simply means that the thread will retain a normal or default scheduling priority.

```
UHD Warning:
    Unable to set the thread priority. Performance may be negatively affected.
    Please see the general application notes in the manual for instructions.
    EnvironmentError: OSError: error in pthread_setschedparam
```

To address this issue, non-privileged (non-root) users need to be given special permission to change the scheduling priority. This can be enabled by creating a group `usrp`, adding your username to it, and then appending the line `@usrp - rtprio 99` to the file `/etc/security/limits.conf`.

```
sudo groupadd usrp
sudo usermod -aG usrp $USER
```

Then add the line below to end of the file `/etc/security/limits.conf`:

```
@usrp - rtprio  99
```

You must log out and log back into the account for this settings to take effect.

Additional information about this is posted in the UHD User Manual here.

Return to the folder that holds all Git repositories. To do this, run the command listed below.

```
cd $HOME/git-repositories
```

Next, clone the OAI Git repository into this folder. To do this, run the command listed below.

```
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git
```

Next, go into the repository, and set your environment to include the OAI repository. To do this, run the command listed below. It is always necessary to have the OAI repository included in your environment whenever you are building or running the OAI eNB software.

```
cd openairinterface5g
source oaienv
```

The `oaienv` script defines several environment variables, including `OPENAIR_HOME`, which points to the top-level folder of your OAI repository.

Use the OAI eNB software build script both to install all the dependencies as well as to build the software. You can invoke the build script with the `-h` option to view all the command-line options available, as shown below.

```
cd $OPENAIR_HOME/cmake_targets
./build_oai -h
```

Install all the dependencies required by the OAI eNB software, using the build script.

```
cd $OPENAIR_HOME/cmake_targets
./build_oai -w USRP -I --install-optional-packages
```

The `-w USRP` option in the command above tells the OAI build script to install UHD. If you have alrady installed UHD manually from source, as was shown above, then this option is not necessary, and the dependencies may be installed as shown below.

```
cd $OPENAIR_HOME/cmake_targets
./build_oai -I --install-optional-packages
```

Next, build the OAI eNB software from source code, using the OAI build script, as shown below. The `-w USRP` option is necessary here whether or not UHD was previously installed manually, as it tells the build script to use the USRP as the target radio hardware. The `--noS1` option tells the build script to build the eNB software without any support for the S1 interface in the EPC.

```
./build_oai -w USRP --eNB -c -C --noS1
```

Watch closely for any errors, and for the `build have failed` message when the build script terminates.

You can optionally include the `--build-doxygen` command-line option to the build script to generate the Doxygen-based documentation.

Once the eNB software has been successfully built, you will need to configure it before running it. Most of the configuration parameters and settings are made through a configuration file. There are example configuration files in the folder listed below.

```
$OPENAIR_HOME/ci-scripts/conf_files
```

In this document, we'll use the configuration file listed below.

```
$OPENAIR_HOME/ci-scripts/conf_files/enb.band7.tm1.25PRB.usrpb210.conf
```

In order to make it work with the USRP 2974, we will add the line listed below to the `RUs = (` section.

```
sdr_addrs = "type=x300,addr=192.168.40.2";
```

The final `RUs` section should be edited as shown below. Note that the `max_rxgain` parameter also needs to be changed.

```
RUs = (
    {
       local_rf       = "yes"
       nb_tx          = 1
       nb_rx          = 1
       att_tx         = 5
       att_rx         = 0;
       bands          = [7];
       max_pdschReferenceSignalPower = -27;
       max_rxgain                    = 117;
       eNB_instances  = [0];
       sdr_addrs      = "type=x300,addr=192.168.40.2";
    }
);
```

Since we are not using the S1 interface in the EPC, we will need to install the `nasmesh.ko` kernel module. To do this, run the `init_nas_nos1` utility, as shown below. This should be run before invoking the eNB software.

```
sudo -E $OPENAIR_HOME/targets/bin/init_nas_nos1 eNB
```

Running this command should create the `oai0` network interface on the system. Verify this by running `ifconfig`. The output should be similar to what is shown below.

```
oai0: flags=4291<UP,BROADCAST,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet 10.0.1.1  netmask 255.255.255.0  broadcast 10.0.1.255
        netrom     txqueuelen 100  (AMPR NET/ROM)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Invoke the OAI eNB software with the command listed below. Replace the configuration file specified with the `-O` option below with the correct path and filename of your edited configuration file.

```
sudo -E $OPENAIR_HOME/targets/bin/lte-softmodem-nos1.Rel14 -O openairinterface5g/ci-scripts/conf_files/enb.band7.tm1.25PRB.usrpb210.conf -
```

If you see the message listed below, you can safely ignore it, as we are not using the LTE NB-IoT.

```
[LOADER] library libNB_IoT.so is not loaded: libNB_IoT.so: cannot open shared object file: No such file or directory
```

If you see the error listed below, then you have not loaded the kernel module `nasmesh.ko` using the `$OPENAIR_HOME/cmake_targets/tools/init_nas_nos1` tool, and you do not have `oai0` network interface created.

```
[NETLINK] Error opening socket -1 (93:Protocol not supported)
```

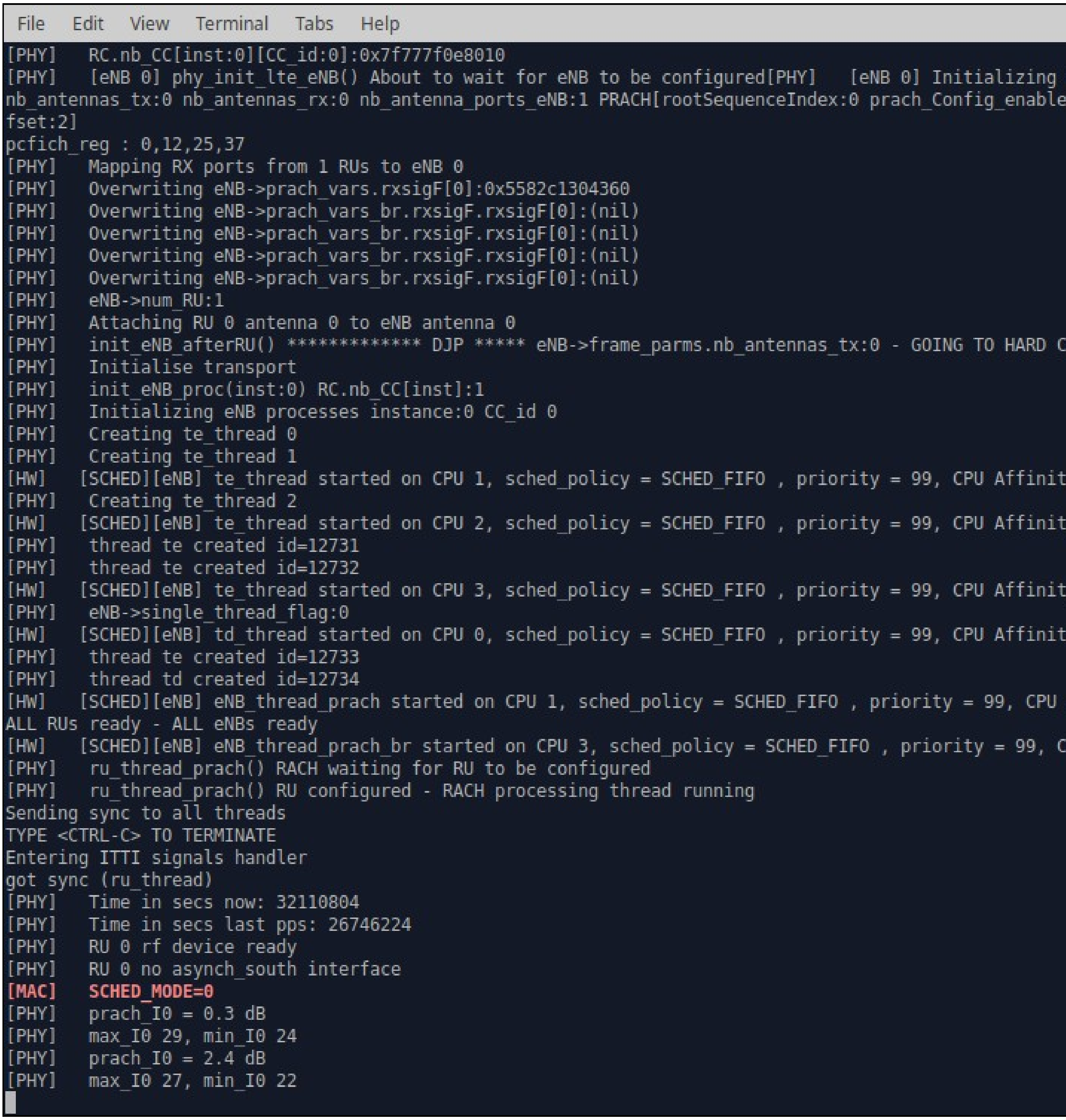
Once the OAI eNB software is running, there will be many messages displayed to the console window. You should see something similar to the output listed below.

```
user@hostname:~/Desktop/git/openairinterface5g >> sudo -E ./targets/bin/lte-softmodem-nos1.Rel14 -O ~/Desktop/Eurecom_OAI/enb.band7.tm1.25PRB
[CONFIG] get parameters from libconfig /home/user/Desktop/Eurecom_OAI/enb.band7.tm1.25PRB.usrpb210.conf , debug flags: 0x00000000
[CONFIG] function config_libconfig_init returned 0
[CONFIG] config module libconfig loaded
[LIBCONFIG] config: 1/1 parameters successfully set, (1 to default value)
# /dev/cpu_dma_latency set to 0us
[LIBCONFIG] log_config: 3/3 parameters successfully set, (1 to default value)
[LIBCONFIG] log_config: 38/38 parameters successfully set, (32 to default value)
[LIBCONFIG] log_config: 38/38 parameters successfully set, (38 to default value)
[LIBCONFIG] log_config: 15/15 parameters successfully set, (15 to default value)
[LIBCONFIG] log_config: 15/15 parameters successfully set, (15 to default value)
log init done
Reading in command-line options
[LIBCONFIG] (root): 19/19 parameters successfully set, (16 to default value)
[LIBCONFIG] (root): 4/4 parameters successfully set, (4 to default value)
Getting ENBSParams
[LIBCONFIG] (root): 3/3 parameters successfully set, (1 to default value)
[LIBCONFIG] THREAD_STRUCT.[0]: 2/2 parameters successfully set, (0 to default value)
[LIBCONFIG] THREAD_STRUCT.[0]: 2/2 parameters successfully set, (0 to default value)
Configuration: nb_rrc_inst 1, nb_L1_inst 1, nb_ru 1
[LIBCONFIG] loader: 2/2 parameters successfully set, (2 to default value)
[LIBCONFIG] loader.NB_IoT: 2/2 parameters successfully set, (1 to default value)
[LOADER] library libNB_IoT.so is not loaded: libNB_IoT.so: cannot open shared object file: No such file or directory
               nb_nbiot_rrc_inst 0, nb_nbiot_L1_inst 0, nb_nbiot_macrlc_inst 0
[LIBCONFIG] TTracer: 4/4 parameters successfully set, (4 to default value)
configuring for RAU/RRU
```

```
CPU Freq is 1.992111
ITTI init, useMME: 0
[TMR]   Starting itti queue: TASK_UNKNOWN as task 0
[TMR]   Starting itti queue: TASK_TIMER as task 1
[TMR]   Starting itti queue: TASK_L2L1 as task 2
[TMR]   Starting itti queue: TASK_BM as task 3
[TMR]   Starting itti queue: TASK_PHY_ENB as task 4
[TMR]   Starting itti queue: TASK_MAC_ENB as task 5
[TMR]   Starting itti queue: TASK_RLC_ENB as task 6
[TMR]   Starting itti queue: TASK_RRC_ENB_NB_IoT as task 7
[TMR]   Starting itti queue: TASK_PDCP_ENB as task 8
[TMR]   Starting itti queue: TASK_RRC_ENB as task 9
[TMR]   Starting itti queue: TASK_RAL_ENB as task 10
[TMR]   Starting itti queue: TASK_S1AP as task 11
[TMR]   Starting itti queue: TASK_X2AP as task 12
[TMR]   Starting itti queue: TASK_SCTP as task 13
[TMR]   Starting itti queue: TASK_ENB_APP as task 14
[TMR]   Starting itti queue: TASK_FLEXRAN_AGENT as task 15
[TMR]   Starting itti queue: TASK_PHY_UE as task 16
[TMR]   Starting itti queue: TASK_MAC_UE as task 17
[TMR]   Starting itti queue: TASK_RLC_UE as task 18
[TMR]   Starting itti queue: TASK_PDCP_UE as task 19
[TMR]   Starting itti queue: TASK_RRC_UE as task 20
[TMR]   Starting itti queue: TASK_NAS_UE as task 21
[TMR]   Starting itti queue: TASK_RAL_UE as task 22
[TMR]   Starting itti queue: TASK_MSC as task 23
[TMR]   Starting itti queue: TASK_GTPV1_U as task 24
[TMR]   Starting itti queue: TASK_UDP as task 25
[LIBCONFIG] opt: 3/3 parameters successfully set, (3 to default value)
[OPT]   OPT disabled
PDCP netlink
[NETLINK]Opened socket with fd 56
reported resolution = 1 ns
[HW]   Version: Branch: HEAD Abrev. Hash: 69e4901e4 Date: Fri Jun 7 14:45:18 2019 +0200
Runtime table
NO deadline scheduler
[HW]   CPU Affinity of main() function is...  CPU_0  CPU_1  CPU_2  CPU_3
[TMR]   Created Posix thread TASK_ENB_APP
[PHY]   eNB_app_task() Task ready initialise structures
[PHY]   RC.eNB = 0x7f7778000b20
[LIBCONFIG] L1s.[0]: 9/9 parameters successfully set, (7 to default value)
[PHY]   RC.eNB[0] = 0x7f7778000bd0
[PHY]   RC.eNB[0][0] = 0x7f777f0e8010
[ENB_APP]   Initializing northbound interface for L1
[PHY]   l1_north_init_eNB() RC.nb_L1_inst:1
[PHY]   l1_north_init_eNB() RC.nb_L1_CC[0]:1
[PHY]   l1_north_init_eNB() RC.eNB[0][0] installing callbacks
[RRC]   Creating RRC eNB Task
[LIBCONFIG] MACRLCs.[0]: 21/21 parameters successfully set, (15 to default value)
[MAC]   [MAIN] Init function start:nb_macrlc_inst=1
[RRC]   Entering main loop of RRC message task
[TMR]   Created Posix thread TASK_RRC_ENB
ITTI tasks created
[LIBCONFIG] (root): 3/3 parameters successfully set, (1 to default value)
[LIBCONFIG] NETWORK_CONTROLLER: 6/6 parameters successfully set, (0 to default value)
[FLEXRAN_AGENT]   FlexRAN Agent for eNB 0 is DISABLED
[PDCP]   PDCP layer has been initialized
[ENB_APP]   sched mode = default 0 [default]
[PHY]   eNB_app_task() RC.nb_L1_inst:1
[PHY]   l1_north_init_eNB() RC.nb_L1_inst:1
[PHY]   l1_north_init_eNB() RC.nb_L1_CC[0]:1
[PHY]   l1_north_init_eNB() RC.eNB[0][0] installing callbacks
[ENB_APP]   Allocating eNB_RRC_INST for 1 instances
[PHY]   eNB_app_task() RC.nb_inst:1 RC.rrc:0x7f7778002ea0
[PHY]   eNB_app_task() Creating RRC instance RC.rrc[0]:0x7f7778002ec0 (1 of 1)
[LIBCONFIG] (root): 3/3 parameters successfully set, (1 to default value)
[LIBCONFIG] eNBs.[0]: 14/14 parameters successfully set, (7 to default value)
[ENB_APP]   RRC 0: Southbound Transport local_mac
[LIBCONFIG] eNBs.[0].plmn_list.[0]: 3/3 parameters successfully set, (0 to default value)
[RRC]   num component carriers 1
[RRC]   enb_config::RCconfig_RRC() parameter number: 0, total number of parameters: 107, ccspath: eNBs.[0].component_carriers.[0]
[LIBCONFIG] eNBs.[0].component_carriers.[0]: 107/107 parameters successfully set, (21 to default value)
phich.resource 0 (ONESIXTH), phich.duration 0 (NORMAL)
[LIBCONFIG] eNBs.[0].srb1_parameters: 6/6 parameters successfully set, (0 to default value)
[ENB_APP]   Sending configuration message to RRC task
[LIBCONFIG] (root): 3/3 parameters successfully set, (1 to default value)
[RRC]   Received message RRC_CONFIGURATION_REQ
[RRC]   [eNB 0] Received RRC_CONFIGURATION_REQ : 0x7f77780090ac
[LIBCONFIG] eNBs.[0]: 14/14 parameters successfully set, (7 to default value)
[LIBCONFIG] eNBs.[0].plmn_list.[0]: 3/3 parameters successfully set, (0 to default value)
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] Init...
[LIBCONFIG] eNBs.[0].component_carriers.[0]: 107/107 parameters successfully set, (21 to default value)
[LIBCONFIG] list eNBs.[0].target_enb_x2_ip_address not found in config file /home/user/Desktop/Eurecom_OAI/enb.band7.tm1.25PRB.usrpb210.conf
[LIBCONFIG] eNBs.[0].NETWORK_INTERFACES: 7/7 parameters successfully set, (0 to default value)
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] Checking release
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] Rel14 RRC detected, MBMS flag 0
[RRC]   Configuring MIB (N_RB_DL 25,phich_Resource 0,phich_Duration 0)
[RRC]   [MIB] systemBandwidth 2, phich_duration 0, phich_resource 0, sfn 0
[RRC]   [eNB 0] Configuration SIB2/3, MBMS = 0
[RRC]   [SIB2] With ITTI. Basic config of paging cycle DRX: radio frame cycle length 2, paging occasion number 2
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] Contents of SIB18 1/1
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB18 rxPool_sc_CP_Len: 0
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB18 sc_Period_r12: 0
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB18 data_CP_Len_r12: 0
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB18 prb_Num_r12: 20
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB18 prb_Start_r12: 5
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB18 prb_End_r12: 44
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB18 offsetIndicator: 0
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB18 subframeBitmap_choice_bs_buf: 00000000000000000000
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] Contents of SIB19 1/1
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB19 cp_Len_r12: 0
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB19 discPeriod_r12: 0
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB19 numRetx_r12: 1
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB19 numRepetition_r12: 2
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB19 tf_ResourceConfig_r12 prb_Num_r12: 5
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB19 tf_ResourceConfig_r12 prb_Start_r12: 3
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB19 tf_ResourceConfig_r12 prb_End_r12: 21
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB19 tf_ResourceConfig_r12 offsetIndicator: 0
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] SIB19 tf_ResourceConfig_r12 subframeBitmap_choice_bs_buf: f0ffffffff
[MAC]   Configuring MIB for instance 0, CCid 0 : (band 7,N_RB_DL 25,Nid_cell 0,p 1,DL freq 2680000000,phich_config.resource 0, phich_config.d
[MAC]   config_mib() NFAPI_CONFIG_REQUEST(num_tlv:16) DL_BW:25 UL_BW:25 Ncp 0,p_eNB 1,earfcn 3350,band 7,phich_resource 0 phich_duration 0 ph
[MAC]   [CONFIG]SIB2/3 Contents (partial)
[MAC]   [CONFIG]pusch_config_common.n_SB = 1
[MAC]   [CONFIG]pusch_config_common.hoppingMode = 0
[MAC]   [CONFIG]pusch_config_common.pusch_HoppingOffset = 0
[MAC]   [CONFIG]pusch_config_common.enable64QAM = 0
[MAC]   [CONFIG]pusch_config_common.groupHoppingEnabled = 1
[MAC]   [CONFIG]pusch_config_common.groupAssignmentPUSCH = 0
[MAC]   [CONFIG]pusch_config_common.sequenceHoppingEnabled = 0
[MAC]   [CONFIG]pusch_config_common.cyclicShift  = 1
[PHY]   Configuring MIB for instance 0, CCid 0 : (band 7,N_RB_DL 25, N_RB_UL 25, Nid_cell 0,eNB_tx_antenna_ports 1,Ncp 0,DL freq 3350,phich_c
[PHY]   Initializing frame parms for N_RB_DL 25, Ncp 0, osf 1
[PHY]   lte_parms.c: Setting N_RB_DL to 25, ofdm_symbol_size 512
[LIBCONFIG] loader.coding: 2/2 parameters successfully set, (1 to default value)
[LOADER] library libcoding.so successfully loaded
XFORMS
[PHY]   prach_config_common.rootSequenceIndex = 0
[PHY]   prach_config_common.prach_ConfigInfo.prach_ConfigIndex = 0
[PHY]   prach_config_common.prach_ConfigInfo.highSpeedFlag = 0
[PHY]   prach_config_common.prach_ConfigInfo.zeroCorrelationZoneConfig = 1
[PHY]   prach_config_common.prach_ConfigInfo.prach_FreqOffset = 2
[PHY]   pusch_config_common.n_SB = 1
[PHY]   pusch_config_common.hoppingMode = 0
[PHY]   pusch_config_common.pusch_HoppingOffset = 0
[PHY]   pusch_config_common.enable64QAM = 0
[PHY]   pusch_config_common.ul_ReferenceSignalsPUSCH.groupHoppingEnabled = 1
[PHY]   pusch_config_common.ul_ReferenceSignalsPUSCH.groupAssignmentPUSCH = 0
[PHY]   pusch_config_common.ul_ReferenceSignalsPUSCH.sequenceHoppingEnabled = 0
[PHY]   pusch_config_common.ul_ReferenceSignalsPUSCH.cyclicShift = 2
[PHY]   eNB 0/0 configured
[RRC]   [eNB] handover active state is 0
[RRC]   [eNB] eMBMS active state is 0
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 0] ENB:OPENAIR RRC IN....
NFAPI MODE:MONOLITHIC
START MAIN THREADS
RC.nb_L1_inst:1
Initializing eNB threads single_thread_flag:0 wait_for_sync:0
[PHY]   [lte-softmodem.c] eNB structure about to allocated RC.nb_L1_inst:1 RC.nb_L1_CC[0]:1
[PHY]   [lte-softmodem.c] eNB structure RC.eNB allocated
[PHY]   Initializing eNB 0 CC_id 0 single_thread_flag:0
[PHY]   Initializing eNB 0 CC_id 0
[PHY]   Registering with MAC interface module
[PHY]   Setting indication lists
[PHY]   [lte-softmodem.c] eNB structure allocated
wait_eNBs()
Waiting for eNB L1 instances to all get configured ... sleeping 50ms (nb_L1_inst 1)
RC.nb_L1_CC[0]:1
eNB L1 are configured
About to Init RU threads RC.nb_RU:1
Initializing RU threads
configuring RU from file
[LIBCONFIG] RUs.[0]: 20/20 parameters successfully set, (10 to default value)
Set RU mask to 1
Creating RC.ru[0]:0x5582c07d8140
Setting function for RU 0 to eNodeB_3GPP
[PHY]   number of L1 instances 1, number of RU 1, number of CPU cores 4
[PHY]   DJP - delete code above this /home/user/Desktop/git/openairinterface5g/targets/RT/USER/lte-ru.c:2781
[PHY]   Copying frame parms from eNB 0 to ru 0
[PHY]   Initializing RRU descriptor 0 : (local RF,synch_to_ext_device,0)
configuring ru_id 0 (start_rf 0x5582be3ade00)
[PHY]   Starting ru_thread 0
[PHY]   Initializing RU proc 0 (eNodeB_3GPP,synch_to_ext_device),
[PHY]   init_RU_proc() DJP - added creation of pthread_prach
[HW]   [SCHED][eNB] ru_thread started on CPU 3, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
[HW]   [SCHED][eNB] ru_thread_prach started on CPU 2, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
[PHY]   thread ru created id=12711
[PHY]   Starting RU 0 (eNodeB_3GPP,synch_to_ext_device),
channel 0, Setting tx_gain offset 0.000000, rx_gain offset 117.000000, tx_freq 2680000000.000000, rx_freq 2560000000.000000
[PHY]   Initializing frame parms for N_RB_DL 25, Ncp 0, osf 1
[PHY]   lte_parms.c: Setting N_RB_DL to 25, ofdm_symbol_size 512
[PHY]   Initializing RU signal buffers (if_south local RF) nb_tx 1
[HW]   [SCHED][eNB] fep_thread started on CPU 1, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
[PHY]   [INIT] common.txdata[0] = 0x7f778387f040 (307200 bytes)
[PHY]   thread fep created id=12713
wait RUs
[ENB_APP]   Waiting for RUs to be configured ... RC.ru_mask:01
[HW]   [SCHED][eNB] feptx_thread started on CPU 1, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
[PHY]   thread feptx created id=12714
[PHY]   nb_tx 1
[PHY]   rxdata_7_5kHz[0] 0x7f77740518e0 for RU 0
[PHY]   [INIT] common.txdata_BF= 0x7f777406f960 (8 bytes)
[PHY]   txdataF_BF[0] 0x7f777406f9e0 for RU 0
[PHY]   rxdataF[0] 0x7f7774076ae0 for RU 0
[LIBCONFIG] loader.oai_device: 2/2 parameters successfully set, (1 to default value)
[LOADER] library liboai_device.so successfully loaded
[PHY]   Checking for USRPs : UHD 3.14.1.HEAD-0-gbfb9c1c7 (3.14.1)
[INFO] [UHD] linux; GNU C++ version 7.4.0; Boost_106501; UHD_3.14.1.HEAD-0-gbfb9c1c7
[HW]   Found USRP x300
Found USRP x300
[INFO] [X300] X300 initialization sequence...
[INFO] [X300] Maximum frame size: 8000 bytes.
[INFO] [X300] Radio 1x clock: 184.32 MHz
[INFO] [GPS] Found an internal GPSDO: LC_XO, Firmware Rev 0.929a
[PHY]   ru_thread_prach() RACH waiting for RU to be configured
[INFO] [0/DmaFIFO_0] Initializing block control (NOC ID: 0xF1F0D00000000000)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1318 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1320 MB/s)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000000001)
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000000001)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000000)
[PHY]   ru_thread_prach() RACH waiting for RU to be configured
[PHY]   ru_thread_prach() RACH waiting for RU to be configured
[PHY]   device_init() sample_rate:7680000
[PHY]   cal 0: freq 3500000000.000000, offset 77.000000, diff 940000000.000000
[PHY]   cal 1: freq 2660000000.000000, offset 81.000000, diff 100000000.000000
[PHY]   cal 2: freq 2300000000.000000, offset 81.000000, diff 260000000.000000
[PHY]   cal 3: freq 1880000000.000000, offset 82.000000, diff 680000000.000000
```

```
[PHY]   cal 4: freq 816000000.000000, offset 85.000000, diff 1744000000.000000
[PHY]   RX Gain 0 117.000000 (81.000000) => 36.000000 (max 37.500000)
[PHY]   USRP TX_GAIN:31.50 gain_range:31.50 tx_gain:0.00
[PHY]   Actual master clock: 184.320000MHz...
[PHY]   ru_thread_prach() RACH waiting for RU to be configured
[PHY]   RF board max packet size 1996, size for 100µs jitter 768
[PHY]   rx_max_num_samps 768
[PHY]   RX Channel 0
[PHY]     Actual RX sample rate: 7.680000MSps...
[PHY]     Actual RX frequency: 2.560000GHz...
[PHY]     Actual RX gain: 36.000000...
[PHY]     Actual RX bandwidth: 5.000000M...
[PHY]     Actual RX antenna: RX2...
[PHY]   TX Channel 0
[PHY]     Actual TX sample rate: 7.680000MSps...
[PHY]     Actual TX frequency: 2.680000GHz...
[PHY]     Actual TX gain: 31.500000...
[PHY]     Actual TX bandwidth: 5.000000M...
[PHY]     Actual TX antenna: TX/RX...
[PHY]   Device timestamp: 2.892506...
[RAU] has loaded USRP X300 device.
setup_RU_buffers: frame_parms = 0x5582c07d81d0
[PHY]   Signaling main thread that RU 0 is ready
waiting for sync (ru_thread,-1/0x5582be9b3348,0x5582bef6faa0,0x5582bee25c00)
RC.ru_mask:00
[PHY]   RUs configured
ALL RUs READY!
RC.nb_RU:1
ALL RUs ready - init eNBs
Not NFAPI mode - call init_eNB_afterRU()
[PHY]   init_eNB_afterRU() RC.nb_inst:1
[PHY]   RC.nb_CC[inst]:1
[PHY]   RC.nb_CC[inst:0][CC_id:0]:0x7f777f0e8010
[PHY]   [eNB 0] phy_init_lte_eNB() About to wait for eNB to be configured[PHY]   [eNB 0] Initializing DL_FRAME_PARMS : N_RB_DL 25, PHICH Reso
pcfich_reg : 0,12,25,37
[PHY]   Mapping RX ports from 1 RUs to eNB 0
[PHY]   Overwriting eNB->prach_vars.rxsigF[0]:0x5582c1304360
[PHY]   Overwriting eNB->prach_vars_br.rxsigF.rxsigF[0]:(nil)
[PHY]   Overwriting eNB->prach_vars_br.rxsigF.rxsigF[0]:(nil)
[PHY]   Overwriting eNB->prach_vars_br.rxsigF.rxsigF[0]:(nil)
[PHY]   Overwriting eNB->prach_vars_br.rxsigF.rxsigF[0]:(nil)
[PHY]   eNB->num_RU:1
[PHY]   Attaching RU 0 antenna 0 to eNB antenna 0
[PHY]   init_eNB_afterRU() ************* DJP ***** eNB->frame_parms.nb_antennas_tx:0 - GOING TO HARD CODE TO 1[PHY]   inst 0, CC_id 0 : nb_an
[PHY]   Initialise transport
[PHY]   init_eNB_proc(inst:0) RC.nb_CC[inst]:1
[PHY]   Initializing eNB processes instance:0 CC_id 0
[PHY]   Creating te_thread 0
[PHY]   Creating te_thread 1
[HW]   [SCHED][eNB] te_thread started on CPU 1, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
[PHY]   Creating te_thread 2
[HW]   [SCHED][eNB] te_thread started on CPU 2, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
[PHY]   thread te created id=12731
[PHY]   thread te created id=12732
[HW]   [SCHED][eNB] te_thread started on CPU 3, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
[PHY]   eNB->single_thread_flag:0
[HW]   [SCHED][eNB] td_thread started on CPU 0, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
[PHY]   thread te created id=12733
[PHY]   thread td created id=12734
[HW]   [SCHED][eNB] eNB_thread_prach started on CPU 1, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
ALL RUs ready - ALL eNBs ready
[HW]   [SCHED][eNB] eNB_thread_prach_br started on CPU 3, sched_policy = SCHED_FIFO , priority = 99, CPU Affinity= CPU_0 CPU_1 CPU_2 CPU_3
[PHY]   ru_thread_prach() RACH waiting for RU to be configured
[PHY]   ru_thread_prach() RU configured - RACH processing thread running
Sending sync to all threads
TYPE <CTRL-C> TO TERMINATE
Entering ITTI signals handler
got sync (ru_thread)
[PHY]   Time in secs now: 32110804
[PHY]   Time in secs last pps: 26746224
[PHY]   RU 0 rf device ready
[PHY]   RU 0 no asynch_south interface
[MAC]   SCHED_MODE=0
[PHY]   prach_I0 = 0.3 dB
[PHY]   max_I0 29, min_I0 24
[PHY]   prach_I0 = 2.4 dB
[PHY]   max_I0 27, min_I0 22
```

Shown below is a screenshot of the terminal window with the output listed.



```
[PHY]   RC.nb_CC[inst:0][CC_id:0]:0x7f777f0e8010
[PHY]   [eNB 0] phy_init_lte_eNB() About to wait for eNB to be configured[PHY]   [eNB 0] Initializing
nb_antennas_tx:0 nb_antennas_rx:0 nb_antenna_ports_eNB:1 PRACH[rootSequenceIndex:0 prach_Config_enable
fset:2]
pcfich_reg : 0,12,25,37
[PHY]   Mapping RX ports from 1 RUs to eNB 0
[PHY]   Overwriting eNB->prach_vars.rxsigF[0]:0x5582c1304360
[PHY]   Overwriting eNB->prach_vars_br.rxsigF.rxsigF[0]:(nil)
[PHY]   Overwriting eNB->prach_vars_br.rxsigF.rxsigF[0]:(nil)
[PHY]   Overwriting eNB->prach_vars_br.rxsigF.rxsigF[0]:(nil)
[PHY]   Overwriting eNB->prach_vars_br.rxsigF.rxsigF[0]:(nil)
[PHY]   eNB->num_RU:1
[PHY]   Attaching RU 0 antenna 0 to eNB antenna 0
[PHY]   init_eNB_afterRU() ************* DJP ***** eNB->frame_parms.nb_antennas_tx:0 - GOING TO HARD C
[PHY]   Initialise transport
[PHY]   init_eNB_proc(inst:0) RC.nb_CC[inst]:1
[PHY]   Initializing eNB processes instance:0 CC_id 0
[PHY]   Creating te_thread 0
[PHY]   Creating te_thread 1
[HW]   [SCHED][eNB] te_thread started on CPU 1, sched_policy = SCHED_FIFO , priority = 99, CPU Affinit
[PHY]   Creating te_thread 2
[HW]   [SCHED][eNB] te_thread started on CPU 2, sched_policy = SCHED_FIFO , priority = 99, CPU Affinit
[PHY]   thread te created id=12731
[PHY]   thread te created id=12732
[HW]   [SCHED][eNB] te_thread started on CPU 3, sched_policy = SCHED_FIFO , priority = 99, CPU Affinit
[PHY]   eNB->single_thread_flag:0
[HW]   [SCHED][eNB] td_thread started on CPU 0, sched_policy = SCHED_FIFO , priority = 99, CPU Affinit
[PHY]   thread te created id=12733
[PHY]   thread td created id=12734
[HW]   [SCHED][eNB] eNB_thread_prach started on CPU 1, sched_policy = SCHED_FIFO , priority = 99, CPU
ALL RUs ready - ALL eNBs ready
[HW]   [SCHED][eNB] eNB_thread_prach_br started on CPU 3, sched_policy = SCHED_FIFO , priority = 99, C
[PHY]   ru_thread_prach() RACH waiting for RU to be configured
[PHY]   ru_thread_prach() RU configured - RACH processing thread running
Sending sync to all threads
TYPE <CTRL-C> TO TERMINATE
Entering ITTI signals handler
got sync (ru_thread)
[PHY]   Time in secs now: 32110804
[PHY]   Time in secs last pps: 26746224
[PHY]   RU 0 rf device ready
[PHY]   RU 0 no asynch_south interface
[MAC]   SCHED_MODE=0
[PHY]   prach_I0 = 0.3 dB
[PHY]   max_I0 29, min_I0 24
[PHY]   prach_I0 = 2.4 dB
[PHY]   max_I0 27, min_I0 22
```

The T tracer is a framework to debug and monitor the OAI eNB software. It provides logging, timinig analysis, and signal visualization capabilities, and can be used with Wireshark for MAC (PDU) analysis. It consists of two main parts:

- an events collector, integrated to the real-time processing
- a separate set of programs to receive, record, display, replay and analyze the events sent by the collector

The eNB side of the T tracer is automatically built when the OAI eNB software is built with the `build_oai --eNB` command.

The GUI side of the T tracer is built separately with the commands listed below.

```
cd $OPENAIR_HOME/common/utils/T/tracer
make
```

The binary for the GUI side of the T tracer will be `openairinterface5g/common/utils/T/tracer/enb`.

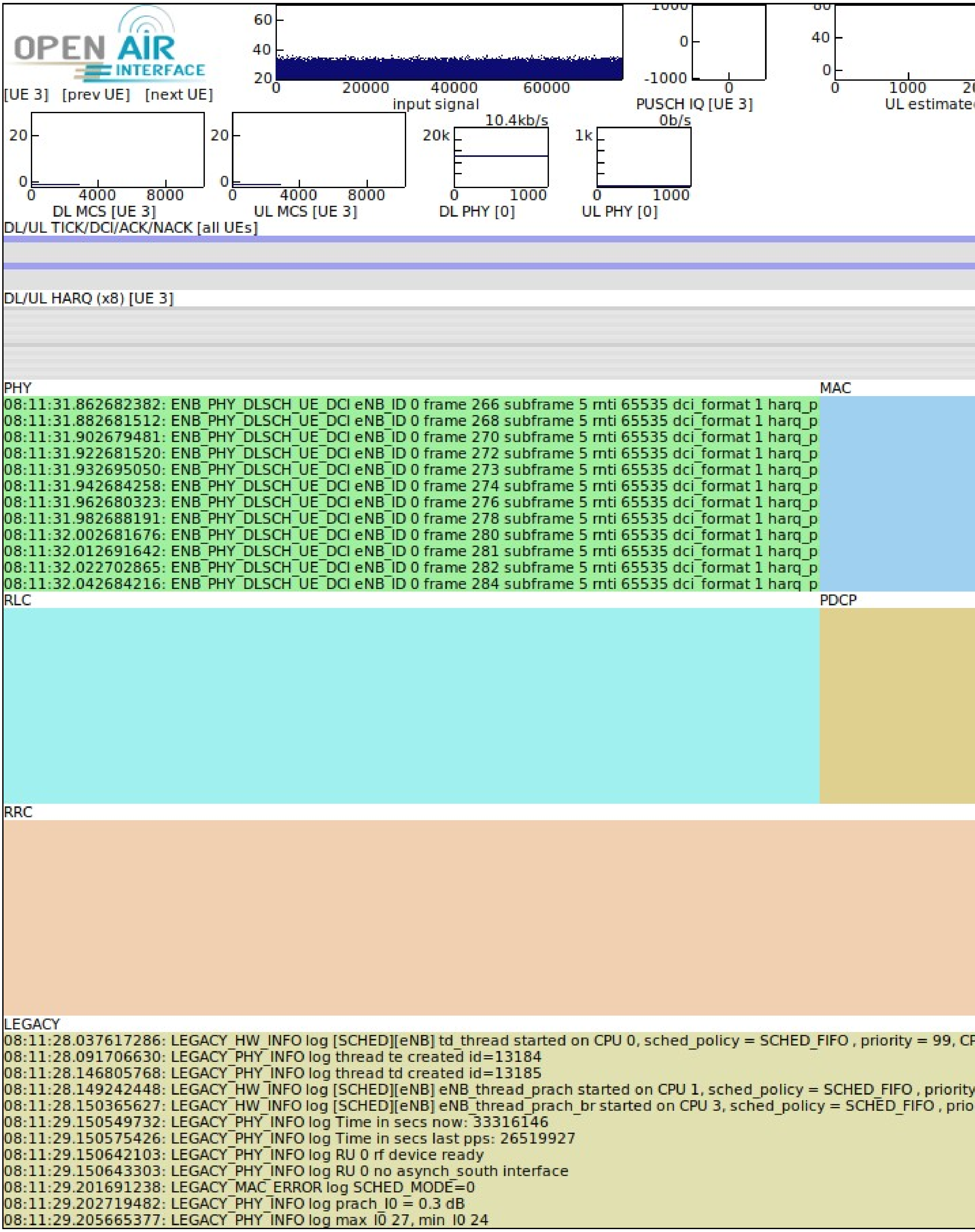
To use the T tracer, first run the eNB software with the `--T_stdout 0` command-line option, and then run the T tracer, and specify a log file, as shown in the commands listed below.

```
   sudo -E $OPENAIR_HOME/targets/bin/lte-softmodem-nos1.Rel14 -O $OPENAIR_HOME/ci-scripts/conf_files/enb.band7.tm1.25PRB.usrpb210.conf --noS1
   $OPENAIR_HOME/common/utils/T/tracer/enb -d ../T_tracer_messages.txt
```

The GUI for the T tracer will appear, from which you can observe and control the monitoring of the eNB software.

More information about the T tracer can be found at the OAI repository here and here, and a screenshot can be found here.

A screenshot of the T tracer running on the USRP 2974 is shown below. Note that there is minimal activity shown, as only the eNB software itself is running, without any emulated UE or real UE handset.



PHY

```
08:11:31.862682382: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 266 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:31.882681512: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 268 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:31.902679481: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 270 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:31.922681520: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 272 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:31.932695050: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 273 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:31.942684258: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 274 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:31.962680323: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 276 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:31.982688191: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 278 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:32.002681676: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 280 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:32.012691642: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 281 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:32.022702865: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 282 subframe 5 rnti 65535 dci_format 1 harq_p
08:11:32.042684216: ENB_PHY_DLSCH_UE_DCI eNB_ID 0 frame 284 subframe 5 rnti 65535 dci_format 1 harq_p
```

LEGACY

```
08:11:28.037617286: LEGACY_HW_INFO log [SCHED][eNB] td_thread started on CPU 0, sched_policy = SCHED_FIFO , priority = 99, CP
08:11:28.091706630: LEGACY_PHY_INFO log thread te created id=13184
08:11:28.146805768: LEGACY_PHY_INFO log thread td created id=13185
08:11:28.149242448: LEGACY_HW_INFO log [SCHED][eNB] eNB_thread_prach started on CPU 1, sched_policy = SCHED_FIFO , priority
08:11:28.150365627: LEGACY_HW_INFO log [SCHED][eNB] eNB_thread_prach_br started on CPU 3, sched_policy = SCHED_FIFO , prio
08:11:29.150549732: LEGACY_PHY_INFO log Time in secs now: 33316146
08:11:29.150575426: LEGACY_PHY_INFO log Time in secs last pps: 26519927
08:11:29.150642103: LEGACY_PHY_INFO log RU 0 rf device ready
08:11:29.150643303: LEGACY_PHY_INFO log RU 0 no asynch_south interface
08:11:29.201691238: LEGACY_MAC_ERROR log SCHED_MODE=0
08:11:29.202719482: LEGACY_PHY_INFO log prach_I0 = 0.3 dB
08:11:29.205665377: LEGACY_PHY_INFO log max_I0 27, min_I0 24
```

In a future document, we will show how to configure OAI eNB software to communicate over-the-air with a real UE handset, as well as how to configure OAI eNB software to work with the OAI UE emulator software.

General technical support and answers to questions can be provided by mailing lists. They are a very good resource for help. There are two relevant mailing lists.

The **usrp-users** mailing list is for USRP-specific and UHD-specific questions, problems, and issues, and can be found here.

There are several OAI mailing lists, which can be found here. The **openair5g-user** list is probably the most appropriate one for users of the OAI eNB software.

An listing of relevant mailing lists can be found on the Ettus Research Knowledge Base here.

If you have any questions, comments, or feedback about this document, then please send us an email at `support@ettus.com`, and we will respond to you promptly.