Overview

- Overview and Ecosystem
- Use cases
- Features
- Hardware targets
- Installation & Usage
- Debugging tools
What is OpenAirInterface?

- **Open-source software-based implementation of 3GPP Technologies**
  - Starting at LTE (Rel 8), including features from LTE-Advanced (Rel 10/11/12), LTE-Advanced-Pro (Rel 13/14), going on to 5G Rel (15/16/…)
  - Spanning the full protocol stack of 3GPP standard
    - E-UTRAN (eNB, UE)
    - EPC (MME, S+P-GW, HSS)
  - Realtime RF and scalable emulation platforms
  - Works with many SDR platforms (ExpressMIMO2, USRP, LimeSDR, …)

- **Makes it is feasible to put a fully-compliant 4G eNodeB and EPC in a commodity x86-based computer (or data center)**

- **Objectives**
  - Building a community of individual developers, academics and major industrials embracing open-source for 5G
  - Become a strong voice and maybe a game-changer in the 3GPP world
    - Real impact from “the little guys” on 3GPP systems
Collaborative Web Tools

- Main page: https://www.openairinterface.org

- Code available from
  - RAN (eNB + UE) https://gitlab.eurecom.fr/oai/openairinterface5g
  - EPC https://github.com/OPENAIRINTERFACE/openair-cn

- Mailing lists
  - https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/MailingList

- Forum in Chinese
  - http://bbs.opensource5g.org/forum.php

- Other tools:
  - https://openairinterface.slack.com
  - https://trello.com/oaidev
The OpenAirInterface Software Alliance

- Launched in 2014 as an endowment fund (French “Fonds de Dotation”)
- Current strategic members (Orange, TCL, Nokia Bell Labs, Fujitsu)
- Many associate members (Samsung, Interdigital, ng4t, Cisco, B–COM, INRIA, IMT, TNO, III, Rutgers WINLAB, U. Washington, IITH, BUPT, etc.)
- Goals:
  - Promote OpenAirInterface and its open-source licensing model
  - Support the community of developers and users
The OAI Licensing model

- **FRAND License** is based on Apache V2.0 but allows committing software with patent rights into OSA and still keep licensing rights → Inline with 3GPP fair use licensing policy

- We work closely with ETSI on implications of open-source for licensing/certification
Use case I: classical 3GPP network

- OAI EPC
- Commercial/3rd party EPC
- OAI eNB
- Commercial/3rd party eNB
- OAI UE
- COTS UE
Use case II: simplified network

- Non-3GPP setup (no-S1 mode):
  - OAI eNB <-- OAI UE
Use case III: cloud-RAN

Main target of EURECOM deployment
OpenAirInterface eNB features (PHY)

- The Physical layer implements 3GPP 36.211, 36.212, 36.213 and provides the following features:
  - LTE release 8.6 compliant, and implements a subset of release 10;
  - FDD and TDD configurations 1 (experimental) and 3;
  - Bandwidth: 5, 10, and 20 MHz;
  - Transmission modes: 1, 2 (stable), 3, 4, 5, 6, 7 (experimental);
  - Max number of antennas: 2
  - CQI/PMI reporting: aperiodic, feedback mode 3-0 and 3-1;
  - PRACH preamble format 0
  - All downlink (DL) channels are supported: PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH, PMCH;
  - All uplink (UL) channels are supported: PRACH, PUSCH, PUCCH (format 1/1a/1b), SRS, DRS;
  - HARQ support (UL and DL);
  - Highly optimized base band processing (including turbo decoder).
  - Expected throughputs DL
    - 5 MHz, 25 PRBS/ MCS 28 = 16-17 Mbit/s (measured with COTS UE Cat 3/4)
    - 10 MHz, 50 PRBS/MCS 28 = 34-35 Mbit/s (measured with COTS UE Cat 3/4)
    - 20 MHz, 100 PRBS/MCS 28 = ~70 Mbit/s (measured with COTS UE Cat 3/4)
  - Expected throughputs UL
    - 5 MHz, 20 PRBs / MCS 20 = 9 Mbit/s (measured with COTS UE Cat 3/4)
    - 10 MHz, 45 PRBs / MCS 20 = 17 Mbit/s (measured with COTS UE Cat 3/4)
    - 20 MHz, 96 PRBs / MCS 20 = ~35 Mbit/s (measured with COTS UE Cat 3/4)
OpenAirInterface eNB features (MAC)

- The MAC layer implements a subset of the 3GPP 36-321 release 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
  - HARQ Support
  - RA procedures and RNTI management
  - RLC interface (AM, UM)
  - UL power control
  - Link adaptation
OpenAirInterface eNB features (PDCP)

- The current PDCP is header compliant with 3GPP 36-323 Rel 10.1.0 and implement 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
OpenAirInterface eNB features (RLC)

- The RLC layer implements a full specification of the 3GPP 36-322 release v9.3

- **RLC TM (mainly used for BCCH and CCCH)**
  - Neither segment nor concatenate RLC SDUs
  - Do not include a RLC header in the RLC PDU
  - Delivery of received RLC PDUs to upper layers

- **RLC UM (mainly used for DTCH)**
  - Segment or concatenate RLC SDUs according to the TB size selected by MAC
  - Include a RLC header in the RLC PDU
  - Duplication detection
  - PDU reordering and reassembly

- **RLC AM, compatible with 9.3**
  - Segmentation, re-segmentation, concatenation, and reassembly
  - Padding
  - Data transfer to the user
  - RLC PDU retransmission in support of error control and correction
  - Generation of data/control PDUs
OpenAirInterface eNB features (RRC)

- Based on 3GPP 36.331 v14.3.0.
  - System Information broadcast (SIB 1, 2, 3, and 13)
  - RRC connection establishment
  - RRC connection reconfiguration (addition and removal of radio bearers, connection release)
  - RRC connection release
  - RRC connection re-establishment
  - inter-frequency measurement collection and reporting (experimental)
  - eMBMS for multicast and broadcast (experimental)
  - Handover (experimental)
  - Paging (soon)
Experimental/upcoming eNB features

- X2 handover
- Rel 10 carrier aggregation
  - 2DL carriers, 1 UL carrier
  - In old branch, to be reactivated
- Rel 10 Transmission modes 3/4/7/8/9
- Rel 13 LTE–M
- Rel 14 NB–IoT
- Rel 14 D2D
- Rel 15 5G–NR
**eNB Functional Splits**

- **IF4.5/IF5**: similar to IEEE P1914.1
- **FAPI (IF2)**: specified by small cell forum, implementation (open-nFAPI) by CISCO
- **IF1 (F1 in 3GPP Rel 15)**: under development
eNB Functional Split Architecture
HARDWARE TARGETS
Hardware Requirements

- **SDR platform**
  - ExpressMIMO2 (discontinued)
  - USRP B200, X300, N300 (recommended)
  - Blade RF
  - LMS-SDR
  - Epiq Sidekiq (experimental)
  - Skylark Iris (experimental)
  - Syrtem (experimental)

- **Host PC**
  - A powerful x86 PC (recommended)
    - Intel Core i5, i7, i9
    - Intel Xeon
    - Intel Atom
    - 4 cores, > 3GHz, SSE 4, AVX
  - Low-cost x86 PC
    - Up board (up2), Euclid board
  - ARM (experimental)
    - Odroid

- **Antennas, Duplexers, etc**
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>USRP B210</th>
<th>USRP X310</th>
<th>USRP N310</th>
<th>Blade RF</th>
<th>LMS SDR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data acquisition</strong></td>
<td>USB3</td>
<td>Gbit EtherNet, PCIexpress</td>
<td>Gbit EtherNet</td>
<td>USB3</td>
<td>USB3</td>
</tr>
<tr>
<td><strong>MIMO and bandwidth capabilities</strong></td>
<td>2x1 MIMO 20MHz or 2x2 MIMO 10MHz</td>
<td>2x2 MIMO, 120MHz</td>
<td>4x4 MIMO 100MHz</td>
<td>1x1 SISO 20MHz</td>
<td>2x2 MIMO 20MHz</td>
</tr>
<tr>
<td><strong>RF chip</strong></td>
<td>AD9361</td>
<td>n/a**</td>
<td>AD9371 (x2)</td>
<td>LMS6002D</td>
<td>LMS7002M</td>
</tr>
<tr>
<td><strong>Frequency range</strong></td>
<td>70MHz - 6GHz</td>
<td>DC-6GHz (depends on daughterboard)</td>
<td>10 MHz - 6GHz</td>
<td>300 MHz - 3.8GHz</td>
<td>300 MHz - 3.8GHz</td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>€1,130</td>
<td>~€5,000</td>
<td>~€10,000</td>
<td>$420</td>
<td>$299</td>
</tr>
<tr>
<td><strong>Duplexing</strong></td>
<td>FDD* or TDD*</td>
<td>FDD* or TDD*</td>
<td>FDD* or TDD</td>
<td>FDD*</td>
<td>FDD* or TDD*</td>
</tr>
<tr>
<td><strong>Output power</strong></td>
<td>10dBm</td>
<td>n/a**</td>
<td>12-18dBm</td>
<td>6dBm</td>
<td>10dBm</td>
</tr>
<tr>
<td><strong>Noise figure</strong></td>
<td>&lt;8dB</td>
<td>n/a**</td>
<td>5.5-7.5dB</td>
<td>?</td>
<td>&lt;7dB</td>
</tr>
<tr>
<td><strong>EVM</strong>*</td>
<td>Very good</td>
<td>Excellent</td>
<td>???</td>
<td>Poor</td>
<td>Average</td>
</tr>
<tr>
<td><strong>Open source</strong></td>
<td>FPGA/driver</td>
<td>FPGA/driver</td>
<td>FPGA/Driver</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td>4G</td>
<td>4G/5G (80MHz with 1/4 sampling)</td>
<td>5G up to 100MHz</td>
<td>4G</td>
<td>4G</td>
</tr>
</tbody>
</table>

*needs external RF elements
** depends on daughterboard
*** subjective to the author 😊
Epiq Sidekiq

- **Based on AD 9361 chipset**
  - 70MHz – 6GHz with up to 50MHz bandwidth per channel

**Sidekiq™ - MiniPCIe**
- MiniPCIe card form factor (30mm x 51mm x 5mm)
- 2 independent RF channels (2xRx or Tx+Rx)
- PCIe Gen1.1 x1 (2.5 Gbps) interface to host + USB 2.0 interface

**Sidekiq™ - M.2**
- M.2 T3042-D3-B card form factor (30mm x 42mm x 4mm)
- Up to 2x2 MIMO
- PCIe Gen2 x1 (5 Gbps) interface to host + USB 2.0 interface

- **Under beta-testing**
Other experimental targets

- **CPRI - PCIexpress**
  - IT Avero
  - Based on Xilinx eval board

- **CPRI gateway**
  - Bell Labs
  - Based on Xilinx or Intel platform

- **Skylark Iris platform**
  - Based on Lime platform
  - Scalable for massive MIMO

- **SYRTEM UED platform**
  - Based on Xilinx ZC706 eval board + AD9371 daughterboard
  - 2 full duplex channels with up to 122.88 MHz sampling
  - Not 100% open source
OAI EPC + OAI eNB + OAI UE

INSTALLATION
Software Requirements

- **Operating system**
  - Ubuntu 16.04.2, kernel 4.8
    - works for both openairinterface5g and openair-cn
    - For real-time operation, a low-latency kernel is recommended
    - For P/S-GW, gtp kernel module needs to be patched
    - See details on Wiki
  - CentOS Linux release 7.4.1708 (Core)
    - Better real-time performance than Ubuntu low-latency

- **Get code from our gitlab server**
  - RAN (eNB+UE): [https://gitlab.eurecom.fr/oai/openairinterface5g](https://gitlab.eurecom.fr/oai/openairinterface5g)
    - Branch master most stable
    - Branch develop latest features (recommended)
    - Several feature branches for cutting-edge developments
  - EPC: [https://gitlab.eurecom.fr/oai/openair-cn](https://gitlab.eurecom.fr/oai/openair-cn)
    - Tag 0.5.0 most stable (recommended)
    - Branch develop latest features
Example setup for eNB + EPC

**eNB**: 192.168.100.102

- eth1
  - IP: 192.168.100.102/24

**SGW**

- S1-U
  - eth1
  - IP: 192.168.100.102/24
  - Port: 2152

**mME**

- S1-C
  - eth1
  - IP: 192.168.100.103/24

**HSS**

- S6a

**EPC + HSS**: 192.168.100.103

- eth1
  - IP: 192.168.100.103/24

**PGW**

- S11
  - Virtual Interface via ITTI
    - “none”, “0.0.0.0/24”

- eth1
  - IP: 192.168.100.103/24
  - Port: 2152

**INTERNET**
OpenAirInterface5G directories

- **cmake_targets**
  - New directory for building all the targets
  - Contains “mother” build_oai script

- **targets**
  - Hardware specific code (drivers, tools, etc)
  - lte-softmodem, oaisim

- **openair1**
  - Basic DSP routines for implementing subset of LTE specifications under x86 (36.211, 36.212, 36.213 3GPP specifications)
  - Channel simulation, sounding and PHY abstraction software,

- **openair2**
  - MAC/RLC/PDCP/RRC

- **openair3**
  - Contains interfaces S1-C, S1-U (GTP, SCTP, S1AP) and NAS UE

- **common/utils**
  - Utilities such as the T tracer or the ITTI
Compiling OpenAirInterface5G

- **Top-level build script located in**
  - `cd openairinterface5g/cmake_targets`

- **Compile lte-softmodem**
  - `.build_oai`
  - `-I` installs additional required software
  - `-w <hw_target>` select HW target
  - `--eNB` compiles the lte-softmodem (for UE and eNB)
  - `-x` compiles with support for xforms softscope
  - `-V` compiles with support for VCD debugging
  - `--UE` compiles UE specific NAS parts
  - `--T-tracer` compiles with T support
  - `--lte-simulators` compiles the unitary simulators
  - `-h` help

- **This creates executables in**
  - `openairinterface5g/targets/bin`
Compiling OAI EPC + HSS

- **Top-level build script located in**
  - `cd openairCN/SCRIPTS`

- **For dedicated RB support**
  - Install kernel sources
  - Patch kernel using patch in /build/tools

- **Compile EPC**
  - `.build_spgw`
    - `-i` installs additional software packages*
  - `.build_mme`
    - `-i` installs additional software packages*
    - requires FQDN to be set in /etc/hosts
  - `.build_hss`
    - `-i` installs additional software packages*
    - `-I` installs database

*when asked to set password for root, use “linux” (or change it later in config file).
HSS and SIM card configuration

- Configuration file: /usr/local/etc/oai/hss.conf
- Use PHPmyadmin: http://yourhsshost/phpmyadmin
  - User: hssadmin, password: admin
- Add your MME
- Add your user
- Configure your SIM card
  - Use a blank SIM card, card reader and programming tool,
  - or a pre-programmed SIM card
- See Wiki for details
EPC and eNB configuration

- **EPC configuration**
  - `/usr/local/etc/oai/mme.conf, /usr/local/etc/oai/spgw.conf`
    - Check MCC, MNC, TAC
    - Check IP addresses for interfaces
    - Take care of S-GW list selection

- **eNB configuration**
  - `targets/PROJECTS/GENERIC-LTE-EPC/CONF/`
  - Select the config file that is most appropriate for your configuration (Band and Hardware)
  - Check
    - MCC, MNC, TAC
    - `downlink_frequency`
    - `mme_ip_address`
    - IP addresses of S1-MME and S1-U interfaces
Running OAI

- **Running EPC**
  - ./run_hss
  - ./run_mme
  - ./run_spgw

- **Run eNB**
  - sudo ./lte-softmodem -O <file.conf> -d -V

- **Run UE**
  - sudo ./lte-softmodem -U -C <freq> -r [25|50|100] -ue-scan-carrier -ue-txgain xx -ue-rxgain yy

- **Have fun!**
Troubleshooting

- **eNB not connection to MME / RRH**
  - Check IP addresses in config files
  - Check MCC, MNC matching

- **I get a lot of UUUs and LLLs**
  - Check the performance setting of CPU (C-states, CPU frequency)
  - Check USB3 connection (some cables are bad)

- **Phone does not connect**
  - Analyze S1AP messages in wireshark
  - Check keys in SIM card and HSS
  - ...

- **Throughput is very low**
  - Check radio conditions: duplexer, antennas, interference
DEBUGGING TOOLS
Debug tools

- **The T tracer**
  - Monitor the eNB in real-time, simulation, or playback mode

- **Telnet server**
  - Monitor and change parameters of the eNB in real-time or simulation

- **Simulators**
  - ulsim/dlsim
  - Basic simulator
  - L1 simulator*
  - L2 FAPI simulator*

*being integrated into develop
The T tracer

- The T tracer is a framework to debug and monitor the softmodem.

- Combines logging, timing analysis, signal visualization, MAC PDU analysis (with wireshark)

- It is made 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.

- Can work locally or over network
The T tracer: usage of GUI

- Compile eNB with \(-T\)-tracer option:
  - ./build_oai -w USRP -eNB -T-tracer

- Compile eNB GUI:
  - cd openairinterface5g/common/utils/T
  - make

- Run lte-softmodem normally
  - sudo ./lte-softmodem -0 <…>

- Run T tracer GUI
  - ./enb -d ../T_messages
- HARQ ACK
- HARQ NAK
- New DCI
- Retr. DCI
Telnet server

- Telnet server can be used to show and change parameters at runtime
  - Log level and verbosity
  - Threads and their priority
  - Some PHY parameters (e.g. turbo iterations)

- Easily extendable

- Usage
  - `./build_oai -w USRP -eNB -T-tracer`
  - `sudo ./lte-softmodem -0 <...> --telnetsrv`
  - Telnet 127.0.0.1 8080
  - Use online help
Simulators

- **dlsim/ulsim**
  - `.build_oai - phy_simulators`

- **Basic simulator**
  - `.build_oai - basic-simulator`
  - See targets/ARCH/tcp_bridge/README.tcp_bridge_oai

- **L1 simulator (ex oaisim)**
  - Based in IF5/IF4.5
  - `.build_oai --eNB -t ETHERNET --noS1`
  - eNB: `sudo ./lte-softmodem -0 ../..../targets/PROJECTS GENERIC-LTE-EPC/CONF/rcc.band7.tml.if4p5.50PRB.lo.conf`
  - `.build_oai --UE -t ETHERNET --noS1`
  - UE: `sudo ./lte-uesoftmodem -0 ../..../targets/PROJECTS GENERIC-LTE-EPC/CONF/rru.oaisim.conf -A AWGN -r 50 -s 25 --siml1`

- **L2 simulator**
  - Based on nFAPI (IF2)
  - Same build process as L1 simulator but config files for nFAPI
  - See targets/DOCS/nfapi-L2-emulator-setup.txt for details
BACKUP
Debug tools

- **Spectrum Analyzer (UL and DL)**
  - Shows RF performance and signal integrity

- **Logs**
  - Verbosity can be adjusted in config file
  - Shows L2/L3 events

- **PHY scope**
  - Signals in time and frequency domain
  - Constellation plots of PUSCH, PUCCH

- **Stats window**
  - eNB measurements (noise, signal power, etc)
  - UE feedback (CQI, etc.)
  - UL and DL HARQ statistics

- **VCD file**
  - Analyze real-time behavior
  - `gtkwave -a ~/openairinterface5g/targets/RT/USER/eNB_usrp.gtk`

- **Wireshark**
  - To analyze messages over S1 interface
  - Can also analyze MAC, RLC, PDCP, RRC if enables (see twiki for details)

- **Iperf/speedtest**
  - Shows throughput for UDP and IP
OAI Packet tracer API
Interface wireshark

- **Supported information**
  - MAC_LTE_RNTI_TAG; MAC_LTE_UEID_TAG; MAC_LTE_SUBFRAME_TAG; MAC_LTE_PAYLOAD_TAG

- **How to enable**
  - Lte-softmodem
    - Wireshark: lte-softmodem -W (capture in localhost)
    - Pcap: lte-softmodem -P /tmp/oai.pcap
  - Oaisim
    - ./oaisim -P wireshark (capture in localhost)
    - ./oaisim -P pcap (output goes to /tmp/oai_opt.pcap)

- **How to configure wireshark**
  - try heuristics for the UDP protocol, MAC-LTE, RLC-LTE, and PDCP-LTE

- **More information can be found at**
  - https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/IttiAnalyzer
Openairinterface5g LOG APIs

- LOG_X(COMPONENT, format_string, args …)
- Logs are formatted as follows:
  
  \[
  \text{[COMPONENT][LOG LEVEL][FUNC][FILE][NODE ID][FRAME NUM][CONTENT]} 
  \]
  
  - COMPONENT: RRC, PDCP, RLC, MAC, PHY, …
  - LOG LEVEL: Emerge, Alert, Critic, Error, Warning, Notice, Info, Debug, Trace
  - FUNC: name of the function inside which the log is called. This is optional
  - FILE: add the file name
  - NODE ID: eNB or UE with their ID
  - FRAME NUM: frame and subframe number
  - CONTENT: content of the log message
  - LOG_FULL include FILE line

- Log verbosity mask
  - LOG_LOW: include the component
  - LOG_MED includes include the level of the log
  - LOG_HIGH includes include function name
  - LOG_FULL include the file name

- LOG Level
  - LOG_EMERG :: LOG_G
  - LOG_ALERT :: LOG_A
  - LOG_CRIT :: LOG_C
  - LOG_ERR :: LOG_E
  - LOG_WARNING :: LOG_W
  - LOG_NOTICE :: LOG_N
  - LOG_INFO :: LOG_I
  - LOG_DEBUG :: LOG_D
Openairinterface5g LOG APIs
How to configure

- Option “-l” with the level as a number
  - 0 lowest
  - 9 highers

- Configuration file (for the moment, only valid for lte-softmodem)
  log_config :
  
  { 
      global_log_level = "trace";
      global_log_verbosity = "medium";
      ...
  }

- Manually in oaisim_config.c (func olg_config) or in lte-softmodem local variables

- oaisim with option “-c” and xml configuration file
  <EMULATION_CONFIG>
  <LOG> <!-- set the global log level -->
  <LEVEL>debug</LEVEL>
  <INTERVAL>1</INTERVAL>
  </LOG>
  <SEED_VALUE>1234</SEED_VALUE> <!-- value 0 means randomly generated by OAI -->
  </EMULATION_CONFIG>

- Source files
  - openairinterface5g/openair2/UTIL/LOG/
Openair-CN Log API

- OAILOG_FUNC_IN(pROTO)
- OAILOG_FUNC_RETURN(pROTO, RC)
- OAILOG_EMERGENCY(pROTO, args...)
- OAILOG_ALERT(pROTO, args...)
- OAILOG_CRITICAL(pROTO, args...)
- OAILOG_ERROR(pROTO, args...)
- OAILOG_WARNING(pROTO, args...)
- OAILOG_NOTICE(pROTO, args...)
- OAILOG_INFO(pROTO, args...)
- OAILOG_DEBUG(pROTO, args...)
- OAILOG_TRACE(pROTO, args...)

Source
   - openair-cn/SRC/UTIL/

Logging:
```cpp
LOGGING :
{
    # OUTPUT choice in {"CONSOLE"}
    # 'path to file' must start with '.' or '/'
    # if TCP stream choice, then you can easily dump the
    traffic on the remote or local host: nc -l 'TCP port num' >
    received.txt
    OUTPUT = "CONSOLE";

    # COLOR choice in {"yes", "no"} means use of ANSI styling
    codes or no
    COLOR = "yes";
    # TODO

    # Log level choice in {"EMERGENCY", "ALERT", "CRITICAL",
    "ERROR", "WARNING", "NOTICE", "INFO", "DEBUG", "TRACE"}
    SCTP_LOG_LEVEL = "TRACE";
    S1AP_LOG_LEVEL = "TRACE";
    NAS_LOG_LEVEL = "TRACE";
    MME_APP_LOG_LEVEL = "TRACE";
    S6A_LOG_LEVEL = "TRACE";
    UTIL_LOG_LEVEL = "TRACE";
    MSC_LOG_LEVEL = "ERROR";
    ITTI_LOG_LEVEL = "ERROR";

    # ASN1 VERBOSITY: none, info, annoying
    # for S1AP protocol
    ASN1_VERBOSITY = "none";
};
```
OAI time analyzer
Format: VCD Value Change Dump

- tracks the execution time of each function working as a common profiler for performance improvement.
  - code optimization, bottleneck detection, and processing time measurements
- Output format is read by gtkwave to view the signal transition and timing.
OAI time analyzer API
Format: VCD Value Change Dump

```
#include "UTIL/LOG/vcd_signal_dumper.h"

Main()
{
    VCD_SIGNAL_DUMPER_INIT("/tmp/openair_dump.vcd");
    VCD_SIGNAL_DUMPER_DUMP_VARIABLE_BY_NAME(VCD_SIGNAL_DUMPER_VARIABLES_HW_FRAME, frame);
    VCD_SIGNAL_DUMPER_DUMP_FUNCTION_BY_NAME(VCD_SIGNAL_DUMPER_FUNCTIONS_ENB_DLSCH_ULSCH_SCHEDULER, VCD_FUNCTION_IN);
    ...;
    VCD_SIGNAL_DUMPER_DUMP_FUNCTION_BY_NAME(VCD_SIGNAL_DUMPER_FUNCTIONS_ENB_DLSCH_ULSCH_SCHEDULER, VCD_FUNCTION_OUT);

    VCD_SIGNAL_DUMPER_CLOSE();
}
```

- **Two type of signals**
  - variables
  - Functions
- **Used with gtkwave to view the signal transition and timing**
- **Source code**
  - Openair2/UTILS/LOG
- **To enable use option “-V”, then open with gtkwave and preconfigured file**
  - eNB_exmimo2.gtkw  eNB_usrp.gtkw  rrh.gtkw  ue_exmimo2.gtkw
Inter-task interface (ITTI)

- Intra-process communication system through async message passing
- Source code:
  - common/utils/itti
Inter-task interface (ITTI)

- Task = thread + Queue intra-process communication
  - #define TASK_DEF( name of the task, priority, queue size)
- Thread management, Task priority, Timer service
- Message definitions
  MESSAGE_DEF (S1AP_SCTP_NEW_MESSAGE_IND, TASK_PRIORITY_MED, S1apSctpNewMessageInd
  s1apSctpNewMessageInd)

  Typedef struct {
    uint8_t * buffer; ///< SCTP buffer
    uint32_t bufLen; ///< SCTP buffer length
    int32_t assocId; ///< SCTP physical
  } s1apSctpNewMessageInd

- ITTI can wait for
  - Messages
  - Timeout
  - External events such as sockets (FD)
void * s1ap_mme_thread ( void * args ) {
while (1) {
    receive_msg ( TASK_S1AP , & receivedMessage );
    assert ( receivedMessage != NULL );
    switch ( receivedMessage -> messageId ) {
        case S1AP_SCTP_NEW_MESSAGE_IND :
            // Some processing
            break ;
        default :
            S1AP_DEBUG ("Unknown message ID %d\n", receivedMessage -> messageId );
            break ;
    }
    free ( receivedMessage );
    receivedMessage = NULL ;
}
return NULL ;
}
ITTI Analyzer
Analyzing protocols PDU/SDU logs

- Complementary to Wireshark
- source file
  - common/utils/itti_analyzer

![Diagram of ITTI Analyzer process with TCP sockets and remote hosts]
Softscope

- Monitor PHY layer for both eNB and UE
- The tool plots:
  - received signal power, channel impulse response, channel frequency response, channel frequency response, LLRs, throughput and I/Q components (e.g., 4-QAM constellation)

- source file
  - openair1/PHY/TOOLS
MAC/PHY statistics

- **Online statistics for the status of the network**
  - successful transmissions, errors per HARQ per round, average throughput, ULSCH/DLSCH errors per HARQ process (8 in LTE FDD) per round (4 is maximum).

- **Source file**
  - Openair1/PHY/LTE_TRANSPORT/print_stats.c
  - Openair2/LAYER2/openair2_proc.c
Message Sequence Chart API

- Represents the internal function calls across layers and entities in a form of chart
- It is an ITTI task
- Make use of MSC lib – [http://www.mcternan.me.uk/mscgen/](http://www.mcternan.me.uk/mscgen/)
**MSC API**

- **MSC_LOG_EVENT**(proto, format, args...)
  - Failure events, new UE attached, congestion, ...
- **MSC_LOG_RX_MESSAGE**(receiver, sender, bytes, numBytes, format, args...)
- **MSC_LOG_RX_DISCARDED_MESSAGE**(receiver, sender, bytes, numBytes, format, args...)
- **MSC_LOG_TX_MESSAGE**(sender, receiver, bytes, numBytes, format, args...)
- **MSC_LOG_TX_MESSAGE_FAILED**(sender, receiver, bytes, numBytes, format, args...)

**Example**
- MSC_LOG_TX_MESSAGE(MSC_S1AP_ENB, MSC_S1AP_MME, NULL, 0, MSC_AS_TIME_FMT" S1AP_NAS_FIRST_REQ eNB %u UE %x", MSC_AS_TIME_ARGS(ctxt_pP), ctxt_pP->module_id, ctxt_pP->rnti);

**Usage**
- usage: msc_gen [-h] [--dir DIR] [--profile PROFILE] [--no_message NO_MESSAGE][--no_pdu NO_PDU] [--no_event NO_EVENT] [--type TYPE]
  - Dir: Directory where msc logs can be found
  - Profile: E-UTRAN, EPC
  - type: 'png', 'eps', 'svg' or 'ismap'

**Source code**
- Openair-cn/SRC/UTIL/MSC/ and openair-cn/SCRIPTS/msc_gen
- Openairinterface5g/common/util/msc and openairinterface5g/targets/SCRIPTS/msc_gen
**CLI**

- **Allow interactive interface with the OAI**
  - Debugging
  - Monitoring
  - Configuration

- **Now only available in oaisim with limited commands**

- **Plans**
  - Extend the commands
  - Apply to all OAI targets: lte-softmodem, RRH
BACKUP
Splits under construction in OAI Community

- **vEPC**
- **MEC**

**RRC**
- **PDCP**
  - **IF1**
  - **IF1′**

**LTE-RLC**
- **LTE-MAC**
  - **LTE-L1H**

**RAU** (Radio Aggregation Unit)
- (vCell Precoder)
- **802.11 MAC PHY**

**IF4.5 / UDP or raw (Nokia)**

**IF2 / UDP or raw (N-FAPI – Cisco)**

**RCC** (Radio Cloud Center)

**IF4.5 / UDP**

6/25/2018
Key Ingredients (How does OAI work)

- **Real-time extensions to Linux OS**
  - Today we rely on the lowlatency kernel provided by Ubuntu (since Ubuntu 14.04)
  - In earlier Ubuntu versions RTAI was used

- **Real-time data acquisition to/from PC**
  - ExpressMIMO uses DMA to transfer signals in and out of PC memory without hogging CPU → very efficient
  - USRP transfers data over USB and therefore requires extra CPU time for (de-)packetization of signals

- **Highly optimized DSP routines running on Intel GPP**
  - Exploiting vector processing (SIMD)
  - 64-bit MMX → 128-bit SSE2/3/4 → 256-bit AVX2
  - OAI features fastest FFT and Turbo decoder of its kind

- **Multi-threaded parallel processing**
OSA Strategic Areas

- Softwarization of Networks
- Internet of Things (IoT)
- 5G Modem
- Large Scale Network Emulation
- Heterogeneous Networks
- Test and Measurement