OpenAirInterface developer doc

Open Cells Project

- Provide OpenAir information for a new project
  - Aim to use and modify OAI

OpenAir features status

- OAI development
  - Basic 4G working with commercial UE or OAI UE, commercial EPC or OAI EPC
  - 4G features
    - FDD & TDD
    - 5/10/20MHz SISO
    - MIMO has been working, transmission modes partially coded
    - Carrier aggregation (Partially coded)
    - Handover, paging (partially)
    - HARQ
    - Multi-UE: working but unstable, static declaration of max UE
    - radio scheduling: basic
    - QCI and bearers: not developed
    - RLC UM: working, AM (acknowledged mode): existing but bugged
  - The introduction of NFAPI (small cells forum API) disturbed a lot the OAI eNB
- A lot of pending work to merge (expecting this NFAPI version progress) from all OSA members
  - 5G, new radio
  - NB-IoT
  - Multi-UE improvements
  - Several private versions or branches exist

OpenAir Code availability

Open Cells Project

- All eNB+UE code is in a dedicated gitlab repository
  - specific FRAND for eNB and UE licence
    - Fair, Reasonable and Non Discriminatory (FRAND) is the general way to use 3GPP standards. These standards embed “essential patents”. 3GPP standard implementation must use these patents.
- EPC migrated to Github
  - Apache 2 licence
- Third party
  - Run in Linux, several distributions
  - Extensive usage of other open source

OpenAir RAN source management

Open Cells Project

- https://gitlab.eurecom.fr/oai/openairinterface5g
- GitLab process
  - Issues, merge, … as per GitLab process
  - Continuous integration in a Jenkins server
    https://oailab.eurecom.fr:8083/jenkins/job/eNb-CI/
  - Master branch is called “develop”
  - Master branch contains old major releases

OpenAir RAN binary generation

Open Cells Project

- Cmake driven
  - the directory cmake_targets
    - Cmake file that drives the compilation.
- Head script
  - build_oai script creates a set of specific build trees from the cmake file to produce each use case binaries
    - simulation,
    - RF boards,
    - reduced versions (like without the 3GPP core network)

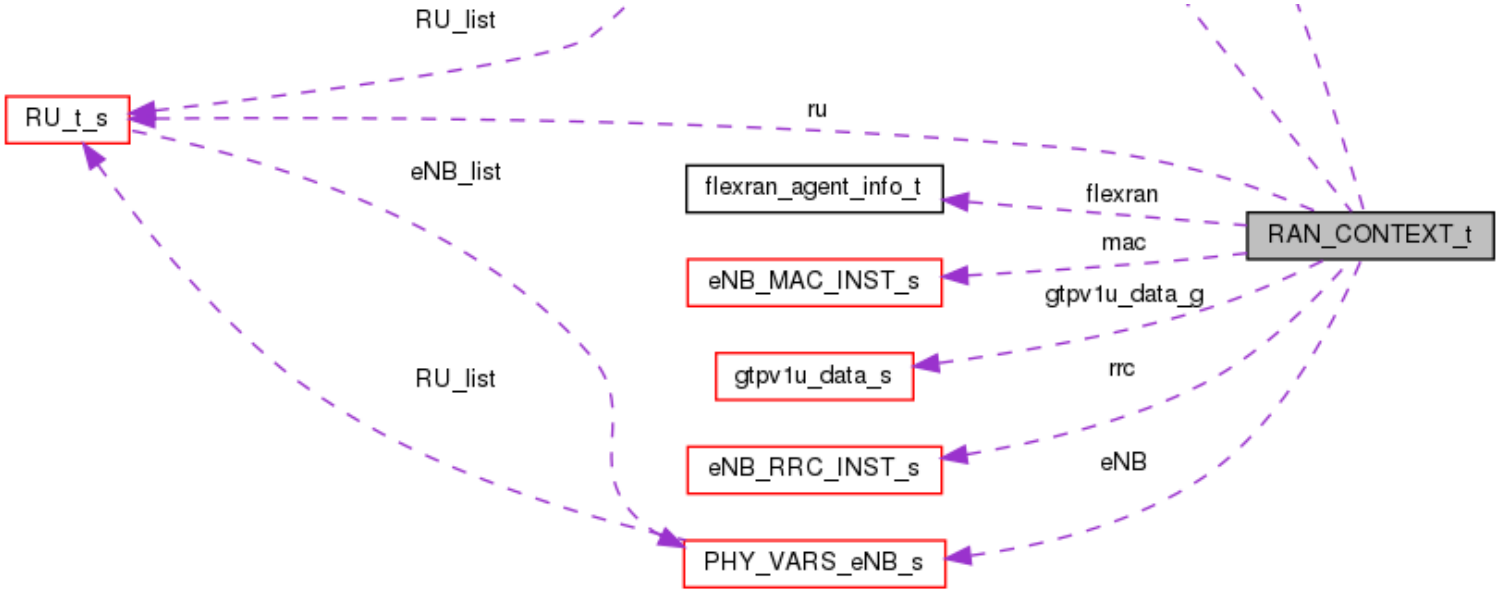
OpenAir RAN source tree

Open Cells Project

- Openair1
  - Contains the L1 code
- Openair2
  - L2 code and most of common code
- Openair3
  - L3 code
- Common
  - Common services
- Targets
  - Main function (process entry) C code
  - Per RF board code (targets/ARCH)
  - Runtime configuration files:In source tree, the directory cmake_targets contains a Cmake file that drives the compilation.
  - A script build_oai creates a set of specific build trees from the cmake file to produce each use case binaries: simulation, RF boards, reduced versions (like without the 3GPP core network)

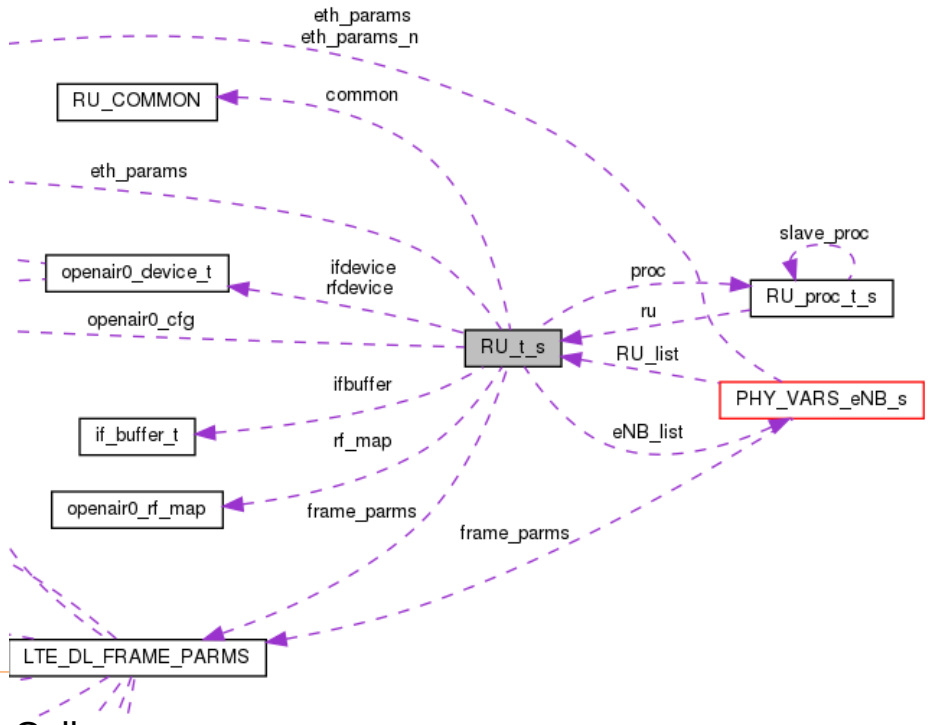
OpenAir RAN datamodel

- All pieces of code access a global structure
  - RAN_CONTEXT_t RC;



OpenAir data model per layer

- Each layer has structures pointed under “RC”
- Example
- See doc



OpenAir RAN multi-thread

Open Cells Project

- A Middleware called “itti” (interthread interface)
  - Classical send/receive message queues
  - Able to manage in the main loop
    - external sockets
    - Timers
- Dedicated hard coded threads in L1
  - Example: fep (front end processing)
    - Creates a second thread to process in // the two halves of a LTE sub-frame

OpenAir itti (middleware)

Open Cells Project

- All queues are created in .h files, as static permanent queues
  - The queues are called tasks because it is often read in a single thread that have the same name
- itti_send_msg_to_task, itti_receive_msg, itti_poll_msg
  - Standard messages queues, thread safe
  - The reader is responsible of freeing the message
- itti_subscribe_event_fd, itti_get_events
  - Add external sockets to itti based main loop
- timer_setup timer_remove
  - When a timer expire, it adds a message in a itti queue
- itti_terminate_tasks
  - Calling this function push a “terminate” message on each itti queue

OpenAir generic main loop

```
Thread_aLayer_main(void * context) {
    // initialize the layer data
    // and register in itti the external sockets (like S1AP, GTP)
    aLayer_init();
    while (1) {
    MessageDef * msg
          itti_receive_msg(TASK_aLayer, &msg);
          // itti receive released, so we have incoming data
          // Can be a itti message
          if ( msg !=  NULL) {
               switch(ITTI_MSG_ID(msg)) {
                    case …:
                         processThisMsgType1(msg);
                         break;
                    case …:
                         processThisMsgType2(msg);
                         break;
               }
               Itti_free(msg);
       }
          // or data/event on external entries (sockets)
          Struct eppol_events* events;
          int nbEvents=itti_get_events(TASK_aLayer, &events);
          if ( nbEvents > 0)
             processLayerEvents(events, nbEvents);
    }
}
```

Threads running itti design pattern

- UDP, SCTP
  - Simple threads that interface itti internal messages to Linux
- GTPV1-I, S1AP
  - Threads to implement S1QP and GTP protocols
- PDCP
  - Pdcp is not following itti design pattern, but it uses itti internally
- RRC
  - Implements 3GPP Rrc features
  - Interfaces the layer 1 and the above itti tasks
- X2, ENP_APP (configuration), L1L2
  - Are declared itti queues with a dedicated thread, but they are almost empty

Layer 1 design

Open Cells Project

- General conception
  - No threads/messaging
  - Functions calls, may vary by setting function pointer
    - Example: RF board interface
      - trx_xxxx_func where xxxx can be start, read, write, end
      - These functions implements the RF broad interface
  - Main loop
    - Is the thread “ru_thread”

Layer 1 main loop

Open Cells Project

- In ru_thread, while(!oai_exit) {
  - Increment the sub-frame counter
    - OpenAir RAN processes sub-frames, all the code is organized for this
  - Read input data: ru→fh_south_in reads the input
  - Either sends dedicated posix signals (pthread_cond_signal()) or calls directly processing functions
  - ru→north_out() sends output data
  - Loop back on the while()

L1 on multi-core CPU

Open Cells Project

- L1 pieces are multi-threaded as in above L1 main loop description
- Example: “fep” is doing “front end processing” work
  - It can be a call to “fep_full()”
    - that do all work in sequence
  - Or it can be a call to “ru_fep_full_2thread()”
    - That send a signal to thread fep_thread() to process a slot (half of the sub-frame)
    - Process the second slot inside the main thread
    - Wait the thread fep_thread() processing end signal to continue in sequence

OpenAir RAN initialization

Open Cells Project

- 3GPP implementation leads to create a lot of constants tables, of look-up tables
  - Some tables are generated of line by matlab code
  - Some non parameter dependent tables are generated in init_lte_top() (but rarely uses parameters)
  - Some are generated in various functions depending on where is processed the configuration file information
- Memory allocation
  - 95% of the code uses permanently allocated memory chained behind the global variable “RC”
  - It is initialized in many places

OpenAir RAN eNB configuration

Open Cells Project

- The eNB uses a configuration file
  - TBD