#### gNB和UE连接：

#### openairinterface5g/targets/ARCH/rfsimulator/simulator.c

#include <sys/socket.h>

#include <netinet/in.h>

#include <netinet/tcp.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdbool.h>

#include <errno.h>

#include <sys/epoll.h>

#include <string.h>

#include <common/utils/assertions.h>

#include <common/utils/LOG/log.h>

#include <common/utils/load\_module\_shlib.h>

#include <common/utils/telnetsrv/telnetsrv.h>

#include <common/config/config\_userapi.h>

#include "common\_lib.h"

#include <openair1/PHY/defs\_eNB.h>

#include "openair1/PHY/defs\_UE.h"

#define CHANNELMOD\_DYNAMICLOAD

#include <openair1/SIMULATION/TOOLS/sim.h>

#include <targets/ARCH/rfsimulator/rfsimulator.h>

#define PORT 4043 //default TCP port for this simulator

#define CirSize 6144000 // 100ms is enough

#define sampleToByte(a,b) ((a)\*(b)\*sizeof(sample\_t))

#define byteToSample(a,b) ((a)/(sizeof(sample\_t)\*(b)))

#define MAX\_SIMULATION\_CONNECTED\_NODES 5

#define GENERATE\_CHANNEL 10 //each frame in DL

#define RFSIMU\_SECTION "rfsimulator"

#define RFSIMU\_OPTIONS\_PARAMNAME "options"

#define RFSIM\_CONFIG\_HELP\_OPTIONS

" list of comma separated options to enable rf simulator functionalities. Available options: \n"\

" chanmod: enable channel modelisation\n"\

" saviq: enable saving written iqs to a file\n"

#define RFSIMULATOR\_PARAMS\_DESC { \

{"serveraddr", "<ip address to connect to>\n", simOpt, strptr:&(rfsimulator->ip), defstrval:"127.0.0.1", TYPE\_STRING, 0 },\

{"serverport", "<port to connect to>\n", simOpt, u16ptr:&(rfsimulator->port), defuintval:PORT, TYPE\_UINT16, 0 },\

{RFSIMU\_OPTIONS\_PARAMNAME, RFSIM\_CONFIG\_HELP\_OPTION 0, strlistptr:NULL, defstrlistval:NULL, TYPE\_STRINGLIST,0 },\

{"IQfile", "<file path to use when saving IQs>\n",simOpt, strptr:&(saveF), defstrval:"/tmp/rfsimulator.iqs",TYPE\_STRING, 0 },\

{"modelname", "<channel model name>\n", simOpt, strptr:&(modelname), defstrval:"AWGN", TYPE\_STRING, 0 },\

{"ploss", "<channel path loss in dB>\n", simOpt, dblptr:&(rfsimulator->chan\_pathloss), defdblval:0, TYPE\_DOUBLE, 0 },\

{"forgetfact", "<channel forget factor ((0 to 1)>\n", simOpt, dblptr:&(rfsimulator->chan\_forgetfact), defdblval:0, TYPE\_DOUBLE, 0 },\

{"offset", "<channel offset in samps>\n", simOpt, iptr:&(rfsimulator->chan\_offset), defintval:0, TYPE\_INT, 0 }\

};

static int rfsimu\_setchanmod\_cmd(char \*buff, int debug, telnet\_printfunc\_t prnt, void \*arg);

static telnetshell\_cmddef\_t rfsimu\_cmdarray[] = {

{"setmodel","<model name> <model type>", (cmdfunc\_t) rfsimu\_setchanmod\_cmd, TELNETSRV\_CMDFLAG\_PUSHINTPOOLQ},

{"","",NULL},

};

static telnetshell\_vardef\_t rfsimu\_vardef[] = {

{"",0,NULL}

};

pthread\_mutex\_t Sockmutex;

typedef c16\_t sample\_t; // 2\*16 bits complex number

typedef struct buffer\_s {

int conn\_sock;

openair0\_timestamp lastReceivedTS;

bool headerMode;

bool trashingPacket;

samplesBlockHeader\_t th;

char \*transferPtr;

uint64\_t remainToTransfer;

char \*circularBufEnd;

sample\_t \*circularBuf;

channel\_desc\_t \*channel\_model;

} buffer\_t;

typedef struct {

int listen\_sock, epollfd;

openair0\_timestamp nextRxTstamp;

openair0\_timestamp lastWroteTS;

uint64\_t typeStamp;

char \*ip;

uint16\_t port;

int saveIQfile;

buffer\_t buf[FD\_SETSIZE];

int rx\_num\_channels;

int tx\_num\_channels;

double sample\_rate;

double tx\_bw;

int channelmod;

double chan\_pathloss;

double chan\_forgetfact;

int chan\_offset;

float noise\_power\_dB;

void \*telnetcmd\_qid;

poll\_telnetcmdq\_func\_t poll\_telnetcmdq;

} rfsimulator\_state\_t;

static void allocCirBuf(rfsimulator\_state\_t \*bridge, int sock) {

buffer\_t \*ptr=&bridge->buf[sock];

AssertFatal ( (ptr->circularBuf=(sample\_t \*) malloc(sampleToByte(CirSize,1))) != NULL, "");

ptr->circularBufEnd=((char \*)ptr->circularBuf)+sampleToByte(CirSize,1);

ptr->conn\_sock=sock;

ptr->lastReceivedTS=0;

ptr->headerMode=true;

ptr->trashingPacket=false;

ptr->transferPtr=(char \*)&ptr->th;

ptr->remainToTransfer=sizeof(samplesBlockHeader\_t);

int sendbuff=1000\*1000\*100;

AssertFatal ( setsockopt(sock, SOL\_SOCKET, SO\_SNDBUF, &sendbuff, sizeof(sendbuff)) == 0, "");

struct epoll\_event ev= {0};

ev.events = EPOLLIN | EPOLLRDHUP;

ev.data.fd = sock;

AssertFatal(epoll\_ctl(bridge->epollfd, EPOLL\_CTL\_ADD, sock, &ev) != -1, "");

if ( bridge->channelmod > 0) {

static bool init\_done=false;

if (!init\_done) {

uint64\_t rand;

FILE \*h=fopen("/dev/random","r");

if ( 1 != fread(&rand,sizeof(rand),1,h) )

LOG\_W(HW, "Simulator can't read /dev/random\n");

fclose(h);

randominit(rand);

tableNor(rand);

init\_done=true;

}

char \*modelname = (bridge->typeStamp == ENB\_MAGICDL) ? "rfsimu\_channel\_ue0": "rfsimu\_channel\_enB0";

ptr->channel\_model=find\_channel\_desc\_fromname(modelname); // path\_loss in dB

AssertFatal((ptr->channel\_model!= NULL),"Channel model %s not found, check config file\n",modelname);

set\_channeldesc\_owner(ptr->channel\_model, RFSIMU\_MODULEID);

random\_channel(ptr->channel\_model,false);

}

}

static void removeCirBuf(rfsimulator\_state\_t \*bridge, int sock) {

AssertFatal( epoll\_ctl(bridge->epollfd, EPOLL\_CTL\_DEL, sock, NULL) != -1, "");

close(sock);

free(bridge->buf[sock].circularBuf);

memset(&bridge->buf[sock], 0, sizeof(buffer\_t));

bridge->buf[sock].conn\_sock=-1;

}

static void socketError(rfsimulator\_state\_t \*bridge, int sock) {

if (bridge->buf[sock].conn\_sock!=-1) {

LOG\_W(HW,"Lost socket \n");

removeCirBuf(bridge, sock);

if (bridge->typeStamp==UE\_MAGICDL)

exit(1);

}

}

enum blocking\_t {

notBlocking,

blocking

};

static void setblocking(int sock, enum blocking\_t active) {

int opts;

AssertFatal( (opts = fcntl(sock, F\_GETFL)) >= 0,"");

if (active==blocking)

opts = opts & ~O\_NONBLOCK;

else

opts = opts | O\_NONBLOCK;

AssertFatal(fcntl(sock, F\_SETFL, opts) >= 0, "");

}

static bool flushInput(rfsimulator\_state\_t \*t, int timeout, int nsamps);

static void fullwrite(int fd, void \*\_buf, ssize\_t count, rfsimulator\_state\_t \*t) {

if (t->saveIQfile != -1) {

if (write(t->saveIQfile, \_buf, count) != count )

LOG\_E(HW,"write in save iq file failed (%s)\n",strerror(errno));

}

AssertFatal(fd>=0 && \_buf && count >0 && t,

"Bug: %d/%p/%zd/%p", fd, \_buf, count, t);

char \*buf = \_buf;

ssize\_t l;

setblocking(fd, notBlocking);

while (count) {

l = write(fd, buf, count);

if (l <= 0) {

if (errno==EINTR)

continue;

if(errno==EAGAIN) {

usleep(500);

continue;

} else

return;

}

count -= l;

buf += l;

}

}

static void rfsimulator\_readconfig(rfsimulator\_state\_t \*rfsimulator) {

char \*saveF=NULL;

char \*modelname=NULL;

paramdef\_t rfsimu\_params[] = RFSIMULATOR\_PARAMS\_DESC;

int p = config\_paramidx\_fromname(rfsimu\_params,sizeof(rfsimu\_params)/sizeof(paramdef\_t), RFSIMU\_OPTIONS\_PARAMNAME) ;

int ret = config\_get( rfsimu\_params,sizeof(rfsimu\_params)/ sizeof(paramdef\_t), RFSIMU\_SECTI ON);

AssertFatal(ret >= 0, "configuration couldn't be performed");

rfsimulator->saveIQfile = -1;

for(int i=0; i<rfsimu\_params[p].numelt ; i++) {

if (strcmp(rfsimu\_params[p].strlistptr[i],"saviq") == 0) {

rfsimulator->saveIQfile=open(saveF,O\_APPEND| O\_CREAT|O\_TRUNC | O\_WRONLY, 0666);

if ( rfsimulator->saveIQfile != -1 )

LOG\_I(HW,"rfsimulator: will save written IQ samples in %s\n", saveF);

else

LOG\_E(HW, "can't open %s for IQ saving (%s)\n", saveF, strerror(errno));

break;

} else if (strcmp(rfsimu\_params[p].strlistptr[i],"chanmod") == 0) {

init\_channelmod();

load\_channellist(rfsimulator->tx\_num\_channels, rfsimulator->rx\_num\_channels, rfsimulator->sample\_rate, rfsimulator->tx\_bw);

rfsimulator->channelmod=true;

} else {

fprintf(stderr,"Unknown rfsimulator option: %s\n",rfsimu\_params[p].strlistptr[i]);

exit(-1);

}

}

if ( getenv("RFSIMULATOR") != NULL ) {

rfsimulator->ip=getenv("RFSIMULATOR");

}

if ( strncasecmp(rfsimulator->ip,"enb",3) == 0 ||

strncasecmp(rfsimulator->ip,"server",3) == 0 )

rfsimulator->typeStamp = ENB\_MAGICDL;

else

rfsimulator->typeStamp = UE\_MAGICDL;

}

static int rfsimu\_setchanmod\_cmd(char \*buff, int debug, telnet\_printfunc\_t prnt, void \*arg) {

char \*modelname=NULL;

char \*modeltype=NULL;

if (debug)

prnt("rfsimu\_setchanmod\_cmd buffer \"%s\"\n",buff);

int s = sscanf(buff,"%m[^ ] %ms\n",&modelname, &modeltype);

if (s == 2) {

int channelmod=modelid\_fromstrtype(modeltype);

if (channelmod<0)

prnt("ERROR: model type %s unknown\n",modeltype);

else {

rfsimulator\_state\_t \*t = (rfsimulator\_state\_t \*)arg;

int found=0;

for (int i=0; i<FD\_SETSIZE; i++) {

buffer\_t \*b=&t->buf[i];

if ( b->channel\_model==NULL)

continue;

if (b->channel\_model->model\_name==NULL)

continue;

if (b->conn\_sock >= 0 && (strcmp(b->channel\_model->model\_name,modelname)==0)) {

\*newmodel=new\_channel\_desc\_scm(t->tx\_num\_channels,t->rx\_num\_channels,

channelmod,

t->sample\_rate,

t->tx\_bw,

30e-9, // TDL delay-spread parameter

t->chan\_forgetfact, // forgetting\_factor

t->chan\_offset, // maybe used for TA

t->chan\_pathloss,

t->noise\_power\_dB

); // path\_loss in dB

set\_channeldesc\_owner(newmodel, RFSIMU\_MODULEID);

set\_channeldesc\_name(newmodel,modelname);

random\_channel(newmodel,false);

channel\_desc\_t \*oldmodel=b->channel\_model;

b->channel\_model=newmodel;

free\_channel\_desc\_scm(oldmodel);

prnt("New model type %s applied to channel %s connected to sock %d\n", modeltype, modelname,i);

found=1;

break;

}

}

if (found==0)

prnt("Channel %s not found or not currently used\n",modelname);

}

} else {

prnt("ERROR: 2 parameters required: model name and model type (%i found)\n",s);

}

free(modelname);

free(modeltype);

return CMDSTATUS\_FOUND;

}

static int startServer(openair0\_device \*device) {

rfsimulator\_state\_t \*t = (rfsimulator\_state\_t \*) device->priv;

t->typeStamp=ENB\_MAGICDL;

AssertFatal((t->listen\_sock = socket(AF\_INET, SOCK\_STREAM, 0)) >= 0, "");

int enable = 1;

AssertFatal(setsockopt(t->listen\_sock, SOL\_SOCKET, SO\_REUSEADDR, &enable, sizeof(int)) == 0, "");

struct sockaddr\_in addr = {

sin\_family:

AF\_INET,

sin\_port:

htons(t->port),

sin\_addr:

{ s\_addr: INADDR\_ANY }

};

bind(t->listen\_sock, (struct sockaddr \*)&addr, sizeof(addr));

AssertFatal(listen(t->listen\_sock, 5) == 0, "");

struct epoll\_event ev= {0};/

ev.events = EPOLLIN;

ev.data.fd = t->listen\_sock;

AssertFatal(epoll\_ctl(t->epollfd, EPOLL\_CTL\_ADD, t->listen\_sock, &ev) != -1, "");

return 0;

}

static int startClient(openair0\_device \*device) {

rfsimulator\_state\_t \*t = device->priv;

t->typeStamp=UE\_MAGICDL;

int sock;

AssertFatal((sock = socket(AF\_INET, SOCK\_STREAM, 0)) >= 0, "");

struct sockaddr\_in addr = {

sin\_family:

AF\_INET,

sin\_port:

htons(t->port),

sin\_addr:

{ s\_addr: INADDR\_ANY }

};

addr.sin\_addr.s\_addr = inet\_addr(t->ip);

bool connected=false;

while(!connected) {

LOG\_I(HW,"rfsimulator: trying to connect to %s:%d\n", t->ip, t->port);

if (connect(sock, (struct sockaddr \*)&addr, sizeof(addr)) == 0) {

LOG\_I(HW,"rfsimulator: connection established\n");

connected=true;

}

perror("rfsimulator");

sleep(1);

}

setblocking(sock, notBlocking);

allocCirBuf(t, sock);

return 0;

}

static int rfsimulator\_write\_internal(rfsimulator\_state\_t \*t, openair0\_timestamp timestamp, void \*\*samplesVoid, int nsamps, int nbAnt, int flags, bool alreadyLocked) {

if (!alreadyLocked)

pthread\_mutex\_lock(&Sockmutex);

LOG\_D(HW,"sending %d samples at time: %ld, nbAnt %d\n", nsamps, timestamp, nbAnt);

for (int i=0; i<FD\_SETSIZE; i++) {

buffer\_t \*b=&t->buf[i];

if (b->conn\_sock >= 0 ) {

samplesBlockHeader\_t header= {t->typeStamp, nsamps, nbAnt, timestamp};

fullwrite(b->conn\_sock,&header, sizeof(header), t);

sample\_t tmpSamples[nsamps][nbAnt];

for(int a=0; a<nbAnt; a++) {

sample\_t \*in=(sample\_t \*)samplesVoid[a];

for(int s=0; s<nsamps; s++)

tmpSamples[s][a]=in[s];

}

if (b->conn\_sock >= 0 ) {

fullwrite(b->conn\_sock, (void \*)tmpSamples, sampleToByte(nsamps,nbAnt), t);

}

}

}

if ( t->lastWroteTS != 0 && abs((double)t->lastWroteTS-timestamp) > (double)CirSize)

LOG\_E(HW,"Discontinuous TX gap too large Tx:%lu, %lu\n", t->lastWroteTS, timestamp);

if (t->lastWroteTS > timestamp+nsamps)

LOG\_E(HW,"Not supported to send Tx out of order (same in USRP) %lu, %lu\n",

t->lastWroteTS, timestamp);

t->lastWroteTS=timestamp+nsamps;

if (!alreadyLocked)

pthread\_mutex\_unlock(&Sockmutex);

LOG\_D(HW,"sent %d samples at time: %ld->%ld, energy in first antenna: %d\n",

nsamps, timestamp, timestamp+nsamps, signal\_energy(samplesVoid[0], nsamps) );

return nsamps;

}

static int rfsimulator\_write(openair0\_device \*device, openair0\_timestamp timestamp, void \*\*samplesVoid, int nsamps, int nbAnt, int flags) {

return rfsimulator\_write\_internal(device->priv, timestamp, samplesVoid, nsamps, nbAnt, flags, false);

}

static bool flushInput(rfsimulator\_state\_t \*t, int timeout, int nsamps\_for\_initial) {

struct epoll\_event events[FD\_SETSIZE]= {{0}};

int nfds = epoll\_wait(t->epollfd, events, FD\_SETSIZE, timeout);

if ( nfds==-1 ) {

if ( errno==EINTR || errno==EAGAIN ) {

return false;

} else

AssertFatal(false,"error in epoll\_wait\n");

}

for (int nbEv = 0; nbEv < nfds; ++nbEv) {

int fd=events[nbEv].data.fd;

if (events[nbEv].events & EPOLLIN && fd == t->listen\_sock) {

int conn\_sock;

AssertFatal( (conn\_sock = accept(t->listen\_sock,NULL,NULL)) != -1, "");

setblocking(conn\_sock, notBlocking);

allocCirBuf(t, conn\_sock);

LOG\_I(HW,"A client connected, sending the current time\n");

c16\_t v= {0};

void \*samplesVoid[t->tx\_num\_channels];

for ( int i=0; i < t->tx\_num\_channels; i++)

samplesVoid[i]=(void \*)&v;

rfsimulator\_write\_internal(t, t->lastWroteTS > 1 ? t->lastWroteTS-1 : 0,

samplesVoid, 1,

t->tx\_num\_channels, 1, false);

} else {

if ( events[nbEv].events & (EPOLLHUP | EPOLLERR | EPOLLRDHUP) ) {

socketError(t,fd);

continue;

}

buffer\_t \*b=&t->buf[fd];

if ( b->circularBuf == NULL ) {

LOG\_E(HW, "received data on not connected socket %d\n", events[nbEv].data.fd);

continue;

}

ssize\_t blockSz;

if ( b->headerMode)

blockSz=b->remainToTransfer;

else

blockSz= b->transferPtr + b->remainToTransfer <= b->circularBufEnd ?

b->remainToTransfer :

b->circularBufEnd - b->transferPtr ;

ssize\_t sz=recv(fd, b->transferPtr, blockSz, MSG\_DONTWAIT);

if ( sz < 0 ) {

if ( errno != EAGAIN ) {

LOG\_E(HW,"socket failed %s\n", strerror(errno));

//abort();

}

} else if ( sz == 0 )

continue;

LOG\_D(HW, "Socket rcv %zd bytes\n", sz);

AssertFatal((b->remainToTransfer-=sz) >= 0, "");

b->transferPtr+=sz;

if (b->transferPtr==b->circularBufEnd )

b->transferPtr=(char \*)b->circularBuf;

if ( b->headerMode==true && b->remainToTransfer==0) {

AssertFatal( (t->typeStamp == UE\_MAGICDL && b->th.magic==ENB\_MAGICDL) ||

(t->typeStamp == ENB\_MAGICDL && b->th.magic==UE\_MAGICDL), "Socket. Error in protocol");

b->headerMode=false;

if ( t->nextRxTstamp == 0 ) { // First block in UE, resync with the eNB current TS

t->nextRxTstamp=b->th.timestamp> nsamps\_for\_initial ?

b->th.timestamp - nsamps\_for\_initial :

b->lastReceivedTS=b->th.timestamp> nsamps\_for\_initial ?

b->th.timestamp :

nsamps\_for\_initial;

LOG\_W(HW,"UE got first timestamp: starting at %lu\n", t->nextRxTstamp);

b->trashingPacket=true;

} else if ( b->lastReceivedTS < b->th.timestamp) {

int nbAnt= b->th.nbAnt;

if ( b->th.timestamp-b->lastReceivedTS < CirSize ) {

for (uint64\_t index=b->lastReceivedTS; index < b->th.timestamp; index++ ) {

for (int a=0; a < nbAnt; a++) {

b->circularBuf[(index\*nbAnt+a)%CirSize].r = 0;

b->circularBuf[(index\*nbAnt+a)%CirSize].i = 0;

}

}

} else {

memset(b->circularBuf, 0, sampleToByte(CirSize,1));

}

if (b->lastReceivedTS != 0 && b->th.timestamp-b->lastReceivedTS < 1000)

LOG\_W(HW,"UEsock: %d gap of: %ld in reception\n", fd,

b->th.timestamp-b. ->lastReceivedTS );

b->lastReceivedTS=b->th.timestamp;

} else if ( b->lastReceivedTS > b->th.timestamp && b->th.size == 1 ) {

LOG\_W(HW,"Received Rx/Tx synchro out of order\n");

b->trashingPacket=true;

} else if ( b->lastReceivedTS == b->th.timestamp ) {

} else {

LOG\_E(HW, "received data in past: current is %lu, new reception: %lu!\n", b->lastReceivedTS, b->th.timestamp);

b->trashingPacket=true;

}

pthread\_mutex\_lock(&Sockmutex);

if(t->lastWroteTS != 0 && ( abs((double)t->lastWroteTS-b->lastReceivedTS) >. (double)CirSize))

LOG\_E(HW,"UEsock: %d Tx/Rx shift too large Tx:%lu, Rx:%lu\n", fd, t->lastWroteTS, b->lastReceivedTS);

pthread\_mutex\_unlock(&Sockmutex);

b->transferPtr=(char \*)&b->circularBuf[(b->lastReceivedTS\*b->th.nbAnt)%CirSize];

b->remainToTransfer=sampleToByte(b->th.size, b->th.nbAnt);

}

if ( b->headerMode==false ) {

if ( ! b->trashingPacket ) {

b->lastReceivedTS=b->th.timestamp+b->th.size-byteToSample(b->remainToTransfer,b- >th.nbAnt);

LOG\_D(HW,"UEsock: %d Set b->lastReceivedTS %ld\n", fd, b->lastReceivedTS);

}

if ( b->remainToTransfer==0) {

LOG\_D(HW,"UEsock: %d Completed block reception: %ld\n", fd, b->lastReceivedTS);

b->headerMode=true;

b->transferPtr=(char \*)&b->th;

b->remainToTransfer=sizeof(samplesBlockHeader\_t);

b->th.magic=-1;

b->trashingPacket=false;

}

}

}

}

return nfds>0;

}

static int rfsimulator\_read(openair0\_device \*device, openair0\_timestamp \*ptimestamp, void. \*\*samplesVoid, int nsamps, int nbAnt) {

if (nbAnt > 4) {

LOG\_W(HW, "rfsimulator: only 4 antenna tested\n");

}

rfsimulator\_state\_t \*t = device->priv;

LOG\_D(HW, "Enter rfsimulator\_read, expect %d samples, will release at TS: %ld, nbAnt %d\n", nsamps, t->nextRxTstamp+nsamps, nbAnt);

int first\_sock;

for (first\_sock=0; first\_sock<FD\_SETSIZE; first\_sock++)

if (t->buf[first\_sock].circularBuf != NULL )

break;

if ( first\_sock == FD\_SETSIZE ) {

if ( t->nextRxTstamp == 0)

LOG\_W(HW,"No connected device, generating void samples...\n");

if (!flushInput(t, 10, nsamps)) {

for (int x=0; x < nbAnt; x++)

memset(samplesVoid[x],0,sampleToByte(nsamps,1));

t->nextRxTstamp+=nsamps;

if ( ((t->nextRxTstamp/nsamps)%100) == 0)

LOG\_D(HW,"No UE, Generated void samples for Rx: %ld\n", t->nextRxTstamp);

\*ptimestamp = t->nextRxTstamp-nsamps;

return nsamps;

}

} else {

bool have\_to\_wait;

do {

have\_to\_wait=false;

for ( int sock=0; sock<FD\_SETSIZE; sock++) {

buffer\_t \*b=&t->buf[sock];

if ( b->circularBuf )

if ( t->nextRxTstamp+nsamps > b->lastReceivedTS ) {

have\_to\_wait=true;

break;

}

}

if (have\_to\_wait)

flushInput(t, 3, nsamps);

} while (have\_to\_wait);

}

for (int a=0; a<nbAnt; a++)

memset(samplesVoid[a],0,sampleToByte(nsamps,1));

for (int sock=0; sock<FD\_SETSIZE; sock++) {

buffer\_t \*ptr=&t->buf[sock];

if ( ptr->circularBuf ) {

bool reGenerateChannel=false;

if (reGenerateChannel)

random\_channel(ptr->channel\_model,0);

if (t->poll\_telnetcmdq)

t->poll\_telnetcmdq(t->telnetcmd\_qid,t);

for (int a=0; a<nbAnt; a++) {

if ( ptr->channel\_model != NULL ) {

rxAddInput(ptr->circularBuf, (c16\_t \*) samplesVoid[a],

a,

ptr->channel\_model,

nsamps,

t->nextRxTstamp,

CirSize);

}

else {

double H\_awgn\_mimo[4][4] ={{1.0, 0.2, 0.1, 0.05}, {0.2, 1.0, 0.2, 0.1}, {0.1, 0.2, 1.0, 0.2}, {0.05, 0.1, 0.2, 1.0}};

sample\_t \*out=(sample\_t \*)samplesVoid[a];

int nbAnt\_tx = ptr->th.nbAnt;//number of Tx antennas

for (int i=0; i < nsamps; i++) {//loop over nsamps

for (int a\_tx=0; a\_tx<nbAnt\_tx; a\_tx++) { //sum up signals from nbAnt\_tx antennas out[i].r+=(short)(ptr->circularBuf[((t->nextRxTstamp+i)\*nbAnt\_tx+a\_tx)%CirSize].r\*H\_awgn\_mimo[a][a\_tx]);

out[i].i+=(short)(ptr->circularBuf[((t->nextRxTstamp+i)\*nbAnt\_tx+a\_tx)%CirSize].

i\*H\_awgn\_mimo[a][a\_tx]);

}

}

}

}

}

}

\*ptimestamp = t->nextRxTstamp; // return the time of the first sample

t->nextRxTstamp+=nsamps;

LOG\_D(HW,"Rx to upper layer: %d from %ld to %ld, energy in first antenna %d\n",

nsamps,

\*ptimestamp, t->nextRxTstamp,

signal\_energy(samplesVoid[0], nsamps));

return nsamps;

}

static int rfsimulator\_get\_stats(openair0\_device \*device) {

return 0;

}

static int rfsimulator\_reset\_stats(openair0\_device \*device) {

return 0;

}

static void rfsimulator\_end(openair0\_device \*device) {}

static int rfsimulator\_stop(openair0\_device \*device) {

return 0;

}

static int rfsimulator\_set\_freq(openair0\_device \*device, openair0\_config\_t \*openair0\_cfg,int exmimo\_dump\_config) {

return 0;

}

static int rfsimulator\_set\_gains(openair0\_device \*device, openair0\_config\_t \*openair0\_cfg) {

return 0;

}

static int rfsimulator\_write\_init(openair0\_device \*device) {

return 0;

}

\_\_attribute\_\_((\_\_visibility\_\_("default")))

int device\_init(openair0\_device \*device, openair0\_config\_t \*openair0\_cfg) {

rfsimulator\_state\_t \*rfsimulator = (rfsimulator\_state\_t \*)calloc(sizeof(rfsimulator\_state\_t),1);

rfsimulator->tx\_num\_channels=openair0\_cfg->tx\_num\_channels;

rfsimulator->rx\_num\_channels=openair0\_cfg->rx\_num\_channels;

rfsimulator->sample\_rate=openair0\_cfg->sample\_rate;

rfsimulator->tx\_bw=openair0\_cfg->tx\_bw;

rfsimulator\_readconfig(rfsimulator);

pthread\_mutex\_init(&Sockmutex, NULL);

LOG\_I(HW,"rfsimulator: running as %s\n", rfsimulator-> typeStamp == ENB\_MAGICDL ? "server. waiting opposite rfsimulators to connect" : "client: will connect to a rfsimulator server side");

device->trx\_start\_func = rfsimulator->typeStamp == ENB\_MAGICDL ? startServer : startClient;

device->trx\_get\_stats\_func = rfsimulator\_get\_stats;

device->trx\_reset\_stats\_func = rfsimulator\_reset\_stats;

device->trx\_end\_func = rfsimulator\_end;

device->trx\_stop\_func = rfsimulator\_stop;

device->trx\_set\_freq\_func = rfsimulator\_set\_freq;

device->trx\_set\_gains\_func = rfsimulator\_set\_gains;

device->trx\_write\_func = rfsimulator\_write;

device->trx\_read\_func = rfsimulator\_read;

device->type = RFSIMULATOR;

openair0\_cfg[0].rx\_gain[0] = 0;

device->openair0\_cfg=&openair0\_cfg[0];

device->priv = rfsimulator;

device->trx\_write\_init = rfsimulator\_write\_init;

for (int i=0; i<FD\_SETSIZE; i++)

rfsimulator->buf[i].conn\_sock=-1;

AssertFatal((rfsimulator->epollfd = epoll\_create1(0)) != -1,"");

randominit(0);

set\_taus\_seed(0);

add\_telnetcmd\_func\_t addcmd = (add\_telnetcmd\_func\_t)get\_shlibmodule\_fptr("telnetsrv", TELNET\_ADDCMD\_FNAME);

if (addcmd != NULL) {

rfsimulator->poll\_telnetcmdq = (poll\_telnetcmdq\_func\_t)get\_shlibmodule\_fptr("telnetsrv", TELNET\_POLLCMDQ\_FNAME);

addcmd("rfsimu",rfsimu\_vardef,rfsimu\_cmdarray);

for(int i=0; rfsimu\_cmdarray[i].cmdfunc != NULL; i++) {

if ( rfsimu\_cmdarray[i].qptr != NULL) {

rfsimulator->telnetcmd\_qid = rfsimu\_cmdarray[i].qptr;

break;

}

}

}

return 0;

}

#### 配置文件：

#### openairinterface5g/targets/PROJECTS/GENERIC-NR-5GC/CONF/gnb.sa.band78.fr1. 106PRB.usrpb210.conf

Active\_gNBs = ( "gNB-OAI");

# Asn1\_verbosity, choice in: none, info, annoying

Asn1\_verbosity = "none";

gNBs =

(

{

gNB\_ID = 0xe00;

gNB\_name = "gNB-OAI";

tracking\_area\_code = 1;

plmn\_list = ({

mcc = 208;

mnc = 99;

mnc\_length = 2;

snssaiList = (

{

sst = 1;

sd = 0x1; // 0 false, else true

}

);

});

nr\_cellid = 12345678L;

ssb\_SubcarrierOffset = 0;

min\_rxtxtime = 6;

pdcch\_ConfigSIB1 = (

{

controlResourceSetZero = 12;

searchSpaceZero = 0;

}

);

servingCellConfigCommon = (

{

#spCellConfigCommon

physCellId = 0;

# this is 3600 MHz + 43 PRBs@30kHz SCS (same as initial BWP)

absoluteFrequencySSB = 641280;

dl\_frequencyBand = 78;

# this is 3600 MHz

dl\_absoluteFrequencyPointA = 640008;

#scs-SpecificCarrierList

dl\_offstToCarrier = 0;

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

dl\_subcarrierSpacing = 1;

dl\_carrierBandwidth = 106;

#initialDownlinkBWP

#genericParameters

# this is RBstart=27,L=48 (275\*(L-1))+RBstart

initialDLBWPlocationAndBandwidth = 28875; # 6366 12925 12956 28875 12952

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

initialDLBWPsubcarrierSpacing = 1;

#pdcch-ConfigCommon

initialDLBWPcontrolResourceSetZero = 12;

initialDLBWPsearchSpaceZero = 0;

#uplinkConfigCommon

#frequencyInfoUL

ul\_frequencyBand = 78;

#scs-SpecificCarrierList

ul\_offstToCarrier = 0;

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

ul\_subcarrierSpacing = 1;

ul\_carrierBandwidth = 106;

pMax = 20;

#initialUplinkBWP

#genericParameters

initialULBWPlocationAndBandwidth = 28875;

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

initialULBWPsubcarrierSpacing = 1;

#rach-ConfigCommon

#rach-ConfigGeneric

prach\_ConfigurationIndex = 98;

#prach\_msg1\_FDM

#0 = one, 1=two, 2=four, 3=eight

prach\_msg1\_FDM = 0;

prach\_msg1\_FrequencyStart = 0;

zeroCorrelationZoneConfig = 13;

preambleReceivedTargetPower = -96;

#preamblTransMax (0...10) = (3,4,5,6,7,8,10,20,50,100,200)

preambleTransMax = 6;

#powerRampingStep

# 0=dB0,1=dB2,2=dB4,3=dB6

powerRampingStep = 1;

#ra\_ReponseWindow

#1,2,4,8,10,20,40,80

ra\_ResponseWindow = 4;

#ssb\_perRACH\_OccasionAndCB\_PreamblesPerSSB\_PR

#1=oneeighth,2=onefourth,3=half,4=one,5=two,6=four,7=eight,8=sixteen

ssb\_perRACH\_OccasionAndCB\_PreamblesPerSSB\_PR = 4;

#oneHalf (0..15) 4,8,12,16,...60,64

ssb\_perRACH\_OccasionAndCB\_PreamblesPerSSB = 14;

#ra\_ContentionResolutionTimer

#(0..7) 8,16,24,32,40,48,56,64

ra\_ContentionResolutionTimer = 7;

rsrp\_ThresholdSSB = 19;

#prach-RootSequenceIndex\_PR

#1 = 839, 2 = 139

prach\_RootSequenceIndex\_PR = 2;

prach\_RootSequenceIndex = 1;

# SCS for msg1, can only be 15 for 30 kHz < 6 GHz, takes precendence over the one derived from prach-ConfigIndex

msg1\_SubcarrierSpacing = 1,

# restrictedSetConfig

# 0=unrestricted, 1=restricted type A, 2=restricted type B

restrictedSetConfig = 0,

msg3\_DeltaPreamble = 1;

p0\_NominalWithGrant = -90;

# pucch-ConfigCommon setup :

# pucchGroupHopping

# 0 = neither, 1= group hopping, 2=sequence hopping

pucchGroupHopping = 0;

hoppingId = 40;

p0\_nominal = -90;

# ssb\_PositionsInBurs\_BitmapPR

# 1=short, 2=medium, 3=long

ssb\_PositionsInBurst\_PR = 2;

ssb\_PositionsInBurst\_Bitmap = 1;

# ssb\_periodicityServingCell

# 0 = ms5, 1=ms10, 2=ms20, 3=ms40, 4=ms80, 5=ms160, 6=spare2, 7=spare1

ssb\_periodicityServingCell = 2;

# dmrs\_TypeA\_position

# 0 = pos2, 1 = pos3

dmrs\_TypeA\_Position = 0;

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

subcarrierSpacing = 1;

#tdd-UL-DL-ConfigurationCommon

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

referenceSubcarrierSpacing = 1;

# pattern1

# dl\_UL\_TransmissionPeriodicity

# 0=ms0p5, 1=ms0p625, 2=ms1, 3=ms1p25, 4=ms2, 5=ms2p5, 6=ms5, 7=ms10

dl\_UL\_TransmissionPeriodicity = 6;

nrofDownlinkSlots = 7;

nrofDownlinkSymbols = 6;

nrofUplinkSlots = 2;

nrofUplinkSymbols = 4;

ssPBCH\_BlockPower = -25;

}

);

# Dedicated Serving Cell Configuration

servingCellConfigDedicated = ({

# BWP-Downlink

# BWP 1 Configuration

dl\_bwp-Id\_1 = 1;

dl\_bwp1\_locationAndBandwidth = 28875; // RBstart=0, L=106 (40 MHz BW)

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

dl\_bwp1\_subcarrierSpacing = 1;

# BWP 2 Configuration

dl\_bwp-Id\_2 = 2;

dl\_bwp2\_locationAndBandwidth = 13750; // RBstart=0, L=51 (20 MHz BW)

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

dl\_bwp2\_subcarrierSpacing = 1;

# BWP 3 Configuration

dl\_bwp-Id\_3 = 3;

dl\_bwp3\_locationAndBandwidth = 6325; // RBstart=0, L=24 (10 MHz BW)

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

dl\_bwp3\_subcarrierSpacing = 1;

firstActiveDownlinkBWP-Id = 1; #BWP-Id

defaultDownlinkBWP-Id = 1; #BWP-Id

# bwp-InactivityTimer ENUMERATED {ms2, ms3, ms4, ms5, ms6, ms8, ms10, ms20, ms30,

# ms40,ms50, ms60, ms80,ms100, ms200,ms300, ms500, ms750, ms1280, ms1920, ms2560,

# spare10, spare9, spare8, spare7, spare6, spare5, spare4, spare3, spare2, spare1 }

# UplinkConfig

# BWP-Uplink

# BWP 1 Configuration

ul\_bwp-Id\_1 = 1;

ul\_bwp1\_locationAndBandwidth = 28875; // RBstart=0, L=106 (40 MHz BW)

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

ul\_bwp1\_subcarrierSpacing = 1;

# BWP 2 Configuration

ul\_bwp-Id\_2 = 2;

ul\_bwp2\_locationAndBandwidth = 13750; // RBstart=0, L=51 (20 MHz BW)

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

ul\_bwp2\_subcarrierSpacing = 1;

# BWP 3 Configuration

ul\_bwp-Id\_3 = 3;

ul\_bwp3\_locationAndBandwidth = 6325; // RBstart=0, L=24 (10 MHz BW)

# subcarrierSpacing

# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120

ul\_bwp3\_subcarrierSpacing = 1;

firstActiveUplinkBWP-Id = 1; #BWP-Id

}

);

# ------- SCTP definitions

SCTP :

{

# Number of streams to use in input/output

SCTP\_INSTREAMS = 2;

SCTP\_OUTSTREAMS = 2;

};

// AMF parameters:

amf\_ip\_address = ( { ipv4 = "192.168.70.132";

ipv6 = "192:168:30::17";

active = "yes";

preference = "ipv4";

}

);

NETWORK\_INTERFACES :

{

GNB\_INTERFACE\_NAME\_FOR\_NG\_AMF = "eno2";

GNB\_IPV4\_ADDRESS\_FOR\_NG\_AMF = "192.168.0.107/666";

GNB\_INTERFACE\_NAME\_FOR\_NGU = "eno2";

GNB\_IPV4\_ADDRESS\_FOR\_NGU = "192.168.0.107/666";

GNB\_PORT\_FOR\_S1U = 2152; # Spec 2152

};

}

);

MACRLCs = (

{

num\_cc = 1;

tr\_s\_preference = "local\_L1";

tr\_n\_preference = "local\_RRC";

pusch\_TargetSNRx10 = 150;

pucch\_TargetSNRx10 = 200;

ulsch\_max\_frame\_inactivity = 0;

}

);

L1s = (

{

num\_cc = 1;

tr\_n\_preference = "local\_mac";

prach\_dtx\_threshold = 120;

pucch0\_dtx\_threshold = 100;

ofdm\_offset\_divisor = 8; #set this to UINT\_MAX for offset 0

}

);

RUs = (

{

local\_rf = "yes"

nb\_tx = 1

nb\_rx = 1

att\_tx = 6;

att\_rx = 6;

bands = [78];

max\_pdschReferenceSignalPower = -27;

max\_rxgain = 114;

eNB\_instances = [0];

#beamforming 1x4 matrix:

bf\_weights = [0x00007fff, 0x0000, 0x0000, 0x0000];

clock\_src = "internal";

}

);

THREAD\_STRUCT = (

{

#three config for level of parallelism "PARALLEL\_SINGLE\_THREAD", "PARALLEL\_RU\_L1\_SPLIT", or "PARALLEL\_RU\_L1\_TRX\_SPLIT"

parallel\_config = "PARALLEL\_SINGLE\_THREAD";

#two option for worker "WORKER\_DISABLE" or "WORKER\_ENABLE"

worker\_config = "WORKER\_ENABLE";

}

);

rfsimulator :

{

serveraddr = "server";

serverport = "4043";

options = (); #("saviq"); or/and "chanmod"

modelname = "TDL\_D";

IQfile = "/tmp/rfsimulator.iqs";

};

security = {

# preferred ciphering algorithms

# the first one of the list that an UE supports in chosen

# valid values: nea0, nea1, nea2, nea3

ciphering\_algorithms = ( "nea0" );

# preferred integrity algorithms

# the first one of the list that an UE supports in chosen

# valid values: nia0, nia1, nia2, nia3

integrity\_algorithms = ( "nia2", "nia0" );

# setting 'drb\_ciphering' to "no" disables ciphering for DRBs, no matter

# what 'ciphering\_algorithms' configures; same thing for 'drb\_integrity'

drb\_ciphering = "yes";

drb\_integrity = "no";

};

log\_config :

{

global\_log\_level ="info";

hw\_log\_level ="info";

phy\_log\_level ="info";

mac\_log\_level ="info";

rlc\_log\_level ="info";

pdcp\_log\_level ="info";

rrc\_log\_level ="debug";

ngap\_log\_level ="debug";

f1ap\_log\_level ="debug";

};

#### 信道模型：

#### openairinterface5g/openair1/SIMULATION/TOOLS/random\_channel.c

#include <math.h>

#include <cblas.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "PHY/TOOLS/tools\_defs.h"

#include "sim.h"

#include "scm\_corrmat.h"

#include "common/utils/LOG/log.h"

#include "common/config/config\_userapi.h"

#include "common/utils/telnetsrv/telnetsrv.h"

#include "common/utils/load\_module\_shlib.h"

#include "assertions.h"

extern void print\_shorts(char \*s,\_\_m128i \*x);

static mapping channelmod\_names[] = {

CHANNELMOD\_MAP\_INIT

};

static int channelmod\_show\_cmd(char \*buff, int debug, telnet\_printfunc\_t prnt);

static int channelmod\_modify\_cmd(char \*buff, int debug, telnet\_printfunc\_t prnt);

static int channelmod\_print\_help(char \*buff, int debug, telnet\_printfunc\_t prnt);

static telnetshell\_cmddef\_t channelmod\_cmdarray[] = {

{"help","",channelmod\_print\_help},

{"show","<predef,current>",channelmod\_show\_cmd},

{"modify","<channelid> <param> <value>",channelmod\_modify\_cmd},

{"","",NULL},

};

static telnetshell\_vardef\_t channelmod\_vardef[] = {

{"",0,NULL}

};

static double snr\_dB=25;

static double sinr\_dB=0;

static unsigned int max\_chan;

static channel\_desc\_t \*\*defined\_channels;

static char modellist\_name[MAX\_OPTNAME\_SIZE]= {0};

void fill\_channel\_desc(channel\_desc\_t \*chan\_desc,

uint8\_t nb\_tx,

uint8\_t nb\_rx,

uint8\_t nb\_taps,

uint8\_t channel\_length,

double \*amps,

double \*delays,

struct complexd \*R\_sqrt,

double Td,

double sampling\_rate,

double channel\_bandwidth,

double ricean\_factor,

double aoa,

double forgetting\_factor,

double max\_Doppler,

int32\_t channel\_offset,

double path\_loss\_dB,

uint8\_t random\_aoa) {

uint16\_t i,j;

double delta\_tau;

LOG\_I(OCM,"[CHANNEL] Getting new channel descriptor, nb\_tx %d, nb\_rx %d, nb\_taps %d, channel\_length %d\n",

nb\_tx,nb\_rx,nb\_taps,channel\_length);

chan\_desc->nb\_tx = nb\_tx;

chan\_desc->nb\_rx = nb\_rx;

chan\_desc->nb\_taps = nb\_taps;

chan\_desc->channel\_length = channel\_length;

chan\_desc->amps = amps;

LOG\_D(OCM,"[CHANNEL] Doing delays ...\n");

if (delays==NULL) {

chan\_desc->delays = (double \*) malloc(nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_DELAY ;

delta\_tau = Td/nb\_taps;

for (i=0; i<nb\_taps; i++)

chan\_desc->delays[i] = ((double)i)\*delta\_tau;

} else

chan\_desc->delays = delays;

chan\_desc->Td = Td;

chan\_desc->sampling\_rate = sampling\_rate;

chan\_desc->channel\_bandwidth = channel\_bandwidth;

chan\_desc->ricean\_factor = ricean\_factor;

chan\_desc->aoa = aoa;

chan\_desc->random\_aoa = random\_aoa;

chan\_desc->forgetting\_factor = forgetting\_factor;

chan\_desc->channel\_offset = channel\_offset;

chan\_desc->path\_loss\_dB = path\_loss\_dB;

chan\_desc->first\_run = 1;

chan\_desc->ip = 0.0;

chan\_desc->max\_Doppler = max\_Doppler;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a = (struct complexd \*\*) malloc(nb\_taps\*sizeof(struct complexd \*));

LOG\_D(OCM,"[CHANNEL] Filling ch \n");

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i] = (struct complexd \*) malloc(channel\_length \* sizeof(struct complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(275 \* 12 \* sizeof(struct complexd));

LOG\_D(OCM,"[CHANNEL] Filling a (nb\_taps %d)\n",nb\_taps);

for (i = 0; i<nb\_taps; i++) {

LOG\_D(OCM,"tap%d(%p,%zu)\n",i,&chan\_desc->a[i],nb\_tx\*nb\_rx\* sizeof(struct complexd));

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

}

LOG\_D(OCM,"[CHANNEL] Doing R\_sqrt ...\n");

if (R\_sqrt == NULL) {

chan\_desc->R\_sqrt = (struct complexd \*\*) calloc(nb\_taps,sizeof(struct complexd \*));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_RSQRT\_NTAPS ;

for (i = 0; i<nb\_taps; i++) {

chan\_desc->R\_sqrt[i] = (struct complexd \*) calloc(nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx,sizeof(struct complexd));

for (j = 0; j<nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx; j+=(nb\_tx\*nb\_rx+1)) {

chan\_desc->R\_sqrt[i][j].r = 1.0;

chan\_desc->R\_sqrt[i][j].i = 0.0;

}

}

} else {

chan\_desc->R\_sqrt = (struct complexd \*\*) calloc(nb\_taps,sizeof(struct complexd \*));

for (i = 0; i<nb\_taps; i++) {

chan\_desc->R\_sqrt[i] = R\_sqrt;

}

}

for (i = 0; i<nb\_taps; i++) {

for (j = 0; j<nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx; j+=(nb\_tx\*nb\_rx+1)) {

LOG\_D(OCM,"Rsqrt[%d][%d] %f %f\n",i,j,chan\_desc->R\_sqrt[i][j].r,chan\_desc->R\_sqrt[i][j].i);

}

}

LOG\_D(OCM,"[CHANNEL] RF %f\n",chan\_desc->ricean\_factor);

for (i=0; i<chan\_desc->nb\_taps; i++)

LOG\_D(OCM,"[CHANNEL] tap %d: amp %f, delay %f\n",i,chan\_desc->amps[i] , chan\_desc->delays[i]);

chan\_desc->nb\_paths=10;

reset\_meas(&chan\_desc->random\_channel);

reset\_meas(&chan\_desc->interp\_time);

reset\_meas(&chan\_desc->interp\_freq);

reset\_meas(&chan\_desc->convolution);

}

static double mbsfn\_delays[] = {0,.03,.15,.31,.37,1.09,12.490, 12.52,12.64,12.80,12.86,13.58, 27.49,27.52,27.64,27.80,27.86,28.58};

static double mbsfn\_amps\_dB[] = {0,-1.5,-1.4,-3.6,-0.6,-7.0,-10,-11.5,-11.4,-13.6, -10.6,-17.0,-20, -21.5,-21.4,-23.6,-20.6,-27};

static double scm\_c\_delays[] = {0, 0.0125, 0.0250, 0.3625, 0.3750, 0.3875, 0.2500, 0.2625, 0.2750, 1.0375, 1.0500, 1.0625, 2.7250, 2.7375, 2.7500, 4.6000, 4.6125, 4.6250};

static double scm\_c\_amps\_dB[] = {0.00, -2.22, -3.98, -1.86, -4.08, -5.84, -1.08, -3.30, -5.06, -9.08, -11.30, -13.06, -15.14, -17.36, -19.12, -20.64, -22.85, -24.62};

static double tdl\_a\_delays[] = {0.0000, 0.3819, 0.4025, 0.5868, 0.4610, 0.5375, 0.6708, 0.5750, 0.7618, 1.5375, 1.8978, 2.2242, 2.1718, 2.4942, 2.5119, 3.0582, 4.0810, 4.4579, 4.5695, 4.7966, 5.0066, 5.3043, 9.6586};

static double tdl\_a\_amps\_dB[] = {-13.4, 0, -2.2, -4, -6, -8.2, -9.9, -10.5, -7.5, -15.9, -6.6, -16.7, -12.4, -15.2, -10.8, -11.3, -12.7, -16.2, -18.3, -18.9, -16.6, -19.9, -29.7 };

static double tdl\_b\_delays[] = {0.0000, 0.1072, 0.2155, 0.2095, 0.2870, 0.2986, 0.3752, 0.5055, 0.3681, 0.3697, 0.5700, 0.5283, 1.1021, 1.2756, 1.5474, 1.7842, 2.0169, 2.8294, 3.0219, 3.6187, 4.1067, 4.2790, 4.7834 };

static double tdl\_b\_amps\_dB[] = {0, -2.2, -4, -3.2, -9.8, -1.2, -3.4, -5.2, -7.6, -3, -8.9, -9, -4.8, -5.7, -7.5,-1.9, -7.6, -12.2, -9.8, -11.4, -14.9, -9.2, -11.3};

static double tdl\_c\_delays[] = {0, 0.2099, 0.2219, 0.2329, 0.2176, 0.6366, 0.6448, 0.6560, 0.6584, 0.7935, 0.8213, 0.9336, 1.2285, 1.3083, 2.1704, 2.7105, 4.2589, 4.6003, 5.4902, 5.6077, 6.3065, 6.6374, 7.0427, 8.6523};

static double tdl\_c\_amps\_dB[] = {-4.4, -1.2, -3.5, -5.2, -2.5, 0, -2.2, -3.9, -7.4, -7.1, -10.7, -11.1, -5.1, -6.8, -8.7, -13.2, -13.9, -13.9, -15.8, -17.1, -16, -15.7, -21.6, -22.8};

static double tdl\_d\_delays[] = {0, 0.035, 0.612, 1.363, 1.405, 1.804, 2.596, 1.775, 4.042, 7.937, 9.424, 9.708, 12.525};

static double tdl\_d\_amps\_dB[] = {-0.2, -13.5,} -.00147, -18.8, -21, -22.8, -17.9, -20.1, -21.9, -22.9, -27.8, -23.6, -24.8, -30.0, -27.7 };

static double tdl\_e\_delays[] = {0, 0.5133, 0.5440, 0.5630, 0.5440, 0.7112, 1.9092, 1.9293, 1.9589, 2.6426, 3.7136, 5.4524, 12.0034, 20.6519 };

static double tdl\_e\_amps\_dB[] = {-0.03, -22.03, -.00433, -15.8, -18.1, -19.8, -22.9, -22.4, -18.6, -20.8, -22.6, -22.3, -25.6, -20.2, -29.8, -29.2};

#define TDL\_E\_RICEAN\_FACTOR 0.0063096\

static double epa\_delays[] = { 0,.03,.07,.09,.11,.19,.41};

static double epa\_amps\_dB[] = {0.0,-1.0,-2.0,-3.0,-8.0,-17.2,-20.8};

static double eva\_delays[] = { 0,.03,.15,.31,.37,.71,1.09,1.73,2.51};

static double eva\_amps\_dB[] = {0.0,-1.5,-1.4,-3.6,-0.6,-9.1,-7.0,-12.0,-16.9};

static double etu\_delays[] = { 0,.05,.12,.2,.23,.5,1.6,2.3,5.0};

static double etu\_amps\_dB[] = {-1.0,-1.0,-1.0,0.0,0.0,0.0,-3.0,-5.0,-7.0};

static double default\_amps\_lin[] = {0.3868472, 0.3094778, 0.1547389, 0.0773694, 0.0386847, 0.0193424, 0.0096712, 0.0038685};

static double default\_amp\_lin[] = {1};

//correlation matrix for a 2x2 channel with full Tx correlation

static struct complexd R\_sqrt\_22\_corr[16] = {{0.70711,0}, {0.0, 0.0}, {0.70711,0}, {0.0, 0.0},

{0.0, 0.0}, {0.70711,0}, {0.0, 0.0}, {0.70711,0},{0.70711,0}, {0.0, 0.0}, {0.70711,0}, {0.0, 0.0},

{0.0, 0.0}, {0.70711,0}, {0.0, 0.0}, {0.70711,0}

};

//correlation matrix for a fully correlated 2x1 channel (h1==h2)

static struct complexd R\_sqrt\_21\_corr[] = {{0.70711,0}, {0.70711,0}, {0.70711,0}, {0.70711,0}};

//correlation matrix for a 2x2 channel with full Tx anti-correlation

static struct complexd R\_sqrt\_22\_anticorr[16] = {{0.70711,0}, {0.0, 0.0}, {-0.70711,0}, {0.0, 0.0},

{0.0, 0.0}, {0.70711,0}, {0.0, 0.0}, {-0.70711,0},{-0.70711,0}, {0.0, 0.0}, {0.70711,0}, {0.0, 0.0},

{0.0, 0.0}, {-0.70711,0}, {0.0, 0.0}, {0.70711,0}

};

//correlation matrix for a fully anti-correlated 2x1 channel (h1==-h2)

static struct complexd R\_sqrt\_21\_anticorr[4] = {{0.70711,0}, {-0.70711,0}, {-0.70711,0}, {0.70711,0}};

static struct complexd R\_sqrt\_22\_orthogonal[16] = {{0.70711,0.0}, {0.0, 0.0}, {0.0,0.0}, {0.0, 0.0},

{0.0, 0.0}, {0.0,0.0}, {0.0, 0.0}, {0.0,0.0},{0.0,0.0}, {0.0, 0.0}, {0.0,0.0}, {0.0, 0.0},{0.0, 0.0}, {0.0,0.0}, {0.0, 0.0}, {0.70711,0.0}

};

// full correlation matrix for TM4 to make orthogonal effective channel

static struct complexd R\_sqrt\_22\_orth\_eff\_ch\_TM4\_prec\_real[16] = {{0.70711,0.0}, {0.0, 0.0}, {0.70711,0.0}, {0.0, 0.0},{0.0, 0.0}, {0.70711,0.0}, {0.0, 0.0}, {-0.70711,0.0},{0.70711,0.0}, {0.0, 0.0}, {0.70711,0.0}, {0.0, 0.0},{0.0, 0.0}, {-0.70711,0.0}, {0.0, 0.0}, {0.70711,0.0}

};

static struct complexd R\_sqrt\_22\_orth\_eff\_ch\_TM4\_prec\_imag[16] = {{0.70711,0.0}, {0.0,0.0}, {0.0, -0.70711}, {0.0,0.0},{0.0, 0.0}, {0.70711,0.0}, {0.0, 0.0}, {0.0,0.70711},{0.0,-0.70711}, {0.0, 0.0}, {-0.70711,0.0}, {0.0, 0.0},{0.0, 0.0}, {0.0,0.70711}, {0.0, 0.0}, {-0.70711,0.0}

};

//Correlation matrix for EPA channel

static struct complexd R\_sqrt\_22\_EPA\_low[16] = {{1.0,0.0}, {0.0,0.0}, {0.0,0.0}, {0.0,0.0},{0.0,0.0}, {1.0,0.0}, {0.0,0.0}, {0.0,0.0},{0.0,0.0}, {0.0,0.0}, {1.0,0.0}, {0.0,0.0},{0.0,0.0}, {0.0,0.0}, {0.0,0.0}, {1.0,0.0}

};

static struct complexd R\_sqrt\_22\_EPA\_high[16] = {{0.7179,0.0}, {0.4500,0.0}, {0.4500,0.0}, {0.2821,0.0},{0.4500,0.0}, {0.7179,0.0}, {0.2821,0.0}, {0.4500,0.0},{0.4500,0.0}, {0.2821,0.0}, {0.7179,0.0}, {0.4500,0.0},{0.2821,0.0}, {0.4500,0.0}, {0.4500,0.0}, {0.7179,0.0}

};

static struct complexd R\_sqrt\_22\_EPA\_medium[16] = {{0.8375,0.0}, {0.5249,0.0}, {0.1286,0.0}, {0.0806,0.0},{0.5249,0.0}, {0.8375,0.0}, {0.0806,0.0}, {0.1286,0.0},{0.1286,0.0}, {0.0806,0.0}, {0.8375,0.0}, {0.5249,0.0},{0.0806,0.0}, {0.1286,0.0}, {0.5249,0.0}, {0.8375,0.0}

};

void tdlModel(int tdl\_paths, double \*tdl\_delays, double \*tdl\_amps\_dB, double DS\_TDL, channel\_desc\_t \*chan\_desc ) {

int nb\_rx=chan\_desc-> nb\_rx;

int nb\_tx=chan\_desc-> nb\_tx;

int tdl\_pathsby3 = tdl\_paths/3;

chan\_desc->nb\_taps= tdl\_paths;

chan\_desc->Td=tdl\_delays[tdl\_paths-1]\*DS\_TDL;printf("lastpath(%d)at%f\*%e =%e\n",tdl\_paths-1, tdl\_delays[tdl\_paths-1],DS\_TDL,chan\_desc->Td);

double sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

for (int i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*tdl\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (int i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] /= sum\_amps;

tdl\_delays[i] \*= DS\_TDL;

}

chan\_desc->delays = tdl\_delays;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a= (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd \*));

for (int i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i] = (struct complexd \*) malloc(chan\_desc->channel\_length \* sizeof(struct complexd))；

for (int i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc((2+(275\*12)) \* sizeof(struct complexd));

for (int i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(tdl\_pathsby3\*sizeof(struct complexd \*\*));

if (nb\_tx==2 && nb\_rx==2) {

for (int i = 0; i<(tdl\_pathsby3); i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R22\_sqrt[i][0];

} else if (nb\_tx==2 && nb\_rx==1) {

for (int i = 0; i<(tdl\_pathsby3); i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R21\_sqrt[i][0];

} else if (nb\_tx==1 && nb\_rx==2) {

for (int i = 0; i<(tdl\_pathsby3); i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R12\_sqrt[i][0];

} else {

for (int i = 0; i<(tdl\_pathsby3); i++) {

chan\_desc->R\_sqrt[i]=(struct complexd\*)malloc(nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx \* sizeof(struct complexd));

for (int j = 0; j<nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx; j+=(nb\_tx\*nb\_rx+1)) {

chan\_desc->R\_sqrt[i][j].r = 1.0;

chan\_desc->R\_sqrt[i][j].i = 0.0;

}

LOG\_W(OCM,"correlation matrix not implemented for nb\_tx==%d and nb\_rx==%d, using identity\n", nb\_tx, nb\_rx);

}

}

}

channel\_desc\_t \*new\_channel\_desc\_scm(uint8\_t nb\_tx,

uint8\_t nb\_rx,

SCM\_t channel\_model,

double sampling\_rate,

double channel\_bandwidth,

double DS\_TDL,

double forgetting\_factor,

int32\_t channel\_offset,

double path\_loss\_dB,

float noise\_power\_dB) {

channel\_desc\_t \*chan\_desc = (channel\_desc\_t \*)calloc(1,sizeof(channel\_desc\_t));

for(int i=0; i<max\_chan; i++) {

if (defined\_channels[i] == NULL) {

defined\_channels[i]=chan\_desc;

chan\_desc->chan\_idx=i;

break;

} else {

AssertFatal(i<(max\_chan-1),

"No more channel descriptors available, increase channelmod.max\_chan parameter above %u\n",max\_chan);

}

}

uint16\_t i,j;

double sum\_amps;

double aoa,ricean\_factor,Td,maxDoppler;

int channel\_length,nb\_taps;

struct complexd \*R\_sqrt\_ptr2;

chan\_desc->modelid = channel\_model;

chan\_desc->nb\_tx = nb\_tx;

chan\_desc->nb\_rx = nb\_rx;

chan\_desc->sampling\_rate = sampling\_rate;

chan\_desc->channel\_bandwidth = channel\_bandwidth;

chan\_desc->forgetting\_factor = forgetting\_factor;

chan\_desc->channel\_offset = channel\_offset;

chan\_desc->path\_loss\_dB = path\_loss\_dB;

chan\_desc->first\_run = 1;

chan\_desc->ip = 0.0;

chan\_desc->noise\_power\_dB = noise\_power\_dB;

LOG\_I(OCM,"Channel Model (inside of new\_channel\_desc\_scm)=%d\n\n", channel\_model);

int tdl\_paths=0;

double \*tdl\_amps\_dB;

double \*tdl\_delays;

switch (channel\_model) {

case SCM\_A:

LOG\_W(OCM,"channel model not yet supported\n");

free(chan\_desc);

return(NULL);

case SCM\_B:

LOG\_W(OCM,"channel model not yet supported\n");

free(chan\_desc);

return(NULL);

case SCM\_C:

chan\_desc->nb\_taps = 18;

chan\_desc->Td = 4.625;

chan\_desc->channel\_length = (int) (2\*chan\_desc->sampling\_rate\*chan\_desc->Td + 1 + 2/(M\_PI\*M\_PI)\*log(4\*M\_PI\*chan\_desc->sampling\_rate\*chan\_desc->Td));

sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_AMPS ;

for (i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*scm\_c\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->amps[i] /= sum\_amps;

chan\_desc->delays = scm\_c\_delays;

chan\_desc->ricean\_factor = 1;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a = (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd \*));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i] = (struct complexd \*) malloc(chan\_desc->channel\_length \* sizeof(struct complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(1200 \* sizeof(struct complexd));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(6\*sizeof(struct complexd \*\*));

if (nb\_tx==2 && nb\_rx==2) {

for (i = 0; i<6; i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R22\_sqrt[i][0];

} else if (nb\_tx==2 && nb\_rx==1) {

for (i = 0; i<6; i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R21\_sqrt[i][0];

} else if (nb\_tx==1 && nb\_rx==2) {

for (i = 0; i<6; i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R12\_sqrt[i][0];

} else {

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_RSQRT\_6 ;

for (i = 0; i<6; i++) {

chan\_desc->R\_sqrt[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx \* sizeof(struct complexd));

for (j = 0; j<nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx; j+=(nb\_tx\*nb\_rx+1)) {

chan\_desc->R\_sqrt[i][j].r = 1.0;

chan\_desc->R\_sqrt[i][j].i = 0.0;

}

LOG\_W(OCM,"correlation matrix not implemented for nb\_tx==%d and nb\_rx==%d, using. identity\n", nb\_tx, nb\_rx);

}

}

break;

case SCM\_D:

LOG\_W(OCM,"This is not the real SCM-D model! It is just SCM-C with an additional Rice. factor!\n");

chan\_desc->nb\_taps = 18;

chan\_desc->Td = 4.625;

chan\_desc->channel\_length = (int) (2\*chan\_desc->sampling\_rate\*chan\_desc->Td + 1 +. 2/(M\_PI\*M\_PI)\*log(4\*M\_PI\*chan\_desc->sampling\_rate\*chan\_desc->Td));

sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_AMPS ;

for (i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*scm\_c\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->amps[i] /= sum\_amps;

chan\_desc->delays = scm\_c\_delays;

chan\_desc->ricean\_factor = 0.1;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a=(struct complexd\*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd \*));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i]=(struct complexd \*)malloc(chan\_desc->channel\_length \* sizeof(struct. complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(1200 \* sizeof(struct complexd));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(6\*sizeof(struct complexd \*\*));

if (nb\_tx==2 && nb\_rx==2) {

for (i = 0; i<6; i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R22\_sqrt[i][0];

} else if (nb\_tx==2 && nb\_rx==1) {

for (i = 0; i<6; i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R21\_sqrt[i][0];

} else if (nb\_tx==1 && nb\_rx==2) {

for (i = 0; i<6; i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R12\_sqrt[i][0];

} else {

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_RSQRT\_6 ;

for (j = 0; j<nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx; j+=(nb\_tx\*nb\_rx+1)) {

chan\_desc->R\_sqrt[i][j].r = 1.0;

chan\_desc->R\_sqrt[i][j].i = 0.0;

}

LOG\_W(OCM,"correlation matrix not implemented for nb\_tx==%d and nb\_rx==%d, using. identity\n", nb\_tx, nb\_rx);

}

}

break;

DevAssert(sizeof(tdl\_ ## MoDel ## \_amps\_dB) == sizeof(tdl\_ ## MoDel ## \_delays));

tdl\_paths=sizeof(tdl\_ ## MoDel ## \_amps\_dB)/sizeof(\*tdl\_ ## MoDel ## \_amps\_dB);

tdl\_delays=tdl\_ ## MoDel ## \_delays;

tdl\_amps\_dB=tdl\_ ## MoDel ## \_amps\_dB

case TDL\_A:

chan\_desc->ricean\_factor = 1;

tdl\_m(a);

tdlModel(tdl\_paths, tdl\_delays, tdl\_amps\_dB, DS\_TDL, chan\_desc);

break;

case TDL\_B:

chan\_desc->ricean\_factor = 1;

tdl\_m(b);

tdlModel(tdl\_paths, tdl\_delays, tdl\_amps\_dB, DS\_TDL, chan\_desc);

break;

case TDL\_C:

chan\_desc->ricean\_factor = 1;

tdl\_m(c);

tdlModel(tdl\_paths, tdl\_delays, tdl\_amps\_dB, DS\_TDL, chan\_desc);

break;

case TDL\_D:

chan\_desc->ricean\_factor = TDL\_D\_RICEAN\_FACTOR;

tdl\_m(d);

tdlModel(tdl\_paths, tdl\_delays, tdl\_amps\_dB, DS\_TDL, chan\_desc);

break;

case TDL\_E:

chan\_desc->ricean\_factor = TDL\_E\_RICEAN\_FACTOR;

tdl\_m(e);

tdlModel(tdl\_paths, tdl\_delays, tdl\_amps\_dB, DS\_TDL, chan\_desc);

break;

case EPA:

chan\_desc->nb\_taps = 7;

chan\_desc->Td = .410;

chan\_desc->channel\_length = (int) (2\*chan\_desc->sampling\_rate\*chan\_desc->Td + 1 +. 2/(M\_PI\*M\_PI)\*log(4\*M\_PI\*chan\_desc->sampling\_rate\*chan\_desc->Td));

sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_AMPS ;

for (i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*epa\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->amps[i] /= sum\_amps;

chan\_desc->delays = epa\_delays;

chan\_desc->ricean\_factor = 1;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a = (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd. \*));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i]=(struct complexd \*)malloc(chan\_desc->channel\_length \* sizeof(struct. complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(1200 \* sizeof(struct complexd));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

if (nb\_tx==2 && nb\_rx==2) {

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(6\*sizeof(struct complexd \*\*));

for (i = 0; i<6; i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R22\_sqrt[i][0];

} else {

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(6\*sizeof(struct complexd \*\*));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_RSQRT\_6 ;

for (i = 0; i<6; i++) {

chan\_desc->R\_sqrt[i]=(structcomplexd\*)malloc(nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx\*sizeof(struct complexd));

for (j = 0; j<nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx; j+=(nb\_tx\*nb\_rx+1)) {

chan\_desc->R\_sqrt[i][j].r = 1.0;

chan\_desc->R\_sqrt[i][j].i = 0.0;

}

LOG\_W(OCM,"correlation matrix only implemented for nb\_tx==2 and nb\_rx==2, using identity\n");

}

}

break;

case EPA\_low:

chan\_desc->nb\_taps = 7;

chan\_desc->Td = .410;

chan\_desc->channel\_length = (int) (2\*chan\_desc->sampling\_rate\*chan\_desc->Td + 1 +. 2/(M\_PI\*M\_PI)\*log(4\*M\_PI\*chan\_desc->sampling\_rate\*chan\_desc->Td));

sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_AMPS ;

for (i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*epa\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->amps[i] /= sum\_amps;

chan\_desc->delays = epa\_delays;

chan\_desc->ricean\_factor = 1;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a= (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd. \*));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i]=(struct complexd \*)malloc(chan\_desc->channel\_length \* sizeof(struct. complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(1200 \* sizeof(struct complexd));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

if (nb\_tx==2 && nb\_rx==2) {

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct. complexd \*\*));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->R\_sqrt[i] = R\_sqrt\_22\_EPA\_low;

} else {

printf("Correlation matrices are implemented for 2 x 2 only");

}

break;

case EPA\_high:

chan\_desc->nb\_taps = 7;

chan\_desc->Td = .410;

chan\_desc->channel\_length = (int) (2\*chan\_desc->sampling\_rate\*chan\_desc->Td + 1 +. 2/(M\_PI\*M\_PI)\*log(4\*M\_PI\*chan\_desc->sampling\_rate\*chan\_desc->Td));

sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_AMPS ;

for (i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*epa\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->amps[i] /= sum\_amps;

chan\_desc->delays = epa\_delays;

chan\_desc->ricean\_factor = 1;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a = (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd. \*));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i]=(struct complexd \*)malloc(chan\_desc->channel\_length \* sizeof(struct complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(1200 \* sizeof(struct complexd));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

if (nb\_tx==2 && nb\_rx==2) {

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd \*\*));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->R\_sqrt[i] = R\_sqrt\_22\_EPA\_high;

} else {

printf("Correlation matrices are implemented for 2 x 2 only");

}

break;

case EPA\_medium:

chan\_desc->nb\_taps = 7;

chan\_desc->Td = .410;

chan\_desc->channel\_length = (int) (2\*chan\_desc->sampling\_rate\*chan\_desc->Td + 1 +. 2/(M\_PI\*M\_PI)\*log(4\*M\_PI\*chan\_desc->sampling\_rate\*chan\_desc->Td));

sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_AMPS ;

for (i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*epa\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->amps[i] /= sum\_amps;

chan\_desc->delays = epa\_delays;

chan\_desc->ricean\_factor = 1;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a = (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd \*));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i]=(struct complexd \*)malloc(chan\_desc->channel\_length \* sizeof(struct complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(1200 \* sizeof(struct complexd));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

if (nb\_tx==2 && nb\_rx==2) {

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd \*\*));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->R\_sqrt[i] = R\_sqrt\_22\_EPA\_medium;

} else {

printf("Correlation matrices are implemented for 2 x 2 only");

}

break;

case EVA:

chan\_desc->nb\_taps = 9;

chan\_desc->Td = 2.51;

chan\_desc->channel\_length = (int) (2\*chan\_desc->sampling\_rate\*chan\_desc->Td + 1 +. 2/(M\_PI\*M\_PI)\*log(4\*M\_PI\*chan\_desc->sampling\_rate\*chan\_desc->Td));

sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_AMPS ;

for (i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*eva\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->amps[i] /= sum\_amps;

chan\_desc->delays = eva\_delays;

chan\_desc->ricean\_factor = 1;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a = (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd. \*));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i]=(struct complexd \*)malloc(chan\_desc->channel\_length \* sizeof(struct complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(1200 \* sizeof(struct complexd));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

if (nb\_tx==2 && nb\_rx==2) {

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(6\*sizeof(struct complexd \*\*));

for (i = 0; i<6; i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R22\_sqrt[i][0];

} else {

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(6\*sizeof(struct complexd \*\*));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_RSQRT\_6 ;

for (i = 0; i<6; i++) {

chan\_desc->R\_sqrt[i]=(struct complexd\*)malloc(nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx\*

sizeof(struct complexd));

for (j = 0; j<nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx; j+=(nb\_tx\*nb\_rx+1)) {

chan\_desc->R\_sqrt[i][j].r = 1.0;

chan\_desc->R\_sqrt[i][j].i = 0.0;

}

LOG\_W(OCM,"correlation matrix only implemented for nb\_tx==2 and nb\_rx==2, using identity\n");

}

}

break;

case ETU:

chan\_desc->nb\_taps = 9;

chan\_desc->Td = 5.0;

chan\_desc->channel\_length = (int) (2\*chan\_desc->sampling\_rate\*chan\_desc->Td + 1 + 2/(M\_PI\*M\_PI)\*log(4\*M\_PI\*chan\_desc->sampling\_rate\*chan\_desc->Td));

sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_AMPS ;

for (i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*etu\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->amps[i] /= sum\_amps;

chan\_desc->delays = etu\_delays;

chan\_desc->ricean\_factor = 1;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a= (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd \*));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i]=(struct complexd\*) malloc(chan\_desc->channel\_length \* sizeof(struct complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(1200 \* sizeof(struct complexd));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

if (nb\_tx==2 && nb\_rx==2) {

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(6\*sizeof(struct complexd \*\*));

for (i = 0; i<6; i++)

chan\_desc->R\_sqrt[i] = (struct complexd \*) &R22\_sqrt[i][0];

} else {

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(6\*sizeof(struct complexd \*\*));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_RSQRT\_6 ;

for (i = 0; i<6; i++) {

chan\_desc->R\_sqrt[i]=(structcomplexd\*)malloc(nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx\*sizeof(struct complexd));

for (j = 0; j<nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx; j+=(nb\_tx\*nb\_rx+1)) {

chan\_desc->R\_sqrt[i][j].r = 1.0;

chan\_desc->R\_sqrt[i][j].i = 0.0;

}

LOG\_W(OCM,"correlation matrix only implemented for nb\_tx==2 and nb\_rx==2, using identity\n");

}

}

break;

case MBSFN:

chan\_desc->nb\_taps = 18;

chan\_desc->Td = 28.58;

chan\_desc->channel\_length = (int) (2\*chan\_desc->sampling\_rate\*chan\_desc->Td + 1 + 2/(M\_PI\*M\_PI)\*log(4\*M\_PI\*chan\_desc->sampling\_rate\*chan\_desc->Td));

sum\_amps = 0;

chan\_desc->amps = (double \*) malloc(chan\_desc->nb\_taps\*sizeof(double));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_AMPS ;

for (i = 0; i<chan\_desc->nb\_taps; i++) {

chan\_desc->amps[i] = pow(10,.1\*mbsfn\_amps\_dB[i]);

sum\_amps += chan\_desc->amps[i];

}

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->amps[i] /= sum\_amps;

chan\_desc->delays = mbsfn\_delays;

chan\_desc->ricean\_factor = 1;

chan\_desc->aoa = 0;

chan\_desc->random\_aoa = 0;

chan\_desc->ch = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->chF = (struct complexd \*\*) malloc(nb\_tx\*nb\_rx\*sizeof(struct complexd \*));

chan\_desc->a= (struct complexd \*\*) malloc(chan\_desc->nb\_taps\*sizeof(struct complexd \*));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->ch[i]=(struct complexd \*)malloc(chan\_desc->channel\_length \* sizeof(struct complexd));

for (i = 0; i<nb\_tx\*nb\_rx; i++)

chan\_desc->chF[i] = (struct complexd \*) malloc(1200 \* sizeof(struct complexd));

for (i = 0; i<chan\_desc->nb\_taps; i++)

chan\_desc->a[i] = (struct complexd \*) malloc(nb\_tx\*nb\_rx \* sizeof(struct complexd));

chan\_desc->R\_sqrt = (struct complexd \*\*) malloc(6\*sizeof(struct complexd \*));

chan\_desc->free\_flags=chan\_desc->free\_flags|CHANMODEL\_FREE\_RSQRT\_6;

for (i = 0; i<6; i++) {

chan\_desc->R\_sqrt[i]=(struct complexd\*)malloc(nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx\*sizeof(struct complexd));

for (j = 0; j<nb\_tx\*nb\_rx\*nb\_tx\*nb\_rx; j+=(nb\_tx\*nb\_rx+1)) {

chan\_desc->R\_sqrt[i][j].r = 1.0;

chan\_desc->R\_sqrt[i][j].i = 0.0;

}

LOG\_W(OCM,"correlation matrix only implemented for nb\_tx==2 and nb\_rx==2, using identity\n");

}

break;

case Rayleigh8:

nb\_taps = 8;

Td = 0.8;

channel\_length = (int)11+2\*sampling\_rate\*Td;

ricean\_factor = 1;

aoa = .03;

maxDoppler = 0;

fill\_channel\_desc(chan\_desc,

nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amps\_lin,

NULL,

NULL,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

case Rice8:

nb\_taps = 8;

Td = 0.8;

channel\_length = (int)11+2\*sampling\_rate\*Td;

ricean\_factor = 0.1;

aoa = 0.7854;

maxDoppler = 0;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amps\_lin,

NULL,

NULL,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

1);

break;

case Rayleigh1://MIMO Test uses Rayleigh1

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 0.0;

aoa = .03;

maxDoppler = 0;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

NULL,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

case Rayleigh1\_800:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 1;

aoa = .03;

maxDoppler = 800;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

NULL,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

case Rayleigh1\_corr:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 1;

aoa = .03;

maxDoppler = 0;

if ((nb\_tx==2) && (nb\_rx==1)) {

R\_sqrt\_ptr2 = R\_sqrt\_21\_corr;

} else if ((nb\_tx==2) && (nb\_rx==2)) {

R\_sqrt\_ptr2 = R\_sqrt\_22\_corr;

} else

R\_sqrt\_ptr2 = NULL;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

R\_sqrt\_ptr2,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

case Rayleigh1\_anticorr:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 1;

aoa = .03;

maxDoppler = 0;

if ((nb\_tx==2) && (nb\_rx==1)) { //check this

R\_sqrt\_ptr2 = R\_sqrt\_21\_anticorr;

} else if ((nb\_tx==2) && (nb\_rx==2)) {

R\_sqrt\_ptr2 = R\_sqrt\_22\_anticorr;

} else

R\_sqrt\_ptr2 = NULL;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

R\_sqrt\_ptr2,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

case Rice1:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 0.1;

aoa = 0.7854;

maxDoppler = 0;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

NULL,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

case AWGN:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 0.0;

aoa = 0.0;

maxDoppler = 0;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

NULL,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

printf("AWGN: ricean\_factor %f\n",chan\_desc->ricean\_factor);

break;

case TS\_SHIFT:

nb\_taps = 2;

double ts\_shift\_delays[] = {0, 1/7.68};

Td = ts\_shift\_delays[1];

channel\_length = 10;

ricean\_factor = 0.0;

aoa = 0.0;

maxDoppler = 0;

double ts\_shift\_amps[] = {0, 1};

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

ts\_shift\_amps,

ts\_shift\_delays,

NULL,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

printf("TS\_SHIFT: ricean\_factor %f\n",chan\_desc->ricean\_factor);

break;

case Rice1\_corr:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 0.1;

aoa = .03;

maxDoppler = 0;

if ((nb\_tx==2) && (nb\_rx==1)) {

R\_sqrt\_ptr2 = R\_sqrt\_21\_corr;

} else if ((nb\_tx==2) && (nb\_rx==2)) {

R\_sqrt\_ptr2 = R\_sqrt\_22\_corr;

} else

R\_sqrt\_ptr2 = NULL;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

R\_sqrt\_ptr2,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

1);

break;

case Rice1\_anticorr:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 0.1;

aoa = .03;

maxDoppler = 0;

if ((nb\_tx==2) && (nb\_rx==1)) {

R\_sqrt\_ptr2 = R\_sqrt\_21\_anticorr;

} else if ((nb\_tx==2) && (nb\_rx==2)) {

R\_sqrt\_ptr2 = R\_sqrt\_22\_anticorr;

} else

R\_sqrt\_ptr2 = NULL;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

R\_sqrt\_ptr2,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

1);

break;

case Rayleigh1\_orthogonal:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 1;

aoa = 0.03;

maxDoppler = 0;

if ((nb\_tx==2) && (nb\_rx==2)) {

R\_sqrt\_ptr2 = R\_sqrt\_22\_orthogonal;

} else

R\_sqrt\_ptr2 = NULL;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

R\_sqrt\_ptr2,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

case Rayleigh1\_orth\_eff\_ch\_TM4\_prec\_real:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 1;

aoa = 0.03;

maxDoppler = 0;

if ((nb\_tx==2) && (nb\_rx==2)) {

R\_sqrt\_ptr2 = R\_sqrt\_22\_orth\_eff\_ch\_TM4\_prec\_real;

} else

R\_sqrt\_ptr2 = NULL;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

R\_sqrt\_ptr2,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

1);

break;

case Rayleigh1\_orth\_eff\_ch\_TM4\_prec\_imag:

nb\_taps = 1;

Td = 0;

channel\_length = 1;

ricean\_factor = 1;

aoa = 0.03;

maxDoppler = 0;

if ((nb\_tx==2) && (nb\_rx==2)) {

R\_sqrt\_ptr2 = R\_sqrt\_22\_orth\_eff\_ch\_TM4\_prec\_imag;

} else

R\_sqrt\_ptr2 = NULL;

fill\_channel\_desc(chan\_desc,nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amp\_lin,

NULL,

R\_sqrt\_ptr2,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

case Rayleigh8\_orth\_eff\_ch\_TM4\_prec\_real:

if ((nb\_tx==2) && (nb\_rx==2)) {

R\_sqrt\_ptr2 = R\_sqrt\_22\_orth\_eff\_ch\_TM4\_prec\_real;

//R\_sqrt\_ptr2 = NULL;

} else

R\_sqrt\_ptr2 = NULL;

nb\_taps = 8;

Td = 0.8;

channel\_length = (int)11+2\*sampling\_rate\*Td;

ricean\_factor = 1;

aoa = .03;

maxDoppler = 0;

fill\_channel\_desc(chan\_desc,

nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amps\_lin,

NULL,

R\_sqrt\_ptr2,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

case Rayleigh8\_orth\_eff\_ch\_TM4\_prec\_imag:

nb\_taps = 8;

Td = 0.8;

channel\_length = (int)11+2\*sampling\_rate\*Td;

ricean\_factor = 1;

aoa = .03;

maxDoppler = 0;

if ((nb\_tx==2) && (nb\_rx==2)) {

R\_sqrt\_ptr2 = R\_sqrt\_22\_orth\_eff\_ch\_TM4\_prec\_imag;

} else

R\_sqrt\_ptr2 = NULL;

fill\_channel\_desc(chan\_desc,

nb\_tx,

nb\_rx,

nb\_taps,

channel\_length,

default\_amps\_lin,

NULL,

R\_sqrt\_ptr2,

Td,

sampling\_rate,

channel\_bandwidth,

ricean\_factor,

aoa,

forgetting\_factor,

maxDoppler,

channel\_offset,

path\_loss\_dB,

0);

break;

default:

LOG\_W(OCM,"channel model not yet supported\n");

free(chan\_desc);

return(NULL);

}

LOG\_D(OCM,"[CHANNEL] RF %f\n",chan\_desc->ricean\_factor);

for (i=0; i<chan\_desc->nb\_taps; i++)

LOG\_D(OCM,"[CHANNEL] tap %d: amp %f, delay %f\n",i,chan\_desc->amps[i], chan\_desc->. delays[i]);

chan\_desc->nb\_paths = 10;

return(chan\_desc);

}

channel\_desc\_t \*find\_channel\_desc\_fromname( char \*modelname ) {

for(int i=0; i<max\_chan; i++) {

if (defined\_channels[i] != NULL) {

if (strcmp(defined\_channels[i]->model\_name,modelname) == 0)

return defined\_channels[i];

}

}

LOG\_E(OCM,"Model %s not found \n", modelname);

return NULL;

}

void free\_channel\_desc\_scm(channel\_desc\_t \*ch) {

if (max\_chan != 0) defined\_channels[ch->chan\_idx]=NULL;

if(ch->free\_flags&CHANMODEL\_FREE\_AMPS)

free(ch->amps);

for (int i = 0; i<ch->nb\_tx\*ch->nb\_rx; i++) {

free(ch->ch[i]);

free(ch->chF[i]);

}

for (int i = 0; i<ch->nb\_taps; i++) {

free(ch->a[i]);

}

if(ch->free\_flags&CHANMODEL\_FREE\_DELAY)

free(ch->delays);

if(ch->free\_flags&CHANMODEL\_FREE\_RSQRT\_6)

for (int i = 0; i<6; i++)

free(ch->R\_sqrt[i]);

if(ch->free\_flags&CHANMODEL\_FREE\_RSQRT\_NTAPS)

for (int i = 0; i<ch->nb\_taps; i++)

free(ch->R\_sqrt[i]);

free(ch->R\_sqrt);

free(ch->ch);

free(ch->chF);

free(ch->a);

free(ch->model\_name);

free(ch);

}

void set\_channeldesc\_owner(channel\_desc\_t \*cdesc, uint32\_t module\_id) {

cdesc->module\_id=module\_id;

}

void set\_channeldesc\_name(channel\_desc\_t \*cdesc,char \*modelname) {

if(cdesc->model\_name != NULL)

free(cdesc->model\_name);

cdesc->model\_name=strdup(modelname);

}

int random\_channel(channel\_desc\_t \*desc, uint8\_t abstraction\_flag) {

double s;

int i,k,l,aarx,aatx;

struct complexd anew[desc->nb\_tx\*desc->nb\_rx];

struct complexd acorr[desc->nb\_tx\*desc->nb\_rx];

struct complexd phase, alpha, beta;

start\_meas(&desc->random\_channel);

for (i=0; i<(int)desc->nb\_taps; i++) {

for (aarx=0; aarx<desc->nb\_rx; aarx++) {

for (aatx=0; aatx<desc->nb\_tx; aatx++) {

anew[aarx+(aatx\*desc->nb\_rx)].r = sqrt(desc->ricean\_factor\*desc->amps[i]/2) \*. gaussdouble(0.0,1.0);

anew[aarx+(aatx\*desc->nb\_rx)].i = sqrt(desc->ricean\_factor\*desc->amps[i]/2) \*. gaussdouble(0.0,1.0);

if ((i==0) && (desc->ricean\_factor != 1.0)) {

if (desc->random\_aoa==1) {

desc->aoa = uniformrandom()\*2\*M\_PI;

}

phase.r = cos(M\_PI\*((aarx-aatx)\*sin(desc->aoa)));

phase.i = sin(M\_PI\*((aarx-aatx)\*sin(desc->aoa)));

anew[aarx+(aatx\*desc->nb\_rx)].r += phase.r \* sqrt(1.0-desc->ricean\_factor);

anew[aarx+(aatx\*desc->nb\_rx)].i += phase.i \* sqrt(1.0-desc->ricean\_factor);

}

if (desc->first\_run==1) {

cblas\_zcopy(desc->nb\_tx\*desc->nb\_rx, (void \*) acorr, 1, (void \*) desc->a[i], 1);

} else {

alpha.r = sqrt(1-desc->forgetting\_factor);

alpha.i = 0;

beta.r = sqrt(desc->forgetting\_factor);

beta.i = 0;

cblas\_zscal(desc->nb\_tx\*desc->nb\_rx, (void \*) &beta, (void \*) desc->a[i], 1);

cblas\_zaxpy(desc->nb\_tx\*desc->nb\_rx, (void \*) &alpha, (void \*) acorr, 1, (void \*). desc->a[i], 1);

}

}

if (desc->first\_run==1)

desc->first\_run = 0;

return (0);

}

double N\_RB2sampling\_rate(uint16\_t N\_RB) {

double sampling\_rate;

switch (N\_RB) {

case 6:

sampling\_rate = 1.92;

break;

case 25:

sampling\_rate = 7.68;

break;

case 50:

sampling\_rate = 15.36;

break;

case 100:

sampling\_rate = 30.72;

break;

default:

AssertFatal(1==0,"Unknown N\_PRB %d",N\_RB);

}

return(sampling\_rate);

}

double N\_RB2channel\_bandwidth(uint16\_t N\_RB) {

double channel\_bandwidth;

switch (N\_RB) {

case 6:

channel\_bandwidth = 1.25;

break;

case 25:

channel\_bandwidth = 5.00;

break;

case 50:

channel\_bandwidth = 10.00;

break;

case 100:

channel\_bandwidth = 20.00;

break;

default:

LOG\_E(OCM,"Unknown N\_PRB\n");

return(-1);

}

return(channel\_bandwidth);

}

#ifdef RANDOM\_CHANNEL\_MAIN

#define sampling\_rate 5.0

#define Td 2.0

main(int argc,char \*\*argv) {

double amps[8] = {.8,.2,.1,.04,.02,.01,.005};

struct complexd ch[(int)(1+2\*sampling\_rate\*Td)],phase;

int i;

randominit();

phase.x = 1.0;

phase.y = 0;

random\_channel(amps,Td, 8,sampling\_rate,ch,(double)1.0,&phase);

for (i=0;i<(11+2\*sampling\_rate\*Td);i++){

printf("%f + sqrt(-1)\*%f\n",ch[i].x,ch[i].y);

}

}

#endif

#### 运行启动、时延、可靠性：

#### 1、 openairinterface5g /executables/nr-ue.c

#define \_GNU\_SOURCE

#include <pthread.h>

#include <openair1/PHY/impl\_defs\_top.h>

#include "executables/nr-uesoftmodem.h"

#include "PHY/phy\_extern\_nr\_ue.h"

#include "PHY/INIT/phy\_init.h"

#include "NR\_MAC\_UE/mac\_proto.h"

#include "RRC/NR\_UE/rrc\_proto.h"

#include "SCHED\_NR\_UE/phy\_frame\_config\_nr.h"

#include "SCHED\_NR\_UE/defs.h"

#include "PHY/NR\_UE\_TRANSPORT/nr\_transport\_proto\_ue.h"

#include "executables/softmodem-common.h"

#include "LAYER2/nr\_pdcp/nr\_pdcp\_entity.h"

#include "SCHED\_NR\_UE/pucch\_uci\_ue\_nr.h"

#include "openair2/NR\_UE\_PHY\_INTERFACE/NR\_IF\_Module.h"

#define RX\_JOB\_ID 0x1010

#define TX\_JOB\_ID 100

typedef enum {

pss = 0,

pbch = 1,

si = 2

} sync\_mode\_t;

queue\_t nr\_rach\_ind\_queue;

static void \*NRUE\_phy\_stub\_standalone\_pnf\_task(void \*arg);

static size\_t dump\_L1\_UE\_meas\_stats(NR\_UE\_MAC\_INST\_t \*mac, PHY\_VARS\_NR\_UE \*ue, char \*output, size\_t max\_len)

{

int stroff = 0;

stroff += print\_meas\_log(&ue->phy\_proc\_tx, "L1 TX processing", NULL, NULL, output);

stroff += print\_meas\_log(&ue->ulsch\_encoding\_stats, "ULSCH encoding", NULL, NULL, output. + stroff);

stroff += print\_meas\_log(&ue->phy\_proc\_rx[0], "L1 RX processing t0", NULL, NULL, output +. stroff);

stroff += print\_meas\_log(&ue->phy\_proc\_rx[1], "L1 RX processing t1", NULL, NULL, output +. stroff);

stroff += print\_meas\_log(&ue->ue\_ul\_indication\_stats, "UL Indication——processSlotTX", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->rx\_pdsch\_stats, "PDSCH receiver", NULL, NULL, output +. stroff);

stroff += print\_meas\_log(&ue->dlsch\_decoding\_stats[0], "PDSCH decoding t0", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->dlsch\_decoding\_stats[1], "PDSCH decoding t1", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->dlsch\_deinterleaving\_stats, " -> Deinterleive", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->dlsch\_rate\_unmatching\_stats, " -> Rate Unmatch", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->dlsch\_ldpc\_decoding\_stats, " -> LDPC Decode", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->dlsch\_unscrambling\_stats, "PDSCH unscrambling", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->dlsch\_rx\_pdcch\_stats, "PDCCH handling", NULL, NULL, output. + stroff);

stroff += print\_meas\_log(&ue\_MAC\_stats, "NRUE\_phy\_stub\_standalone\_pnf\_task", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue\_dl\_indication\_stats, "ue\_dl\_indication\_stats", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue\_ul\_indication\_status, "ue\_ul\_indication\_status", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&pdcp\_run\_stats, "pdcp\_run\_stats", NULL, NULL, output + stroff);

stroff+=print\_meas\_log(&process\_queued\_nr\_nfapi\_msgs\_stats,"process\_queued\_nr\_nfapi\_msgs\_stats", NULL, NULL, output + stroff);

stroff+= print\_meas\_log(&ue->pbch\_procedures\_stats, "pbch\_procedures\_stats", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->pdcch\_procedures\_stats, "pdcch\_procedures\_stats", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->pdcch\_ack\_stats, "pdcch\_ack\_stats", NULL, NULL, output +. stroff);

stroff += print\_meas\_log(&ue->pdsch\_procedures\_stats, "pdsch\_procedures\_stats", NULL, NULL, output + stroff);

stroff += print\_meas\_log(&ue->pdsch\_ack\_stats, "pdsch\_ack\_stats", NULL, NULL, output +. stroff);

return stroff;

}

static void \*nrL1\_UE\_stats\_thread(void \*param)

{

PHY\_VARS\_NR\_UE \*ue = (PHY\_VARS\_NR\_UE \*) param;

const int max\_len = 16384;

char output[max\_len];

char filename[30];

snprintf(filename, 29, "nrL1\_UE\_stats-%d.log", ue->Mod\_id);

filename[29] = 0;

FILE \*fd = fopen(filename, "w");

AssertFatal(fd != NULL, "Cannot open %s\n", filename);

reset\_meas(&ue->phy\_proc\_rx[0]);

reset\_meas(&ue->phy\_proc\_rx[1]);

reset\_meas(&ue->dlsch\_decoding\_stats[0]);

reset\_meas(&ue->dlsch\_decoding\_stats[1]);

reset\_meas(&ue->dlsch\_deinterleaving\_stats);

reset\_meas(&ue->dlsch\_rate\_unmatching\_stats);

reset\_meas(&ue->dlsch\_unscrambling\_stats);

reset\_meas(&ue->pbch\_procedures\_stats);

reset\_meas(&ue->pdcch\_procedures\_stats);

reset\_meas(&ue->pdcch\_ack\_stats);

reset\_meas(&ue->pdsch\_procedures\_stats);

reset\_meas(&ue->pdsch\_ack\_stats);

while (!oai\_exit) {

sleep(1);

const int len = dump\_L1\_UE\_meas\_stats(mac, ue, output, max\_len);

AssertFatal(len < max\_len, "exceeded length\n");

fwrite(output, len + 1, 1, fd); // + 1 for terminating NULL byte

fflush(fd);

fseek(fd, 0, SEEK\_SET);

}

fclose(fd);

return NULL;

}

void init\_nr\_ue\_vars(PHY\_VARS\_NR\_UE \*ue,

uint8\_t UE\_id,

uint8\_t abstraction\_flag)

{

int nb\_connected\_gNB = 1, gNB\_id;

ue->Mod\_id = UE\_id;

ue->mac\_enabled = 1;

ue->if\_inst = nr\_ue\_if\_module\_init(0);

ue->dci\_thres = 0;

for (gNB\_id = 0; gNB\_id < nb\_connected\_gNB; gNB\_id++){

ue->UE\_mode[gNB\_id]=NOT\_SYNCHED; ue->prach\_resources[gNB\_id]=(NR\_PRACH\_RESOURCES\_t\*)malloc16\_clear(sizeof(NR\_PRACH\_RESOURCES\_t));

}

init\_nr\_ue\_signal(ue, nb\_connected\_gNB);

init\_nr\_ue\_transport(ue);

init\_N\_TA\_offset(ue);

}

void init\_nrUE\_standalone\_thread(int ue\_idx)

{

int standalone\_tx\_port = 3611 + ue\_idx \* 2;

int standalone\_rx\_port = 3612 + ue\_idx \* 2;

nrue\_init\_standalone\_socket(standalone\_tx\_port, standalone\_rx\_port);

NR\_UE\_MAC\_INST\_t \*mac = get\_mac\_inst(0);

pthread\_mutex\_init(&mac->mutex\_dl\_info, NULL);

pthread\_t thread;

if (pthread\_create(&thread, NULL, nrue\_standalone\_pnf\_task, NULL) != 0) {

LOG\_E(NR\_MAC, "pthread\_create failed for calling nrue\_standalone\_pnf\_task");

}

pthread\_setname\_np(thread, "oai:nrue-stand");

pthread\_t phy\_thread;

if (pthread\_create(&phy\_thread, NULL, NRUE\_phy\_stub\_standalone\_pnf\_task, NULL) != 0) {

LOG\_E(NR\_MAC, "pthread\_create failed for calling NRUE\_phy\_stub\_standalone\_pnf\_task");

}

pthread\_setname\_np(phy\_thread, "oai:nrue-stand-phy");

}

static void L1\_nsa\_prach\_procedures(frame\_t frame, int slot, fapi\_nr\_ul\_config\_prach\_pdu \*prach\_pdu)

{

NR\_UE\_MAC\_INST\_t \*mac = get\_mac\_inst(0);

nfapi\_nr\_rach\_indication\_t \*rach\_ind = CALLOC(1, sizeof(\*rach\_ind));

rach\_ind->sfn = frame;

rach\_ind->slot = slot;

rach\_ind->header.message\_id = NFAPI\_NR\_PHY\_MSG\_TYPE\_RACH\_INDICATION;

uint8\_t pdu\_index = 0;

rach\_ind->pdu\_list = CALLOC(1, sizeof(\*rach\_ind->pdu\_list));

rach\_ind->number\_of\_pdus = 1;

rach\_ind->pdu\_list[pdu\_index].phy\_cell\_id = prach\_pdu->phys\_cell\_id;

rach\_ind->pdu\_list[pdu\_index].symbol\_index = prach\_pdu->prach\_start\_symbol;

rach\_ind->pdu\_list[pdu\_index].slot\_index = prach\_pdu->prach\_slot;

rach\_ind->pdu\_list[pdu\_index].freq\_index = prach\_pdu->num\_ra;

rach\_ind->pdu\_list[pdu\_index].avg\_rssi = 128;

rach\_ind->pdu\_list[pdu\_index].avg\_snr = 0xff;

rach\_ind->pdu\_list[pdu\_index].num\_preamble = 1;

const int num\_p = rach\_ind->pdu\_list[pdu\_index].num\_preamble;

rach\_ind->pdu\_list[pdu\_index].preamble\_list=calloc(num\_p,sizeof(nfapi\_nr\_prach\_indication\_preamble\_t));

uint8\_t preamble\_index = get\_softmodem\_params()->nsa ?

mac->ra.rach\_ConfigDedicated->cfra->resources.choice.ssb->ssb\_ResourceList.list.array[0]->ra\_PreambleIndex :

mac->ra.ra\_PreambleIndex;

rach\_ind->pdu\_list[pdu\_index].preamble\_list[0].preamble\_index = preamble\_index;

rach\_ind->pdu\_list[pdu\_index].preamble\_list[0].timing\_advance = 0;

rach\_ind->pdu\_list[pdu\_index].preamble\_list[0].preamble\_pwr = 0xffffffff;

if (!put\_queue(&nr\_rach\_ind\_queue, rach\_ind))

{

for (int pdu\_index = 0; pdu\_index < rach\_ind->number\_of\_pdus; pdu\_index++)

{

free(rach\_ind->pdu\_list[pdu\_index].preamble\_list);

}

free(rach\_ind->pdu\_list);

free(rach\_ind);

}

LOG\_D(NR\_MAC, "We have successfully filled the rach\_ind queue with the recently filled rach. ind\n");

}

static void process\_queued\_nr\_nfapi\_msgs(NR\_UE\_MAC\_INST\_t \*mac, int sfn\_slot)

{

nfapi\_nr\_rach\_indication\_t\*rach\_ind=unqueue\_matching(&nr\_rach\_ind\_queue, MAX\_QUEUE\_SIZE, sfn\_slot\_matcher, &sfn\_slot);

nfapi\_nr\_dl\_tti\_request\_t \*dl\_tti\_request = get\_queue(&nr\_dl\_tti\_req\_queue);

nfapi\_nr\_ul\_dci\_request\_t \*ul\_dci\_request = get\_queue(&nr\_ul\_dci\_req\_queue);

start\_meas(&mac->nfapi\_stats);

for (int i = 0; i < NR\_MAX\_HARQ\_PROCESSES; i++) {

LOG\_D(NR\_MAC, "Try to get a ul\_tti\_req by matching CRC active SFN %d/SLOT %d from queue with %lu items\n",

nfapi\_nr\_ul\_tti\_request\_t \*ul\_tti\_request\_crc = unqueue\_matching(&nr\_ul\_tti\_req\_queue, MAX\_QUEUE\_SIZE, sfn\_slot\_matcher, &mac->nr\_ue\_emul\_l1.harq[i].active\_ul\_harq\_sfn\_slot);

if (ul\_tti\_request\_crc && ul\_tti\_request\_crc->n\_pdus > 0)

{

check\_and\_process\_dci(NULL, NULL, NULL, ul\_tti\_request\_crc);

}

}

stop\_meas(&mac->nfapi\_stats);

if (rach\_ind && rach\_ind->number\_of\_pdus > 0)

{

NR\_UL\_IND\_t UL\_INFO = {

.rach\_ind = \*rach\_ind,

};

send\_nsa\_standalone\_msg(&UL\_INFO, rach\_ind->header.message\_id);

for (int i = 0; i < rach\_ind->number\_of\_pdus; i++)

{

free(rach\_ind->pdu\_list[i].preamble\_list);

}

free(rach\_ind->pdu\_list);

free(rach\_ind);

nr\_Msg1\_transmitted(0, 0, NFAPI\_SFNSLOT2SFN(sfn\_slot), 0);

}

if (dl\_tti\_request)

{

int dl\_tti\_sfn\_slot = NFAPI\_SFNSLOT2HEX(dl\_tti\_request->SFN, dl\_tti\_request->Slot);

nfapi\_nr\_tx\_data\_request\_t \*tx\_data\_request = unqueue\_matching(&nr\_tx\_req\_queue, MAX\_QUEUE\_SIZE, sfn\_slot\_matcher, &dl\_tti\_sfn\_slot);

if (!tx\_data\_request)

{

LOG\_E(NR\_MAC, "[%d %d] No corresponding tx\_data\_request for given dl\_tti\_request. sfn/slot\n",

NFAPI\_SFNSLOT2SFN(dl\_tti\_sfn\_slot), NFAPI\_SFNSLOT2SLOT(dl\_tti\_sfn\_slot));

if (get\_softmodem\_params()->nsa)

save\_nr\_measurement\_info(dl\_tti\_request);

free(dl\_tti\_request);

dl\_tti\_request = NULL;

}

if (get\_softmodem\_params()->nsa)

save\_nr\_measurement\_info(dl\_tti\_request);

check\_and\_process\_dci(dl\_tti\_request, tx\_data\_request, NULL, NULL);

}

else

{

AssertFatal(false, "We dont have PDUs in either dl\_tti %d or tx\_req %d\n", dl\_tti\_request->dl\_tti\_request\_body.nPDUs,tx\_data\_request->Number\_of\_PDUs);

}

}

if (ul\_dci\_request && ul\_dci\_request->numPdus > 0)

{

check\_and\_process\_dci(NULL, NULL, ul\_dci\_request, NULL);

}

}

static void check\_nr\_prach(NR\_UE\_MAC\_INST\_t \*mac, nr\_uplink\_indication\_t \*ul\_info, NR\_PRACH\_RESOURCES\_t \*prach\_resources)

{

fapi\_nr\_ul\_config\_request\_t \*ul\_config = get\_ul\_config\_request(mac, ul\_info->slot\_tx);

if (!ul\_config)

{

LOG\_E(NR\_MAC, "mac->ul\_config is null! \n");

return;

}

if (mac->ra.ra\_state != RA\_SUCCEEDED)

{

AssertFatal(ul\_config->number\_pdus<sizeof(ul\_config->ul\_config\_list)/sizeof(ul\_config->ul\_config\_list[0]),

fapi\_nr\_ul\_config\_prach\_pdu\*prach\_pdu=&ul\_config->ul\_config\_list[ul\_config->number\_pdus].prach\_config\_pdu;

uint8\_t nr\_prach = nr\_ue\_get\_rach(prach\_resources,

prach\_pdu,

ul\_info->module\_id,

ul\_info->cc\_id,

ul\_info->frame\_tx,

ul\_info->gNB\_index,

ul\_info->slot\_tx);

if (nr\_prach == 1)

{

L1\_nsa\_prach\_procedures(ul\_info->frame\_tx, ul\_info->slot\_tx, prach\_pdu);

ul\_config->number\_pdus = 0;

ul\_info->ue\_sched\_mode = SCHED\_ALL;

}

else if (nr\_prach == 2)

{

LOG\_I(NR\_PHY, "In %s: [UE %d] RA completed, setting UE mode to PUSCH\n", \_\_FUNCTION\_\_, ul\_info->module\_id);

}

else if(nr\_prach == 3)

{

LOG\_I(NR\_PHY, "In %s: [UE %d] RA failed, setting UE mode to PRACH\n", \_\_FUNCTION\_\_, ul\_info->module\_id);

}

}

}

static void \*NRUE\_phy\_stub\_standalone\_pnf\_task(void \*arg)

{

LOG\_I(MAC, "Clearing Queues\n");

reset\_queue(&nr\_rach\_ind\_queue);

reset\_queue(&nr\_rx\_ind\_queue);

reset\_queue(&nr\_crc\_ind\_queue);

reset\_queue(&nr\_uci\_ind\_queue);

reset\_queue(&nr\_dl\_tti\_req\_queue);

reset\_queue(&nr\_tx\_req\_queue);

reset\_queue(&nr\_ul\_dci\_req\_queue);

reset\_queue(&nr\_ul\_tti\_req\_queue);

NR\_PRACH\_RESOURCES\_t prach\_resources;

memset(&prach\_resources, 0, sizeof(prach\_resources));

NR\_UL\_TIME\_ALIGNMENT\_t ul\_time\_alignment;

memset(&ul\_time\_alignment, 0, sizeof(ul\_time\_alignment));

int last\_sfn\_slot = -1;

uint16\_t sfn\_slot = 0;

module\_id\_t mod\_id = 0;

NR\_UE\_MAC\_INST\_t \*mac = get\_mac\_inst(mod\_id);

for (int i = 0; i < NR\_MAX\_HARQ\_PROCESSES; i++) {

mac->nr\_ue\_emul\_l1.harq[i].active = false;

mac->nr\_ue\_emul\_l1.harq[i].active\_ul\_harq\_sfn\_slot = -1;

}

while (!oai\_exit)

{

if (sem\_wait(&sfn\_slot\_semaphore) != 0)

{

LOG\_E(NR\_MAC, "sem\_wait() error\n");

abort();

}

uint16\_t \*slot\_ind = get\_queue(&nr\_sfn\_slot\_queue);

nr\_phy\_channel\_params\_t \*ch\_info = get\_queue(&nr\_chan\_param\_queue);

if (!slot\_ind && !ch\_info)

{

LOG\_D(MAC, "get nr\_sfn\_slot\_queue and nr\_chan\_param\_queue == NULL!\n");

continue;

}

if (slot\_ind) {

sfn\_slot = \*slot\_ind;

free\_and\_zero(slot\_ind);

}

else if (ch\_info) {

sfn\_slot = ch\_info->sfn\_slot;

free\_and\_zero(ch\_info);

}

frame\_t frame = NFAPI\_SFNSLOT2SFN(sfn\_slot);

int slot = NFAPI\_SFNSLOT2SLOT(sfn\_slot);

if (sfn\_slot == last\_sfn\_slot)

{

LOG\_D(NR\_MAC, "repeated sfn\_sf = %d.%d\n",frame, slot);

continue;

}

last\_sfn\_slot = sfn\_slot;

LOG\_D(NR\_MAC, "The received sfn/slot [%d %d] from proxy\n",frame, slot);

if (get\_softmodem\_params()->sa && mac->mib == NULL)

{

LOG\_D(NR\_MAC, "We haven't gotten MIB. Lets see if we received it\n");

nr\_ue\_dl\_indication(&mac->dl\_info, &ul\_time\_alignment);

process\_queued\_nr\_nfapi\_msgs(mac, sfn\_slot);

}

if (mac->scc == NULL && mac->scc\_SIB == NULL)

{

LOG\_D(MAC, "[NSA] mac->scc == NULL and [SA] mac->scc\_SIB == NULL!\n");

continue;

}

mac->ra.generate\_nr\_prach = 0;

int CC\_id = 0;

uint8\_t gNB\_id = 0;

nr\_uplink\_indication\_t ul\_info;

int slots\_per\_frame = 20; //30 kHZ subcarrier spacing

int slot\_ahead = 2; // TODO: Make this dynamic

ul\_info.cc\_id = CC\_id;

ul\_info.gNB\_index = gNB\_id;

ul\_info.module\_id = mod\_id;

ul\_info.frame\_rx = frame;

ul\_info.slot\_rx = slot;

ul\_info.slot\_tx = (slot + slot\_ahead) % slots\_per\_frame;

ul\_info.frame\_tx = (ul\_info.slot\_rx + slot\_ahead >= slots\_per\_frame) ? ul\_info.frame\_rx + 1 : ul\_info.frame\_rx;

ul\_info.ue\_sched\_mode = SCHED\_ALL;

if (pthread\_mutex\_lock(&mac->mutex\_dl\_info)) abort();

memset(&mac->dl\_info, 0, sizeof(mac->dl\_info));

mac->dl\_info.cc\_id = CC\_id;

mac->dl\_info.gNB\_index = gNB\_id;

mac->dl\_info.module\_id = mod\_id;

mac->dl\_info.frame = frame;

mac->dl\_info.slot = slot;

mac->dl\_info.thread\_id = 0;

mac->dl\_info.dci\_ind = NULL;

mac->dl\_info.rx\_ind = NULL;

if(is\_nr\_DL\_slot(get\_softmodem\_params()->nsa?mac->scc->tdd\_UL\_DL\_ConfigurationCommon : mac->scc\_SIB->tdd\_UL\_DL\_ConfigurationCommon, ul\_info.slot\_rx))

{

nr\_ue\_dl\_indication(&mac->dl\_info, &ul\_time\_alignment);

}

if(is\_nr\_UL\_slot(get\_softmodem\_params()->nsa?mac->scc->tdd\_UL\_DL\_ConfigurationCommon: mac->scc\_SIB->tdd\_UL\_DL\_ConfigurationCommon, ul\_info.slot\_tx, mac->frame\_type))

{

LOG\_D(NR\_MAC, "Slot %d. calling nr\_ue\_ul\_ind() and nr\_ue\_pucch\_scheduler() f rom %s\n", ul\_info.slot\_tx, \_\_FUNCTION\_\_);

nr\_ue\_scheduler(NULL, &ul\_info);

nr\_ue\_prach\_scheduler(mod\_id, ul\_info.frame\_tx, ul\_info.slot\_tx, ul\_info.thread\_id);

nr\_ue\_pucch\_scheduler(mod\_id, ul\_info.frame\_tx, ul\_info.slot\_tx, ul\_info.thread\_id);

check\_nr\_prach(mac, &ul\_info, &prach\_resources);

}

if (!IS\_SOFTMODEM\_NOS1 && get\_softmodem\_params()->sa) {

NR\_UE\_MAC\_INST\_t \*mac = get\_mac\_inst(0);

protocol\_ctxt\_t ctxt;

PROTOCOL\_CTXT\_SET\_BY\_MODULE\_ID(&ctxt, 0, ENB\_FLAG\_NO, mac->crnti, frame, slot, 0);

pdcp\_run(&ctxt);

}

process\_queued\_nr\_nfapi\_msgs(mac, sfn\_slot);

}

return NULL;

}

static void UE\_synch(void \*arg) {

syncData\_t \*syncD=(syncData\_t \*) arg;

int i, hw\_slot\_offset;

PHY\_VARS\_NR\_UE \*UE = syncD->UE;

sync\_mode\_t sync\_mode = pbch;

static int freq\_offset=0;

UE->is\_synchronized = 0;

if (UE->UE\_scan == 0) {

for (i=0; i<openair0\_cfg[UE->rf\_map.card].rx\_num\_channels; i++) {

LOG\_I( PHY, "[SCHED][UE] Check absolute frequency DL %f, UL %f (RF card %d, oai\_exit %d, channel %d, rx\_num\_channels %d)\n",

openair0\_cfg[UE->rf\_map.card].rx\_freq[UE->rf\_map.chain+i],

openair0\_cfg[UE->rf\_map.card].tx\_freq[UE->rf\_map.chain+i],

UE->rf\_map.card,

oai\_exit,

i,

openair0\_cfg[0].rx\_num\_channels);

}

} else {

LOG\_E(PHY,"Fixme!\n");

}

case pbch:

LOG\_I(PHY, "[UE thread Synch] Running Initial Synch (mode %d)\n",UE->mode);

uint64\_t dl\_carrier, ul\_carrier;

nr\_get\_carrier\_frequencies(UE, &dl\_carrier, &ul\_carrier);

if (nr\_initial\_sync(&syncD->proc, UE, 2, get\_softmodem\_params()->sa) == 0) {

freq\_offset = UE->common\_vars.freq\_offset; // frequency offset computed with pss in initial sync

hw\_slot\_offset = ((UE->rx\_offset<<1) / UE->frame\_parms.samples\_per\_subframe \* UE->frame\_parms.slots\_per\_subframe)+round((float)((UE->rx\_offset<<1)% UE->frame\_parms.samples\_per\_subframe)/UE->frame\_parms.samples\_per\_slot0);

nr\_rf\_card\_config\_freq(&openair0\_cfg[UE->rf\_map.card],ul\_carrier,dl\_carrier,freq\_offet);

LOG\_I(PHY,"Got synch: hw\_slot\_offset %d, carrier off %d Hz, rxgain %f (DL %f Hz, UL %f Hz)\n",hw\_slot\_offset,freq\_offset,openair0\_cfg[UE->rf\_map.card].rx\_gain[0],openair0\_cfg [UE->rf\_map.card].rx\_freq[0], openair0\_cfg[UE->rf\_map.card].tx\_freq[0]);

UE->rfdevice.trx\_set\_freq\_func(&UE->rfdevice,&openair0\_cfg[0],0);

if (UE->UE\_scan\_carrier == 1) {

UE->UE\_scan\_carrier = 0;

} else {

UE->is\_synchronized = 1;

}

}

else {

if (UE->UE\_scan\_carrier == 1) {

if (freq\_offset >= 0)

freq\_offset += 100;

freq\_offset \*= -1；

LOG\_I(PHY, "Initial sync failed: trying carrier off %d Hz\n", freq\_offset);

UE->rfdevice.trx\_set\_freq\_func(&UE->rfdevice,&openair0\_cfg[0],0);

}

break;

}

}

}

void processSlotTX(void \*arg) {

nr\_rxtx\_thread\_data\_t \*rxtxD = (nr\_rxtx\_thread\_data\_t \*) arg;

UE\_nr\_rxtx\_proc\_t \*proc = &rxtxD->proc;

PHY\_VARS\_NR\_UE\*UE= rxtxD->UE;

fapi\_nr\_config\_request\_t \*cfg = &UE->nrUE\_config;

int tx\_slot\_type = nr\_ue\_slot\_select(cfg, proc->frame\_tx, proc->nr\_slot\_tx);

uint8\_t gNB\_id = 0;

if (tx\_slot\_type == NR\_UPLINK\_SLOT || tx\_slot\_type == NR\_MIXED\_SLOT){

if(UE->if\_inst != NULL && UE->if\_inst->ul\_indication != NULL) {

start\_meas(&UE->ue\_ul\_indication\_stats);

nr\_uplink\_indication\_t ul\_indication;

memset((void\*)&ul\_indication, 0, sizeof(ul\_indication));

ul\_indication.module\_id = UE->Mod\_id;

ul\_indication.gNB\_index = gNB\_id;

ul\_indication.cc\_id = UE->CC\_id;

ul\_indication.frame\_rx = proc->frame\_rx;

ul\_indication.slot\_rx = proc->nr\_slot\_rx;

ul\_indication.frame\_tx = proc->frame\_tx;

ul\_indication.slot\_tx = proc->nr\_slot\_tx;

ul\_indication.thread\_id = proc->thread\_id;

ul\_indication.ue\_sched\_mode = rxtxD->ue\_sched\_mode;

UE->if\_inst->ul\_indication(&ul\_indication);

stop\_meas(&UE->ue\_ul\_indication\_stats);

}

if (rxtxD->ue\_sched\_mode != NOT\_PUSCH) {

(UE,proc,0);

}

}

}

void processSlotRX(void \*arg) {

nr\_rxtx\_thread\_data\_t \*rxtxD = (nr\_rxtx\_thread\_data\_t \*) arg;

UE\_nr\_rxtx\_proc\_t \*proc = &rxtxD->proc;

PHY\_VARS\_NR\_UE \*UE = rxtxD->UE;

fapi\_nr\_config\_request\_t \*cfg = &UE->nrUE\_config;

int rx\_slot\_type = nr\_ue\_slot\_select(cfg, proc->frame\_rx, proc->nr\_slot\_rx);

int tx\_slot\_type = nr\_ue\_slot\_select(cfg, proc->frame\_tx, proc->nr\_slot\_tx);

uint8\_t gNB\_id = 0;

NR\_UE\_PDCCH\_CONFIG phy\_pdcch\_config={0};

static double cpuf = 0.0;

cpuf = get\_cpu\_freq\_GHz();

time\_stats\_t rx\_procedures\_stats;

if (IS\_SOFTMODEM\_NOS1 || get\_softmodem\_params()->sa) {

if (proc->nr\_slot\_rx % UE->frame\_parms.slots\_per\_subframe == 0) {

void nr\_rlc\_tick(int frame, int subframe);

void nr\_pdcp\_tick(int frame, int subframe);

nr\_rlc\_tick(proc->frame\_rx, proc->nr\_slot\_rx / UE->frame\_parms.slots\_per\_subframe);

nr\_pdcp\_tick(proc->frame\_rx, proc->nr\_slot\_rx / UE->frame\_parms.slots\_per\_subframe);

}

}

if (rx\_slot\_type == NR\_DOWNLINK\_SLOT || rx\_slot\_type == NR\_MIXED\_SLOT){

if(UE->if\_inst != NULL && UE->if\_inst->dl\_indication != NULL) {

nr\_downlink\_indication\_t dl\_indication;

nr\_fill\_dl\_indication(&dl\_indication, NULL, NULL, proc, UE, gNB\_id, &phy\_pdcch\_config);

UE->if\_inst->dl\_indication(&dl\_indication, NULL);

}

start\_meas(&rx\_procedures\_stats);phy\_procedures\_nrUE\_RX(UE,proc,gNB\_id,get\_nrUE\_par. ams()->nr\_dlsch\_parallel, &phy\_pdcch\_config, &rxtxD->txFifo);

stop\_meas(&rx\_procedures\_stats);

if (cpumeas(CPUMEAS\_GETSTATE)) {

LOG\_D(PHY, "In %s: slot %d, time %5.2f\n", \_\_FUNCTION\_\_, proc->nr\_slot\_rx, rx\_procedures\_stats.p\_time/(cpuf \* 1000.0));

}

if(IS\_SOFTMODEM\_NOS1 || get\_softmodem\_params()->sa){

NR\_UE\_MAC\_INST\_t \*mac = get\_mac\_inst(0);

protocol\_ctxt\_t ctxt;

PROTOCOL\_CTXT\_SET\_BY\_MODULE\_ID(&ctxt, UE->Mod\_id, ENB\_FLAG\_NO, mac->crnti, proc->frame\_rx, proc->nr\_slot\_rx, 0);

pdcp\_run(&ctxt);

}

} else {

rxtxD->ue\_sched\_mode = SCHED\_ALL;

processSlotTX(rxtxD);

}

if (tx\_slot\_type == NR\_UPLINK\_SLOT || tx\_slot\_type == NR\_MIXED\_SLOT){

if (UE->UE\_mode[gNB\_id] <= PUSCH) {

if (get\_softmodem\_params()->usim\_test==0) {

pucch\_procedures\_ue\_nr(UE, gNB\_id, proc);

}

LOG\_D(PHY, "Sending Uplink data \n");

nr\_ue\_pusch\_common\_procedures(UE,proc->nr\_slot\_tx,&UE->frame\_parms,UE->frame\_parms.nb\_antennas\_tx);

}

if (UE->UE\_mode[gNB\_id] > NOT\_SYNCHED && UE->UE\_mode[gNB\_id] < PUSCH) {

nr\_ue\_prach\_procedures(UE, proc, gNB\_id);

}

LOG\_D(PHY,"\*\*\*\*\*\* end TX-Chain for AbsSubframe %d.%d \*\*\*\*\*\*\n", proc->frame\_tx, proc->nr\_slot\_tx);

}

ue\_ta\_procedures(UE, proc->nr\_slot\_tx, proc->frame\_tx);

}

void \*UE\_thread(void \*arg) {

PHY\_VARS\_NR\_UE \*UE = (PHY\_VARS\_NR\_UE \*) arg;

openair0\_timestamp timestamp, writeTimestamp;

void \*rxp[NB\_ANTENNAS\_RX], \*txp[NB\_ANTENNAS\_TX];

int start\_rx\_stream = 0;

AssertFatal(0== i(&(UE->rfdevice), &openair0\_cfg[0]), "");

UE->rfdevice.host\_type = RAU\_HOST;

UE->lost\_sync = 0;

UE->is\_synchronized = 0;

AssertFatal(UE->rfdevice.trx\_start\_func(&UE->rfdevice) == 0, "Could not start the device\n");

notifiedFIFO\_t nf;

initNotifiedFIFO(&nf);

notifiedFIFO\_t freeBlocks;

initNotifiedFIFO\_nothreadSafe(&freeBlocks);

int nbSlotProcessing=0;

int thread\_idx=0;

NR\_UE\_MAC\_INST\_t \*mac = get\_mac\_inst(0);

int timing\_advance = UE->timing\_advance;

bool syncRunning=false;

const int = UE->frame\_parms.slots\_per\_frame;

int absolute\_slot=0, decoded\_frame\_rx=INT\_MAX, trashed\_frames=0;

for (int i=0; i<NR\_RX\_NB\_TH+1; i++) {

notifiedFIFO\_elt\_t\*newElt=newNotifiedFIFO\_elt(sizeof(nr\_rxtx\_thread\_data\_t),RX\_JOB\_ID,&nf,processSlotRX);

nr\_rxtx\_thread\_data\_t \*curMsg=(nr\_rxtx\_thread\_data\_t \*)NotifiedFifoData(newElt);

initNotifiedFIFO(&curMsg->txFifo);

pushNotifiedFIFO\_nothreadSafe(&freeBlocks, newElt);

}

while (!oai\_exit) {

if (UE->lost\_sync) {

int nb = abortTpool(&(get\_nrUE\_params()->Tpool),RX\_JOB\_ID);

nb += abortNotifiedFIFO(&nf, RX\_JOB\_ID);

LOG\_I(PHY,"Number of aborted slots %d\n",nb);

for (int i=0; i<nb; i++)

pushNotifiedFIFO\_nothreadSafe(&freeBlocks, newNotifiedFIFO\_elt(sizeof(nr\_rxtx\_thread\_data\_t), RX\_JOB\_ID,&nf,processSlotRX));

nbSlotProcessing = 0;

UE->is\_synchronized = 0;

UE->lost\_sync = 0;

}

if (syncRunning) {

notifiedFIFO\_elt\_t \*res=tryPullTpool(&nf,&(get\_nrUE\_params()->Tpool));

if (res) {

syncRunning=false;

syncData\_t \*tmp=(syncData\_t \*)NotifiedFifoData(res);

if (UE->is\_synchronized) {

decoded\_frame\_rx=(((mac->mib->systemFrameNumber.buf[0] >>mac->mib->systemFrameNumber.bits\_unused)<<4) | tmp->proc.decoded\_frame\_rx);

decoded\_frame\_rx=(decoded\_frame\_rx + UE->init\_sync\_frame + trashed\_frames) % MAX\_FRAME\_NUMBER;

}

delNotifiedFIFO\_elt(res);

start\_rx\_stream=0;

} else {

readFrame(UE, &timestamp, true);

trashed\_frames+=2;

continue;

}

}

AssertFatal( !syncRunning, "At this point synchronization can't be running\n");

if (!UE->is\_synchronized) {

readFrame(UE, &timestamp, false);

notifiedFIFO\_elt\_t \*Msg=newNotifiedFIFO\_elt(sizeof(syncData\_t),0,&nf,UE\_synch);

syncData\_t \*syncMsg=(syncData\_t \*)NotifiedFifoData(Msg);

syncMsg->UE=UE;

memset(&syncMsg->proc, 0, sizeof(syncMsg->proc));

pushTpool(&(get\_nrUE\_params()->Tpool), Msg);

trashed\_frames=0;

syncRunning=true;

continue;

}

if (start\_rx\_stream==0) {

start\_rx\_stream=1;

syncInFrame(UE, &timestamp);

UE->rx\_offset=0;

UE->time\_sync\_cell=0;

AssertFatal (UE->frame\_parms.ofdm\_symbol\_size+UE->frame\_parms.nb\_prefix\_samples0 ==UE->rfdevice.trx\_read\_func(&UE->rfdevice,&timestamp,(void\*\*)UE->common\_vars.rxdata,UE->frame\_parms.ofdm\_symbol\_size+UE->frame\_parms.nb\_prefix\_samples0,UE->fram \_parms.nb\_antennas\_rx),"");

decoded\_frame\_rx++;

absolute\_slot=decoded\_frame\_rx\*nb\_slot\_frame -1;

continue;

}

absolute\_slot++;

thread\_idx = absolute\_slot % NR\_RX\_NB\_TH;

int slot\_nr = absolute\_slot % nb\_slot\_frame;

notifiedFIFO\_elt\_t \*msgToPush;

AssertFatal((msgToPush=pullNotifiedFIFO\_nothreadSafe(&freeBlocks)) != NULL,"chained list failure");

nr\_rxtx\_thread\_data\_t \*curMsg=(nr\_rxtx\_thread\_data\_t \*)NotifiedFifoData(msgToPush);

curMsg->UE=UE;

curMsg->proc.thread\_id = thread\_idx;

curMsg->proc.CC\_id = UE->CC\_id;

curMsg->proc.nr\_slot\_rx = slot\_nr;

curMsg->proc.nr\_slot\_tx = (absolute\_slot + DURATION\_RX\_TO\_TX) % nb\_slot\_frame;

curMsg->proc.frame\_rx = (absolute\_slot/nb\_slot\_frame) % MAX\_FRAME\_NUMBER;

curMsg->proc.decoded\_frame\_rx=-1;

int firstSymSamp = get\_firstSymSamp(slot\_nr, &UE->frame\_parms);

for (int i=0; i<UE->frame\_parms.nb\_antennas\_rx; i++)

rxp[i] = (void \*)&UE->common\_vars.rxdata[i][firstSymSamp+

UE->frame\_parms.get\_samples\_slot\_timestamp(slot\_nr,&UE->frame\_parms,0)];

for (int i=0; i<UE->frame\_parms.nb\_antennas\_tx; i++)

txp[i]=(void\*)&UE->common\_vars.txdata[i][UE->frame\_parms.get\_samples\_slot\_timestamp(((slot\_nr+DURATION\_RX\_TO\_TX-NR\_RX\_NB\_TH)%nb\_slot\_frame),&UE->frame\_parms,0)];

int readBlockSize, writeBlockSize;、

if (slot\_nr<(nb\_slot\_frame - 1)) {

readBlockSize=get\_readBlockSize(slot\_nr, &UE->frame\_parms);

writeBlockSize=UE->frame\_parms.get\_samples\_per\_slot((slot\_nr+ DURATION\_RX\_TO\_TX- NR\_RX\_NB\_TH) % nb\_slot\_frame, &UE->frame\_parms);

} else {

UE->rx\_offset\_diff = computeSamplesShift(UE);

readBlockSize=get\_readBlockSize(slot\_nr, &UE->frame\_parms) - UE->rx\_offset\_diff;

writeBlockSize=UE->frame\_parms.get\_samples\_per\_slot((slot\_nr + DURATION\_RX\_TO\_TX - NR\_RX\_NB\_TH) % nb\_slot\_frame, &UE->frame\_parms)- UE->rx\_offset\_diff;

}

AssertFatal(readBlockSize ==UE->rfdevice.trx\_read\_func(&UE->rfdevice, &timestamp, rxp, readBlockSize, UE->frame\_parms.nb\_antennas\_rx),"");

UE->frame\_parms.nb\_antennas\_rx),"");

} else

LOG\_E(PHY,"can't compensate: diff =%d\n", first\_symbols);

}

curMsg->proc.timestamp\_tx = timestamp+

UE->frame\_parms.get\_samples\_slot\_timestamp(slot\_nr,&UE->frame\_parms,DURATION\_RX\_TO\_TX) - firstSymSamp;

notifiedFIFO\_elt\_t \*res;

while (nbSlotProcessing >= NR\_RX\_NB\_TH) {

res=pullTpool(&nf, &(get\_nrUE\_params()->Tpool));

nbSlotProcessing--;

nr\_rxtx\_thread\_data\_t \*tmp=(nr\_rxtx\_thread\_data\_t \*)res->msgData;

if (tmp->proc.decoded\_frame\_rx != -1)

decoded\_frame\_rx=(((mac->mib->systemFrameNumber.buf[0]>>mac->mib->systemFrameNumber.bits\_unused)<<4) | tmp->proc.decoded\_frame\_rx);

else

decoded\_frame\_rx=-1;

pushNotifiedFIFO\_nothreadSafe(&freeBlocks,res);

}

if (decoded\_frame\_rx>0 && decoded\_frame\_rx != curMsg->proc.frame\_rx)

LOG\_E(PHY,"Decoded frame index (%d) is not compatible with current context (%d), UE should go back to synch mode\n",decoded\_frame\_rx, curMsg->proc.frame\_rx);

writeTimestamp = timestamp+

if (UE->timing\_advance != timing\_advance) {

writeBlockSize -= UE->timing\_advance - timing\_advance;

timing\_advance = UE->timing\_advance;

}

int flags = 0;

if(openair0\_cfg[0].duplex\_mode==duplex\_mode\_TDD&& !get\_softmodem\_params()->continuous\_tx) {

uint8\_t tdd\_period = mac->phy\_config.config\_req.tdd\_table.tdd\_period\_in\_slots;

int nrofUplinkSlots, nrofUplinkSymbols;

if (mac->scc) {

nrofUplinkSlots=mac->scc->tdd\_UL\_DL\_ConfigurationCommon->pattern1.nrofUplinkSlots;

nrofUplinkSymbols=mac->scc->tdd\_UL\_DL\_ConfigurationCommon->pattern1.nrofUplinkSymbols;

}

else {

nrofUplinkSlots=mac->scc\_SIB->tdd\_UL\_DL\_ConfigurationCommon->pattern1.nrofUplinkSlots;

nrofUplinkSymbols=mac->scc\_SIB->tdd\_UL\_DL\_ConfigurationCommon->pattern1.nrof UplinkSymbols;

}

int slot\_tx\_usrp = slot\_nr + DURATION\_RX\_TO\_TX - NR\_RX\_NB\_TH;

uint8\_t num\_UL\_slots = nrofUplinkSlots + (nrofUplinkSymbols != 0);

uint8\_t first\_tx\_slot = tdd\_period - num\_UL\_slots;

if (slot\_tx\_usrp % tdd\_period == first\_tx\_slot)

flags = 2;

else if (slot\_tx\_usrp % tdd\_period == first\_tx\_slot + num\_UL\_slots - 1)

flags = 3;

else if (slot\_tx\_usrp % tdd\_period > first\_tx\_slot)

flags = 1;

} else {

flags = 1;

}

for (int i=0; i<UE->frame\_parms.nb\_antennas\_tx; i++)

memset(txp[i], 0, writeBlockSize);

nbSlotProcessing++;

LOG\_D(PHY,"Number of slots being processed at the moment: %d\n",nbSlotProcessing);

pushTpool(&(get\_nrUE\_params()->Tpool), msgToPush);

}

return NULL;

}

void init\_NR\_UE(int nb\_inst,

char\* uecap\_file,

char\* rrc\_config\_path) {

int inst;

NR\_UE\_MAC\_INST\_t \*mac\_inst;

NR\_UE\_RRC\_INST\_t\* rrc\_inst;

for (inst=0; inst < nb\_inst; inst++) {

AssertFatal((rrc\_inst = nr\_l3\_init\_ue(uecap\_file,rrc\_config\_path)) != NULL, "can not initialize RRC module\n");

AssertFatal((mac\_inst = nr\_l2\_init\_ue(rrc\_inst)) != NULL, "can not initialize L2 module\n");

AssertFatal((mac\_inst->if\_module = nr\_ue\_if\_module\_init(inst)) != NULL, "can not initialize IF module\n");

}

}

void init\_NR\_UE\_threads(int nb\_inst) {

int inst;

pthread\_t threads[nb\_inst];

for (inst=0; inst < nb\_inst; inst++) {

PHY\_VARS\_NR\_UE \*UE = PHY\_vars\_UE\_g[inst][0];

threadCreate(&threads[inst],UE\_thread,(void\*)UE, "UEthread", -1, OAI\_PRIORITY\_RT\_MAX);

pthread\_t stat\_pthread;

threadCreate(&stat\_pthread,nrL1\_UE\_stats\_thread,UE,"L1\_UE\_stats",-1,OAI\_PRIORITY\_RT\_LOW);

}

}

#### 2、openairinterface5g /openair1/SCHED\_NR\_UE/phy\_procedures\_nr\_ue.c

#define \_GNU\_SOURCE

#include "nr/nr\_common.h"

#include "assertions.h"

#include "defs.h"

#include "PHY/defs\_nr\_UE.h"

#include "PHY/NR\_REFSIG/dmrs\_nr.h"

#include "PHY/phy\_extern\_nr\_ue.h"

#include "PHY/MODULATION/modulation\_UE.h"

#include "PHY/NR\_UE\_TRANSPORT/nr\_transport\_ue.h"

#include "PHY/NR\_UE\_TRANSPORT/nr\_transport\_proto\_ue.h"

#include "PHY/NR\_UE\_TRANSPORT/srs\_modulation\_nr.h"

#include "SCHED\_NR/extern.h"

#include "SCHED\_NR\_UE/phy\_sch\_processing\_time.h"

#include "PHY/NR\_UE\_ESTIMATION/nr\_estimation.h"

#include "SCHED/phy\_procedures\_emos.h"

#include "executables/softmodem-common.h"

#include "executables/nr-uesoftmodem.h"

#include "LAYER2/NR\_MAC\_UE/mac\_proto.h"

#include "LAYER2/NR\_MAC\_UE/nr\_l1\_helpers.h"

#define NR\_PDCCH\_SCHED

#ifndef PUCCH

#define PUCCH

#include "common/utils/LOG/log.h"

fifo\_dump\_emos\_UE emos\_dump\_UE;

#include "common/utils/LOG/vcd\_signal\_dumper.h"

#include "UTIL/OPT/opt.h"

#include "intertask\_interface.h"

#include "T.h"

void nr\_fill\_dl\_indication(nr\_downlink\_indication\_t \*dl\_ind,

fapi\_nr\_dci\_indication\_t \*dci\_ind,

fapi\_nr\_rx\_indication\_t \*rx\_ind,

UE\_nr\_rxtx\_proc\_t \*proc,

PHY\_VARS\_NR\_UE \*ue,

uint8\_t gNB\_id,

void \*phy\_data){

memset((void\*)dl\_ind, 0, sizeof(nr\_downlink\_indication\_t));

dl\_ind->gNB\_index = gNB\_id;

dl\_ind->module\_id = ue->Mod\_id;

dl\_ind->cc\_id = ue->CC\_id;

dl\_ind->frame = proc->frame\_rx;

dl\_ind->slot = proc->nr\_slot\_rx;

dl\_ind->thread\_id = proc->thread\_id;

dl\_ind->phy\_data = phy\_data;

if (dci\_ind) {

dl\_ind->rx\_ind = NULL;

dl\_ind->dci\_ind = dci\_ind;

} else if (rx\_ind) {

dl\_ind->rx\_ind = rx\_ind; // hang on rx\_ind instance

dl\_ind->dci\_ind = NULL;

}

}

void nr\_fill\_rx\_indication(fapi\_nr\_rx\_indication\_t \*rx\_ind,

uint8\_t pdu\_type,

uint8\_t gNB\_id,

PHY\_VARS\_NR\_UE \*ue,

NR\_UE\_DLSCH\_t \*dlsch0,

NR\_UE\_DLSCH\_t \*dlsch1,

uint16\_t n\_pdus,

UE\_nr\_rxtx\_proc\_t \*proc,

void \* typeSpecific){

NR\_DL\_FRAME\_PARMS \*frame\_parms = &ue->frame\_parms;

if (n\_pdus > 1){

LOG\_E(PHY, "In %s: multiple number of DL PDUs not supported yet...\n", \_\_FUNCTION\_\_);

}

if (pdu\_type != FAPI\_NR\_RX\_PDU\_TYPE\_SSB)

trace\_NRpdu(DIRECTION\_DOWNLINK,

dlsch0->harq\_processes[dlsch0->current\_harq\_pid]->b,

dlsch0->harq\_processes[dlsch0->current\_harq\_pid]->TBS / 8,

WS\_C\_RNTI,

dlsch0->rnti,

proc->frame\_rx,

proc->nr\_slot\_rx,

0,0);

switch (pdu\_type){

case FAPI\_NR\_RX\_PDU\_TYPE\_SIB:

rx\_ind->rx\_indication\_body[n\_pdus - 1].pdsch\_pdu.harq\_pid = dlsch0->current\_harq\_pid;

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.ack\_nack=dlsch0->harq\_processes[dlsch0->current\_harq\_pid]->ack;

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.pdu=dlsch0->harq\_processes[dlsch0->current\_harq\_pid]->b;

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.pdu\_length=dlsch0->harq\_processes[dlsch0->current\_harq\_pid]->TBS / 8;

break;

case FAPI\_NR\_RX\_PDU\_TYPE\_DLSCH:

if(dlsch0) {

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.harq\_pid=dlsch0->current\_harq\_pid;

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.ack\_nack=dlsch0->harq\_processes[d lsch0->current\_harq\_pid]->ack;

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.pdu=dlsch0->harq\_processes[dlsch0 ->current\_harq\_pid]->b;

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.pdu\_length=dlsch0->harq\_processes [dlsch0->current\_harq\_pid]->TBS / 8;

}

if(dlsch1) {

AssertFatal(1==0,"Second codeword currently not supported\n");

}

break;

case FAPI\_NR\_RX\_PDU\_TYPE\_RAR:

rx\_ind->rx\_indication\_body[n\_pdus - 1].pdsch\_pdu.harq\_pid = dlsch0->current\_harq\_pid;

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.ack\_nack=dlsch0->harq\_processes[dlsc h0->current\_harq\_pid]->ack;

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.pdu=dlsch0->harq\_processes[dlsch0->c urrent\_harq\_pid]->b;

rx\_ind->rx\_indication\_body[n\_pdus-1].pdsch\_pdu.pdu\_length=dlsch0->harq\_processes[dlsch0->current\_harq\_pid]->TBS / 8;

break;

case FAPI\_NR\_RX\_PDU\_TYPE\_SSB:

rx\_ind->rx\_indication\_body[n\_pdus1].ssb\_pdu.pdu=malloc(sizeof(((fapiPbch\_t\*)typeSpecific)->decoded\_output));

memcpy(rx\_ind->rx\_indication\_body[n\_pdus-1].ssb\_pdu.pdu,((fapiPbch\_t\*)typeSpecific)->decoded\_output, sizeof(((fapiPbch\_t\*)typeSpecific)->decoded\_output));

rx\_ind->rx\_indication\_body[n\_pdus-1].ssb\_pdu.additional\_bits=((fapiPbch\_t\*)typeSpecific)->xtra\_byte;

rx\_ind->rx\_indication\_body[n\_pdus-1].ssb\_pdu.ssb\_index=(frame\_parms->ssb\_index);

rx\_ind->rx\_indication\_body[n\_pdus - 1].ssb\_pdu.ssb\_length = frame\_parms->Lmax;

rx\_ind->rx\_indication\_body[n\_pdus - 1].ssb\_pdu.cell\_id = frame\_parms->Nid\_cell;

rx\_ind->rx\_indication\_body[n\_pdus-1].ssb\_pdu.ssb\_start\_subcarrier=frame\_parms->ssb\_ start\_subcarrier;

rx\_ind->rx\_indication\_body[n\_pdus-1].ssb\_pdu.rsrp\_dBm=ue->measurements.rsrp\_dBm[gNB\_id];

break;

default:

break;

}

}

int get\_tx\_amp\_prach(int power\_dBm, int power\_max\_dBm, int N\_RB\_UL){

int gain\_dB = power\_dBm - power\_max\_dBm, amp\_x\_100 = -1;

switch (N\_RB\_UL) {

case 6:

amp\_x\_100 = AMP;

break;

case 15:

amp\_x\_100 = 158\*AMP;

break;

case 25:

amp\_x\_100 = 204\*AMP;

break;

case 50:

amp\_x\_100 = 286\*AMP;

break;

case 75:

amp\_x\_100 = 354\*AMP;

break;

case 100:

amp\_x\_100 = 408\*AMP;

break;

default:

LOG\_E(PHY, "Unknown PRB size %d\n", N\_RB\_UL);

return (amp\_x\_100);

break;

}

if (gain\_dB < -30) {

return (amp\_x\_100/3162);

} else if (gain\_dB > 0)

return (amp\_x\_100);

else

return (amp\_x\_100/gain\_table[-gain\_dB]);

return (amp\_x\_100);

}

void ue\_ta\_procedures(PHY\_VARS\_NR\_UE \*ue, int slot\_tx, int frame\_tx){

if (ue->mac\_enabled == 1) {

uint8\_t gNB\_id = 0;

NR\_UL\_TIME\_ALIGNMENT\_t \*ul\_time\_alignment = &ue->ul\_time\_alignment[gNB\_id];

if (frame\_tx == ul\_time\_alignment->ta\_frame && slot\_tx == ul\_time\_alignment->ta\_slot) {

uint16\_t ofdm\_symbol\_size = ue->frame\_parms.ofdm\_symbol\_size;

uint16\_t bw\_scaling = get\_bw\_scaling(ofdm\_symbol\_size);

ue->timing\_advance += (ul\_time\_alignment->ta\_command - 31) \* bw\_scaling;

ul\_time\_alignment->ta\_frame = -1;

ul\_time\_alignment->ta\_slot = -1;

}

}

}

void phy\_procedures\_nrUE\_TX(PHY\_VARS\_NR\_UE \*ue, UE\_nr\_rxtx\_proc\_t \*proc, uint8\_t gNB\_id) {

int slot\_tx = proc->nr\_slot\_tx;

int frame\_tx = proc->frame\_tx;

for(int i=0; i< ue->frame\_parms.nb\_antennas\_tx; ++i)

memset(ue->common\_vars.txdataF[i],0,sizeof(int)\*14\*ue->frame\_parms.ofdm\_symbol\_size);

LOG\_D(PHY,"\*\*\*\*\*\* start TX-Chain for AbsSubframe %d.%d \*\*\*\*\*\*\n", frame\_tx, slot\_tx);

start\_meas(&ue->phy\_proc\_tx);

}

if (ue->UE\_mode[gNB\_id] == PUSCH) {

ue\_srs\_procedures\_nr(ue, proc, gNB\_id);

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_UE\_TX, VCD\_FUNCTION\_OUT);

stop\_meas(&ue->phy\_proc\_tx);

}

void nr\_ue\_measurement\_procedures(uint16\_t l,

PHY\_VARS\_NR\_UE \*ue,

UE\_nr\_rxtx\_proc\_t \*proc,

uint8\_t gNB\_id,

uint16\_t slot){

NR\_DL\_FRAME\_PARMS \*frame\_parms=&ue->frame\_parms;

int frame\_rx = proc->frame\_rx;

int nr\_slot\_rx = proc->nr\_slot\_rx;

if (l==2) {

nr\_ue\_measurements(ue, proc, nr\_slot\_rx);

}

if (( slot == 2) && (l==(2-frame\_parms->Ncp))) {

phy\_adjust\_gain\_nr (ue,ue->measurements.rx\_power\_avg\_dB[gNB\_id],gNB\_id);

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_UE\_MEASUREMENT\_PROCEDURES, VCD\_FUNCTION\_OUT);

}

static void nr\_ue\_pbch\_procedures(uint8\_t gNB\_id,

PHY\_VARS\_NR\_UE \*ue,

UE\_nr\_rxtx\_proc\_t\*proc,

int estimateSz,

struct complex16 dl\_ch\_estimates[][estimateSz], NR\_UE\_PDCCH\_CONFIG \*phy\_pdcch\_config) {

int ret = 0;

DevAssert(ue);

int frame\_rx = proc->frame\_rx;

int nr\_slot\_rx = proc->nr\_slot\_rx;

fapiPbch\_t result;

ret = nr\_rx\_pbch(ue, proc,

estimateSz, dl\_ch\_estimates,

ue->pbch\_vars[gNB\_id],

&ue->frame\_parms,

gNB\_id,

(ue->frame\_parms.ssb\_index)&7,

SISO,

phy\_pdcch\_config,

&result);

if (ret==0) {

ue->pbch\_vars[gNB\_id]->pdu\_errors\_conseq = 0;

if (ue->UE\_mode[gNB\_id] == NOT\_SYNCHED && ue->no\_timing\_correction == 1){

if (get\_softmodem\_params()->do\_ra) {

ue->UE\_mode[gNB\_id] = PRACH;

ue->prach\_resources[gNB\_id]->sync\_frame = frame\_rx;

ue->prach\_resources[gNB\_id]->init\_msg1 = 0;

} else {

ue->UE\_mode[gNB\_id] = PUSCH;

}

}

}

if (frame\_rx % 100 == 0) {

ue->pbch\_vars[gNB\_id]->pdu\_errors\_last = ue->pbch\_vars[gNB\_id]->pdu\_errors;

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_UE\_PBCH\_PROCEDURES, VCD\_FUNCTION\_OUT);

}

int nr\_ue\_pdcch\_procedures(uint8\_t gNB\_id,

PHY\_VARS\_NR\_UE \*ue,

UE\_nr\_rxtx\_proc\_t \*proc,

int32\_t pdcch\_est\_size,

int32\_t pdcch\_dl\_ch\_estimates[][pdcch\_est\_size],

NR\_UE\_PDCCH\_CONFIG \*phy\_pdcch\_config,

int n\_ss)

{

int frame\_rx = proc->frame\_rx;

int nr\_slot\_rx = proc->nr\_slot\_rx;

unsigned int dci\_cnt=0;

fapi\_nr\_dci\_indication\_t \*dci\_ind = calloc(1, sizeof(\*dci\_ind));

nr\_downlink\_indication\_t dl\_indication;

fapi\_nr\_dl\_config\_dci\_dl\_pdu\_rel15\_t \*rel15 = &phy\_pdcch\_config->pdcch\_config[n\_ss];

start\_meas(&ue->dlsch\_rx\_pdcch\_stats);

int32\_t pdcch\_e\_rx\_size = NR\_MAX\_PDCCH\_SIZE;

int16\_t pdcch\_e\_rx[pdcch\_e\_rx\_size];

nr\_rx\_pdcch(ue, proc, pdcch\_est\_size, pdcch\_dl\_ch\_estimates, pdcch\_e\_rx, rel15);

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_RX\_PDCCH, VCD\_FUNCTION\_OUT);

dci\_cnt = nr\_dci\_decoding\_procedure(ue, proc, pdcch\_e\_rx, dci\_ind, rel15, phy\_pdcch\_config);

ue->pdcch\_vars[proc->thread\_id][gNB\_id]->dci\_received += dci\_cnt;

dci\_ind->number\_of\_dcis = dci\_cnt;

nr\_fill\_dl\_indication(&dl\_indication, dci\_ind, NULL, proc, ue, gNB\_id, phy\_pdcch\_config);

ue->if\_inst->dl\_indication(&dl\_indication, NULL);

stop\_meas(&ue->dlsch\_rx\_pdcch\_stats);

return(dci\_cnt);

}

int nr\_ue\_pdsch\_procedures(PHY\_VARS\_NR\_UE \*ue,

UE\_nr\_rxtx\_proc\_t \*proc,

int gNB\_id,

PDSCH\_t pdsch,

NR\_UE\_DLSCH\_t \*dlsch0,

NR\_UE\_DLSCH\_t \*dlsch1) {

int frame\_rx = proc->frame\_rx;

int nr\_slot\_rx = proc->nr\_slot\_rx;

int m;

int i\_mod,gNB\_id\_i,dual\_stream\_UE;

int first\_symbol\_flag=0;

ue->Mod\_id,pdsch,nr\_slot\_rx,harq\_pid,dlsch0\_harq->status,pdsch\_start\_rb,pdsch\_nb\_rb,s0,s1,dlsch0\_harq->dlDmrsSymbPos, dlsch0\_harq->Nl);

for (m = s0; m < (s0 +s1); m++) {

if (dlsch0\_harq->dlDmrsSymbPos & (1 << m)) {

for (uint8\_t aatx=0; aatx<dlsch0\_harq->Nl; aatx++) {

nr\_pdsch\_channel\_estimation(ue,

proc,

gNB\_id,

is\_SI,

nr\_slot\_rx,

get\_dmrs\_port(aatx,dlsch0\_harq->dmrs\_ports),

m,

dlsch0\_harq->nscid,

dlsch0\_harq->dlDmrsScramblingId,

BWPStart,

dlsch0\_harq->dmrsConfigType,

ue->frame\_parms.first\_carrier\_offset

pdsch\_nb\_rb);

int nr\_frame\_rx = proc->frame\_rx;

char filename[100];

for (uint8\_t aarx=0; aarx<ue->frame\_parms.nb\_antennas\_rx; aarx++) {

int\*\*dl\_ch\_estimates=ue->pdsch\_vars[proc->thread\_id][gNB\_id]->dl\_ch\_estimates;

}

}

}

}

if (ue->chest\_time == 1) {

nr\_chest\_time\_domain\_avg(&ue->frame\_parms,

ue->pdsch\_vars[proc->thread\_id][gNB\_id]->dl\_ch\_estimates,

dlsch0\_harq->nb\_symbols,

dlsch0\_harq->start\_symbol,

dlsch0\_harq->dlDmrsSymbPos,

pdsch\_nb\_rb);

}

uint16\_t first\_symbol\_with\_data = s0;

uint32\_t dmrs\_data\_re;

if (dlsch0\_harq->dmrsConfigType == NFAPI\_NR\_DMRS\_TYPE1)

dmrs\_data\_re = 12 - 6 \* dlsch0\_harq->n\_dmrs\_cdm\_groups;

else

dmrs\_data\_re = 12 - 4 \* dlsch0\_harq->n\_dmrs\_cdm\_groups;

while((dmrs\_data\_re==0)&&(dlsch0\_harq->dlDmrsSymbPos&(1 << first\_symbol\_with\_data))) {

first\_symbol\_with\_data++;

}

start\_meas(&ue->rx\_pdsch\_stats);

for (m = s0; m < (s1 + s0); m++) {

dual\_stream\_UE = 0;

gNB\_id\_i = gNB\_id+1;

i\_mod = 0;

if (m==first\_symbol\_with\_data)

first\_symbol\_flag = 1;

else

first\_symbol\_flag = 0;

uint8\_t slot = 0;

if(m >= ue->frame\_parms.symbols\_per\_slot>>1)

slot = 1;

start\_meas(&ue->dlsch\_llr\_stats\_parallelization[proc->thread\_id][slot]);

if (pdsch == PDSCH || pdsch == SI\_PDSCH || pdsch == RA\_PDSCH) {

if (nr\_rx\_pdsch(ue,

proc,

pdsch,

gNB\_id,

gNB\_id\_i,

frame\_rx,

nr\_slot\_rx,

m,

first\_symbol\_flag,

dual\_stream\_UE,

i\_mod,

harq\_pid) < 0)

return -1;

} else AssertFatal(1==0,"Not RA\_PDSCH, SI\_PDSCH or PDSCH\n");

stop\_meas(&ue->dlsch\_llr\_stats\_parallelization[proc->thread\_id][slot]);

if(first\_symbol\_flag) {

proc->first\_symbol\_available = 1;

}

} // CRNTI active

stop\_meas(&ue->rx\_pdsch\_stats);

}

return 0;

}

bool nr\_ue\_dlsch\_procedures(PHY\_VARS\_NR\_UE \*ue,

UE\_nr\_rxtx\_proc\_t \*proc,

int gNB\_id,

PDSCH\_t pdsch,

NR\_UE\_DLSCH\_t \*dlsch0,

NR\_UE\_DLSCH\_t \*dlsch1,

int \*dlsch\_errors) {

bool dec = false;

int harq\_pid = dlsch0->current\_harq\_pid;

int frame\_rx = proc->frame\_rx;

int nr\_slot\_rx = proc->nr\_slot\_rx;

uint32\_t ret = UINT32\_MAX, ret1 = UINT32\_MAX;

NR\_UE\_PDSCH \*pdsch\_vars;

nr\_downlink\_indication\_t dl\_indication;

fapi\_nr\_rx\_indication\_t \*rx\_ind = calloc(1, sizeof(\*rx\_ind));

uint16\_t number\_pdus = 1;

NR\_UL\_TIME\_ALIGNMENT\_t \*ul\_time\_alignment = &ue->ul\_time\_alignment[gNB\_id];

uint8\_t is\_cw0\_active = dlsch0->harq\_processes[harq\_pid]->status;

uint16\_t nb\_symb\_sch = dlsch0->harq\_processes[harq\_pid]->nb\_symbols;

uint16\_t start\_symbol = dlsch0->harq\_processes[harq\_pid]->start\_symbol;

uint8\_t dmrs\_type = dlsch0->harq\_processes[harq\_pid]->dmrsConfigType;

uint8\_t nb\_re\_dmrs;

if (dmrs\_type==NFAPI\_NR\_DMRS\_TYPE1) {

nb\_re\_dmrs = 6\*dlsch0->harq\_processes[harq\_pid]->n\_dmrs\_cdm\_groups;

}

else {

nb\_re\_dmrs = 4\*dlsch0->harq\_processes[harq\_pid]->n\_dmrs\_cdm\_groups;

}

if (1) {

switch (pdsch) {

case SI\_PDSCH:

case RA\_PDSCH:

case P\_PDSCH:

case PDSCH:

pdsch\_vars = ue->pdsch\_vars[proc->thread\_id][gNB\_id];

break;

case PMCH:

case PDSCH1:

LOG\_E(PHY,"Illegal PDSCH %d for ue\_pdsch\_procedures\n",pdsch);

pdsch\_vars = NULL;

return false;

break;

default:

pdsch\_vars = NULL;

return false;

break;

}

start\_meas(&ue->dlsch\_unscrambling\_stats);

nr\_dlsch\_unscrambling(pdsch\_vars->llr[0],

dlsch0->harq\_processes[harq\_pid]->G,

0,

ue->frame\_parms.Nid\_cell,

dlsch0->rnti)；

stop\_meas(&ue->dlsch\_unscrambling\_stats);

start\_meas(&ue->dlsch\_decoding\_stats[proc->thread\_id]);

ret = nr\_dlsch\_decoding(ue,

proc,

gNB\_id,

pdsch\_vars->llr[0],

&ue->frame\_parms,

dlsch0,

dlsch0->harq\_processes[harq\_pid],

frame\_rx,

nb\_symb\_sch,

nr\_slot\_rx,

harq\_pid,

pdsch==PDSCH);

if(ret<dlsch0->max\_ldpc\_iterations+1)

dec = true;

switch (pdsch) {

case RA\_PDSCH:

nr\_fill\_dl\_indication(&dl\_indication, NULL, rx\_ind, proc, ue, gNB\_id, NULL);

nr\_fill\_rx\_indication(rx\_ind, FAPI\_NR\_RX\_PDU\_TYPE\_RAR, gNB\_id, ue, dlsch0, NULL, number\_pdus, proc, NULL);

ue->UE\_mode[gNB\_id] = RA\_RESPONSE;

break;

case PDSCH:

nr\_fill\_dl\_indication(&dl\_indication, NULL, rx\_ind, proc, ue, gNB\_id, NULL);

nr\_fill\_rx\_indication(rx\_ind, FAPI\_NR\_RX\_PDU\_TYPE\_DLSCH, gNB\_id, ue, dlsch0, NULL, number\_pdus, proc, NULL);

break;

case SI\_PDSCH:

nr\_fill\_dl\_indication(&dl\_indication, NULL, rx\_ind, proc, ue, gNB\_id, NULL);

nr\_fill\_rx\_indication(rx\_ind, FAPI\_NR\_RX\_PDU\_TYPE\_SIB, gNB\_id, ue, dlsch0, NULL, number\_pdus, proc, NULL);

break;

default:

break;

}

stop\_meas(&ue->dlsch\_decoding\_stats[proc->thread\_id]);

}

start\_meas(&ue->dlsch\_unscrambling\_stats);

nr\_dlsch\_unscrambling(pdsch\_vars->llr[1],

dlsch1->harq\_processes[harq\_pid]->G,

0,

ue->frame\_parms.Nid\_cell,

dlsch1->rnti);

stop\_meas(&ue->dlsch\_unscrambling\_stats);

start\_meas(&ue->dlsch\_decoding\_stats[proc->thread\_id]);

ret1 = nr\_dlsch\_decoding(ue,

proc,

gNB\_id,

pdsch\_vars->llr[1],

&ue->frame\_parms,

dlsch1,

dlsch1->harq\_processes[harq\_pid],

frame\_rx,

nb\_symb\_sch,

nr\_slot\_rx,

harq\_pid,

pdsch==PDSCH);

stop\_meas(&ue->dlsch\_decoding\_stats[proc->thread\_id]);

if (cpumeas(CPUMEAS\_GETSTATE)) {

LOG\_D(PHY, " --> Unscrambling for CW1 %5.3f\n",

(ue->dlsch\_unscrambling\_stats.p\_time)/(cpuf\*1000.0));

LOG\_D(PHY, "AbsSubframe %d.%d --> ldpc Decoding for CW1 %5.3f\n",

frame\_rx%1024,nr\_slot\_rx,(ue->dlsch\_decoding\_stats[proc->thread\_id].p\_time)/(cpuf\*1000.0));

}

LOG\_D(PHY, "harq\_pid: %d, TBS expected dlsch1: %d \n", harq\_pid, dlsch1->harq\_processes[harq\_pid]->TBS);

}

const int numerology = ue->frame\_parms.numerology\_index;

const int ofdm\_symbol\_size = ue->frame\_parms.ofdm\_symbol\_size;

const int nb\_prefix\_samples = ue->frame\_parms.nb\_prefix\_samples;

const int samples\_per\_subframe = ue->frame\_parms.samples\_per\_subframe;

const int slots\_per\_frame = ue->frame\_parms.slots\_per\_frame;

const int slots\_per\_subframe = ue->frame\_parms.slots\_per\_subframe;

const double tc\_factor = 1.0 / samples\_per\_subframe;

const uint16\_t bw\_scaling = get\_bw\_scaling(ofdm\_symbol\_size);

const int Ta\_max = 3846; // Max value of 12 bits TA Command

const double N\_TA\_max = Ta\_max \* bw\_scaling \* tc\_factor;

NR\_UE\_MAC\_INST\_t \*mac = get\_mac\_inst(0);

NR\_BWP\_Id\_t dl\_bwp = mac->DL\_BWP\_Id;

NR\_BWP\_Id\_t ul\_bwp = mac->UL\_BWP\_Id;

start\_meas(&ue->pdsch\_procedures\_per\_slot\_stat[proc->thread\_id][1]);

ue\_pdsch\_procedures(ue,

proc,

gNB\_id,

PDSCH,

ue->dlsch[proc->thread\_id][gNB\_id][0],

NULL,

(ue->frame\_parms.symbols\_per\_slot>>1),

ue->frame\_parms.symbols\_per\_slot-1,

abstraction\_flag);

LOG\_D(PHY," ------ end PDSCH ChannelComp/LLR slot 0: AbsSubframe %d.%d ------ \n", frame\_rx%1024, nr\_slot\_rx);

LOG\_D(PHY," ------ --> PDSCH Turbo Decoder slot 0/1: AbsSubframe %d.%d ------ \n", frame\_rx%1024, nr\_slot\_rx);

}

if ((ue->dlsch\_SI[gNB\_id]) && (ue->dlsch\_SI[gNB\_id]->active == 1)) {

ue\_pdsch\_procedures(ue,

proc,

gNB\_id,

SI\_PDSCH,

ue->dlsch\_SI[gNB\_id],

NULL,

(ue->frame\_parms.symbols\_per\_slot>>1),

ue->frame\_parms.symbols\_per\_slot-1,

abstraction\_flag);

}

if ((ue->dlsch\_p[gNB\_id]) && (ue->dlsch\_p[gNB\_id]->active == 1)) {

ue\_pdsch\_procedures(ue,

proc,

gNB\_id,

P\_PDSCH,

ue->dlsch\_p[gNB\_id],

NULL,

(ue->frame\_parms.symbols\_per\_slot>>1),

ue->frame\_parms.symbols\_per\_slot-1,

abstraction\_flag);

}

if ((ue->dlsch\_ra[gNB\_id]) && (ue->dlsch\_ra[gNB\_id]->active == 1) && (UE\_mode != PUSCH)) {

ue\_pdsch\_procedures(ue,

proc,

gNB\_id,

RA\_PDSCH,

ue->dlsch\_ra[gNB\_id],

NULL,

(ue->frame\_parms.symbols\_per\_slot>>1),

ue->frame\_parms.symbols\_per\_slot-1,

abstraction\_flag);

}

stop\_meas(&ue->pdsch\_procedures\_per\_slot\_stat[proc->thread\_id][1]);

if (cpumeas(CPUMEAS\_GETSTATE))

LOG\_D(PHY, "[AbsSFN %d.%d] Slot1: LLR Computation %5.2f \n",frame\_rx,nr\_slot\_rx,ue->pdsch\_procedures\_per\_slot\_stat[proc->thread\_id][1].p\_time/(cpuf\*1000.0));

}

}

free(arg);

return &UE\_dl\_slot1\_processing\_retval;

}

int phy\_procedures\_nrUE\_RX(PHY\_VARS\_NR\_UE \*ue,

UE\_nr\_rxtx\_proc\_t \*proc,

uint8\_t gNB\_id,

uint8\_t dlsch\_parallel,

NR\_UE\_PDCCH\_CONFIG \*phy\_pdcch\_config,

notifiedFIFO\_t \*txFifo

){

uint64\_t a=rdtsc\_oai();

int frame\_rx = proc->frame\_rx;

int nr\_slot\_rx = proc->nr\_slot\_rx;

int slot\_pbch;

int slot\_ssb;

fapi\_nr\_config\_request\_t \*cfg = &ue->nrUE\_config;

uint8\_t dci\_cnt = 0;

NR\_DL\_FRAME\_PARMS \*fp = &ue->frame\_parms;

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_UE\_RX, VCD\_FUNCTION\_IN);

start\_meas(&ue->phy\_proc\_rx[proc->thread\_id]);

LOG\_D(PHY," \*\*\*\*\*\* start RX-Chain for Frame.Slot %d.%d (energy %d dB)\*\*\*\*\*\* \n",frame\_rx%1024,nr\_slot\_rx,dB\_fixed(signal\_energy(ue->common\_vars.common\_vars\_rx\_data\_per\_thread[proc->thread\_id].rxdataF[0],2048\*14)));

int coreset\_nb\_rb=0;

int coreset\_start\_rb=0;

if (phy\_pdcch\_config->nb\_search\_space > 0)

get\_coreset\_rballoc(phy\_pdcch\_config->pdcch\_config[0].coreset.frequency\_domain\_resource,&coreset\_nb\_rb,&coreset\_start\_rb);

slot\_pbch = is\_pbch\_in\_slot(cfg, frame\_rx, nr\_slot\_rx, fp);

slot\_ssb = is\_ssb\_in\_slot(cfg, frame\_rx, nr\_slot\_rx, fp);

start\_meas(&ue->dlsch\_channel\_estimation\_stats);

nr\_pbch\_channel\_estimation(ue,

estimateSz,

dl\_ch\_estimates, dl\_ch\_estimates\_time,

proc,

gNB\_id,

nr\_slot\_rx,

(ue->symbol\_offset+i)%(fp->symbols\_per\_slot),

i-1,

(fp->ssb\_index)&7,

fp->half\_frame\_bit);

stop\_meas(&ue->dlsch\_channel\_estimation\_stats);

}

nr\_ue\_rsrp\_measurements(ue, gNB\_id, proc, nr\_slot\_rx, 0);

if ((ue->decode\_MIB == 1) && slot\_pbch) {

nr\_ue\_pbch\_procedures(gNB\_id,ue,proc,estimateSz,dl\_ch\_estimates, phy\_pdcch\_config);

if (ue->no\_timing\_correction==0) {

LOG\_D(PHY,"start adjust sync slot = %d no timing %d\n", nr\_slot\_rx, ue->no\_timing\_correction);

nr\_adjust\_synch\_ue(fp,

ue,

gNB\_id,

fp->ofdm\_symbol\_size,

dl\_ch\_estimates\_time,

frame\_rx,

nr\_slot\_rx,

0,

16384);

}

LOG\_D(PHY, "Doing N0 measurements in %s\n", \_\_FUNCTION\_\_);

nr\_ue\_rrc\_measurements(ue, proc, nr\_slot\_rx);

}

}

if (coreset\_nb\_rb > 0)

nr\_pdcch\_channel\_estimation(ue,

proc,

gNB\_id,

nr\_slot\_rx,

l,

phy\_pdcch\_config->pdcch\_config[n\_ss].coreset.pdcch\_dmrs\_scrambling\_id,fp->first\_carrier\_offset+(phy\_pdcch\_config->pdcch\_config[n\_ss].BWPStart+coreset\_start\_rb)\*12,coreset\_nb\_rb, pdcch\_est\_size, pdcch\_dl\_ch\_estimates);

}

dci\_cnt=dci\_cnt+nr\_ue\_pdcch\_procedures(gNB\_id,ue,proc,pdcch\_est\_size,pdcch\_dl\_ch\_estimates, phy\_pdcch\_config, n\_ss);

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_UE\_SLOT\_FEP\_PDCCH, VCD\_FUNCTION\_OUT);

if (dci\_cnt > 0) {

LOG\_D(PHY,"[UE %d] Frame %d, nr\_slot\_rx %d: found %d DCIs\n", ue->Mod\_id, frame\_rx, nr\_slot\_rx, dci\_cnt);

NR\_UE\_DLSCH\_t \*dlsch = NULL;

if (ue->dlsch[proc->thread\_id][gNB\_id][0]->active == 1){

dlsch = ue->dlsch[proc->thread\_id][gNB\_id][0];

} else if (ue->dlsch\_SI[0]->active == 1){

dlsch = ue->dlsch\_SI[0];

} else if (ue->dlsch\_ra[0]->active == 1){

dlsch = ue->dlsch\_ra[0];

}

if (dlsch) {

uint8\_t harq\_pid = dlsch->current\_harq\_pid;

NR\_DL\_UE\_HARQ\_t \*dlsch0\_harq = dlsch->harq\_processes[harq\_pid];

uint16\_t nb\_symb\_sch = dlsch0\_harq->nb\_symbols;

uint16\_t start\_symb\_sch = dlsch0\_harq->start\_symbol;

LOG\_D(PHY," ------ --> PDSCH ChannelComp/LLR Frame.slot %d.%d ------ \n", frame\_rx%1024, nr\_slot\_rx);

for (uint16\_t m=start\_symb\_sch;m<(nb\_symb\_sch+start\_symb\_sch) ; m++){

nr\_slot\_fep(ue,

proc,

m,

nr\_slot\_rx);

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_UE\_SLOT\_FEP\_PDSCH, VCD\_FUNCTION\_OUT);

}

} else {

LOG\_D(PHY,"[UE %d] Frame %d, nr\_slot\_rx %d: No DCIs found\n", ue->Mod\_id, frame\_rx, nr\_slot\_rx);

LOG\_D(PHY, "In harq ack : slot %d, time %llu\n", \_\_FUNCTION\_\_, proc->nr\_slot\_rx, (rdtsc\_oai()-a)/3500);

}

notifiedFIFO\_elt\_t\*newElt=newNotifiedFIFO\_elt(sizeof(nr\_rxtx\_thread\_data\_t),proc->nr\_slot\_tx,txFifo,processSlotTX);

nr\_rxtx\_thread\_data\_t \*curMsg=(nr\_rxtx\_thread\_data\_t \*)NotifiedFifoData(newElt);

curMsg->proc = \*proc;

curMsg->UE = ue;

curMsg->ue\_sched\_mode = ONLY\_PUSCH;

pushTpool(&(get\_nrUE\_params()->Tpool), newElt);

start\_meas(&ue->generic\_stat);

int ret\_pdsch = 0;

if (ue->dlsch[proc->thread\_id][gNB\_id][0]->active == 1) {

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PDSCH\_PROC\_C, VCD\_FUNCTION\_IN);

ret\_pdsch = nr\_ue\_pdsch\_procedures(ue,

proc,

gNB\_id,

PDSCH,

ue->dlsch[proc->thread\_id][gNB\_id][0],

NULL);

nr\_ue\_measurement\_procedures(2, ue, proc, gNB\_id, nr\_slot\_rx);

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PDSCH\_PROC\_C, VCD\_FUNCTION\_OUT);

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PDSCH\_PROC\_SI, VCD\_FUNCTION\_IN);

nr\_ue\_pdsch\_procedures(ue,

proc,

gNB\_id,

SI\_PDSCH,

ue->dlsch\_SI[gNB\_id],

NULL);

nr\_ue\_dlsch\_procedures(ue,

proc,

gNB\_id,

SI\_PDSCH,

ue->dlsch\_SI[gNB\_id],

NULL,

&ue->dlsch\_SI\_errors[gNB\_id]);

ue->dlsch\_SI[gNB\_id]->active = 0;

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PDSCH\_PROC\_SI, VCD\_FUNCTION\_OUT);

}

if ((ue->dlsch\_p[gNB\_id]) && (ue->dlsch\_p[gNB\_id]->active == 1)) {

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PDSCH\_PROC\_P, VCD\_FUNCTION\_IN);

nr\_ue\_pdsch\_procedures(ue,

proc,

gNB\_id,

P\_PDSCH,

ue->dlsch\_p[gNB\_id],

NULL);

nr\_ue\_dlsch\_procedures(ue,

proc,

gNB\_id,

P\_PDSCH,

ue->dlsch\_p[gNB\_id],

NULL,

&ue->dlsch\_p\_errors[gNB\_id]);

ue->dlsch\_p[gNB\_id]->active = 0;

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PDSCH\_PROC\_P, VCD\_FUNCTION\_OUT);

}

if((ue->dlsch\_ra[gNB\_id]&& (ue->dlsch\_ra[gNB\_id]->active == 1) && (ue->UE\_mode[gNB\_id] != PUSCH)) {

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PDSCH\_PROC\_RA, VCD\_FUNCTION\_IN);

nr\_ue\_pdsch\_procedures(ue,

proc,

gNB\_id,

RA\_PDSCH,

ue->dlsch\_ra[gNB\_id],

NULL);

nr\_ue\_dlsch\_procedures(ue,

proc,

gNB\_id,

RA\_PDSCH,

ue->dlsch\_ra[gNB\_id],

NULL,

&ue->dlsch\_ra\_errors[gNB\_id]);

ue->dlsch\_ra[gNB\_id]->active = 0;

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PDSCH\_PROC\_RA, VCD\_FUNCTION\_OUT);

}

start\_meas(&ue->dlsch\_procedures\_stat[proc->thread\_id])；

NR\_UE\_DLSCH\_t \*dlsch1 = NULL;

if (NR\_MAX\_NB\_LAYERS>4)

dlsch1 = ue->dlsch[proc->thread\_id][gNB\_id][1];

if (ret\_pdsch >= 0)

nr\_ue\_dlsch\_procedures(ue,

proc,

gNB\_id,

PDSCH,

ue->dlsch[proc->thread\_id][gNB\_id][0],

dlsch1,

&ue->dlsch\_errors[gNB\_id]);

stop\_meas(&ue->dlsch\_procedures\_stat[proc->thread\_id]);

}

ue->dlsch[proc->thread\_id][gNB\_id][0]->active = 0;

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PDSCH\_PROC, VCD\_FUNCTION\_OUT);

}

if ((ue->csiim\_vars[gNB\_id]) && (ue->csiim\_vars[gNB\_id]->active == 1)) {

nr\_ue\_csi\_im\_procedures(ue, proc, gNB\_id);

ue->csiim\_vars[gNB\_id]->active = 0;

}

if ((ue->csirs\_vars[gNB\_id]) && (ue->csirs\_vars[gNB\_id]->active == 1)) {

for(int symb = 0; symb < NR\_SYMBOLS\_PER\_SLOT; symb++) {

if(is\_csi\_rs\_in\_symbol(ue->csirs\_vars[gNB\_id]->csirs\_config\_pdu,symb)) {

nr\_slot\_fep(ue, proc, symb, nr\_slot\_rx);

}

}

nr\_ue\_csi\_rs\_procedures(ue, proc, gNB\_id);

ue->csirs\_vars[gNB\_id]->active = 0;

}

start\_meas(&ue->generic\_stat);

if (nr\_slot\_rx==9) {

if (frame\_rx % 10 == 0) {

if ((ue->dlsch\_received[gNB\_id] - ue->dlsch\_received\_last[gNB\_id]) != 0)

ue->dlsch\_fer[gNB\_id]=(100\*(ue->dlsch\_errors[gNB\_id]-ue->dlsch\_errors\_last[gNB\_id]))/(ue->dlsch\_received[gNB\_id] - ue->dlsch\_received\_last[gNB\_id]);

ue->dlsch\_errors\_last[gNB\_id] = ue->dlsch\_errors[gNB\_id];

ue->dlsch\_received\_last[gNB\_id] = ue->dlsch\_received[gNB\_id];

}

ue->bitrate[gNB\_id] = (ue->total\_TBS[gNB\_id] - ue->total\_TBS\_last[gNB\_id])\*100;

ue->total\_TBS\_last[gNB\_id] = ue->total\_TBS[gNB\_id];

LOG\_D(PHY,"[UE %d] Calculating bitrate Frame %d: total\_TBS = %d, total\_TBS\_last = %d, bitrate %f kbits\n",

ue->Mod\_id,frame\_rx,ue->total\_TBS[gNB\_id],

ue->total\_TBS\_last[gNB\_id],(float) ue->bitrate[gNB\_id]/1000.0);

}

stop\_meas(&ue->generic\_stat);

if (cpumeas(CPUMEAS\_GETSTATE))

LOG\_D(PHY,"after tubo until end of Rx %5.2f \n",ue->generic\_stat.p\_time/(cpuf\*1000.0));

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_UE\_RX, VCD\_FUNCTION\_OUT);

stop\_meas(&ue->phy\_proc\_rx[proc->thread\_id]);

return (0);

}

void nr\_ue\_prach\_procedures(PHY\_VARS\_NR\_UE \*ue, UE\_nr\_rxtx\_proc\_t \*proc, uint8\_t gNB\_id) {

int frame\_tx = proc->frame\_tx, nr\_slot\_tx = proc->nr\_slot\_tx, prach\_power;

uint8\_t mod\_id = ue->Mod\_id;

NR\_PRACH\_RESOURCES\_t \* prach\_resources = ue->prach\_resources[gNB\_id];

AssertFatal(prach\_resources != NULL, "ue->prach\_resources[%u] == NULL\n", gNB\_id);

uint8\_t nr\_prach = 0;

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_UE\_TX\_PRACH, VCD\_FUNCTION\_IN);

if (ue->mac\_enabled == 0){

prach\_resources->ra\_TDD\_map\_index = 0;

prach\_resources->ra\_PREAMBLE\_RECEIVED\_TARGET\_POWER = 10;

prach\_resources->ra\_RNTI = 0x1234;

nr\_prach = 1;

prach\_resources->init\_msg1 = 1;

} else {

nr\_prach=nr\_ue\_get\_rach(prach\_resources,&ue->prach\_vars[0]->prach\_pdu,mod\_id,ue->CC\_id, frame\_tx, gNB\_id, nr\_slot\_tx);

LOG\_D(PHY, "In %s:[%d.%d] getting PRACH resources : %d\n", \_\_FUNCTION\_\_, frame\_tx, nr\_slot\_tx,nr\_prach);

}

if (nr\_prach == GENERATE\_PREAMBLE) {

if (ue->mac\_enabled == 1) {

int16\_t pathloss = get\_nr\_PL(mod\_id, ue->CC\_id, gNB\_id);

ue->tx\_power\_dBm[nr\_slot\_tx] = min(nr\_get\_Pcmax(mod\_id), ra\_preamble\_rx\_power);

}

ue->prach\_vars[gNB\_id]->amp = AMP;

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_UE\_GENERATE\_PRACH, VCD\_FUNCTION\_IN);

prach\_power = generate\_nr\_prach(ue, gNB\_id, frame\_tx, nr\_slot\_tx);

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_UE\_GENERATE\_PRACH, VCD\_FUNCTION\_OUT);

if (ue->mac\_enabled == 1)

nr\_Msg1\_transmitted(mod\_id, ue->CC\_id, frame\_tx, gNB\_id);

} else if (nr\_prach == WAIT\_CONTENTION\_RESOLUTION) {

LOG\_D(PHY, "In %s: [UE %d] RA waiting contention resolution\n", \_\_FUNCTION\_\_, mod\_id);

ue->UE\_mode[gNB\_id] = RA\_WAIT\_CR;

} else if (nr\_prach == RA\_SUCCEEDED) {

LOG\_D(PHY, "In %s: [UE %d] RA completed, setting UE mode to PUSCH\n", \_\_FUNCTION\_\_, mod\_id);

ue->UE\_mode[gNB\_id] = PUSCH;

} else if(nr\_prach == RA\_FAILED){

LOG\_D(PHY, "In %s: [UE %d] RA failed, setting UE mode to PRACH\n", \_\_FUNCTION\_\_, mod\_id);

ue->UE\_mode[gNB\_id] = PRACH;

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_UE\_TX\_PRACH, VCD\_FUNCTION\_OUT);

}

#### 3、openairinterface5g /openair1/SCHED\_NR/phy\_procedures\_nr\_gNB.c

#include "PHY/defs\_gNB.h"

#include "sched\_nr.h"

#include "PHY/NR\_TRANSPORT/nr\_transport\_proto.h"

#include "PHY/NR\_TRANSPORT/nr\_dlsch.h"

#include "PHY/NR\_TRANSPORT/nr\_ulsch.h"

#include "PHY/NR\_TRANSPORT/nr\_dci.h"

#include "PHY/NR\_ESTIMATION/nr\_ul\_estimation.h"

#include "nfapi/open-nFAPI/nfapi/public\_inc/nfapi\_interface.h"

#include "nfapi/open-nFAPI/nfapi/public\_inc/nfapi\_nr\_interface.h"

#include "fapi\_nr\_l1.h"

#include "common/utils/LOG/log.h"

#include "common/utils/LOG/vcd\_signal\_dumper.h"

#include "PHY/INIT/phy\_init.h"

#include "PHY/MODULATION/nr\_modulation.h"

#include "PHY/NR\_UE\_TRANSPORT/srs\_modulation\_nr.h"

#include "T.h"

#include "executables/nr-softmodem.h"

#include "executables/softmodem-common.h"

#include "assertions.h"

#include <time.h>

#include "intertask\_interface.h"

uint8\_t SSB\_Table[38]={0,2,4,6,8,10,12,14,254,254,16,18,20,22,24,26,28,30,254,254,32,34,36,38,40,42,44,46,254,254,48,50,52,54,56,58,60,62};

extern uint8\_t nfapi\_mode;

void nr\_set\_ssb\_first\_subcarrier(nfapi\_nr\_config\_request\_scf\_t \*cfg, NR\_DL\_FRAME\_PARMS \*fp) {

uint8\_t sco = 0;

if (((fp->freq\_range == nr\_FR1) && (cfg->ssb\_table.ssb\_subcarrier\_offset.value<24)) ||

((fp->freq\_range == nr\_FR2) && (cfg->ssb\_table.ssb\_subcarrier\_offset.value<12)) ) {

if (fp->freq\_range == nr\_FR1)

sco = cfg->ssb\_table.ssb\_subcarrier\_offset.value>>cfg->ssb\_config.scs\_common.value;

}

fp->ssb\_start\_subcarrier = (12 \* cfg->ssb\_table.ssb\_offset\_point\_a.value + sco);

}

void nr\_common\_signal\_procedures (PHY\_VARS\_gNB \*gNB,int frame,int slot,nfapi\_nr\_dl\_tti\_ssb\_pdu ssb\_pdu) {

NR\_DL\_FRAME\_PARMS \*fp=&gNB->frame\_parms;

nfapi\_nr\_config\_request\_scf\_t \*cfg = &gNB->gNB\_config;

int \*\*txdataF = gNB->common\_vars.txdataF;

uint8\_t ssb\_index, n\_hf;

uint16\_t ssb\_start\_symbol;

int txdataF\_offset = slot\*fp->samples\_per\_slot\_wCP;

uint16\_t slots\_per\_hf = (fp->slots\_per\_frame)>>1;

ssb\_index = ssb\_pdu.ssb\_pdu\_rel15.SsbBlockIndex;

int ssb\_start\_symbol\_abs = nr\_get\_ssb\_start\_symbol(fp,ssb\_index);

ssb\_start\_symbol = ssb\_start\_symbol\_abs % fp->symbols\_per\_slot;

nr\_set\_ssb\_first\_subcarrier(cfg, fp);

nr\_generate\_pss(&txdataF[0][txdataF\_offset], AMP, ssb\_start\_symbol, cfg, fp);

nr\_generate\_sss(&txdataF[0][txdataF\_offset], AMP, ssb\_start\_symbol, cfg, fp);

if (fp->Lmax == 4)

nr\_generate\_pbch\_dmrs(gNB->nr\_gold\_pbch\_dmrs[n\_hf][ssb\_index&7],&txdataF[0][txdataF\_offset], AMP, ssb\_start\_symbol, cfg, fp);

else nr\_generate\_pbch\_dmrs(gNB->nr\_gold\_pbch\_dmrs[0][ssb\_index&7],&txdataF[0][txdataF\_offset], AMP, ssb\_start\_symbol, cfg, fp);

if (T\_ACTIVE(T\_GNB\_PHY\_MIB)) {

unsigned char bch[3];

bch[0] = ssb\_pdu.ssb\_pdu\_rel15.bchPayload & 0xff;

bch[1] = (ssb\_pdu.ssb\_pdu\_rel15.bchPayload >> 8) & 0xff;

bch[2] = (ssb\_pdu.ssb\_pdu\_rel15.bchPayload >> 16) & 0xff;

T(T\_GNB\_PHY\_MIB, T\_INT(0) /\* module ID \*/, T\_INT(frame), T\_INT(slot), T\_BUFFER(bch, 3));

}

for (int j=0;j<fp->symbols\_per\_slot;j++)

gNB->common\_vars.beam\_id[0][slot\*fp->symbols\_per\_slot+j]=cfg->ssb\_table.ssb\_beam\_id\_list[ssb\_index].beam\_id.value;

}

nr\_generate\_pbch(&ssb\_pdu,

gNB->nr\_pbch\_interleaver,

&txdataF[0][txdataF\_offset],

AMP,

ssb\_start\_symbol,

n\_hf, frame, cfg, fp);

}

void phy\_procedures\_gNB\_TX(processingData\_L1tx\_t \*msgTx,

int frame,

int slot,

int do\_meas) {

int aa;

PHY\_VARS\_gNB \*gNB = msgTx->gNB;

NR\_DL\_FRAME\_PARMS \*fp=&gNB->frame\_parms;

nfapi\_nr\_config\_request\_scf\_t \*cfg = &gNB->gNB\_config;

int offset = gNB->CC\_id;

int txdataF\_offset = slot\*fp->samples\_per\_slot\_wCP;

for (aa=0; aa<cfg->carrier\_config.num\_tx\_ant.value; aa++) {

memset(&gNB->common\_vars.txdataF[aa][txdataF\_offset],0,fp->samples\_per\_slot\_wCP\*sizeof(int32\_t)); memset(&gNB->common\_vars.beam\_id[aa][slot\*fp->symbols\_per\_slot],255,fp->symbols\_per\_slot\*sizeof(uint8\_t));

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_gNB\_COMMON\_TX,1);

for (int i=0; i<fp->Lmax; i++) {

if (msgTx->ssb[i].active) {

nr\_common\_signal\_procedures(gNB,frame,slot,msgTx->ssb[i].ssb\_pdu);

msgTx->ssb[i].active = false;

}

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_gNB\_COMMON\_TX,0);

}

for (int i=0;i<NUMBER\_OF\_NR\_CSIRS\_MAX;i++){

NR\_gNB\_CSIRS\_t \*csirs = &msgTx->csirs\_pdu[i];

if (csirs->active == 1) {

LOG\_D(PHY, "CSI-RS generation started in frame %d.%d\n",frame,slot);

nfapi\_nr\_dl\_tti\_csi\_rs\_pdu\_rel15\_t \*csi\_params = &csirs->csirs\_pdu.csi\_rs\_pdu\_rel15;

nr\_generate\_csi\_rs(gNB->frame\_parms,gNB->common\_vars.txdataF,AMP,gNB->nr\_csi\_rs\_info, csi\_params, slot);

csirs->active = 0;

}

}

for (aa=0; aa<cfg->carrier\_config.num\_tx\_ant.value; aa++) {

apply\_nr\_rotation(fp,(int16\_t\*). &gNB->common\_vars.txdataF[aa][txdataF\_offset],slot,0,fp->Ncp==EXTENDED?12:14,fp->ofdm\_symbol\_size);

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_gNB\_TX+offset,0);

}

void nr\_postDecode(PHY\_VARS\_gNB \*gNB, notifiedFIFO\_elt\_t \*req) {

ldpcDecode\_t \*rdata = (ldpcDecode\_t\*) NotifiedFifoData(req);

NR\_UL\_gNB\_HARQ\_t \*ulsch\_harq = rdata->ulsch\_harq;

NR\_gNB\_ULSCH\_t \*ulsch = rdata->ulsch;

int r = rdata->segment\_r;

nfapi\_nr\_pusch\_pdu\_t\*pusch\_pdu=&gNB->ulsch[rdata->ulsch\_id]->harq\_processes[rdata->harq\_pid]->ulsch\_pdu;

bool decodeSuccess = (rdata->decodeIterations <= rdata->decoderParms.numMaxIter);

ulsch\_harq->processedSegments++;

gNB->nbDecode--;

LOG\_D(PHY,"remain to decoded in subframe: %d\n", gNB->nbDecode);

if (decodeSuccess) {

memcpy(ulsch\_harq->b+rdata->offset,

ulsch\_harq->c[r],

rdata->Kr\_bytes - (ulsch\_harq->F>>3) -((ulsch\_harq->C>1)?3:0));

} else {

if ( rdata->nbSegments != ulsch\_harq->processedSegments ) {

int nb=abortTpool(gNB->threadPool, req->key);

nb+=abortNotifiedFIFO(gNB->respDecode, req->key);

gNB->nbDecode-=nb;

ulsch\_harq->processedSegments, nb, rdata->nbSegments);

ulsch\_harq->processedSegments=rdata->nbSegments;

}

}

if (rdata->nbSegments == ulsch\_harq->processedSegments) {

if (decodeSuccess) {

gNB->Mod\_id,ulsch\_harq->frame,ulsch\_harq->slot,rdata->harq\_pid,pusch\_pdu->pusch\_data.new\_data\_indicator,ulsch\_harq->status,ulsch\_harq->round,ulsch\_harq->TBS,rdata->decodeIterations);

ulsch\_harq->status = SCH\_IDLE;

ulsch\_harq->round = 0;

ulsch->harq\_mask &= ~(1 << rdata->harq\_pid);

LOG\_D(PHY, "ULSCH received ok \n");

nr\_fill\_indication(gNB,ulsch\_harq->frame,ulsch\_harq->slot,rdata->ulsch\_id,rdata->harq\_pid, 0,0);

} else {

ulsch\_harq->round++;

if (ulsch\_harq->round >= ulsch->Mlimit) {

ulsch\_harq->status = SCH\_IDLE;

ulsch\_harq->round = 0;

ulsch\_harq->handled = 0;

ulsch->harq\_mask &= ~(1 << rdata->harq\_pid);

}

ulsch\_harq->handled = 1;

LOG\_D(PHY, "ULSCH %d in error\n",rdata->ulsch\_id);

nr\_fill\_indication(gNB,ulsch\_harq->frame,ulsch\_harq->slot,rdata->ulsch\_id,rdata->harq\_pid, 1,0);

}

}

exit(-1);

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_gNB\_ULSCH\_DECODING,0);

}

void nr\_ulsch\_procedures(PHY\_VARS\_gNB \*gNB, int frame\_rx, int slot\_rx, int ULSCH\_id, uint8\_t harq\_pid)

{

NR\_DL\_FRAME\_PARMS \*frame\_parms = &gNB->frame\_parms;

nfapi\_nr\_pusch\_pdu\_t\*pusch\_pdu=.&gNB->ulsch[ULSCH\_id]->harq\_processes[harq\_pid]->ulsch\_pdu;

uint8\_t l, number\_dmrs\_symbols = 0;

uint32\_t G;

uint16\_t start\_symbol, number\_symbols, nb\_re\_dmrs;

start\_symbol = pusch\_pdu->start\_symbol\_index;

number\_symbols = pusch\_pdu->nr\_of\_symbols;

for (l = start\_symbol; l < start\_symbol + number\_symbols; l++)

number\_dmrs\_symbols += ((pusch\_pdu->ul\_dmrs\_symb\_pos)>>l)&0x01;

if (pusch\_pdu->dmrs\_config\_type==pusch\_dmrs\_type1)

nb\_re\_dmrs = 6\*pusch\_pdu->num\_dmrs\_cdm\_grps\_no\_data;

else

nb\_re\_dmrs = 4\*pusch\_pdu->num\_dmrs\_cdm\_grps\_no\_data;

nr\_ulsch\_layer\_demapping(gNB->pusch\_vars[ULSCH\_id]->llr,

pusch\_pdu->nrOfLayers,

pusch\_pdu->qam\_mod\_order,

G,

gNB->pusch\_vars[ULSCH\_id]->llr\_layers);

start\_meas(&gNB->ulsch\_unscrambling\_stats);

nr\_ulsch\_unscrambling(gNB->pusch\_vars[ULSCH\_id]->llr,

G,

pusch\_pdu->data\_scrambling\_id,

pusch\_pdu->rnti);

stop\_meas(&gNB->ulsch\_unscrambling\_stats);

start\_meas(&gNB->ulsch\_decoding\_stats);

nr\_ulsch\_decoding(gNB,

ULSCH\_id,

gNB->pusch\_vars[ULSCH\_id]->llr,

frame\_parms,

pusch\_pdu,

frame\_rx,

slot\_rx,

harq\_pid,

G);

while (gNB->nbDecode > 0) {

notifiedFIFO\_elt\_t\*req=pullTpool(gNB->respDecode,gNB->threadPool);nr\_postDecode(gNB, req); delNotifiedFIFO\_elt(req);

}

stop\_meas(&gNB->ulsch\_decoding\_stats);

}

uint16\_t num\_crc = gNB->UL\_INFO.crc\_ind.number\_crcs;

gNB->UL\_INFO.crc\_ind.crc\_list = &gNB->crc\_pdu\_list[0];

gNB->UL\_INFO.crc\_ind.sfn = frame;

gNB->UL\_INFO.crc\_ind.slot = slot\_rx;

gNB->crc\_pdu\_list[num\_crc].handle = pusch\_pdu->handle;

gNB->crc\_pdu\_list[num\_crc].rnti = pusch\_pdu->rnti;

gNB->crc\_pdu\_list[num\_crc].harq\_id = harq\_pid;

gNB->crc\_pdu\_list[num\_crc].tb\_crc\_status = crc\_flag;

gNB->crc\_pdu\_list[num\_crc].num\_cb = pusch\_pdu->pusch\_data.num\_cb;

gNB->crc\_pdu\_list[num\_crc].ul\_cqi = cqi;

gNB->crc\_pdu\_list[num\_crc].timing\_advance = timing\_advance\_update;

gNB->UL\_INFO.crc\_ind.number\_crcs++;

uint16\_t num\_rx = gNB->UL\_INFO.rx\_ind.number\_of\_pdus;

gNB->UL\_INFO.rx\_ind.pdu\_list = &gNB->rx\_pdu\_list[0];

gNB->UL\_INFO.rx\_ind.sfn = frame;

gNB->UL\_INFO.rx\_ind.slot = slot\_rx;

gNB->rx\_pdu\_list[num\_rx].handle = pusch\_pdu->handle;

gNB->rx\_pdu\_list[num\_rx].rnti = pusch\_pdu->rnti;

gNB->rx\_pdu\_list[num\_rx].harq\_id = harq\_pid;

gNB->rx\_pdu\_list[num\_rx].ul\_cqi = cqi;

gNB->rx\_pdu\_list[num\_rx].timing\_advance = timing\_advance\_update;

gNB->rx\_pdu\_list[num\_rx].rssi = gNB->crc\_pdu\_list[num\_crc].rssi;

if (crc\_flag)

gNB->rx\_pdu\_list[num\_rx].pdu\_length = 0;

else {

gNB->rx\_pdu\_list[num\_rx].pdu\_length = harq\_process->TBS;

gNB->rx\_pdu\_list[num\_rx].pdu = harq\_process->b;

}

}

for (int ULSCH\_id=0;ULSCH\_id<gNB->number\_of\_nr\_ulsch\_max;ULSCH\_id++) {

NR\_gNB\_ULSCH\_t \*ulsch = gNB->ulsch[ULSCH\_id];

int harq\_pid;

NR\_UL\_gNB\_HARQ\_t \*ulsch\_harq;

if ((ulsch) &&

(ulsch->rnti > 0)) {

for (harq\_pid=0;harq\_pid<NR\_MAX\_ULSCH\_HARQ\_PROCESSES;harq\_pid++) {

ulsch\_harq = ulsch->harq\_processes[harq\_pid];

AssertFatal(ulsch\_harq!=NULL,"harq\_pid %d is not allocated\n",harq\_pid);

if ((ulsch\_harq->status == NR\_ACTIVE) && (ulsch\_harq->frame == frame\_rx) &&. (ulsch\_harq->slot == slot\_rx) && (ulsch\_harq->handled == 0)){

uint8\_t symbol\_start = ulsch\_harq->ulsch\_pdu.start\_symbol\_index;

uint8\_t symbol\_end = symbol\_start + ulsch\_harq->ulsch\_pdu.nr\_of\_symbols;

for (int symbol=symbol\_start ; symbol<symbol\_end ; symbol++) {

if(gNB->frame\_parms.frame\_type==FDD||(gNB->frame\_parms.frame\_type==TD D&&gNB->gNB\_config.tdd\_table.max\_tdd\_periodicity\_list[slot\_rx].max\_num\_of\_symbol\_per\_slot\_list[symbol].slot\_config.value==1)) ;

for (rb=0; rb<ulsch\_harq->ulsch\_pdu.rb\_size; rb++) {

rb2=rb+ulsch\_harq->ulsch\_pdu.rb\_start+ulsch\_harq->ulsch\_pdu.bwp\_start;

gNB->rb\_mask\_ul[symbol][rb2>>5] |= (1<<(rb2&31));

}

}

}

}

}

}

}

for (int i=0;i<NUMBER\_OF\_NR\_SRS\_MAX;i++) {

NR\_gNB\_SRS\_t \*srs = gNB->srs[i];

if (srs) {

if ((srs->active == 1) && (srs->frame == frame\_rx) && (srs->slot == slot\_rx)) {

nfapi\_nr\_srs\_pdu\_t \*srs\_pdu = &srs->srs\_pdu;

for(int symbol = 0; symbol<(1<<srs\_pdu->num\_symbols); symbol++) {

for(rb = srs\_pdu->bwp\_start; rb < (srs\_pdu->bwp\_start+srs\_pdu->bwp\_size); rb++) {

gNB->rb\_mask\_ul[gNB->frame\_parms.symbols\_per\_slot-srs\_pdu->time\_start\_posi tion-1+symbol][rb>>5] |= 1<<(rb&31);

}

}

}

}

}

}

void phy\_procedures\_gNB\_common\_RX(PHY\_VARS\_gNB \*gNB, int frame\_rx, int slot\_rx) {

uint8\_t symbol;

unsigned char aa;

for(symbol = 0; symbol < (gNB->frame\_parms.Ncp==EXTENDED?12:14); symbol++) {

for (aa = 0; aa < gNB->frame\_parms.nb\_antennas\_rx; aa++) {

nr\_slot\_fep\_ul(&gNB->frame\_parms,

gNB->common\_vars.rxdata[aa],

gNB->common\_vars.rxdataF[aa],

symbol,

slot\_rx,

0);

}

}

for (aa = 0; aa < gNB->frame\_parms.nb\_antennas\_rx; aa++) {

apply\_nr\_rotation\_ul(&gNB->frame\_parms,

gNB->common\_vars.rxdataF[aa],

slot\_rx,

0,

gNB->frame\_parms.Ncp==EXTENDED?12:14,

gNB->frame\_parms.ofdm\_symbol\_size);

}

}

int phy\_procedures\_gNB\_uespec\_RX(PHY\_VARS\_gNB \*gNB, int frame\_rx, int slot\_rx) {

int pucch\_decode\_done = 0;

int pusch\_decode\_done = 0;

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_gNB\_UESPEC\_RX,1);

start\_meas(&gNB->phy\_proc\_rx);

for (int i=0;i<NUMBER\_OF\_NR\_PUCCH\_MAX;i++){

NR\_gNB\_PUCCH\_t \*pucch = gNB->pucch[i];

if (pucch) {

if (NFAPI\_MODE == NFAPI\_MODE\_PNF)

pucch->frame = frame\_rx;

if ((pucch->active == 1) && (pucch->frame == frame\_rx) && (pucch->slot == slot\_rx) ) {

pucch\_decode\_done = 1;

nfapi\_nr\_pucch\_pdu\_t \*pucch\_pdu = &pucch->pucch\_pdu;

uint16\_t num\_ucis;

switch (pucch\_pdu->format\_type) {

case 0:

num\_ucis = gNB->UL\_INFO.uci\_ind.num\_ucis;

gNB->UL\_INFO.uci\_ind.uci\_list = &gNB->uci\_pdu\_list[0];

gNB->UL\_INFO.uci\_ind.sfn = frame\_rx;

gNB->UL\_INFO.uci\_ind.slot = slot\_rx;

gNB->uci\_pdu\_list[num\_ucis].pdu\_type = NFAPI\_NR\_UCI\_FORMAT\_0\_1\_PDU\_TYPE;

gNB->uci\_pdu\_list[num\_ucis].pdu\_size=sizeof(nfapi\_nr\_uci\_pucch\_pdu\_format\_0\_1 \_t);

nfapi\_nr\_uci\_pucch\_pdu\_format\_0\_1\_t\*uci\_pdu\_format0=&gNB->uci\_pdu\_list[num. \_ucis].pucch\_pdu\_format\_0\_1;

offset = pucch\_pdu->start\_symbol\_index\*gNB->frame\_parms.ofdm\_symbol\_size + (gNB->frame\_parms.first\_carrier\_offset+pucch\_pdu->prb\_start\*12);

power\_rxF = signal\_energy\_nodc(&gNB->common\_vars.rxdataF[0][offset],12);

nr\_decode\_pucch0(gNB,

frame\_rx,

slot\_rx,

uci\_pdu\_format0,

pucch\_pdu);

gNB->UL\_INFO.uci\_ind.num\_ucis += 1;

pucch->active = 0;

break;

case 2:

num\_ucis = gNB->UL\_INFO.uci\_ind.num\_ucis;

gNB->UL\_INFO.uci\_ind.uci\_list = &gNB->uci\_pdu\_list[0];

gNB->UL\_INFO.uci\_ind.sfn = frame\_rx;

gNB->UL\_INFO.uci\_ind.slot = slot\_rx;

gNB->uci\_pdu\_list[num\_ucis].pdu\_type=NFAPI\_NR\_UCI\_FORMAT\_2\_3\_4PDU\_TYPE;

gNB->uci\_pdu\_list[num\_ucis].pdu\_size=sizeof(nfapi\_nr\_uci\_pucch\_pdu\_format\_2\_3 \_4\_t);

nfapi\_nr\_uci\_pucch\_pdu\_format\_2\_3\_4\_t\*uci\_pdu\_format2=&gNB->uci\_pdu\_list[n um\_ucis].pucch\_pdu\_format\_2\_3\_4;

nr\_decode\_pucch2(gNB,

slot\_rx,

uci\_pdu\_format2,

pucch\_pdu);

gNB->UL\_INFO.uci\_ind.num\_ucis += 1;

pucch->active = 0;

break;

default:

AssertFatal(1==0,"Only PUCCH formats 0 and 2 are currently supported\n");

}

}

}

}

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_NR\_RX\_PUSCH,1);

start\_meas(&gNB->rx\_pusch\_stats);

no\_sig = nr\_rx\_pusch(gNB, ULSCH\_id, frame\_rx, slot\_rx, harq\_pid);

if (no\_sig) {

LOG\_D(PHY, "PUSCH not detected in frame %d, slot %d\n", frame\_rx, slot\_rx);

nr\_fill\_indication(gNB, frame\_rx, slot\_rx, ULSCH\_id, harq\_pid, 1,1);

return 1;

}

gNB->pusch\_vars[ULSCH\_id]->ulsch\_power\_tot=0;

gNB->pusch\_vars[ULSCH\_id]->ulsch\_noise\_power\_tot=0;

for (int aarx=0;aarx<gNB->frame\_parms.nb\_antennas\_rx;aarx++) {

gNB->pusch\_vars[ULSCH\_id]->ulsch\_power[aarx]/=num\_dmrs;

gNB->pusch\_vars[ULSCH\_id]->ulsch\_power\_tot+=gNB->pusch\_vars[ULSCH\_id]-> ulsch\_power[aarx];

gNB->pusch\_vars[ULSCH\_id]->ulsch\_noise\_power[aarx]/=num\_dmrs;

gNB->pusch\_vars[ULSCH\_id]->ulsch\_noise\_power\_tot+=gNB->pusch\_vars[ULSCH \_id]->ulsch\_noise\_power[aarx];

}

if(dB\_fixed\_x10(gNB->pusch\_vars[ULSCH\_id]->ulsch\_power\_tot)<dB\_fixed\_x10(gNB- >pusch\_vars[ULSCH\_id]->ulsch\_noise\_power\_tot) + gNB->pusch\_thres) {

NR\_gNB\_SCH\_STATS\_t \*stats=get\_ulsch\_stats(gNB,ulsch);

LOG\_D(PHY, "PUSCH not detected in %d.%d (%d,%d,%d)\n",frame\_rx,slot\_rx,

dB\_fixed\_x10(gNB->pusch\_vars[ULSCH\_id]->ulsch\_power\_tot),dB\_fixed\_x10(gNB->pusch\_vars[ULSCH\_id]->ulsch\_noise\_power\_tot),gNB->pusch\_thres);

gNB->pusch\_vars[ULSCH\_id]->ulsch\_power\_tot=gNB->pusch\_vars[ULSCH\_id]->ul sch\_noise\_power\_tot;

gNB->pusch\_vars[ULSCH\_id]->DTX=1;

if (stats) stats->DTX++;

if (!get\_softmodem\_params()->phy\_test) {

nr\_fill\_indication(gNB,frame\_rx, slot\_rx, ULSCH\_id, harq\_pid, 1,1);

return 1;

}

} else {

LOG\_D(PHY, "PUSCH detected in %d.%d (%d,%d,%d)\n",frame\_rx,slot\_rx,

dB\_fixed\_x10(gNB->pusch\_vars[ULSCH\_id]->ulsch\_power\_tot),dB\_fixed\_x10(gNB->pusch\_vars[ULSCH\_id]->ulsch\_noise\_power\_tot),gNB->pusch\_thres);

gNB->pusch\_vars[ULSCH\_id]->DTX=0;

}

stop\_meas(&gNB->rx\_pusch\_stats);

nr\_ulsch\_procedures(gNB, frame\_rx, slot\_rx, ULSCH\_id, harq\_pid);

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_NR\_ULSCH\_PROCEDURES\_RX,0);

break;

}

}

}

}

nr\_srs\_channel\_estimation(gNB,frame\_rx,slot\_rx,srs\_pdu,

gNB->nr\_srs\_info[i],

gNB->nr\_srs\_info[i]->srs\_generated\_signal,

gNB->nr\_srs\_info[i]->srs\_received\_signal,

gNB->nr\_srs\_info[i]->srs\_estimated\_channel\_freq,

gNB->nr\_srs\_info[i]->srs\_estimated\_channel\_time, gNB->nr\_srs\_info[i]->srs\_estimated\_channel\_time\_shifted,gNB->nr\_srs\_info[i]->noise\_power);

}

}

}

stop\_meas(&gNB->phy\_proc\_rx);

VCD\_SIGNAL\_DUMPER\_DUMP\_FUNCTION\_BY\_NAME(VCD\_SIGNAL\_DUMPER\_FUNCTIONS\_PHY\_PROCEDURES\_gNB\_UESPEC\_RX,0);

return 0;

}

#### 4、openairinterface5g /openair1/PHY/NR\_ESTIMATION/nr\_measurements\_gNB.c

#include "PHY/types.h"

#include "PHY/defs\_gNB.h"

#include "PHY/phy\_extern.h"

#include "nr\_ul\_estimation.h"

int nr\_est\_timing\_advance\_pusch(PHY\_VARS\_gNB\* gNB, int UE\_id)

{

int max\_pos = 0, max\_val = 0;

NR\_DL\_FRAME\_PARMS \*frame\_parms = &gNB->frame\_parms;

NR\_gNB\_PUSCH \*gNB\_pusch\_vars = gNB->pusch\_vars[UE\_id];

int32\_t \*\*ul\_ch\_estimates\_time = gNB\_pusch\_vars->ul\_ch\_estimates\_time;

const int sync\_pos = 0;

for (int i = 0; i < frame\_parms->ofdm\_symbol\_size; i++) {

int temp = 0;

for (int aa = 0; aa < frame\_parms->nb\_antennas\_rx; aa++) {

int Re = ((int16\_t\*)ul\_ch\_estimates\_time[aa])[(i<<1)];

int Im = ((int16\_t\*)ul\_ch\_estimates\_time[aa])[1+(i<<1)];

temp += (Re\*Re/2) + (Im\*Im/2);

}

if (temp > max\_val) {

max\_pos = i;

max\_val = temp;

}

}

if (max\_pos > frame\_parms->ofdm\_symbol\_size/2)

max\_pos = max\_pos - frame\_parms->ofdm\_symbol\_size;

return max\_pos - sync\_pos;

}

void dump\_nr\_I0\_stats(FILE \*fd,PHY\_VARS\_gNB \*gNB) {

int min\_I0=1000,max\_I0=0;

int amin=0,amax=0;

fprintf(fd,"Blacklisted PRBs %d/%d\n",gNB->num\_ulprbbl,gNB->frame\_parms.N\_RB\_UL);

for (int i=0; i<gNB->frame\_parms.N\_RB\_UL; i++) {

if(gNB->ulprbbl[i] > 0) continue;

if(gNB->measurements.n0\_subband\_power\_tot\_dB[i]<min\_I0){min\_I0=gNB->measurements.n0\_subband\_power\_tot\_dB[i]; amin=i;}

if(gNB->measurements.n0\_subband\_power\_tot\_dB[i]>max\_I0){max\_I0=gNB->measurements.n0\_subband\_power\_tot\_dB[i]; amax=i;}

}

void gNB\_I0\_measurements(PHY\_VARS\_gNB \*gNB,int slot, int first\_symb,int num\_symb) {

NR\_DL\_FRAME\_PARMS \*frame\_parms = &gNB->frame\_parms;

NR\_gNB\_COMMON \*common\_vars = &gNB->common\_vars;

PHY\_MEASUREMENTS\_gNB \*measurements = &gNB->measurements;

int rb, nb\_symb[275]={0};

memset(measurements->n0\_subband\_power,sizeof(measurements->n0\_subband\_power));

for (int s=first\_symb;s<(first\_symb+num\_symb);s++) {

for (rb=0; rb<frame\_parms->N\_RB\_UL; rb++) {

if (s==first\_symb /\*&& ((gNB->rb\_mask\_ul[s][rb>>5]&(1<<(rb&31))) == 0)\*/) {

nb\_symb[rb]=0;

for (int aarx=0; aarx<frame\_parms->nb\_antennas\_rx;aarx++)

measurements->n0\_subband\_power[aarx][rb]=0;

}

if ((gNB->rb\_mask\_ul[s][rb>>5]&(1<<(rb&31))) == 0) { /

nb\_symb[rb]++;

for (int aarx=0; aarx<frame\_parms->nb\_antennas\_rx; aarx++) {

int offset = offset0 + (s\*frame\_parms->ofdm\_symbol\_size);

int32\_t \*ul\_ch = &common\_vars->rxdataF[aarx][offset];

int len = 12;

if (((frame\_parms->N\_RB\_UL&1) == 1) && (rb==(frame\_parms->N\_RB\_UL>>1))) {

len=6;

}

AssertFatal(ul\_ch, "RX signal buffer (freq) problem\n");

measurements->n0\_subband\_power[aarx][rb] += signal\_energy\_nodc(ul\_ch,len);

}

}

}

}

int nb\_rb=0;

int32\_t n0\_subband\_tot=0;

int32\_t n0\_subband\_tot\_perPRB=0;

int32\_t n0\_subband\_tot\_perANT[1+frame\_parms->nb\_antennas\_rx];

for (int rb = 0 ; rb<frame\_parms->N\_RB\_UL;rb++) {

n0\_subband\_tot\_perPRB=0;

if (nb\_symb[rb] > 0) {

for (int aarx=0;aarx<frame\_parms->nb\_antennas\_rx;aarx++) {

measurements->n0\_subband\_power[aarx][rb]/=nb\_symb[rb];

measurements->n0\_subband\_power\_dB[aarx][rb]=dB\_fixed(measurements->n0\_subb and\_power[aarx][rb]);

n0\_subband\_tot\_perPRB+=measurements->n0\_subband\_power[aarx][rb];

if(rb==0). n0\_subband\_tot\_perANT[aarx]=measurements->n0\_subband\_power[aarx][rb];

else n0\_subband\_tot\_perANT[aarx]+=measurements->n0\_subband\_power[aarx][rb];

}

n0\_subband\_tot\_perPRB/=frame\_parms->nb\_antennas\_rx;

measurements->n0\_subband\_power\_tot\_dB[rb] = dB\_fixed(n0\_subband\_tot\_perPRB);

measurements->n0\_subband\_power\_tot\_dBm[rb] =. measurements->n0\_subband\_power\_tot\_dB[rb]-gNB->rx\_total\_gain\_dB-dB\_fixed(frame\_parms->N\_RB\_UL);

n0\_subband\_tot += n0\_subband\_tot\_perPRB;

nb\_rb++;

}

}

if (nb\_rb>0) {

measurements->n0\_subband\_power\_avg\_dB = dB\_fixed(n0\_subband\_tot/nb\_rb);

for (int aarx=0;aarx<frame\_parms->nb\_antennas\_rx;aarx++) {

measurements->n0\_subband\_power\_avg\_perANT\_dB[aarx]=dB\_fixed(n0\_subban d\_tot\_perANT[aarx]/nb\_rb);

}

}

}

void nr\_gnb\_measurements(PHY\_VARS\_gNB \*gNB, uint8\_t ulsch\_id, unsigned char harq\_pid, unsigned char symbol, uint8\_t nrOfLayers){

int rx\_power\_tot[NUMBER\_OF\_NR\_ULSCH\_MAX];

int rx\_power[NUMBER\_OF\_NR\_ULSCH\_MAX][NB\_ANTENNAS\_RX];

unsigned short rx\_power\_avg\_dB[NUMBER\_OF\_NR\_ULSCH\_MAX];

unsigned short rx\_power\_tot\_dB[NUMBER\_OF\_NR\_ULSCH\_MAX];

double rx\_gain = openair0\_cfg[0].rx\_gain[0];

double rx\_gain\_offset = openair0\_cfg[0].rx\_gain\_offset[0];

PHY\_MEASUREMENTS\_gNB \*meas = &gNB->measurements;

NR\_DL\_FRAME\_PARMS \*fp = &gNB->frame\_parms;

int ch\_offset = fp->ofdm\_symbol\_size \* symbol;

int N\_RB\_UL = gNB->ulsch[ulsch\_id]->harq\_processes[harq\_pid]->ulsch\_pdu.rb\_size;

rx\_power\_tot[ulsch\_id] = 0;

for (int aarx = 0; aarx < fp->nb\_antennas\_rx; aarx++){

rx\_power[ulsch\_id][aarx] = 0;

for (int aatx = 0; aatx < nrOfLayers; aatx++){

meas->rx\_spatial\_power[ulsch\_id][aatx][aarx]=(signal\_energy\_nodc(&gNB->pusch\_vars[ulsch\_id]->ul\_ch\_estimates[aatx\*fp->nb\_antennas\_rx+aarx][ch\_offset],N\_RB\_UL\*NR\_NB\_SC\_PER\_RB));

if (meas->rx\_spatial\_power[ulsch\_id][aatx][aarx] < 0) {

meas->rx\_spatial\_power[ulsch\_id][aatx][aarx] = 0;

}

meas->rx\_spatial\_power\_dB[ulsch\_id][aatx][aarx] = (unsigned short) dB\_fixed(meas->rx\_spatial\_power[ulsch\_id][aatx][aarx]);

rx\_power[ulsch\_id][aarx] += meas->rx\_spatial\_power[ulsch\_id][aatx][aarx];

}

rx\_power\_tot[ulsch\_id] += rx\_power[ulsch\_id][aarx];

}

rx\_power\_tot\_dB[ulsch\_id] = (unsigned short) dB\_fixed(rx\_power\_tot[ulsch\_id]);

rx\_power\_avg\_dB[ulsch\_id] = rx\_power\_tot\_dB[ulsch\_id];

meas->wideband\_cqi\_tot[ulsch\_id]=dB\_fixed2(rx\_power\_tot[ulsch\_id], meas->n0\_power\_tot);

meas->rx\_rssi\_dBm[ulsch\_id] = rx\_power\_avg\_dB[ulsch\_id] + 30 - 10 \* log10(pow(2, 30)) - (rx\_gain - rx\_gain\_offset) - dB\_fixed(fp->ofdm\_symbol\_size);

LOG\_D(PHY, "[ULSCH %d] RSSI %d dBm/RE, RSSI (digital) %d dB (N\_RB\_UL %d), WBand CQI tot %d dB, N0 Power tot %d, RX Power tot %d\n",

ulsch\_id,

meas->rx\_rssi\_dBm[ulsch\_id],

rx\_power\_avg\_dB[ulsch\_id],

N\_RB\_UL,

meas->wideband\_cqi\_tot[ulsch\_id],

meas->n0\_power\_tot,

rx\_power\_tot[ulsch\_id]);

}

#### 5、openairinterface5g /openair1/SIMULATION/NR\_PHY/ulsim.c

#include <string.h>

#include <math.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#include <sys/mman.h>

#include "common/ran\_context.h"

#include "common/config/config\_userapi.h"

#include "common/utils/LOG/log.h"

#include "PHY/defs\_gNB.h"

#include "PHY/defs\_nr\_common.h"

#include "PHY/defs\_nr\_UE.h"

#include "PHY/phy\_vars\_nr\_ue.h"

#include "PHY/types.h"

#include "PHY/INIT/phy\_init.h"

#include "PHY/MODULATION/modulation\_UE.h"

#include "PHY/MODULATION/nr\_modulation.h"

#include "PHY/NR\_REFSIG/dmrs\_nr.h"

#include "PHY/NR\_REFSIG/refsig\_defs\_ue.h"

#include "PHY/NR\_TRANSPORT/nr\_dlsch.h"

#include "PHY/NR\_TRANSPORT/nr\_sch\_dmrs.h"

#include "PHY/NR\_TRANSPORT/nr\_transport\_proto.h"

#include "PHY/NR\_TRANSPORT/nr\_ulsch.h"

#include "PHY/NR\_UE\_TRANSPORT/nr\_transport\_proto\_ue.h"

#include "PHY/TOOLS/tools\_defs.h"

#include "SCHED\_NR/fapi\_nr\_l1.h"

#include "SCHED\_NR/sched\_nr.h"

#include "SCHED\_NR\_UE/defs.h"

#include "SCHED\_NR\_UE/fapi\_nr\_ue\_l1.h"

#include "openair1/SIMULATION/TOOLS/sim.h"

#include "openair1/SIMULATION/RF/rf.h"

#include "openair1/SIMULATION/NR\_PHY/nr\_unitary\_defs.h"

#include "openair2/RRC/NR/MESSAGES/asn1\_msg.h"

#include "openair2/LAYER2/NR\_MAC\_UE/mac\_proto.h"

#include "openair2/LAYER2/NR\_MAC\_gNB/mac\_proto.h"

#include "common/utils/threadPool/thread-pool.h"

#include "PHY/NR\_REFSIG/ptrs\_nr.h"

#define inMicroS(a) (((double)(a))/(get\_cpu\_freq\_GHz()\*1000.0))

#include "SIMULATION/LTE\_PHY/common\_sim.h"

#include <openair2/LAYER2/MAC/mac\_vars.h>

#include <openair2/RRC/LTE/rrc\_vars.h>

#include <executables/softmodem-common.h>

#include "PHY/NR\_REFSIG/ul\_ref\_seq\_nr.h"

#include <openair3/ocp-gtpu/gtp\_itf.h>

PHY\_VARS\_gNB \*gNB;

PHY\_VARS\_NR\_UE \*UE;

RAN\_CONTEXT\_t RC;

char \*uecap\_file;

int32\_t uplink\_frequency\_offset[MAX\_NUM\_CCs][4];

uint16\_t sf\_ahead=4 ;

int slot\_ahead=6 ;

uint16\_t sl\_ahead=0;

double cpuf;

uint64\_t downlink\_frequency[MAX\_NUM\_CCs][4];

THREAD\_STRUCT thread\_struct;

nfapi\_ue\_release\_request\_body\_t release\_rntis;

instance\_t DUuniqInstance=0;

instance\_t CUuniqInstance=0;

teid\_t newGtpuCreateTunnel(instance\_t instance, rnti\_t rnti, int incoming\_bearer\_id, int outgoing\_bearer\_id, teid\_t outgoing\_teid, transport\_layer\_addr\_t remoteAddr, int port, gtpCallback callBack) {

return 0;

}

int8\_t nr\_mac\_rrc\_data\_ind\_ue(const module\_id\_t module\_id,

const int CC\_id,

const uint8\_t gNB\_index,

const frame\_t frame,

const sub\_frame\_t sub\_frame,

const rnti\_t rnti,

const channel\_t channel,

const uint8\_t\* pduP,

const sdu\_size\_t pdu\_len)

{

return 0;

}

int generate\_dlsch\_header(unsigned char \*mac\_header,

unsigned char num\_sdus,

unsigned short \*sdu\_lengths,

unsigned char \*sdu\_lcids,

unsigned char drx\_cmd,

unsigned short timing\_advance\_cmd,

unsigned char \*ue\_cont\_res\_id,

unsigned char short\_padding,

unsigned short post\_padding){return 0;}

typedef struct {

uint64\_t optmask;

uint8\_t nr\_dlsch\_parallel;

tpool\_t Tpool;

} nrUE\_params\_t;

void processSlotTX(void \*arg) {}

nrUE\_params\_t nrUE\_params;

nrUE\_params\_t \*get\_nrUE\_params(void) {

return &nrUE\_params;

}

channel\_desc\_t \*UE2gNB[NUMBER\_OF\_UE\_MAX][NUMBER\_OF\_gNB\_MAX];

int main(int argc, char \*\*argv)

{

char c;

int i;

double SNR, snr0 = -2.0, snr1 = 2.0;

double sigma, sigma\_dB;

double snr\_step = .2;

uint8\_t snr1set = 0;

int slot = 8, frame = 1;

FILE \*output\_fd = NULL;

double \*\*s\_re,\*\*s\_im,\*\*r\_re,\*\*r\_im;

int trial, n\_trials = 1, n\_false\_positive = 0, delay = 0;

double maxDoppler = 0.0;

uint8\_t n\_tx = 1, n\_rx = 1;

channel\_desc\_t \*UE2gNB;

uint8\_t extended\_prefix\_flag = 0;

FILE \*input\_fd = NULL;

SCM\_t channel\_model = AWGN;

uint16\_t N\_RB\_DL = 106, N\_RB\_UL = 106, mu = 1;

NB\_UE\_INST = 1;

NR\_DL\_FRAME\_PARMS \*frame\_parms;

int loglvl = OAILOG\_WARNING;

uint16\_t nb\_symb\_sch = 12;

int start\_symbol = 0;

uint16\_t nb\_rb = 50;

int Imcs = 9;

uint8\_t precod\_nbr\_layers = 1;

int gNB\_id = 0;

int ap;

int tx\_offset;

int32\_t txlev\_sum = 0, atxlev[4];

int start\_rb = 0;

int UE\_id =0;

int print\_perf = 0;

cpuf = get\_cpu\_freq\_GHz();

int msg3\_flag = 0;

int rv\_index = 0;

float roundStats[100];

double effRate[100];

double effTP[100];

float eff\_tp\_check = 100;

uint8\_t snrRun;

int chest\_type[2] = {0};

int enable\_ptrs = 0;

int modify\_dmrs = 0;

int ptrs\_arg[2] = {-1,-1};

int dmrs\_arg[4] = {-1,-1,-1,-1};

uint16\_t ptrsSymPos = 0;

uint16\_t ptrsSymbPerSlot = 0;

uint16\_t ptrsRePerSymb = 0;

uint8\_t transform\_precoding = transformPrecoder\_disabled;

uint8\_t num\_dmrs\_cdm\_grps\_no\_data = 1;

uint8\_t mcs\_table = 0;

UE\_nr\_rxtx\_proc\_t UE\_proc;

FILE \*scg\_fd=NULL;

int file\_offset = 0;

double DS\_TDL = .03;

int ibwps=24;

int ibwp\_rboffset=41;

int params\_from\_file = 0;

int max\_ldpc\_iterations = 5;

if ( load\_configmodule(argc,argv,CONFIG\_ENABLECMDLINEONLY) == 0 ) {

exit\_fun("[NR\_ULSIM] Error, configuration module init failed\n");

}

int ul\_proc\_error = 0;

randominit(0);

while((c=getopt(argc,argv,"a:b:c:d:ef:g:h:i:kl:m:n:p:q:r:s:t:u:w:y:z:F:G:H:I:M:N:PR:S:T:U:L:ZW:")) != -1) {

printf("handling optarg %c\n",c);

switch (c) {

case 'a':

start\_symbol = atoi(optarg);

AssertFatal(start\_symbol >= 0 && start\_symbol < 13,"start\_symbol %d is not in 0..12\n",start\_symbol);

break;

case 'b':

nb\_symb\_sch = atoi(optarg);

AssertFatal(nb\_symb\_sch > 0 && nb\_symb\_sch < 15,"start\_symbol %d is not in 1..14\n",nb\_symb\_sch);

break;

case 'c':

n\_rnti = atoi(optarg);

AssertFatal(n\_rnti > 0 && n\_rnti<=65535,"Illegal n\_rnti %x\n",n\_rnti);

break;

case 'd':

delay = atoi(optarg);

break；

case 'e':

msg3\_flag = 1;

break;

case 'f':

scg\_fd = fopen(optarg, "r");

if (scg\_fd == NULL) {

printf("Error opening %s\n", optarg);

exit(-1);

}

break;

case 'g':

switch ((char) \*optarg) {

case 'A':

channel\_model = SCM\_A;

break;

case 'B':

channel\_model = SCM\_B;

break;

case 'C':

channel\_model = SCM\_C;

break;

case 'D':

channel\_model = SCM\_D;

break;

case 'E':

channel\_model = EPA;

break;

case 'F':

channel\_model = EVA;

break;

case 'G':

channel\_model = ETU;

break;

case 'H':

channel\_model = TDL\_C;

DS\_TDL = .030;

break;

case 'I':

channel\_model = TDL\_C;

DS\_TDL = .3; // 300ns

break;

case 'J':

channel\_model=TDL\_D;

DS\_TDL = .03;

break;

default:

printf("Unsupported channel model!\n");

exit(-1);

}

break;

case 'i':

for(i=0; i < atoi(optarg); i++){

chest\_type[i] = atoi(argv[optind++]);

}

break;

case 'k':

printf("Setting threequarter\_fs\_flag\n");

openair0\_cfg[0].threequarter\_fs= 1;

break;

case 'l':

nb\_symb\_sch = atoi(optarg);

break;

case 'm':

Imcs = atoi(optarg);

break;

case 'W':

precod\_nbr\_layers = atoi(optarg);

break;

case 'n':

n\_trials = atoi(optarg);

break;

case 'p':

extended\_prefix\_flag = 1;

break;

case 'q':

mcs\_table = atoi(optarg);

break;

case 'r':

nb\_rb = atoi(optarg);

break;

case 's':

snr0 = atof(optarg);

printf("Setting SNR0 to %f\n", snr0);

break;

case 'u':

mu = atoi(optarg);

break;

case 'w':

start\_rb = atoi(optarg);

break;

case 't':

eff\_tp\_check = (float)atoi(optarg);

break;

case 'y':

n\_tx = atoi(optarg);

if ((n\_tx == 0) || (n\_tx > 4)) {

printf("Unsupported number of tx antennas %d\n", n\_tx);

exit(-1);

}

break;

case 'z':

n\_rx = atoi(optarg);

if ((n\_rx == 0) || (n\_rx > 8)) {

printf("Unsupported number of rx antennas %d\n", n\_rx);

exit(-1);

}

break;

case 'F':

input\_fd = fopen(optarg, "r");

if (input\_fd == NULL) {

printf("Problem with filename %s\n", optarg);

exit(-1);

}

break;

case 'G':

file\_offset = atoi(optarg);

break;

case 'H':

slot = atoi(optarg);

break;

case 'I':

max\_ldpc\_iterations = atoi(optarg);

break;

case 'M':

break;

case 'N':

break;

case 'R':

N\_RB\_DL = atoi(optarg);

N\_RB\_UL = N\_RB\_DL;

break;

case 'S':

snr1 = atof(optarg);

snr1set = 1;

printf("Setting SNR1 to %f\n", snr1);

break;

case 'P':

print\_perf=1;

opp\_enabled=1;

break;

case 'L':

loglvl = atoi(optarg);

break;

case 'T':

enable\_ptrs=1;

for(i=0; i < atoi(optarg); i++){

ptrs\_arg[i] = atoi(argv[optind++]);

}

break;

case 'U':

modify\_dmrs = 1;

for(i=0; i < atoi(optarg); i++){

dmrs\_arg[i] = atoi(argv[optind++]);

}

break;

case 'Q':

params\_from\_file = 1;

break;

case 'Z':

transform\_precoding = transformPrecoder\_enabled;

num\_dmrs\_cdm\_grps\_no\_data = 2;

mcs\_table = 3;

printf("NOTE: TRANSFORM PRECODING (SC-FDMA) is ENABLED in UPLINK (0 - ENABLE, 1 - DISABLE) : %d \n", transform\_precoding);

break;

default:

case 'h':

printf("-d Introduce delay in terms of number of samples\n");

printf("-f Number of frames to simulate\n");

printf("-g [A,B,C,D,E,F,G] Use 3GPP SCM (A,B,C,D) or 36-101 (E-EPA,F-EVA,G-ETU) models (ignores delay spread and Ricean factor)\n");

printf("-h This message\n");

printf("-i Change channel estimation technique. Arguments list: Number of arguments=2, Frequency domain {0:Linear interpolation, 1:PRB based averaging}, Time domain {0:Estimates of last DMRS symbol, 1:Average of DMRS symbols}. e.g. -i 2 1 0\n");

printf("-s Starting SNR, runs from SNR0 to SNR0 + 10 dB if ending SNR isn't given\n");

printf("-m MCS value\n");

printf("-n Number of trials to simulate\n");

printf("-p Use extended prefix mode\n");

printf("-q MCS table\n");

printf("-t Delay spread for multipath channel\n");

printf("-u Set the numerology\n");

printf("-w Start PRB for PUSCH\n");

printf("-y Number of TX antennas used at UE\n");

printf("-z Number of RX antennas used at gNB\n");

printf("-F Input filename (.txt format) for RX conformance testing\n");

printf("-G Offset of samples to read from file (0 default)\n");

printf("-L <log level, 0(errors), 1(warning), 2(info) 3(debug) 4 (trace)>\n");

printf("-I Maximum LDPC decoder iterations\n");

printf("-M Multiple SSB positions in burst\n");

printf("-N Nid\_cell\n");

printf("-O oversampling factor (1,2,4,8,16)\n");

printf("-R N\_RB\_DL\n");

printf("-t Acceptable effective throughput (in percentage)\n");

printf("-S Ending SNR, runs from SNR0 to SNR1\n");

printf("-P Print ULSCH performances\n");

printf("-T Enable PTRS, arguments list: Number of arguments=2 L\_PTRS{0,1,2} K\_PTRS{2,4}, e.g. -T 2 0 2 \n");

printf("-U Change DMRS Config, arguments list: Number of arguments=4, DMRS Mapping Type{0=A,1=B}, DMRS AddPos{0:3}, DMRS Config Type{1,2}, Number of CDM groups without data{1,2,3} e.g. -U 4 0 2 0 1 \n");

printf("-Q If -F used, read parameters from file\n");

printf("-Z If -Z is used, SC-FDMA or transform precoding is enabled in Uplink \n");

printf("-W Num of layer for PUSCH\n")

exit(-1);

break;

}

}

get\_softmodem\_params()->phy\_test = 1;

get\_softmodem\_params()->do\_ra = 0;

get\_softmodem\_params()->usim\_test = 1;

if (snr1set == 0)

snr1 = snr0 + 10;

double sampling\_frequency;

double bandwidth;

if (mu == 0 && N\_RB\_UL == 25 ) {

sampling\_frequency = 7.68;

bandwidth = 5;

}

else if (mu == 1 && N\_RB\_UL == 273) {

sampling\_frequency = 122.88;

bandwidth = 100;

}

else if (mu == 1 && N\_RB\_UL == 217) {

sampling\_frequency = 122.88;

bandwidth = 80;

}

else if (mu == 1 && N\_RB\_UL == 106) {

sampling\_frequency = 61.44;

bandwidth = 40;

}

else if (mu == 1 && N\_RB\_UL == 24) {

sampling\_frequency = 15.36;

bandwidth = 10;

}

else if (mu == 3 && N\_RB\_UL == 32) {

sampling\_frequency = 61.44;

bandwidth = 50;

}

else {

printf("Add N\_RB\_UL %d\n",N\_RB\_UL);

exit(-1);

}

frame\_parms->N\_RB\_DL = N\_RB\_DL;

frame\_parms->N\_RB\_UL = N\_RB\_UL;

frame\_parms->Ncp = extended\_prefix\_flag ? EXTENDED : NORMAL;

s\_re = malloc(n\_tx\*sizeof(double\*));

s\_im = malloc(n\_tx\*sizeof(double\*));

r\_re = malloc(n\_rx\*sizeof(double\*));

r\_im = malloc(n\_rx\*sizeof(double\*));

NR\_ServingCellConfigCommon\_t \*scc = rrc.carrier.servingcellconfigcommon;

NR\_ServingCellConfig\_t \*scd = calloc(1,sizeof(NR\_ServingCellConfig\_t));

NR\_CellGroupConfig\_t \*secondaryCellGroup=calloc(1,sizeof(\*secondaryCellGroup));

prepare\_scc(rrc.carrier.servingcellconfigcommon);

uint64\_t ssb\_bitmap;

fill\_scc\_sim(rrc.carrier.servingcellconfigcommon,&ssb\_bitmap,N\_RB\_DL,N\_RB\_DL,mu,mu);

gNB->if\_inst->NR\_PHY\_config\_req = nr\_phy\_config\_request;

rrc\_mac\_config\_req\_gNB(0,0, conf.pdsch\_AntennaPorts, n\_rx, 0, 6, scc, &rrc.carrier.mib, rrc.carrier.siblock1, 0, 0, NULL);

rrc\_mac\_config\_req\_gNB(0,0, conf.pdsch\_AntennaPorts, n\_rx, 0, 6, scc, &rrc.carrier.mib, rrc.carrier.siblock1, 1,

frame\_parms->nb\_antennas\_tx = 1;

frame\_parms->nb\_antennas\_rx = n\_rx;

nfapi\_nr\_config\_request\_scf\_t \*cfg = &gNB->gNB\_config;

cfg->carrier\_config.num\_tx\_ant.value = 1;

cfg->carrier\_config.num\_rx\_ant.value = n\_rx;

gNB->chest\_freq = chest\_type[0];

gNB->chest\_time = chest\_type[1];

phy\_init\_nr\_gNB(gNB,0,1);

N\_RB\_DL = gNB->frame\_parms.N\_RB\_DL;

NR\_BWP\_Uplink\_t

UE = malloc(sizeof(PHY\_VARS\_NR\_UE));

memset((void\*)UE,0,sizeof(PHY\_VARS\_NR\_UE));

PHY\_vars\_UE\_g = malloc(sizeof(PHY\_VARS\_NR\_UE\*\*));

PHY\_vars\_UE\_g[0] = malloc(sizeof(PHY\_VARS\_NR\_UE\*));

PHY\_vars\_UE\_g[0][0] = UE;

memcpy(&UE->frame\_parms, frame\_parms, sizeof(NR\_DL\_FRAME\_PARMS));

UE->frame\_parms.nb\_antennas\_tx = n\_tx;

UE->frame\_parms.nb\_antennas\_rx = 0;

if (init\_nr\_ue\_signal(UE, 1) != 0) {

printf("Error at UE NR initialisation\n");

exit(-1);

}

init\_nr\_ue\_transport(UE);

for(int n\_scid = 0; n\_scid<2; n\_scid++) {

UE->scramblingID\_ulsch[n\_scid] = frame\_parms->Nid\_cell;

nr\_init\_pusch\_dmrs(UE, frame\_parms->Nid\_cell, n\_scid);

}

NR\_UE\_RRC\_INST\_t rrcue;

memset(&rrcue,0,sizeof(NR\_UE\_RRC\_INST\_t));

rrc.carrier.MIB = (uint8\_t\*) malloc(4);

rrc.carrier.sizeof\_MIB = do\_MIB\_NR(&rrc,0);

rrcue.mib = rrc.carrier.mib.message.choice.mib;

rrcue.scell\_group\_config=secondaryCellGroup;

nr\_l2\_init\_ue(&rrcue);

NR\_UE\_MAC\_INST\_t\* UE\_mac = get\_mac\_inst(0);

UE->if\_inst = nr\_ue\_if\_module\_init(0);

UE->if\_inst->scheduled\_response = nr\_ue\_scheduled\_response;

UE->if\_inst->phy\_config\_request = nr\_ue\_phy\_config\_request;

UE->if\_inst->dl\_indication = nr\_ue\_dl\_indication;

UE->if\_inst->ul\_indication = nr\_ue\_ul\_indication;

UE\_mac->if\_module = nr\_ue\_if\_module\_init(0);

nr\_ue\_phy\_config\_request(&UE\_mac->phy\_config);

unsigned char harq\_pid = 0;

NR\_gNB\_ULSCH\_t \*ulsch\_gNB = gNB->ulsch[UE\_id];

NR\_Sched\_Rsp\_t \*Sched\_INFO = malloc(sizeof(\*Sched\_INFO));

memset((void\*)Sched\_INFO,0,sizeof(\*Sched\_INFO));

Sched\_INFO->UL\_tti\_req=UL\_tti\_req;

NR\_UE\_ULSCH\_t \*ulsch\_ue = UE->ulsch[0][0];

unsigned char \*estimated\_output\_bit;

unsigned char \*test\_input\_bit;

uint32\_t errors\_decoding = 0;

test\_input\_bit = (unsigned char \*) malloc16(sizeof(unsigned char) \* 16 \* 68 \* 384);

estimated\_output\_bit = (unsigned char \*) malloc16(sizeof(unsigned char) \* 16 \* 68 \* 384);

nr\_scheduled\_response\_t scheduled\_response;

memset(&scheduled\_response, 0, sizeof(scheduled\_response));

memset(&ul\_config, 0, sizeof(ul\_config));

memset(&tx\_req, 0, sizeof(tx\_req));

uint8\_t ptrs\_mcs1 = 2;

uint8\_t ptrs\_mcs2 = 4;

uint8\_t ptrs\_mcs3 = 10;

uint16\_t n\_rb0 = 25;

uint16\_t n\_rb1 = 75;

uint8\_t max\_rounds = 4;

uint8\_t crc\_status = 0;

unsigned char mod\_order = nr\_get\_Qm\_ul(Imcs, mcs\_table);

uint16\_t code\_rate = nr\_get\_code\_rate\_ul(Imcs, mcs\_table);

uint8\_t mapping\_type = typeB; // Default Values

pusch\_dmrs\_type\_t dmrs\_config\_type = pusch\_dmrs\_type1; // Default Values

pusch\_dmrs\_AdditionalPosition\_t add\_pos = pusch\_dmrs\_pos0; // Default Values

if(modify\_dmrs) {

if(dmrs\_arg[0] == 0)

mapping\_type = typeA;

else if (dmrs\_arg[0] == 1)

mapping\_type = typeB;

if(dmrs\_arg[1] >= 0 && dmrs\_arg[1] <=3 )

add\_pos = dmrs\_arg[1];

if(dmrs\_arg[2] == 1)

dmrs\_config\_type = pusch\_dmrs\_type1;

else if(dmrs\_arg[2] == 2)

dmrs\_config\_type = pusch\_dmrs\_type2;

num\_dmrs\_cdm\_grps\_no\_data = dmrs\_arg[3];

}

}

uint32\_t errors\_scrambling[4][100];

int n\_errors[4][100];

int round\_trials[4][100];

double blerStats[4][100];

double berStats[4][100];

double snrStats[100];

memset(errors\_scrambling, 0, sizeof(uint32\_t)\*4\*100);

memset(n\_errors, 0, sizeof(int)\*4\*100);

memset(round\_trials, 0, sizeof(int)\*4\*100);

memset(blerStats, 0, sizeof(double)\*4\*100);

memset(berStats, 0, sizeof(double)\*4\*100);

memset(snrStats, 0, sizeof(double)\*100);

for (SNR = snr0; SNR < snr1; SNR += snr\_step) {

varArray\_t \*table\_rx=initVarArray(1000,sizeof(double));

int error\_flag = 0;

n\_false\_positive = 0;

effRate[snrRun] = 0;

effTP[snrRun] = 0;

reset\_meas(&gNB->phy\_proc\_rx);

reset\_meas(&gNB->rx\_pusch\_stats);

reset\_meas(&gNB->ulsch\_decoding\_stats);

reset\_meas(&gNB->ulsch\_deinterleaving\_stats);

reset\_meas(&gNB->ulsch\_rate\_unmatching\_stats);

reset\_meas(&gNB->ulsch\_ldpc\_decoding\_stats);

reset\_meas(&gNB->ulsch\_unscrambling\_stats);

reset\_meas(&gNB->ulsch\_channel\_estimation\_stats);

reset\_meas(&gNB->ulsch\_llr\_stats);

reset\_meas(&gNB->ulsch\_channel\_compensation\_stats);

reset\_meas(&gNB->ulsch\_rbs\_extraction\_stats);

reset\_meas(&UE->ulsch\_ldpc\_encoding\_stats);

reset\_meas(&UE->ulsch\_rate\_matching\_stats);

reset\_meas(&UE->ulsch\_interleaving\_stats);

reset\_meas(&UE->ulsch\_encoding\_stats);

clear\_pusch\_stats(gNB);

for (trial = 0; trial < n\_trials; trial++) {

uint8\_t round = 0;

crc\_status = 1;

errors\_decoding = 0;

memset((void\*)roundStats,0,50\*sizeof(roundStats[0]));

while (round<max\_rounds && crc\_status) {

round\_trials[round][snrRun]++;

ulsch\_ue->harq\_processes[harq\_pid]->round = round;

gNB->ulsch[0]->harq\_processes[harq\_pid]->round = round;

rv\_index = nr\_rv\_round\_map[round];

UE\_proc.thread\_id = 0;

UE\_proc.nr\_slot\_tx = slot;

UE\_proc.frame\_tx = frame;

UL\_tti\_req->SFN = frame;

UL\_tti\_req->Slot = slot;

UL\_tti\_req->n\_pdus = 1;

UL\_tti\_req->pdus\_list[0].pdu\_type = NFAPI\_NR\_UL\_CONFIG\_PUSCH\_PDU\_TYPE;

UL\_tti\_req->pdus\_list[0].pdu\_size = sizeof(nfapi\_nr\_pusch\_pdu\_t);

memset(pusch\_pdu,0,sizeof(nfapi\_nr\_pusch\_pdu\_t));

int ibwp\_size = ibwps;

int ibwp\_start = ibwp\_rboffset;

if (msg3\_flag == 1) {

if ((ibwp\_start < abwp\_start) || (ibwp\_size > abwp\_size))

pusch\_pdu->bwp\_start = abwp\_start;

else

pusch\_pdu->bwp\_start = ibwp\_start;

pusch\_pdu->bwp\_size = ibwp\_size;

start\_rb = (ibwp\_start - abwp\_start);

printf("msg3: ibwp\_size %d, abwp\_size %d, ibwp\_start %d, abwp\_start %d\n",

ibwp\_size,abwp\_size,ibwp\_start,abwp\_start);

}

else {

pusch\_pdu->bwp\_start = abwp\_start;

pusch\_pdu->bwp\_size = abwp\_size;

}

pusch\_pdu->pusch\_data.tb\_size = TBS/8;

pusch\_pdu->pdu\_bit\_map = pdu\_bit\_map;

pusch\_pdu->rnti = n\_rnti;

pusch\_pdu->mcs\_index = Imcs;

pusch\_pdu->mcs\_table = mcs\_table;

pusch\_pdu->target\_code\_rate = code\_rate;

pusch\_pdu->qam\_mod\_order = mod\_order;

pusch\_pdu->transform\_precoding = transform\_precoding;

pusch\_pdu->data\_scrambling\_id = \*scc->physCellId;

pusch\_pdu->nrOfLayers = precod\_nbr\_layers;

pusch\_pdu->ul\_dmrs\_symb\_pos = l\_prime\_mask;

pusch\_pdu->dmrs\_config\_type = dmrs\_config\_type;

pusch\_pdu->ul\_dmrs\_scrambling\_id = \*scc->physCellId;

pusch\_pdu->scid = 0;

pusch\_pdu->dmrs\_ports = ((1<<precod\_nbr\_layers)-1);

pusch\_pdu->num\_dmrs\_cdm\_grps\_no\_data = num\_dmrs\_cdm\_grps\_no\_data;

pusch\_pdu->resource\_alloc = 1;

pusch\_pdu->rb\_start = start\_rb;

pusch\_pdu->rb\_size = nb\_rb;

pusch\_pdu->vrb\_to\_prb\_mapping = 0;

pusch\_pdu->frequency\_hopping = 0;

pusch\_pdu->uplink\_frequency\_shift\_7p5khz = 0;

pusch\_pdu->start\_symbol\_index = start\_symbol;

pusch\_pdu->nr\_of\_symbols = nb\_symb\_sch;

pusch\_pdu->pusch\_data.rv\_index = rv\_index;

pusch\_pdu->pusch\_data.harq\_process\_id = 0;

pusch\_pdu->pusch\_data.new\_data\_indicator = trial & 0x1;

pusch\_pdu->pusch\_data.num\_cb = 0;

pusch\_pdu->pusch\_ptrs.ptrs\_time\_density = ptrs\_time\_density;

pusch\_pdu->pusch\_ptrs.ptrs\_freq\_density = ptrs\_freq\_density;

pusch\_pdu->pusch\_ptrs.ptrs\_ports\_list[0].ptrs\_re\_offset = 0;

scheduled\_response.module\_id = 0;

scheduled\_response.CC\_id = 0;

scheduled\_response.frame = frame;

scheduled\_response.slot = slot;

scheduled\_response.thread\_id = UE\_proc.thread\_id;

scheduled\_response.dl\_config = NULL;

scheduled\_response.ul\_config = &ul\_config;

scheduled\_response.tx\_request = &tx\_req;

tx\_req.slot = slot;

tx\_req.sfn = frame;

tx\_req.number\_of\_pdus = 1;

tx\_req.tx\_request\_body[0].pdu\_length = TBS/8;

tx\_req.tx\_request\_body[0].pdu\_index = 0;

tx\_req.tx\_request\_body[0].pdu = &ulsch\_input\_buffer[0];

ul\_config.slot = slot;

ul\_config.number\_pdus = 1;

ul\_config.ul\_config\_list[0].pdu\_type = FAPI\_NR\_UL\_CONFIG\_TYPE\_PUSCH;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.rnti = n\_rnti;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.pdu\_bit\_map = pdu\_bit\_map;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.qam\_mod\_order = mod\_order;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.rb\_size = nb\_rb;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.rb\_start = start\_rb;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.nr\_of\_symbols = nb\_symb\_sch;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.start\_symbol\_index = start\_symbol;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.ul\_dmrs\_symb\_pos = l\_prime\_mask;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.dmrs\_config\_type = dmrs\_config\_type;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.mcs\_index = Imcs;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.mcs\_table = mcs\_table;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.num\_dmrs\_cdm\_grps\_no\_data=num\_dmrs\_cdm\_grps\_no\_data;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.nrOfLayers = precod\_nbr\_layers;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.dmrs\_ports = ((1<<precod\_nbr\_layers)-1);

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.absolute\_delta\_PUSCH = 0;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.target\_code\_rate = code\_rate;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.pusch\_data.tb\_size = TBS/8;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.pusch\_data.new\_data\_indicator=trial & 0x1;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.pusch\_data.rv\_index = rv\_index;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.pusch\_data.harq\_process\_id = harq\_pid;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.pusch\_ptrs.ptrs\_time\_density=ptrs\_time\_density;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.pusch\_ptrs.ptrs\_freq\_density=ptrs\_freq\_density;

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.pusch\_ptrs.ptrs\_ports\_list=(nfapi\_nr\_ue\_ptrs\_ports\_t \*) malloc(2\*sizeof(nfapi\_nr\_ue\_ptrs\_ports\_t));

ul\_config.ul\_config\_list[0].pusch\_config\_pdu.transform\_precoding= transform\_precoding;

}

phy\_procedures\_nrUE\_TX(UE, &UE\_proc, gNB\_id);

LOG\_D(PHY, "Sending Uplink data \n");

nr\_ue\_pusch\_common\_procedures(UE,

slot,

&UE->frame\_parms,

UE->frame\_parms.nb\_antennas\_tx);

}

}

tx\_offset = frame\_parms->get\_samples\_slot\_timestamp(slot,frame\_parms,0);

txlev\_sum = 0;

for (int aa=0; aa<UE->frame\_parms.nb\_antennas\_tx; aa++) {

atxlev[aa]=signal\_energy(&UE->common\_vars.txdata[aa][tx\_offset+5\*frame\_parms->ofdm\_symbol\_size+4\*frame\_parms->nb\_prefix\_samples+frame\_parms->nb\_prefix\_samples0],frame\_parms->ofdm\_symbol\_size+frame\_parms->nb\_prefix\_samples);

txlev\_sum += atxlev[aa];

}

}

else

n\_trials = 1;

}

}

gNB->UL\_INFO.rx\_ind.number\_of\_pdus = 0;

gNB->UL\_INFO.crc\_ind.number\_crcs = 0;

phy\_procedures\_gNB\_common\_RX(gNB, frame, slot);

ul\_proc\_error = phy\_procedures\_gNB\_uespec\_RX(gNB, frame, slot);

}

}

if((gNB->ulsch[0]->last\_iteration\_cnt>=gNB->ulsch[0]->max\_ldpc\_iterations+1)||ul\_proc\_error == 1) {

error\_flag = 1;

n\_errors[round][snrRun]++;

crc\_status = 1;

} else

crc\_status = 0;

if(n\_trials==1) printf("end of round %d rv\_index %d\n",round, rv\_index);

if ((pusch\_pdu->pdu\_bit\_map & PUSCH\_PDU\_BITMAP\_PUSCH\_PTRS) && (SNR==snr0) && (trial==0) && (round==0)) {

ptrs\_symbols = 0;

for (int i = pusch\_pdu->start\_symbol\_index; i < pusch\_pdu->start\_symbol\_index + pusch\_pdu->nr\_of\_symbols; i++)

ptrs\_symbols += ((gNB->pusch\_vars[UE\_id]->ptrs\_symbols) >> i) & 1;

available\_bits-=2\*ptrs\_symbols\*((nb\_rb+ptrs\_freq\_density-1) /ptrs\_freq\_density);

}

for (i = 0; i < available\_bits; i++) {

if(((ulsch\_ue->harq\_processes[harq\_pid]->f[i] == 0) && (gNB->pusch\_vars[UE\_id]->llr[i] <= 0))||((ulsch\_ue->harq\_processes[harq\_pid]->f[i]==1)&&(gNB->pusch\_vars[UE\_id]->llr[i] >= 0))) {

errors\_scrambling[round][snrRun]++;

}

}

round++;

}

for (i = 0; i < TBS; i++) {

estimated\_output\_bit[i] = (ulsch\_gNB->harq\_processes[harq\_pid]->b[i/8]&(1 << (i & 7))) >> (i & 7);

test\_input\_bit[i]= (ulsch\_ue->harq\_processes[harq\_pid]->b[i/8] & (1 << (i & 7))) >> (i & 7);

errors\_decoding++;

}

}

if (n\_trials == 1) {

for (int r=0;r<ulsch\_ue->harq\_processes[harq\_pid]->C;r++)

for (int i=0;i<ulsch\_ue->harq\_processes[harq\_pid]->K>>3;i++) {

if((ulsch\_ue->harq\_processes[harq\_pid]->c[r][i]^ulsch\_gNB->harq\_processes[harq\_pid]->c[r][i]) != 0) printf("\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

if (errors\_decoding > 0 && error\_flag == 0) {

n\_false\_positive++;

if (n\_trials==1)

}

roundStats[snrRun] += ((float)round);

if (!crc\_status) effRate[snrRun] += ((double)TBS)/(double)round;

}

roundStats[snrRun]/=((float)n\_trials);

effRate[snrRun] /= (double)n\_trials;

printf("SNR%f:n\_errors(%d/%d,%d/%d,%d/%d,%d/%d) (negative CRC), false\_positive %d/%d, errors\_scrambling(%u/%u,%u/%u,%u/%u,%u/%u\n",SNR,n\_errors[0][snrRun],round\_trials[0][snrRun],n\_errors[1][snrRun],round\_trials[1][snrRun],n\_errors[2][snrRun],round\_trials[2][snrRun],n\_errors[3][snrRun],round\_trials[3][snrRun],n\_false\_positive,n\_trials,errors\_scrambling[0][snrRun],available\_bits\*n\_trials,errors\_scrambling[1][snrRun],available\_bits\*n\_trials,errors\_scrambling[2][snrRun],available\_bits\*n\_trials,errors\_scrambling[3][snrRun],available\_bits\*n\_trials);

printf("\n");

blerStats[0][snrRun] = (double)n\_errors[0][snrRun]/round\_trials[0][snrRun];

blerStats[1][snrRun] = (double)n\_errors[1][snrRun]/round\_trials[1][snrRun];

blerStats[2][snrRun] = (double)n\_errors[2][snrRun]/round\_trials[2][snrRun];

blerStats[3][snrRun] = (double)n\_errors[3][snrRun]/round\_trials[3][snrRun];

berStats[0][snrRun]=(double)errors\_scrambling[0][snrRun]/available\_bits/round\_trials[0][snrRun];

berStats[1][snrRun]=(double)errors\_scrambling[1][snrRun]/available\_bits/round\_trials[1][snrRun];

berStats[2][snrRun]=(double)errors\_scrambling[2][snrRun]/available\_bits/round\_trials[2][snrRun];

berStats[3][snrRun]=(double)errors\_scrambling[3][snrRun]/available\_bits/round\_trials[3][snrRun];

effTP[snrRun] = effRate[snrRun]/(double)TBS\*(double)100;

printf("SNR %f: Channel BLER (%e,%e,%e,%e), Channel BER (%e,%e,%e,%e) Avg round %.2f, Eff Rate %.4f bits/slot, Eff Throughput %.2f, TBS %u bits/slot\n", SNR, blerStats[0][snrRun],blerStats[1][snrRun],blerStats[2][snrRun],blerStats[3][snrRun],berStats[0][snrRun],berStats[1][snrRun],berStats[2][snrRun],berStats[3][snrRun],roundStats[snrRun],effRate[snrRun],effTP[snrRun],TBS);

if (print\_perf==1) {

printDistribution(&gNB->phy\_proc\_rx,table\_rx,"Total PHY proc rx");

printStatIndent(&gNB->rx\_pusch\_stats,"RX PUSCH time");

printStatIndent2(&gNB->ulsch\_channel\_estimation\_stats,"ULSCH channel estimation time");

printStatIndent2(&gNB->ulsch\_ptrs\_processing\_stats,"ULSCH PTRS Processing time");

printStatIndent2(&gNB->ulsch\_rbs\_extraction\_stats,"ULSCH rbs extraction time");

printStatIndent2(&gNB->ulsch\_channel\_compensation\_stats,"ULSCH channel compensation time");

printStatIndent2(&gNB->ulsch\_mrc\_stats,"ULSCH mrc computation");

printStatIndent2(&gNB->ulsch\_llr\_stats,"ULSCH llr computation");

printStatIndent(&gNB->ulsch\_unscrambling\_stats,"ULSCH unscrambling");

printStatIndent(&gNB->ulsch\_decoding\_stats,"ULSCH total decoding time");

printStatIndent(&UE->ulsch\_encoding\_stats,"ULSCH total encoding time");

printStatIndent2(&UE->ulsch\_segmentation\_stats,"ULSCH segmentation time");

printStatIndent2(&UE->ulsch\_ldpc\_encoding\_stats,"ULSCH LDPC encoder time");

printStatIndent2(&UE->ulsch\_rate\_matching\_stats,"ULSCH rate-matching time");

printStatIndent2(&UE->ulsch\_interleaving\_stats,"ULSCH interleaving time");

printf("\n");

}

if ((float)effTP[snrRun] >= eff\_tp\_check) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("PUSCH test OK\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

}

snrStats[snrRun] = SNR;

snrRun++;

n\_errs = n\_errors[0][snrRun];

}

printf("\n");

char opStatsFile[50];

sprintf(opStatsFile, "ulsimStats\_z%d.m", n\_rx);

LOG\_M(opStatsFile,"SNR",snrStats,snrRun,1,7);

LOG\_MM(opStatsFile,"BLER\_round0",blerStats[0],snrRun,1,7);

LOG\_MM(opStatsFile,"BLER\_round1",blerStats[1],snrRun,1,7);

LOG\_MM(opStatsFile,"BLER\_round2",blerStats[2],snrRun,1,7);

LOG\_MM(opStatsFile,"BLER\_round3",blerStats[3],snrRun,1,7);

LOG\_MM(opStatsFile,"BER\_round0",berStats[0],snrRun,1,7);

LOG\_MM(opStatsFile,"BER\_round1",berStats[1],snrRun,1,7);

LOG\_MM(opStatsFile,"BER\_round2",berStats[2],snrRun,1,7);

LOG\_MM(opStatsFile,"BER\_round3",berStats[3],snrRun,1,7);

LOG\_MM(opStatsFile,"EffRate",effRate,snrRun,1,7);

LOG\_MM(opStatsFile,"EffTP",effTP,snrRun,1,7);

free(test\_input\_bit);

free(estimated\_output\_bit);

if (output\_fd)

fclose(output\_fd);

if (input\_fd)

fclose(input\_fd);

if (scg\_fd)

fclose(scg\_fd);

return (n\_errs);

}