**TD-LTE Small Cell基站协议栈软件**

**源程序**

目录

[1. Main.c 1](#_Toc353124923)

[2. Lowmac\_ctrl.c 8](#_Toc353124924)

[3. Harq.c 38](#_Toc353124925)

# Main.c

**#include** <getopt.h>

**#include** <time.h>

**#include** <signal.h>

**#include** "henb\_config.h"

**#include** "rrm\_sm.h"

**#include** "rrm\_sm\_out\_stack\_fn.h"

**#include** "rrm\_central\_ctrl.h"

**#include** "pdcp.h"

**#include** "mme\_conn.h"

**#include** "lowmac\_ctrl.h"

**#include** "mac\_pdu\_tx.h"

**#include** "mac\_pdu\_rx.h"

**#include** "harq.h"

**#include** "sim\_phy.h"

**#include** "rlc\_mgmt.h"

**#include** "rrc\_si\_mgmt.h"

**#include** "x2\_core.h"

**#include** "gtpu.h"

**#include** "rrc\_meas.h"

**#include** "lte\_socket.h"

MsgqType rrm\_rxmq;

MsgqType sctp\_txmq;

MsgqType gtpu\_txmq;

MsgqType pdcp\_txmq;

MsgqType si\_rxmq;

MsgqType g\_to\_lowmac\_rx\_msgq;

TaskType tnl\_rx\_task;

TaskType sctptx\_task;

TaskType gtputx\_task;

TaskType pdcp\_task;

TaskType sirx\_task;

TaskType g\_henb\_send\_task\_id;

TaskType rx\_liapi\_msg\_id;

**extern** UINT8 g\_henb\_send\_task\_flag;

**extern** UINT8 g\_si\_sched\_task\_flag;

**extern** UINT8 g\_trigger\_ho\_type;

TaskType tid\_select\_ho\_type;

TaskType henb\_send\_task;

TaskType henb\_receive\_task;

**extern** INT32 init\_ue\_config();

**extern** INT32 init\_ue\_config4ho();

//testing dl ata forward for handover

**extern** INT32 construct\_data\_func(**void**);

**#define** MSGQ\_NAME\_RRM\_RX "/lte\_henb\_rrm\_rx"

**#define** MSGQ\_NAME\_SCTP\_TX "/lte\_henb\_sctp\_tx"

**#define** MSGQ\_NAME\_GTPU\_TX "/lte\_henb\_gtpu\_tx"

**#define** MSGQ\_NAME\_PDCP\_TX "/lte\_henb\_pdcp\_tx"

**#define** MSGQ\_NAME\_LOWMAC\_RX "/lte\_henb\_lowmac\_rx"

**#define** MSGQ\_NAME\_SI\_RX "/lte\_henb\_si\_rx"

**void** sig\_handler(**int** signo)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* release msgq and task

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* rrm \*/

close\_msgq(MSGQ\_NAME\_RRM\_RX, rrm\_rxmq);

close\_msgq(MSGQ\_NAME\_PDCP\_TX, pdcp\_txmq);

delete\_task(pdcp\_task);

/\* s1x2 \*/

close\_msgq(MSGQ\_NAME\_SCTP\_TX, sctp\_txmq);

delete\_task(tnl\_rx\_task);

delete\_task(sctptx\_task);

/\* UL Data-Plane \*/

close\_msgq(MSGQ\_NAME\_LOWMAC\_RX, g\_to\_lowmac\_rx\_msgq);

delete\_task(rx\_liapi\_msg\_id);

/\* DL Data-Plane for construct dl data test \*/

**#if** 0

delete\_task(henb\_send\_task);

**#endif**

/\* socket recv task \*/

delete\_task(henb\_receive\_task);

**#ifdef** MANUAL\_TRIGGER\_HO

delete\_task(tid\_select\_ho\_type);

**#endif**

cleanup\_lowmac();

cleanup\_sim\_phy();

clean\_socket();

cleanup\_sched();

free\_henb\_config();

delete\_all\_mme\_conn();

cleanup\_core();

g\_si\_sched\_task\_flag = 0;

exit(0);

}

**#define** HENB\_INIT(statement) \

**do** { \

**if** (statement) { \

log\_msg(LOG\_ERR, RRM, #statement " failed\n"); \

perror(#statement); \

exit(-1); \

} **else** log\_msg(LOG\_SUMMARY, RRM, #statement " success\n"); \

} **while** (0)

**void** usage(**char** \*prog)

{

fprintf(stderr, "Usage: %s [options] MMEaddrs\n", prog);

fprintf(stderr, "Where options are:\n");

fprintf(stderr, "\t -H local IP address (default: ::0) \n");

fprintf(stderr, "\t -p local SCTP port (default: 12345) \n");

fprintf(stderr, "\t -f config file path (default: henb.cfg) \n");

fprintf(stderr, "\t -P MME serving port (default: 36412)\n");

fprintf(stderr, "\t -h for this help message\n");

exit(1);

}

**void** start\_sctp\_task\_mq()

{

/\* sctptxmq: send sctp\_txmq(s1 x2 sctp msg) to MME/Henb\*/

HENB\_INIT((sctp\_txmq = open\_msgq(MSGQ\_NAME\_SCTP\_TX, MSGQ\_MSG\_SIZE, **sizeof**(sctp\_mq\_item\_t))) == MSGQ\_ERROR);

HENB\_INIT((sctptx\_task = create\_task(NULL, SCHED\_RR, 1 << 10, TASK\_JOIN,

sctp\_tx\_thread\_proc, NULL)) < 0);

/\* recv s1/x2 msg from MME/HeNB,save to rrm\_rxmq \*/

HENB\_INIT((tnl\_rx\_task = create\_task(NULL, SCHED\_RR, 1 << 10, TASK\_JOIN,

tnl\_rx\_thread\_proc, NULL)) < 0);

}

**#ifdef** MANUAL\_TRIGGER\_HO

**void** t\_trigger\_ho(**void** \*arg)

{

**char** ho\_trigger = 0;

**while**(1) {

scanf("%c",&ho\_trigger);

**switch**(ho\_trigger) {

**case** 'x':

{

g\_trigger\_ho\_type = X2HO;

log\_msg(LOG\_SUMMARY,RRM,"trigger X2 handover!\n");

test\_send\_meas\_report(61);

//test\_send\_meas\_report(64);

**break**;

}

**case** 's':

{

g\_trigger\_ho\_type = S1HO;

log\_msg(LOG\_SUMMARY,RRM,"trigger S1 handover!\n");

test\_send\_meas\_report(64);

**break**;

}

**case** 'q': {

**return**;

}

**default**:

**break**;

}

}

}

**#endif**

**#ifdef** PC9608\_CL1

**int** henb\_main(**int** argc, **char** \*\*argv)

**#else**

**int** main(**int** argc, **char** \*\*argv)

**#endif**

{

INT32 opt, i = 0, end\_flag = 0;

**char** \*cfg\_path = "henb.xml";

**char** \*localport = "12345";

**char** \*localaddr = "::0";

**char** \*mmeport = S1AP\_SRV\_PORT;

UINT8 log\_dest = 0, log\_format = 0, log\_level = 0;

UINT8 log\_module[LOG\_MAX\_MODULE\_NAME] = {0};

/\* **TODO**: Setting signal handlers. \*/

**if** (signal(SIGINT, sig\_handler) == SIG\_ERR)

{

**return** -1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Initializing cmd-args, log and henb config

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Parse command-line arguments. \*/

**while** ((opt = getopt(argc, argv, "f:p:H::h:P:")) != -1) {

**switch** (opt) {

**case** 'f':

cfg\_path = optarg;

**break**;

**case** 'p':

localport = optarg;

**break**;

**case** 'H':

localaddr = optarg;

**break**;

**case** 'P':

mmeport = optarg;

**break**;

**case** 'h':

**case** '?':

**default**:

usage(argv[0]);

**break**;

}

}

**if** ((argc - optind) <= 0) {

fprintf(stderr, "Error: at least one MME address required!\n");

usage(argv[0]);

}

/\* **TODO**: Reading configuration \*/

**if** (load\_henb\_config(cfg\_path) < 0) {

fprintf(stderr, "Reading configuration failed\n");

**return** -1;

}

/\* **TODO**: Initializing log output \*/

get\_log\_info(&log\_dest, &log\_format, &log\_level,

log\_module, LOG\_MAX\_MODULE\_NAME);

**if** (log\_init(log\_dest, log\_format, log\_level)) {

fprintf(stderr, "Initializing log output failed\n");

**return** -1;

}

**for** (; i < LOG\_MAX\_MODULE\_NAME; ++i) {

**if** (1 == log\_module[i])

log\_open(i);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Initializing all management structures\RRM\PDCP

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

HENB\_INIT(init\_core() < 0);

/\* init rrm State Table and msg queue \*/

HENB\_INIT(init\_rrm\_sm());

/\* rrm\_rxmq: recv rrc msg from ue,s1 and x2 msg \*/

HENB\_INIT((rrm\_rxmq = open\_msgq(MSGQ\_NAME\_RRM\_RX, MSGQ\_MSG\_SIZE, **sizeof**(RrmRxMsgInfo))) == MSGQ\_ERROR);

/\* pdcp\_txmq: send dl rrc msg, and data plane msg \*/

HENB\_INIT((pdcp\_txmq = open\_msgq(MSGQ\_NAME\_PDCP\_TX, MSGQ\_MSG\_SIZE, **sizeof**(PdcpRxMsgInfo))) == MSGQ\_ERROR);

HENB\_INIT((pdcp\_task = create\_task(NULL, SCHED\_RR, 1 << 10, TASK\_JOIN,

pdcp\_tx\_thread\_proc, NULL)) <0);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Initializing S1 and X2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* init sctp connect for s1 and x2 modules \*/

**#ifndef** PC9608\_PLATFORM

**if**(s1\_init(localaddr, localport, &argv[optind], argc - optind, mmeport) < 0){

log\_msg(LOG\_ERR, RRM, "Fail to init S1 sctp connect:%s\n", strerror(errno));

}**else**{

log\_msg(LOG\_SUMMARY, RRM, "Init S1-ap success! \n");

}

**#if** 1

**if**(x2\_init() < 0){

log\_msg(LOG\_ERR, RRM, "Fail to init X2 sctp connect:%s\n", strerror(errno));

}**else**{

log\_msg(LOG\_SUMMARY, RRM, "Init X2-ap success! \n");

}

**#endif**

/\* sctptxmq: send sctp\_txmq(s1 x2 sctp msg) to MME/Henb\*/

HENB\_INIT((sctp\_txmq = open\_msgq(MSGQ\_NAME\_SCTP\_TX, MSGQ\_MSG\_SIZE, **sizeof**(sctp\_mq\_item\_t))) == MSGQ\_ERROR);

HENB\_INIT((sctptx\_task = create\_task(NULL, SCHED\_RR, 1 << 10, TASK\_JOIN,

sctp\_tx\_thread\_proc, NULL)) < 0);

/\* recv s1/x2 msg from MME/HeNB,save to rrm\_rxmq \*/

HENB\_INIT((tnl\_rx\_task = create\_task(NULL, SCHED\_RR, 1 << 10, TASK\_JOIN,

tnl\_rx\_thread\_proc, NULL)) < 0);

**#endif**

**#ifdef** PC9608\_CL1

/\*use PHY TTI interrupt generate sfn\*/

**#else**

/\* setup timer to generate sfn \*/

HENB\_INIT(init\_sim\_phy());

**#endif**

/\* init data plane variable, should be replaced with a single func \*/

HENB\_INIT(init\_sched());

HENB\_INIT(init\_harq());

HENB\_INIT(init\_ra());

HENB\_INIT(init\_rlc\_mod());

HENB\_INIT(init\_lowmac(get\_tdd\_config()));

HENB\_INIT(init\_socket());

**#ifdef** TEST\_DATA\_PLANE

/\* create ue entity for data plane test, which should be triggered

\* by mac-ra and ctrl-plane process \*/

//init\_ue\_config();

**#endif**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* init msg queue and thread for UL Data-Plane

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* msg queue for lowmac->mac \*/

**#ifndef** PC9608\_PLATFORM

/\* msg queue for phy->lowmac \*/

HENB\_INIT((g\_to\_lowmac\_rx\_msgq = open\_msgq(MSGQ\_NAME\_LOWMAC\_RX, 10,

**sizeof**(RecvMsg))) == MSGQ\_ERROR);

/\* recv and decode all l1api msg \*/

HENB\_INIT((rx\_liapi\_msg\_id = create\_task("tLowMacRx", SCHED\_RR, 1 << 10,TASK\_JOIN,

(FUNCPTR)rx\_liapi\_msg, NULL)) < 0);

**#endif**

usleep(1000);

**#ifdef** TEST\_GTPU

/\* gtpu\_txmq: recv msg from mac->rlc->pdcp, transmit to SGW \*/

HENB\_INIT((gtpu\_txmq = open\_msgq(MSGQ\_NAME\_GTPU\_TX, MSGQ\_MSG\_SIZE, **sizeof**(sctp\_mq\_item\_t))) == MSGQ\_ERROR);

HENB\_INIT((gtputx\_task = create\_task(NULL, SCHED\_RR, 1 << 10, TASK\_JOIN,

gtpu\_tx\_thread\_proc, NULL)) <0);

**#endif**

/\* init socket for simple PHY Tx/Rx \*/

create\_test\_socket();

**#ifndef** PC9608\_PLATFORM

HENB\_INIT((henb\_receive\_task = create\_task("Lte\_HeNB\_Receive", SCHED\_RR, 1 << 10, TASK\_JOIN,

(FUNCPTR)rx\_socket\_msg\_thread, NULL)) < 0);

**#endif**

/\* construct data for DL Data-Plane \*/

**#ifdef** X2HO\_SOURCE

HENB\_INIT((henb\_send\_task = create\_task("Lte\_HeNB\_Send", SCHED\_RR, 1 << 10, TASK\_JOIN,

(FUNCPTR)construct\_data\_func, NULL)) < 0);

**#else**

//test:do nothing

**#endif**

log\_msg(LOG\_SUMMARY, RRM, "=========lte henb init complete =========\n");

/\* when denfie PC9608\_CL1, main will be called by cl1 as

\* henb\_main to init l2 l3, and then return \*/

**#ifdef** PC9608\_CL1

**return** 0;

**#endif**

**#ifdef** MANUAL\_TRIGGER\_HO

HENB\_INIT((tid\_select\_ho\_type = create\_task(NULL, SCHED\_RR, 1 << 10, TASK\_JOIN,

(FUNCPTR)t\_trigger\_ho, NULL)) <0);

**#endif**

/\* **TODO**: Start receiving RRM messages, start main thread \*/

exec\_rrm\_central\_ctrl();

**return** 0;

}

# Lowmac\_ctrl.c

/\* Dependencies ------------------------------------------------------------- \*/

**#include** "lowmac\_ctrl.h"

**#include** "ra.h"

**#include** "sim\_phy.h"

**#include** "rrc\_si\_mgmt.h"

**#include** "lte\_socket.h"

/\* Constants ---------------------------------------------------------------- \*/

/\*frame config type\*/

UINT8 G\_CRNT\_TDD\_DLUL\_CONF = 0;

**#ifdef** TEST\_HENB\_TO\_UE\_DL\_SPS

UINT8 sps\_dci\_flag = 0;

**#endif**

/\*frame config info\*/

**const** UINT8 TDD\_SUBFRAME\_CONF[7][10] = {

{1,2,0,0,0,1,2,0,0,0},

{1,2,0,0,1,1,2,0,0,1},

{1,2,0,1,1,1,2,0,1,1},

{1,2,0,0,0,1,1,1,1,1},

{1,2,0,0,1,1,1,1,1,1},

{1,2,0,1,1,1,1,1,1,1},

{1,2,0,0,0,1,2,0,0,1}

};

/\*when mutipling if receive a ack when sub\_fn=2

\* count=0;

\* find D

\* for(i=FIRST\_ACK[config][sub\_fn]; i <FIRST\_ACK[config][sub\_fn]+10;i++){

\* if (TDD\_HARQ\_ACK[config][i%10] % 10 == sub\_fn && TDD\_SUBFRAME\_CONF[config][i%10] == 1){

\* ack\_node.sub\_fn = i % 10;

\* ack\_node.sub\_fn = (sys\_fn - (TDD\_HARQ\_ACK[config][i%10] / 10) + 1024) % 1024;

\* ack\_node.ack\_result = LI\_API.result[count++]l

\* }

\* }

\* find S

\* for(i=FIRST\_ACK[config][sub\_fn]; i <FIRST\_ACK[config][sub\_fn]+10;i++){

\* if (TDD\_HARQ\_ACK[config][i%10] % 10 == sub\_fn && TDD\_SUBFRAME\_CONF[config][i%10] == 2){

\* ack\_node.sub\_fn = i % 10;

\* ack\_node.sub\_fn = (sys\_fn - (TDD\_HARQ\_ACK[config][i%10] / 10) + 1024) % 1024;

\* ack\_node.ack\_result = LI\_API.result[count++]l

\* }

\* }

\* \*/

**const** INT8 FIRST\_ACK[7][10] = {

{-1,-1,5,-1,0,-1,-1,1,-1,5},

{-1,-1,5,9,-1,-1,-1,0,4,-1},

{-1,-1,4,-1,-1,-1,-1,9,-1,-1},

{-1,-1,2,7,9,-1,-1,-1,-1,-1},

{-1,-1,0,7,-1,-1,-1,-1,-1,-1},

{-1,-1,9,-1,-1,-1,-1,-1,-1,-1},

{-1,-1,5,6,9,-1,-1,0,1,-1}

};

**const** INT8 TDD\_HARQ\_ACK[7][10] = {

{4,7,-1,-1,-1,9,12,-1,-1,-1},

{7,7,-1,-1,8,12,12,-1,-1,13},

{7,7,-1,7,12,12,12,-1,12,17},

{4,12,-1,-1,-1,12,12,13,13,14},

{12,12,-1,-1,12,12,13,13,13,13},

{12,12,-1,12,12,12,12,12,12,22},

{7,8,-1,-1,-1,12,13,-1,-1,14}

};

**const** UINT8 TDD\_UL\_K\_PHICH\_INDEX[7][10] = {

{0,0,4,7,6,0,0,4,7,6},

{0,0,4,6,0,0,0,4,6,0},

{0,0,6,0,0,0,0,6,0,0},

{0,0,6,6,6,0,0,0,0,0},

{0,0,6,6,0,0,0,0,0,0},

{0,0,6,0,0,0,0,0,0,0},

{0,0,4,6,6,0,0,4,7,0}

};

**const** UINT8 MAX\_UL\_NUM[7] = {6,4,2,3,4,1,5};

**const** UINT8 MAX\_DL\_NUM[7] = {2,3,6,6,7,8,3};

UlHarqAck UL\_HARQ\_ACK[MAX\_UE\_NUM][MAX\_UL\_HARQ\_NUM];

UINT8 g\_ulphy\_res[UL\_BANDWIDTH + 1];

/\* array(1-110) indicates rb resources, 0 for idle rb, 1 for occupation,

\* 2 for unavailable, where 1- RAR\_PRB is for RAR,

\* (RAR\_PRB+1) - 110 is for SPS and DYNAMIC SCHED. \*/

/\* Globals ------------------------------------------------------------------ \*/

/\*for test -----------------------\*/

**typedef** **struct** L1apiMsgDestAddr\_t{

UINT8 ip\_addr[4];

UINT32 port\_start;

}L1apiMsgDestAddr\_s;

L1apiMsgDestAddr\_s g\_l1api\_msg\_dest\_addr =

{

{10,21,1,108},//wireshark service addr

8801 //port\_start

};

L1apiMsgDestAddr\_s g\_l1api\_msg\_dest\_addr2 =

{

{10,21,1,48},//wireshark service addr

8899 //RECV\_SERVER\_PORT

};

LowmacInfoDebug\_t g\_mac\_info\_statistic;

/\* FPA POOL 6 \*/

**#define** MSG\_FPA\_POOL\_BIG (6)

**#define** DL\_SUBFRAME\_OFFSET 0

**#define** UL\_SUBFRAME\_OFFSET 0

/\*for test -----------------------end\*/

/\* Types -------------------------------------------------------------------- \*/

**typedef** **struct** {

UINT16 rnti;

UINT8 cqi\_value;

UINT8 dec\_flag;

} CqiInd;

**typedef** **struct** {

NodeType ln;

SFTIMERFUNC \*func\_p;

SfTimerParam sf\_timer\_param;

INT32 expires;

}SfTimerNode;

**typedef** **struct** {

ListType harq\_list;

ListType tc\_list;

}SfTimerLst;

/\* Macros ------------------------------------------------------------------- \*/

/\* Globals ------------------------------------------------------------------ \*/

SemaType g\_mac\_tx\_sem; /\* the semaphore gived to mac\_tx schedule when dl subframe \*/

SemaType g\_si\_sched\_sem; /\* semaphore for si and paging message schedule \*/

UINT8 g\_sub\_frame\_num; /\* global sub-frame number counter 0-9 \*/

UINT16 g\_system\_frame\_num; /\* global frame number counter 0-1023 \*/

**extern** MsgqType g\_to\_lowmac\_rx\_msgq;

**static** SfTimerLst g\_sf\_timer\_lst;

/\* Functions ---------------------------------------------------------------- \*/

**extern** INT32 **rcv\_cqi**(UINT16 rnti, UINT8 cqi\_value);

///\* L1 send interface \*/

//extern void sendWqe(UINT8 msgType, UINT16 length, UINT16 sfnsf, UINT32 pool, UINT8 \*buf);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description: get tx frame number and subframe number of ACK/NACK.

\* Input: None.

\* Output:

\* sys\_fn\_p: ack tx frame number pointer.

\* sub\_fn\_p: ack tx subframe number pointer.

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **get\_ack\_tx\_fn**(UINT16 rx\_sys\_fn, UINT8 rx\_sub\_fn,

UINT16 \*sys\_fn\_p, UINT8 \*sub\_fn\_p)

{

UINT8 conf = get\_crrnt\_tdd\_conf();

**if** (sys\_fn\_p == NULL || sub\_fn\_p == NULL) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "error input func param !\n");

**return** -1;

}

**if** (TDD\_UL\_K\_PHICH\_INDEX[conf][rx\_sub\_fn]) {

\*sys\_fn\_p = (rx\_sys\_fn + (rx\_sub\_fn + TDD\_UL\_K\_PHICH\_INDEX[conf][rx\_sub\_fn]) / 10) % 1024;

\*sub\_fn\_p = (rx\_sub\_fn + TDD\_UL\_K\_PHICH\_INDEX[conf][rx\_sub\_fn]) % 10;

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Illegal UL sub-frame number!\n");

**return** -1;

}

**return** 0;

}

INT32 **send\_phichack\_to\_ue**(UINT16 sys\_fn, UINT8 sub\_fn)

{

UINT8 \*msg\_p = NULL;

UINT8 i = 0, j = 0;

UINT16 tx\_ack\_sys\_fn = 0;

UINT8 tx\_ack\_sub\_fn = 0;

ByteStream\_t bs\_out;

UINT8 ack\_num = 0;

UINT8 ack\_flag = 0;

msg\_p = (UINT8 \*)lte\_malloc(**sizeof**(PHICHACK\_indication));

**for** (i = 0; i < MAX\_UE\_NUM; i++) {

**memset**(msg\_p, 0, **sizeof**(PHICHACK\_indication));

lte\_bstream\_init(&bs\_out, msg\_p, **sizeof**(PHICHACK\_indication));

ack\_flag = 0;

ack\_num = 0;

**for** (j = 0; j < MAX\_UL\_HARQ\_NUM; j++) {

**if** (UL\_HARQ\_ACK[i][j].active == 1) {

get\_ack\_tx\_fn(UL\_HARQ\_ACK[i][j].rx\_sys\_fn, UL\_HARQ\_ACK[i][j].rx\_sub\_fn,

&tx\_ack\_sys\_fn, &tx\_ack\_sub\_fn);

**if** (tx\_ack\_sys\_fn == sys\_fn && tx\_ack\_sub\_fn == sub\_fn) {

ack\_flag = 1;

ack\_num++;

**if** (ack\_num == 1) {

lte\_bs\_put32(&bs\_out, *RX\_PHICHACK\_IND*);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, ack\_num);

lte\_bs\_put16(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, UL\_HARQ\_ACK[i][j].ackNackResult);

lte\_bs\_put8(&bs\_out, UL\_HARQ\_ACK[i][j].rx\_sub\_fn);

lte\_bs\_put16(&bs\_out, UL\_HARQ\_ACK[i][j].rx\_sys\_fn);

} **else** **if** (ack\_num == 2) {

lte\_bs\_put8(&bs\_out, UL\_HARQ\_ACK[i][j].ackNackResult);

lte\_bs\_put8(&bs\_out, UL\_HARQ\_ACK[i][j].rx\_sub\_fn);

lte\_bs\_put16(&bs\_out, UL\_HARQ\_ACK[i][j].rx\_sys\_fn);

bs\_out.offset = 40;

lte\_bs\_put8(&bs\_out, ack\_num);

} **else** {

log\_msg(*LOG\_WARNING*, *L1API*, "error ack\_num !\n");

}

}

**memset**(&(UL\_HARQ\_ACK[i][j]), 0, **sizeof**(UlHarqAck));

}

}

**if** (ack\_flag == 1) {

log\_msg(*LOG\_INFO*, *L1API*, "send ack to ue !\n");

tx\_msg\_between\_lte(0, msg\_p, **sizeof**(PHICHACK\_indication));

}

}

lte\_free(msg\_p);

**return** 0;

}

INT32 **send\_subframe\_ind**(sys\_fn, sub\_fn){

UINT8 \*msg\_p = NULL;

ByteStream\_t bs\_out;

msg\_p = (UINT8 \*)lte\_malloc(8);

lte\_bstream\_init(&bs\_out, msg\_p, 8);

lte\_bs\_put32(&bs\_out, *RX\_SUBFRAME\_IND*);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, sub\_fn);

lte\_bs\_put16(&bs\_out, sys\_fn);

tx\_msg\_between\_lte(0, msg\_p, 8);

**return** 0;

}

INT32 **send\_dci0\_to\_ue**(UINT16 sys\_fn, UINT8 sub\_fn)

{

UINT8 \*msg\_p = NULL;

ByteStream\_t bs\_out;

INT32 rnti\_type = 0;

UINT16 recv\_sys\_fn = 0;

UINT8 recv\_sub\_fn = 0;

UINT8 i = 0;

UlHarqUeInfo \*ulharq\_p = NULL;

UlHarqProcInfo \*ulproc\_p = NULL;

DCIUlschInfo \*dci\_ulsch\_p = NULL;

**for** (i = 0; i < MAX\_UE\_NUM; i++) {

ulharq\_p = get\_ul\_harq(MIN\_C\_RNTI + i);

**if** (NULL == ulharq\_p || 1 != ulharq\_p->flag) {

log\_msg(*LOG\_INFO*, *MAC\_ULSCHED*, "get ul harq fail !\n");

**continue**;

}

get\_ul\_sched\_fn(sys\_fn, sub\_fn, &recv\_sys\_fn, &recv\_sub\_fn);

ulproc\_p = get\_ul\_harq\_proc(MIN\_C\_RNTI + i, recv\_sys\_fn, recv\_sub\_fn);

**if** (!ulproc\_p) {

log\_msg(*LOG\_INFO*, *MAC\_ULSCHED*, "get ulproc fail !\n");

**continue**;

}

**if** (1 == ulproc\_p->active) {

dci\_ulsch\_p = &(ulproc\_p->dci\_ulsch);

rnti\_type = get\_rnti\_type(dci\_ulsch\_p->rnti);

msg\_p = (UINT8 \*)lte\_malloc(**sizeof**(PDCCH\_indication));

**memset**(msg\_p, 0, **sizeof**(PDCCH\_indication));

lte\_bstream\_init(&bs\_out, msg\_p, **sizeof**(PDCCH\_indication));

lte\_bs\_put32(&bs\_out, *RX\_PDCCH\_IND*);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, sub\_fn);

lte\_bs\_put16(&bs\_out, sys\_fn);

**if** (rnti\_type == *SEMI\_C\_RNTI\_TYPE* && dci\_ulsch\_p->rb\_start == 0xff)

/\* sps explicit release \*/

lte\_bs\_put8(&bs\_out, 1);

**else**

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0x02);

lte\_bs\_put16(&bs\_out, 0);

/\* dci1A \*/

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put16(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put16(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put16(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

/\* dci0 \*/

**for** (i = 0; i < 2; i++) {

lte\_bs\_put8(&bs\_out, dci\_ulsch\_p->ndi);

lte\_bs\_put8(&bs\_out, 0);

// printf("size=%d, rb\_len=%d\n", result\_node\_p->size, result\_node\_p->rb\_len);

lte\_bs\_put16(&bs\_out, ulproc\_p->tb\_size);

lte\_bs\_put8(&bs\_out, rnti\_type);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put16(&bs\_out, dci\_ulsch\_p->rnti);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, dci\_ulsch\_p->rb\_start);

lte\_bs\_put8(&bs\_out, dci\_ulsch\_p->rb\_num);

**if** (rnti\_type == *SEMI\_C\_RNTI\_TYPE* && dci\_ulsch\_p->rb\_start == 0xff)

lte\_bs\_put8(&bs\_out, 0);

**else**

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put32(&bs\_out, 0);

**if** (rnti\_type == *SEMI\_C\_RNTI\_TYPE* && dci\_ulsch\_p->rb\_start == 0xff)

lte\_bs\_put8(&bs\_out, 31);

**else**

lte\_bs\_put8(&bs\_out, dci\_ulsch\_p->mcs);

lte\_bs\_put8(&bs\_out, dci\_ulsch\_p->rv);

lte\_bs\_put16(&bs\_out, 0);

} /\* for (i = 0; i < 2; i++) \*/

log\_msg(*LOG\_INFO*, *L1API*, "send dci0 to ue !\n");

show\_memory(msg\_p, **sizeof**(PDCCH\_indication), "data of dci0 send to ue !\n");

tx\_msg\_between\_lte(0, msg\_p, **sizeof**(PDCCH\_indication));

lte\_free(msg\_p);

ulproc\_p->active = 0;

}/\* if (1 == ulproc\_p->active) \*/

}

**return** 0;

}

INT32 **send\_ue\_dlmsg**(UINT16 sys\_fn, UINT8 sub\_fn, LowmacTxPduList \*lowmac\_pdu\_lst\_p){

UINT16 dlsch\_pdu\_len = 0;

PDSCHTB\_indication dl\_pdu\_ind;

PBCHTB\_indication pbch\_pdu\_ind;

ByteStream\_t bs\_out;

UINT8 \*msg\_p = NULL;

UINT32 msg\_len = 0;

INT32 rnti = 0;

UINT32 rnti\_type = 0;

LowmacTxPduNode \*lowmac\_pdu\_p = NULL;

DlHarqUeInfo \*dlharq\_p = NULL;

DlHarqProcInfo \*dlproc\_p = NULL;

lowmac\_pdu\_p = (LowmacTxPduNode \*)get\_list((ListType \*)lowmac\_pdu\_lst\_p);

**while** (lowmac\_pdu\_p) {

**if** (*L1API\_DLSCH\_PDU* == lowmac\_pdu\_p->pdu\_type || *L1API\_PCH\_PDU* == lowmac\_pdu\_p->pdu\_type) {

**memset**(&dl\_pdu\_ind, 0, **sizeof**(PDSCHTB\_indication));

dl\_pdu\_ind.msg\_type = *RX\_PDSCHTB\_IND*;

dlsch\_pdu\_len = lowmac\_pdu\_p->pdu\_len;

dl\_pdu\_ind.err\_code[0] = 0;

dl\_pdu\_ind.decode\_flag[0] = 0;

dl\_pdu\_ind.rx\_subframe\_num = sub\_fn;

dl\_pdu\_ind.rx\_frame\_num = sys\_fn;

dl\_pdu\_ind.rnti\_type = get\_rnti\_type(lowmac\_pdu\_p->rnti);

dl\_pdu\_ind.rnti\_value = lowmac\_pdu\_p->rnti;

dl\_pdu\_ind.message\_length[0] = dlsch\_pdu\_len;

dl\_pdu\_ind.message\_body\_addr[0] = lowmac\_pdu\_p->pdu\_p;

dlsch\_pdu\_len = 0;

msg\_len = ALIGN\_32(**sizeof**(PDSCHTB\_indication) + dl\_pdu\_ind.message\_length[0]);

msg\_p = (UINT8 \*)lte\_malloc(msg\_len);

**memset**(msg\_p, 0, msg\_len);

lte\_bstream\_init(&bs\_out, msg\_p, msg\_len);

dl\_pdu\_ind.decode\_flag[0] = 0;

dl\_pdu\_ind.decode\_flag[1] = 0;

lte\_bs\_put32(&bs\_out, dl\_pdu\_ind.msg\_type);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.err\_code[0]);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.err\_code[1]);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.decode\_flag[0]);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.decode\_flag[1]);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.rx\_subframe\_num);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put16(&bs\_out, dl\_pdu\_ind.rx\_frame\_num);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.Ph);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.PL);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.rnti\_type);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put16(&bs\_out, dl\_pdu\_ind.rnti\_value);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.tb\_num);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put32(&bs\_out, dl\_pdu\_ind.CQI\_value);

lte\_bs\_put16(&bs\_out, dl\_pdu\_ind.message\_length[0]);

lte\_bs\_put16(&bs\_out, dl\_pdu\_ind.message\_length[1]);

lte\_bs\_put32(&bs\_out, 0);

lte\_bs\_put32(&bs\_out, 0);

lte\_bs\_putraw(&bs\_out, lowmac\_pdu\_p->pdu\_p, dl\_pdu\_ind.message\_length[0]);

//lte\_bs\_putraw(&bs\_out, lowmac\_pdu\_p->pdu\_p, dl\_pdu\_ind.message\_length[1]);

//tx\_msg\_to\_enodeb(msg\_p,msg\_len, 61);

tx\_msg\_between\_lte(0, msg\_p, msg\_len);

lte\_free(lowmac\_pdu\_p->pdu\_p);

lte\_free(msg\_p);

} **else** **if** (*L1API\_BCH\_PDU* == lowmac\_pdu\_p->pdu\_type) {

**memset**(&pbch\_pdu\_ind, 0, **sizeof**(PBCHTB\_indication));

pbch\_pdu\_ind.msg\_type = *RX\_PBCHTB\_IND*;

pbch\_pdu\_ind.rx\_frame\_num = sys\_fn;

pbch\_pdu\_ind.rx\_subframe\_num = sub\_fn;

msg\_len = ALIGN\_32(**sizeof**(PBCHTB\_indication));

msg\_p = (UINT8 \*)lte\_malloc(msg\_len);

**memset**(msg\_p, 0, msg\_len);

lte\_bstream\_init(&bs\_out, msg\_p, msg\_len);

lte\_bs\_put32(&bs\_out, pbch\_pdu\_ind.msg\_type);

lte\_bs\_put8(&bs\_out, pbch\_pdu\_ind.err\_code);

lte\_bs\_put8(&bs\_out, pbch\_pdu\_ind.rx\_subframe\_num);

lte\_bs\_put16(&bs\_out, pbch\_pdu\_ind.rx\_frame\_num);

lte\_bs\_put8(&bs\_out, pbch\_pdu\_ind.dl\_bw);

lte\_bs\_put8(&bs\_out, pbch\_pdu\_ind.reserved);

lte\_bs\_put16(&bs\_out, pbch\_pdu\_ind.physCelld);

// log\_msg(LOG\_WARNING, MAC\_TX ,"send msg len=%d, msg\_type:%d\n",

// msg\_len, \*(UINT32 \*)msg\_p);

// log\_msg(LOG\_INFO, L1API, "tx dl msg to ue !\n");

// tx\_msg\_to\_enodeb(msg\_p,msg\_len, 61);

tx\_msg\_between\_lte(0, msg\_p, msg\_len);

lte\_free(msg\_p);

}

lte\_free(lowmac\_pdu\_p);

lowmac\_pdu\_p = (LowmacTxPduNode \*)get\_list((ListType \*)lowmac\_pdu\_lst\_p);

}

**for** (rnti = MIN\_C\_RNTI; rnti < MIN\_C\_RNTI + MAX\_UE\_NUM; rnti++) {

dlharq\_p = get\_dl\_harq(rnti);

**if** (NULL != dlharq\_p && 1 == dlharq\_p->flag) {

**if** (dlharq\_p->data\_type == *SPS\_DATA*) {

dlproc\_p = get\_dl\_harq\_proc(rnti, sys\_fn, sub\_fn, *SPS\_DATA*);

} **else** **if** (dlharq\_p->data\_type == *CRNTI\_DATA*) {

dlproc\_p = get\_dl\_harq\_proc(rnti, sys\_fn, sub\_fn, *CRNTI\_DATA*);

} **else** {

log\_msg(*LOG\_ERR*, *L1API*, "No dl harq process, data type=%d !\n", dlharq\_p->data\_type);

**continue**;

}

} **else** {

log\_msg(*LOG\_INFO*, *L1API*, "No dl harq, rnti = %d !\n", rnti);

**continue**;

}

**if** (NULL == dlproc\_p || 0 == dlproc\_p->active) {

log\_msg(*LOG\_INFO*, *L1API*, "No dl harq process, rnti=%d, sys\_fn=%d, sub\_fn=%d !\n", rnti, sys\_fn, sub\_fn);

**continue**;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* send dl dci

\*/

**#if** 1

DciInfo \*dci\_info\_p = NULL;

/\* send dl dci to ue \*/

dci\_info\_p = &(dlproc\_p->dci\_info);

show\_dl\_dci\_info(dci\_info\_p);

msg\_len = **sizeof**(PDCCH\_indication);

msg\_p = (UINT8 \*)lte\_malloc(msg\_len);

**memset**(msg\_p, 0, msg\_len);

lte\_bstream\_init(&bs\_out, msg\_p, msg\_len);

lte\_bs\_put32(&bs\_out, *RX\_PDCCH\_IND*);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, sub\_fn);

lte\_bs\_put16(&bs\_out, sys\_fn);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 1);

lte\_bs\_put16(&bs\_out, 0);

/\* Dci1A \*/

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put16(&bs\_out, 0);

lte\_bs\_put8(&bs\_out, 0); /\* ra\_flag \*/

rnti\_type = get\_rnti\_type(dlproc\_p->rnti\_value);

lte\_bs\_put8(&bs\_out, rnti\_type);

lte\_bs\_put16(&bs\_out, dlproc\_p->rnti\_value);

lte\_bs\_put8(&bs\_out, dci\_info\_p->rb\_start);

lte\_bs\_put8(&bs\_out, dci\_info\_p->rb\_len);

lte\_bs\_put16(&bs\_out, 0);

lte\_bs\_put8(&bs\_out,dci\_info\_p->mcs\_index);

lte\_bs\_put8(&bs\_out, dci\_info\_p->ndi);

lte\_bs\_put8(&bs\_out, dci\_info\_p->rv);

lte\_bs\_put8(&bs\_out, dlproc\_p->harq\_proc\_id);

**if** (rnti\_type == *C\_RNTI\_TYPE*) {

log\_msg(*LOG\_INFO*, *L1API*, "tx dl dci to ue, rnti\_type = %d, proc\_id = %d !\n",

rnti\_type, dlproc\_p->harq\_proc\_id);

show\_memory(msg\_p, msg\_len, "tx dl dci to ue:");

tx\_msg\_between\_lte(0, msg\_p, msg\_len);

}

**#ifdef** TEST\_HENB\_TO\_UE\_DL\_SPS

**if** (rnti\_type == SEMI\_C\_RNTI\_TYPE && sps\_dci\_flag == 1) {

log\_msg(LOG\_INFO, L1API, "tx dl dci to ue !\n");

show\_memory(msg\_p, msg\_len, "tx dl dci to ue:");

tx\_msg\_between\_lte(0, msg\_p, msg\_len);

sps\_dci\_flag = 0;

}

**#endif**

lte\_free(msg\_p);

**#endif**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* send dl msg

\*/

**if** (NULL == dlproc\_p->pdu\_p) {

log\_msg(*LOG\_INFO*, *L1API*, "No dl mac pdu, rnti=%d, sys\_fn=%d, sub\_fn=%d !\n", rnti, sys\_fn, sub\_fn);

**continue**;

}

**memset**(&dl\_pdu\_ind, 0, **sizeof**(PDSCHTB\_indication));

dl\_pdu\_ind.msg\_type = *RX\_PDSCHTB\_IND*;

dlsch\_pdu\_len = dlproc\_p->tb\_size;

dl\_pdu\_ind.err\_code[0] = 0;

dl\_pdu\_ind.decode\_flag[0] = 0;

dl\_pdu\_ind.rx\_subframe\_num = sub\_fn;

dl\_pdu\_ind.rx\_frame\_num = sys\_fn;

dl\_pdu\_ind.rnti\_type = get\_rnti\_type(dlproc\_p->rnti\_value);

dl\_pdu\_ind.rnti\_value = dlproc\_p->rnti\_value;

dl\_pdu\_ind.message\_length[0] = dlsch\_pdu\_len;

dl\_pdu\_ind.message\_body\_addr[0] = dlproc\_p->pdu\_p;

dlsch\_pdu\_len = 0;

msg\_len = ALIGN\_32(**sizeof**(PDSCHTB\_indication) + dl\_pdu\_ind.message\_length[0]);

msg\_p = (UINT8 \*)lte\_malloc(msg\_len);

**memset**(msg\_p, 0, msg\_len);

lte\_bstream\_init(&bs\_out, msg\_p, msg\_len);

dl\_pdu\_ind.decode\_flag[0] = 0;

dl\_pdu\_ind.decode\_flag[1] = 0;

lte\_bs\_put32(&bs\_out, dl\_pdu\_ind.msg\_type);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.err\_code[0]);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.err\_code[1]);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.decode\_flag[0]);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.decode\_flag[1]);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.rx\_subframe\_num);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put16(&bs\_out, dl\_pdu\_ind.rx\_frame\_num);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.Ph);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.PL);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.rnti\_type);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put16(&bs\_out, dl\_pdu\_ind.rnti\_value);

lte\_bs\_put8(&bs\_out, dl\_pdu\_ind.tb\_num);

lte\_bs\_put8(&bs\_out, 0);

lte\_bs\_put32(&bs\_out, dl\_pdu\_ind.CQI\_value);

lte\_bs\_put16(&bs\_out, dl\_pdu\_ind.message\_length[0]);

lte\_bs\_put16(&bs\_out, dl\_pdu\_ind.message\_length[1]);

lte\_bs\_put32(&bs\_out, 0);

lte\_bs\_put32(&bs\_out, 0);

lte\_bs\_putraw(&bs\_out, dlproc\_p->pdu\_p, dl\_pdu\_ind.message\_length[0]);

//lte\_bs\_putraw(&bs\_out, lowmac\_pdu\_p->pdu\_p, dl\_pdu\_ind.message\_length[1]);

//tx\_msg\_to\_enodeb(msg\_p,msg\_len, 61);

tx\_msg\_between\_lte(0, msg\_p, msg\_len);

lte\_free(msg\_p);

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* create subframe timer.

\*

\* Input:

\* func\_p: time expire function pointer.

\* arg: arguments of function.

\* expires: expire time.

\*

\* Output:

\* none.

\*

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **create\_sf\_timer**(SFTIMERFUNC \*func\_p, SfTimerParam \*timer\_param\_p, INT32 expires,

UINT32 schedul\_time)

{

SfTimerNode \*sf\_timer\_p = NULL;

INT32 ex\_time = 0;

**if** (NULL == func\_p || NULL == timer\_param\_p) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "error input param!\n");

**return** -1;

}

/\*calculate the expire time\*/

ex\_time = (expires + schedul\_time) % MAX\_SYS\_TIME;

/\*calculate the expire time\*/

sf\_timer\_p = (SfTimerNode \*)lte\_malloc(**sizeof**(SfTimerNode));

**if** (!sf\_timer\_p) {

**return** -1;

}

**memset**(sf\_timer\_p, 0, **sizeof**(SfTimerNode));

sf\_timer\_p->func\_p = func\_p;

sf\_timer\_p->expires = ex\_time;

sf\_timer\_p->sf\_timer\_param.param\_value = timer\_param\_p->param\_value;

sf\_timer\_p->sf\_timer\_param.param\_p = timer\_param\_p->param\_p;

sf\_timer\_p->sf\_timer\_param.timer\_type = timer\_param\_p->timer\_type;

/\*add expire timer node to list\*/

**if** (*HARQ\_TIMER* == timer\_param\_p->timer\_type) {

add\_list((ListType \*)&(g\_sf\_timer\_lst.harq\_list), (NodeType \*)sf\_timer\_p);

} **else** **if** (*TC\_TIMER* == timer\_param\_p->timer\_type) {

add\_list((ListType \*)&(g\_sf\_timer\_lst.tc\_list), (NodeType \*)sf\_timer\_p);

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "Parameter type error,type=%d!\n", timer\_param\_p->timer\_type);

**return** -1;

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* delete subframe timer.

\*

\* Input:

\* expire\_time

\*

\* Output:

\* none.

\*

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **delete\_sf\_timer**(SfTimerParam \*timer\_param\_p)

{

SfTimerNode \*sf\_timer\_p = NULL, \*temp\_timer\_p = NULL;

INT32 timer\_num = 0;

**if** (NULL == timer\_param\_p) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "error input param!\n");

**return** -1;

}

**if** (*HARQ\_TIMER* == timer\_param\_p->timer\_type) {

sf\_timer\_p = (SfTimerNode \*)first\_list((ListType \*)&(g\_sf\_timer\_lst.harq\_list));

**while** (NULL != sf\_timer\_p) {

**if** (sf\_timer\_p->expires == timer\_param\_p->param\_value) {

temp\_timer\_p = sf\_timer\_p;

sf\_timer\_p = (SfTimerNode \*)next\_list((NodeType \*)sf\_timer\_p);

delete\_list((ListType \*)&(g\_sf\_timer\_lst.harq\_list), (NodeType \*)temp\_timer\_p);

lte\_free(temp\_timer\_p);

timer\_num++;

}**else** {

sf\_timer\_p = (SfTimerNode \*)next\_list((NodeType \*)sf\_timer\_p);

}

}

} **else** **if** (*TC\_TIMER* == timer\_param\_p->timer\_type){

sf\_timer\_p = (SfTimerNode \*)first\_list((ListType \*)&(g\_sf\_timer\_lst.tc\_list));

**while** (NULL != sf\_timer\_p) {

**if** (sf\_timer\_p->sf\_timer\_param.param\_value == timer\_param\_p->param\_value) {

temp\_timer\_p = sf\_timer\_p;

sf\_timer\_p = (SfTimerNode \*)next\_list((NodeType \*)sf\_timer\_p);

delete\_list((ListType \*)&(g\_sf\_timer\_lst.tc\_list), (NodeType \*)temp\_timer\_p);

lte\_free(temp\_timer\_p);

timer\_num++;

} **else** {

sf\_timer\_p = (SfTimerNode \*)next\_list((NodeType \*)sf\_timer\_p);

}

}

}

**if** (timer\_num == 0) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "Delete the expire error:no this node!");

**return** -1;

} **else** {

log\_msg(*LOG\_SUMMARY*, *MAC\_TX*, "Delete the expire function is %d!", timer\_num);

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* get current frame config.

\*

\* Input:

\* none.

\*

\* Output:

\* current frame config

\*

\* Return:

\* current frame config.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UINT8 **get\_crrnt\_tdd\_conf**(**void**)

{

**return** G\_CRNT\_TDD\_DLUL\_CONF;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* get current sfn\_sf field by system frame number and subframe number.

\*

\* Input:

\* sys\_fn\_p: pointer of system frame number.

\* sub\_fn\_p: pointer of subframe number.

\* Output:

\* sfn\_sf\_p: 2 bytes, 15:4 SFN , 3:0 SF.

\* Return:

\* 0.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **set\_sfn\_sf**(UINT16 sys\_fn, UINT8 sub\_fn, UINT16 \*sfn\_sf\_p)

{

\*sfn\_sf\_p = ((sys\_fn << 4) & 0XFFF0) + sub\_fn;

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* get current system frame number and subframe number.

\*

\* Input:

\* sfn\_sf: .

\* Output:

\* sys\_fn\_p: pointer of system frame number.

\* sub\_fn\_p: pointer of subframe number.

\*

\* Return:

\* 0.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **get\_sysfn\_subfn**(UINT16 sfn\_sf, UINT16 \*sys\_fn\_p, UINT8 \*sub\_fn\_p)

{

\*sub\_fn\_p = sfn\_sf & 0X000F;

\*sys\_fn\_p = (sfn\_sf >> 4) & 0X0FFF;

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* set current system frame and frame nubmer.

\* Input: L1ApiMsgSubFrameInd\_t subframe\_ind\_p.

\* Output: None.

\* Return: 0.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **deal\_subframe\_ind**(L1ApiMsgSubFrameInd\_t subframe\_ind)

{

get\_sysfn\_subfn(subframe\_ind.sfn\_sf, &g\_system\_frame\_num, &g\_sub\_frame\_num);

lowmac\_control();

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* process of receiving PRACH msg .

\*

\* Input:

\* msgq\_info\_p:msg address.

\*

\* Output:

\* none.

\*

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **deal\_rach\_ind**(L1ApiMsgRachInd\_t \*rach\_ind\_p)

{

**if** (!rach\_ind\_p) {

log\_msg(*LOG\_ERR*, *MAC\_RA*, "error input param!\n");

**return** -1;

}

**if**(rcv\_ra\_preamble(&(rach\_ind\_p->preamble\_list))) {

log\_msg(*LOG\_ERR*, *MAC\_RA*, "process prach preamble fail\n");

**return** -1;

}

// l1api\_print\_rach\_ind(rach\_ind\_p);

free\_list((ListType \*)&(rach\_ind\_p->preamble\_list));

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* process of receiving ACK/NACK msg .

\*

\* Input:

\* msgq\_info\_p:msg address.

\*

\* Output:

\* none.

\*

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **deal\_harq\_ind**(L1ApiMsgHarqInd\_t \*harq\_ind\_p)

{

// DlHarqAckList ack\_list;

DlHarqAckNode ack\_node;

L1ApiMsgHarqIndNode\_t \*harq\_ind\_node\_p = NULL;

UINT16 sys\_fn = 0;

UINT8 sub\_fn = 0;

INT32 i =0, j =0, k = 0;

AckType ack\_type = 0;

INT32 count = 0;

get\_sysfn\_subfn(harq\_ind\_p->sfn\_sf, &sys\_fn, &sub\_fn);

// /\*create ACK list for HARQ\*/

// init\_list((ListType \*)&ack\_list);

/\*create ACK list nodes for HARQ\*/

harq\_ind\_node\_p = (L1ApiMsgHarqIndNode\_t \*)first\_list(&(harq\_ind\_p->harq\_ind\_list));

/\* because somes fields are unused, free is in decode.c \*/

**for** (i = 0; i < harq\_ind\_p->harqs\_num; i++)

{

**switch** (harq\_ind\_node\_p->ack\_mode)

{

**case** *L1API\_MSG\_BUNDLING*:

**for** (j = 0; j < 1; j++)

{

**if** (harq\_ind\_node\_p->harq\_data[j] == 1 ||

harq\_ind\_node\_p->harq\_data[j] == 5)

ack\_type = *ACK*;

**else** **if** (harq\_ind\_node\_p->harq\_data[j] == 2 ||

harq\_ind\_node\_p->harq\_data[j] == 6)

ack\_type = *NACK*;

**for**(k = FIRST\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][sub\_fn];

k < FIRST\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][sub\_fn] + 10; k++) {

**if** (TDD\_HARQ\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][k%10] % 10 == sub\_fn) {

**memset**(&ack\_node, 0, **sizeof**(DlHarqAckNode));

ack\_node.rnti = harq\_ind\_node\_p->rnti;

ack\_node.acktype = ack\_type;

ack\_node.sys\_fn= (sys\_fn - TDD\_HARQ\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][k%10] / 10 + 1024) % 1024;

ack\_node.sub\_fn= k % 10;

process\_rcv\_ack(&ack\_node);

}

}

// add\_list((ListType \*)&ack\_list, (NodeType \*)ack\_node\_p);

}

**break**;

**case** *L1API\_MSG\_MULTIPLEXING*:

**#ifdef** TEST\_HENB\_TO\_UE

memset(&ack\_node, 0, **sizeof**(DlHarqAckNode));

ack\_node.rnti = harq\_ind\_node\_p->rnti;

**if** (harq\_ind\_node\_p->harq\_data[j] == 1 ||

harq\_ind\_node\_p->harq\_data[j] == 5)

ack\_node.acktype = ACK;

**else** **if** (harq\_ind\_node\_p->harq\_data[j] == 2 ||

harq\_ind\_node\_p->harq\_data[j] == 6)

ack\_node.acktype = NACK;

ack\_node.sys\_fn= sys\_fn;

ack\_node.sub\_fn= sub\_fn;

process\_rcv\_ack(&ack\_node);

**#else**

**for**(k = FIRST\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][sub\_fn];

k < FIRST\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][sub\_fn] + 10; k++) {

**if** (TDD\_HARQ\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][k%10] % 10 == sub\_fn &&

TDD\_SUBFRAME\_CONF[G\_CRNT\_TDD\_DLUL\_CONF][k%10] == 1) {

**memset**(&ack\_node, 0, **sizeof**(DlHarqAckNode));

ack\_node.rnti = harq\_ind\_node\_p->rnti;

**if**(count >= 4) {

log\_msg(*LOG\_ERR*, *MAC\_RX*, "deal harq error!!");

**return** -1;

}

ack\_node.acktype = harq\_ind\_node\_p->harq\_data[count++];

ack\_node.sys\_fn= (sys\_fn - TDD\_HARQ\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][k%10] / 10 + 1024) % 1024;

ack\_node.sub\_fn= k % 10;

process\_rcv\_ack(&ack\_node);

}

}

**for**(k = FIRST\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][sub\_fn];

k < FIRST\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][sub\_fn] + 10; k++) {

**if** (TDD\_HARQ\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][k%10] % 10 == sub\_fn &&

TDD\_SUBFRAME\_CONF[G\_CRNT\_TDD\_DLUL\_CONF][k%10] == 2) {

**memset**(&ack\_node, 0, **sizeof**(DlHarqAckNode));

ack\_node.rnti = harq\_ind\_node\_p->rnti;

**if**(count >= 4) {

log\_msg(*LOG\_ERR*, *MAC\_RX*, "deal harq error!!");

**return** -1;

}

ack\_node.acktype = harq\_ind\_node\_p->harq\_data[count++];

ack\_node.sys\_fn= (sys\_fn - TDD\_HARQ\_ACK[G\_CRNT\_TDD\_DLUL\_CONF][k%10] / 10 + 1024) % 1024;

ack\_node.sub\_fn= k % 10;

process\_rcv\_ack(&ack\_node);

}

}

**#endif**

**break**;

**case** *L1API\_MSG\_SPECIAL\_BUNDLING*:

**break**;

**default**:

**break**;

}/\*end of switch\*/

harq\_ind\_node\_p = (L1ApiMsgHarqIndNode\_t \*)next\_list((NodeType \*)harq\_ind\_node\_p);

}

harq\_ind\_node\_p = NULL;

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* process of receiving CQI msg .

\*

\* Input:

\* msgq\_info\_p:msg address.

\*

\* Output:

\* none.

\*

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **deal\_cqi\_ind**(UINT8 \*msgq\_info\_p)

{

CqiInd \*cqi\_ind\_p = NULL;

**if** (!msgq\_info\_p ) {

log\_msg(*LOG\_ERR*, *MAC\_RX*, "error input param!\n");

**return** -1;

}

/\*judge the msg whether is decoded successfully or not\*/

cqi\_ind\_p = (CqiInd \*)msgq\_info\_p;

**if** (cqi\_ind\_p->dec\_flag) {

log\_msg(*LOG\_INFO*,*MAC\_RX*,"CQI msg decode fail!\n");

**return** 0;

}

/\*applying interface to deal with CQI msg\*/

**if**(rcv\_cqi(cqi\_ind\_p->rnti, cqi\_ind\_p->cqi\_value)) {

log\_msg(*LOG\_ERR*,*MAC\_RX*,"process rcv cqi fail\n");

**return** -1;

}

**return** 0;

}

/\*for test -----------------------\*/

INT32 **l1api\_msg\_cc2socket2**(UINT32 \* msg\_p, UINT32 len)

{

**static** **struct** sockaddr\_in servaddr;

**static** INT32 sockfd = -1;

INT32 nbytes = 0;

INT8 buf[10000] = {0};

**memcpy**(buf,msg\_p,len);

**if**(sockfd <0) {

sockfd = **socket**(AF\_INET,SOCK\_DGRAM,0);

bzero(&servaddr,**sizeof**(servaddr));

servaddr.sin\_family = AF\_INET;

**memcpy**(&servaddr.sin\_addr,g\_l1api\_msg\_dest\_addr2.ip\_addr,4);

servaddr.sin\_port = **htons**(g\_l1api\_msg\_dest\_addr2.port\_start);

**if**(sockfd >0 ) {

log\_msg(*LOG\_SUMMARY*, *L1API*, "l1api msg test socket init ok !\n");

}

}

nbytes = **sendto**(sockfd,buf,len,0,(**struct** sockaddr\*)&servaddr,**sizeof**(servaddr));

**if**(nbytes < 0)

{

log\_msg(*LOG\_ERR*, *L1API*, "Carbon Copy l1api msg failed:%s\n", **strerror**(errno));

**return** -1;

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* check if mac dl subframe.

\*

\* Input:

\* sub\_fn: .

\*

\* Output:

\* none.

\*

\* Return: 1 : dl subframe.

\* other: not dl subframe.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **check\_mac\_dl**(UINT8 sub\_fn)

{

UINT8 tdd\_conf = get\_crrnt\_tdd\_conf();

**if**(TDD\_SUBFRAME\_CONF[tdd\_conf][sub\_fn] == 1){

**return** 1;

}**else**{

**return** 0;

}

}

INT32 **show\_crnti\_ue\_lst**()

{

UeRntiMsg \*node\_p = NULL;

CRntiUeLst\* ue\_list\_p = &g\_c\_rnti\_ue\_lst;

// typedef struct {

// ResourcePhyInfo phy\_info;

// UINT32 tb\_size; /\* the TB size received in this process. \*/

// UINT32 current\_tx\_num; /\* The times of transmission of one TB. \*/

// SchedTranType sched\_tran\_type;/\* the indication of the sched an tran type.\*/

// // SchedType sched\_type;

// UINT8 pre\_ndi; /\* previous ndi \*/

// // UINT16 reserved;

// } UlProcInfo;

node\_p = (UeRntiMsg \*)first\_list((ListType \*)ue\_list\_p);

**printf**("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

**printf**("\*\*\*\*\*\*\* show c-rnti ue lst \*\*\*\*\*\*\*\*\*\*\*\n");

**while** (node\_p) {

**printf**("rnti type of ue is %d\n", get\_rnti\_type(node\_p->rnti\_type));

**printf**("rnti value of ue is %d\n", node\_p->rnti\_value);

node\_p = (UeRntiMsg \*)next\_list((NodeType \*)node\_p);

}

**printf**("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

**return** MAC\_ALL\_RIGHT;

}

INT32 **show\_ul\_sched\_result**(UINT16 sys\_fn, UINT8 sub\_fn)

{

UINT8 i = 0;

UlHarqUeInfo \*ulharq\_p = NULL;

UlHarqProcInfo \*ulproc\_p = NULL;

DCIUlschInfo \*dci\_ulsch\_p = NULL;

**printf**("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

**printf**("\*\*\*\*\*\*\*show ul sched result \*\*\*\*\*\*\*\*\*\*\*\*\*\n");

**for** (i = 0; i < MAX\_UE\_NUM; i++) {

ulharq\_p = get\_ul\_harq(MIN\_C\_RNTI + i);

**if** (NULL == ulharq\_p || 1 != ulharq\_p->flag) {

log\_msg(*LOG\_INFO*, *MAC\_ULSCHED*, "get ul harq fail !\n");

**continue**;

}

ulproc\_p = get\_ul\_harq\_proc(MIN\_C\_RNTI + i, sys\_fn, sub\_fn);

**if** (!ulproc\_p) {

log\_msg(*LOG\_INFO*, *MAC\_ULSCHED*, "get ulproc fail !\n");

**continue**;

}

log\_msg(*LOG\_INFO*, *MAC\_ULSCHED*, "ulproc\_p->active = %d !\n", ulproc\_p->active );

**if** (1 == ulproc\_p->active) {

dci\_ulsch\_p = &(ulproc\_p->dci\_ulsch);

**printf**("size: %d\n", ulproc\_p->tb\_size);

**printf**("rb\_start: %d\n", dci\_ulsch\_p->rb\_start);

**printf**("rb\_len: %d\n", dci\_ulsch\_p->rb\_num);

**printf**("rnti: %d\n", dci\_ulsch\_p->rnti);

**printf**("mcs: %d\n", dci\_ulsch\_p->mcs);

**printf**("ndi: %d\n", dci\_ulsch\_p->ndi);

**printf**("rv: %d\n", dci\_ulsch\_p->rv);

**printf**("cce\_index: %d\n", dci\_ulsch\_p->cce\_index);

**printf**("aggregation\_level: %d\n", dci\_ulsch\_p->aggregation\_level);

**printf**("cyclic\_shift2\_for\_dmrs: %d\n", dci\_ulsch\_p->cyclic\_shift2\_for\_dmrs);

**printf**("freq\_hopping\_enabled\_flag: %d\n", dci\_ulsch\_p->freq\_hopping\_enabled\_flag);

**printf**("freq\_hopping\_bits: %d\n", dci\_ulsch\_p->freq\_hopping\_bits);

**printf**("ue\_tx\_antenna\_selection: %d\n", dci\_ulsch\_p->ue\_tx\_antenna\_selection);

**printf**("tpc: %d\n", dci\_ulsch\_p->tpc);

**printf**("cqi\_request: %d\n", dci\_ulsch\_p->cqi\_request);

**printf**("ul\_index: %d\n", dci\_ulsch\_p->ul\_index);

**printf**("dl\_assignment\_index: %d\n", dci\_ulsch\_p->dl\_assignment\_index);

**printf**("tpc\_bit\_map: %d\n", dci\_ulsch\_p->tpc\_bit\_map);

}

}

**printf**("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

**return** MAC\_ALL\_RIGHT;

}

INT32 **show\_dl\_dci\_info**(DciInfo \*dci\_info\_p)

{

log\_msg(*LOG\_INFO*, *MAC\_DLSCHED*, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*DCI INFO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

**printf**("dci\_info\_p->mcs\_index= %d \n", dci\_info\_p->mcs\_index);

**printf**("dci\_info\_p->ndi= %d \n", dci\_info\_p->ndi);

**printf**("dci\_info\_p->dci\_format = %d \n", dci\_info\_p->dci\_format);

**printf**("dci\_info\_p->scramble\_type = %d \n", dci\_info\_p->scramble\_type);

**printf**("dci\_info\_p->dci\_res\_alloc\_type = %d \n", dci\_info\_p->dci\_res\_alloc\_type);

**printf**("dci\_info\_p->dci\_vrb\_assign\_flag = %d \n", dci\_info\_p->dci\_vrb\_assign\_flag);

**printf**("dci\_info\_p->tch\_res\_alloc\_type = %d \n", dci\_info\_p->tch\_res\_alloc\_type);

**printf**("dci\_info\_p->tch\_vrb\_assign\_flag = %d \n", dci\_info\_p->tch\_vrb\_assign\_flag);

**printf**("dci\_info\_p->aggregation\_level = %d \n", dci\_info\_p->aggregation\_level);

**printf**("dci\_info\_p->cce\_index = %d \n", dci\_info\_p->cce\_index);

**printf**("dci\_info\_p->rb\_coding = 0x%x \n", dci\_info\_p->rb\_coding);

**printf**("dci\_info\_p->rb\_start = 0x%x \n", dci\_info\_p->rb\_start);

**printf**("dci\_info\_p->rb\_len = 0x%x \n", dci\_info\_p->rb\_len);

**printf**("dci\_info\_p->tb\_size\_index = %d \n", dci\_info\_p->tb\_size\_index);

**printf**("dci\_info\_p->dl\_assgn\_index = %d \n", dci\_info\_p->dl\_assgn\_index);

**printf**("dci\_info\_p->mcs = %d \n", dci\_info\_p->mcs);

**printf**("dci\_info\_p->tx\_scheme = %d \n", dci\_info\_p->tx\_scheme);

**printf**("dci\_info\_p->tb\_num = %d \n", dci\_info\_p->tb\_num);

**printf**("dci\_info\_p->layers\_num = %d \n", dci\_info\_p->layers\_num);

log\_msg(*LOG\_INFO*, *MAC\_DLSCHED*, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*DCI INFO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

**return** MAC\_ALL\_RIGHT;

}

INT32 **print\_space**(INT32 level){

INT32 i = 0;

**for** (i = 0; i < level; i++) {

**printf**(" ");

}

**return** 0;

}

INT32 **print\_lowmac\_node**(LowmacTxPduNode \*lowmac\_pdu\_node\_p, INT32 level) {

print\_space(level);

**printf**("lowmac\_pdu\_node\_p->rnti=%d\n",lowmac\_pdu\_node\_p->rnti);

print\_space(level);

**printf**("lowmac\_pdu\_node\_p->pdu\_index=%d\n",lowmac\_pdu\_node\_p->pdu\_index);

print\_space(level);

**printf**("lowmac\_pdu\_node\_p->sched\_type=%d\n",lowmac\_pdu\_node\_p->sched\_type);

print\_space(level);

**printf**("lowmac\_pdu\_node\_p->pdu\_type=%d\n",lowmac\_pdu\_node\_p->pdu\_type);

print\_space(level);

**printf**("lowmac\_pdu\_node\_p->ndi\_1st\_tb=%d\n",lowmac\_pdu\_node\_p->ndi\_1st\_tb);

print\_space(level);

**printf**("lowmac\_pdu\_node\_p->tx\_num=%d\n",lowmac\_pdu\_node\_p->tx\_num);

print\_space(level);

**printf**("lowmac\_pdu\_node\_p->harq\_process\_num=%d\n",lowmac\_pdu\_node\_p->harq\_process\_num);

print\_space(level);

**printf**("lowmac\_pdu\_node\_p->rv\_1st\_tb=%d\n",lowmac\_pdu\_node\_p->rv\_1st\_tb);

print\_space(level);

**printf**("lowmac\_pdu\_node\_p->pdu\_len=%d\n",lowmac\_pdu\_node\_p->pdu\_len);

print\_space(level);

show\_memory(lowmac\_pdu\_node\_p->pdu\_p, lowmac\_pdu\_node\_p->pdu\_len, "lowmac\_pdu\_node\_p->pdu\_p:");

**return** 0;

}

INT32 **print\_lowmac\_list**(LowmacTxPduList \*lowmac\_pdu\_list\_p) {

LowmacTxPduNode \*lowmac\_pdu\_node\_p = NULL;

INT32 tx\_num = 0, level = 2;

tx\_num = count\_list((ListType \*)lowmac\_pdu\_list\_p);

**printf**("list number=%d\n",tx\_num);

lowmac\_pdu\_node\_p = (LowmacTxPduNode \*)first\_list((ListType \*)lowmac\_pdu\_list\_p);

**while** (NULL != lowmac\_pdu\_node\_p) {

print\_lowmac\_node(lowmac\_pdu\_node\_p, level);

lowmac\_pdu\_node\_p = (LowmacTxPduNode \*)next\_list((NodeType \*)lowmac\_pdu\_node\_p);

}

**return** 0;

}

INT32 **free\_lowmac\_list**(LowmacTxPduList \*lowmac\_pdu\_list\_p) {

LowmacTxPduNode \*lowmac\_pdu\_node\_p = NULL;

lowmac\_pdu\_node\_p = (LowmacTxPduNode \*)get\_list((ListType \*)lowmac\_pdu\_list\_p);

**while** (lowmac\_pdu\_node\_p) {

**if**(lowmac\_pdu\_node\_p->pdu\_p){

lte\_free(lowmac\_pdu\_node\_p->pdu\_p);

}

lte\_free(lowmac\_pdu\_node\_p);

lowmac\_pdu\_node\_p = (LowmacTxPduNode \*)get\_list((ListType \*)lowmac\_pdu\_list\_p);

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* lowmac control when receive subframe indication.

\*

\* Input:

\* none.

\*

\* Output:

\* none.

\*

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **lowmac\_control**(**void**) /\* task function \*/

{

UINT8 tdd\_conf = get\_crrnt\_tdd\_conf();

FrameNumNode \*frame\_node\_p = NULL;

UINT8 mac\_sched\_offset = 0;

UINT8 temp\_sub\_fn;

UINT8 sched\_sub\_fn = 0;

LowmacTxPduList lowmac\_list;

init\_list((ListType \*)&lowmac\_list);

/\* process sf timers \*/

log\_msg(*LOG\_SUMMARY*, *MAC\_TX*,"\*\*\*\*\*\*\*\*\*\*\*\*\*system\_frame\_num = %d\*\*\*\*\*\*\*\*\*\*\*\n",g\_system\_frame\_num);

log\_msg(*LOG\_SUMMARY*, *MAC\_TX*,"\*\*\*\*\*\*\*\*\*\*\*\*\*sub\_frame\_num = %d\*\*\*\*\*\*\*\*\*\*\*\*\*\n",g\_sub\_frame\_num);

sf\_timeout\_process((10\* g\_system\_frame\_num) + g\_sub\_frame\_num);

/\*handle max tx schedule frame\*/

mac\_sched\_offset = 3;

temp\_sub\_fn = g\_sub\_frame\_num + mac\_sched\_offset;

sched\_sub\_fn = temp\_sub\_fn % 10;

**if**(mac\_sched\_offset && TDD\_SUBFRAME\_CONF[tdd\_conf][sched\_sub\_fn] == 1){

/\*if need to do mac schedule set schedule frame\*/

/\*add a new tx frame number to g\_frame\_num\_to\_mactx\_lst\*/

frame\_node\_p = (FrameNumNode \*)lte\_malloc(**sizeof**(FrameNumNode));

**if** (!frame\_node\_p) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "NOT enough memory for FrameNumNode!\n");

}

frame\_node\_p->sys\_fn = (g\_system\_frame\_num + temp\_sub\_fn / 10) % 1024;

frame\_node\_p->sub\_fn = sched\_sub\_fn;

//add\_list((ListType \*)&g\_frame\_num\_to\_mactx\_lst, (NodeType \*)frame\_node\_p);

/\*give sem to start MAC TX schedule task\*/

give\_sem(g\_mac\_tx\_sem);

log\_msg(*LOG\_INFO*, *MAC\_TX*, "give sem to MAC schedule!\n");

}

/\*do si schedule\*/

**if** (handle\_rrc\_si(g\_system\_frame\_num, g\_sub\_frame\_num)) {

log\_msg(*LOG\_INFO*, *SI\_MGMT*, "si did not tx in sys\_fn = %d, "

"sub\_fn = %d\n", g\_system\_frame\_num, g\_sub\_frame\_num);

}

get\_lowmac\_pdu(g\_system\_frame\_num, g\_sub\_frame\_num, &lowmac\_list);

print\_lowmac\_list(&lowmac\_list);

**if** (10 \* g\_system\_frame\_num + g\_sub\_frame\_num == 24)

print\_sched\_info();

**return** 0;

**#if** 0

/\* downlink subframe for tx\*/

**if** (TDD\_SUBFRAME\_CONF[tdd\_conf][g\_sub\_frame\_num] == 1) {

msg\_num = g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_num;

**if**(msg\_num <= 0 || msg\_num > 3){

**return** 0;

}

**for**(i = 0; i< 3; i++){

**if** (g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_len > 0 &&

NULL != g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_p){

**#if** defined TEST\_UE || defined TEST\_HENB

tx\_msg\_to\_enodeb(g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_p,

g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_len, rnti);

**#else**

log\_msg(LOG\_INFO, MAC\_TX, "sucess gen schedule l1api msg:%d, ssys\_fn = %d,sub\_fn = %d!\n",

g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_type, g\_system\_frame\_num, g\_sub\_frame\_num);

l1api\_msg\_cc2socket2((UINT32 \*)g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_p,

g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_len);

**#endif**

lte\_free(g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_p);

g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_p = NULL;

g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_buf[i].msg\_len = 0;

}

}

g\_lowmac\_tx\_buf[g\_sub\_frame\_num].msg\_num = 0;

}

**#endif**

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* initiate lowmac ctrl module and start lowmac ctrl task.

\*

\* Input:

\* none.

\*

\* Output:

\* none.

\*

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **init\_lowmac**(UINT8 tdd\_config)

{

**if** (tdd\_config < 0 || tdd\_config > 6) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "Tdd config error.\n");

**return** -1;

}

G\_CRNT\_TDD\_DLUL\_CONF = tdd\_config;

log\_msg(*LOG\_SUMMARY*, *MAC\_TX*, "TDD CONFIG IS %d\n", G\_CRNT\_TDD\_DLUL\_CONF);

/\* init g\_mac\_tx\_sem \*/

**if** ((g\_mac\_tx\_sem = create\_semb(SEM\_EMPTY)) == NULL) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "error,create semaphore failed\n");

**return** -1;

}

/\* init g\_mac\_tx\_sem \*/

**if** ((g\_si\_sched\_sem = create\_semb(SEM\_EMPTY)) == NULL) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "error,create semaphore failed\n");

**return** -1;

}

/\* init frame number \*/

g\_sub\_frame\_num = 0;

g\_system\_frame\_num = 0;

init\_list(&(g\_sf\_timer\_lst.harq\_list));

init\_list(&(g\_sf\_timer\_lst.tc\_list));

// memset(g\_lowmac\_tx\_buf, 0, sizeof(LowmacBuf\_t) \* 10);

/\* init debug info \*/

**memset**(&g\_mac\_info\_statistic, 0, **sizeof**(LowmacInfoDebug\_t));

g\_mac\_info\_statistic.mib\_tx\_cnt = 0;

g\_mac\_info\_statistic.sib\_tx\_cnt = 0;

**memset**(g\_mac\_info\_statistic.si\_tx\_cnt, 0, 13);

reset\_minmax\_avg\_timing(&g\_mac\_info\_statistic.bcch2bch\_sched\_period);

reset\_minmax\_avg\_timing(&g\_mac\_info\_statistic.bcch2dlsch\_sched\_period);

reset\_minmax\_avg\_timing(&g\_mac\_info\_statistic.dlsch\_crnti\_sched\_period);

reset\_minmax\_avg\_timing(&g\_mac\_info\_statistic.mac\_sched\_period);

reset\_minmax\_avg\_timing(&g\_mac\_info\_statistic.rrc\_si\_sched\_period);

reset\_minmax\_avg\_timing(&g\_mac\_info\_statistic.tti\_period);

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* stop lowmac ctrl task and clean up lowmac ctrl module.

\*

\* Input:

\* none.

\*

\* Output:

\* none.

\*

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **cleanup\_lowmac**(**void**)

{

//printf("send\_all\_num = %d, send\_all\_time=%ld, average=%f\n", send\_all\_num, send\_all\_time, (double)send\_all\_time/send\_all\_num);

/\* delete semaphore \*/

**if** (delete\_sem(g\_mac\_tx\_sem) == -1) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "delete g\_mac\_tx\_sem failed\n");

**return** -1;

}

**if** (delete\_sem(g\_si\_sched\_sem) == -1) {

log\_msg(*LOG\_ERR*, *MAC\_TX*, "delete g\_si\_sched\_sem failed\n");

**return** -1;

}

/\* clean up the allocated memory \*/

free\_list((ListType \*)&g\_sf\_timer\_lst);

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* receive liapi message by message queue.

\*

\* Input:

\* none.

\*

\* Output:

\* none.

\*

\* Return: 0 : succesful.

\* -1: faild.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **rx\_liapi\_msg**(**void**)

{

INT32 ret = 0;

RecvMsg recv\_msg;

**while**(1) {

**memset**(&recv\_msg, 0, **sizeof**(RecvMsg));

ret = receive\_msgq(g\_to\_lowmac\_rx\_msgq, &recv\_msg, **sizeof**(RecvMsg),

WAIT\_FOREVER);

**if** (ret > 0){

l1api\_msg\_decode(recv\_msg.msg\_p, recv\_msg.msg\_len);

}

}

**return** 0;

}

**void** **sf\_timeout\_process**(INT32 time)

{

SfTimerNode \*sf\_timer\_node\_p = NULL;

SfTimerNode \*tmp\_sf\_timer\_node\_p = NULL;

INT32 diff\_time = 0;

/\* process sf timers \*/

sf\_timer\_node\_p = (SfTimerNode \*)first\_list((ListType \*)&g\_sf\_timer\_lst);

**while** (sf\_timer\_node\_p) {

diff\_time = time - sf\_timer\_node\_p->expires;

**if** (diff\_time >=0 && diff\_time <= MAX\_SYS\_TIME / 2) {

/\*if the timer node expires in this subframe\*/

tmp\_sf\_timer\_node\_p = sf\_timer\_node\_p;

(tmp\_sf\_timer\_node\_p->func\_p)(&tmp\_sf\_timer\_node\_p->sf\_timer\_param);

sf\_timer\_node\_p = (SfTimerNode \*)next\_list

((NodeType \*)sf\_timer\_node\_p);

delete\_list((ListType \*)&g\_sf\_timer\_lst,(NodeType \*)tmp\_sf\_timer\_node\_p);

lte\_free(tmp\_sf\_timer\_node\_p);

} **else** {

/\*if the timer node does not expire in this subframe\*/

sf\_timer\_node\_p = (SfTimerNode \*)next\_list

((NodeType \*)sf\_timer\_node\_p);

}

}

}

**void** **print\_sched\_info**()

{

UINT32 i = 0;

MinMaxAvgTiming\_t tmp;

INT8 str\_buf[2048] = {0};

UINT32 str\_len = 0;

str\_len += **sprintf**((**char**\*)str\_buf, "\n/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAC sched info\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\n");

str\_len += **sprintf**(str\_buf+str\_len, "#######Mibcnt:%d\n", g\_mac\_info\_statistic.mib\_tx\_cnt);

str\_len += **sprintf**(str\_buf+str\_len, "#######Sibcnt:%d\n", g\_mac\_info\_statistic.sib\_tx\_cnt);

**for**(i = 0; i< 13;i++){

**if**(g\_mac\_info\_statistic.si\_tx\_cnt[i] >0){

str\_len += **sprintf**(str\_buf+str\_len, "#######si[%d], cnt:%d\n",i, g\_mac\_info\_statistic.si\_tx\_cnt[i]);

}

}

**for**(i = 0; i < 255; i++) {

**if**(g\_mac\_info\_statistic.l1apiMsgStat[i].rxTxOk >0){

str\_len += **sprintf**(str\_buf+str\_len, "#######l1api msg:0x%x cnt:%d\n", i, g\_mac\_info\_statistic.l1apiMsgStat[i].rxTxOk);

}

}

str\_len += **sprintf**(str\_buf+str\_len, "#######pch sched cnt:%d\n",g\_mac\_info\_statistic.pch\_sched\_cnt);

str\_len += **sprintf**(str\_buf+str\_len, "#######bcch2bch sched cnt:%d\n",g\_mac\_info\_statistic.bcch2bch\_sched\_cnt);

str\_len += **sprintf**(str\_buf+str\_len, "#######bcch2dlsch sched cnt:%d\n",g\_mac\_info\_statistic.bcch2dlsch\_sched\_cnt);

/\* mac all sched period \*/

tmp = g\_mac\_info\_statistic.mac\_sched\_period;

**if**( tmp.count != 0 )

{

str\_len += **sprintf**(str\_buf+str\_len, "#######mac all sched period (min:max:avg) %dus : %dus : %dus\n", \

tmp.min, tmp.max, (tmp.sum/tmp.count) );

}

**else**

{

str\_len += **sprintf**(str\_buf+str\_len, "#######mac sched period(min:max:avg) --- : --- : ---\n");

}

/\* bcch2bch sched period \*/

tmp = g\_mac\_info\_statistic.bcch2bch\_sched\_period;

**if**( tmp.count != 0 )

{

str\_len += **sprintf**(str\_buf+str\_len, "#######bcch2bch sched period (min:max:avg) %dus : %dus : %dus\n", \

tmp.min, tmp.max, (tmp.sum/tmp.count) );

}

**else**

{

str\_len += **sprintf**(str\_buf+str\_len, "#######bcch2bch sched period(min:max:avg) --- : --- : ---\n");

}

tmp = g\_mac\_info\_statistic.bcch2dlsch\_sched\_period;

**if**( tmp.count != 0 )

{

str\_len += **sprintf**(str\_buf+str\_len, "#######bcch2dlsch sched period (min:max:avg) %dus : %dus : %dus\n", \

tmp.min, tmp.max, (tmp.sum/tmp.count) );

}

**else**

{

str\_len += **sprintf**(str\_buf+str\_len, "#######bcch2dlsch sched period(min:max:avg) --- : --- : ---\n");

}

/\* SI rrc msg sched time \*/

tmp = g\_mac\_info\_statistic.rrc\_si\_sched\_period;

**if**( tmp.count != 0 )

{

str\_len += **sprintf**(str\_buf+str\_len, "#######si rrc handle sched period (min:max:avg) %dus : %dus : %dus\n", \

tmp.min, tmp.max, (tmp.sum/tmp.count) );

}

**else**

{

str\_len += **sprintf**(str\_buf+str\_len, "#######si rrc handle sched period(min:max:avg) --- : --- : ---\n");

}

str\_len += **sprintf**(str\_buf+str\_len, "/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAC sched info\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\n");

/\* TTI period \*/

tmp = g\_mac\_info\_statistic.tti\_period;

**if**( tmp.count != 0 )

{

str\_len += **sprintf**(str\_buf+str\_len, "#######TTI period(min:max:avg) %dus : %dus : %dus\n", \

tmp.min, tmp.max, (tmp.sum/tmp.count) );

}

**else**

{

str\_len += **sprintf**(str\_buf+str\_len, "#######TTI period period(min:max:avg) --- : --- : ---\n");

}

str\_len += **sprintf**(str\_buf+str\_len, "/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAC sched info\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\n");

log\_msg(*LOG\_SUMMARY*, *MAC\_DLSCHED*, "%s", str\_buf);

}

# Harq.c

/\*Dependencies----------------------------------------------------------------\*/

**#include** "lte\_malloc.h"

**#include** "lte\_list.h"

**#include** "lte\_log.h"

**#include** "harq.h"

**#include** "lowmac\_ctrl.h"

/\* Globals ------------------------------------------------------------------ \*/

//UINT8 dl\_max\_tx\_num; /\* the max number of DL tx \*/

UINT8 ul\_max\_tx\_num; /\* the max number of UL tx \*/

UINT8 ul\_proc\_num;

UINT8 dl\_proc\_num;

DlHarqUeInfo TDD\_HARQ[MAX\_UE\_NUM]; //store dl harq

UlHarqUeInfo TDD\_UL\_HARQ[MAX\_UE\_NUM]; //store ul harq

/\*the dl frame number of every config\*/

UINT8 DL\_NUM[7] = {4,6,8,7,8,9,5};

/\*the sequence of every dl frame number\*/

INT8 DL\_LOCATION[7][10] =

{

{0,1,-1,-1,-1,2,3,-1,-1,-1},

{0,1,-1,-1,2,3,4,-1,-1,5},

{0,1,-1,2,3,4,5,-1,6,7},

{0,1,-1,-1,-1,2,3,4,5,6},

{0,1,-1,-1,2,3,4,5,6,7},

{0,1,-1,2,3,4,5,6,7,8},

{0,1,-1,-1,-1,2,3,-1,-1,4}

};

UINT8 RV\_VALUE[4] = {0,2,3,1};

/\* Functions ---------------------------------------------------------------- \*/

**extern** UlHarqAck UL\_HARQ\_ACK[MAX\_UE\_NUM][MAX\_UL\_HARQ\_NUM];

/\* get ul harq info by sps-c-rnti.\*/

**extern** UlHarqUeInfo \*ul\_sps\_harq\_table[MAX\_UE\_NUM];

**extern** DlHarqUeInfo \*dl\_sps\_harq\_table[MAX\_UE\_NUM];

**const** UINT8 TDD\_DL\_ACK\_INDEX[7][10] =

{ {4,6,0,0,0,4,6,0,0,0},

{7,6,0,0,4,7,6,0,0,4},

{7,6,0,4,8,7,6,0,4,8},

{4,11,0,0,0,7,6,6,5,5},

{12,11,0,0,8,7,7,6,5,4},

{12,11,0,9,8,7,6,5,4,13},

{7,7,0,0,0,7,7,0,0,5} };

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* This ia an internel function to get the next redundancy version.

\*

\* Input: current\_rv: the current redundancy version.

\*

\* Output: \*next\_rv\_p: return the next redundancy version value.

\*

\* Return: 0:success; -1:failure.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **get\_next\_rv**(UINT8 current\_rv, UINT8 \*next\_rv\_p)

{

**if** (current\_rv == 0) {

\*next\_rv\_p = 2;

} **else** **if** (current\_rv == 2) {

\*next\_rv\_p = 3;

} **else** **if** (current\_rv == 3) {

\*next\_rv\_p = 1;

} **else** {

\*next\_rv\_p = 0;

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* the interface provided to do\_ullharq function to get ul process id .

\*

\* Input: sys\_fn: number of system frame.

\* sub\_fn:number of subframe.

\* \*proc\_id\_p: pointer to the MAC\_HARQ process ID.

\*

\* Output: \*proc\_id\_p: pointer to the MAC\_HARQ process ID

\*

\* Return: 0: success; -1: fail.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **get\_ulproc\_id**(UINT16 sys\_fn, UINT8 sub\_fn, UlHarqUeInfo \*ulharq\_p, UINT8 \*proc\_id\_p)

{

UINT8 conf = get\_crrnt\_tdd\_conf();

**switch** (conf) {

**case** 0:

**if** (sub\_fn >= 2 && sub\_fn <= 4) {

sub\_fn -= 2;

} **else** **if** (sub\_fn >= 7 && sub\_fn <= 9) {

sub\_fn -= 4;

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Illegal UL sub-frame number!\n");

**return** -1;

}

**if** (ulharq\_p->tti\_bundle\_enabled == 1) {

\*proc\_id\_p = sub\_fn + sys\_fn \* 6;

\*proc\_id\_p %= 14;// period is 14 subframes

\*proc\_id\_p /= 4;// bundling\_size is 4

**if** (\*proc\_id\_p == 3) {

// \*proc\_id\_p = 17;//spare subframes

**return** -1;

}

} **else** {

\*proc\_id\_p = (sub\_fn + sys\_fn \* 6) % 7;

}

**break**;

**case** 1:

**if** (sub\_fn == 2) sub\_fn = 0;

**else** **if** (sub\_fn == 3) sub\_fn = 1;

**else** **if** (sub\_fn == 7) sub\_fn = 2;

**else** **if** (sub\_fn == 8) sub\_fn = 3;

**else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Illegal UL sub-frame number!\n");

**return** -1;

}

**if** (ulharq\_p->tti\_bundle\_enabled == 1) {

\*proc\_id\_p = sub\_fn + sys\_fn \* 4;

\*proc\_id\_p %= 8;

\*proc\_id\_p /= 4;

\*proc\_id\_p += ulharq\_p->tti\_bundling\_proc\_num;

\*proc\_id\_p %= 2;

log\_msg(*LOG\_INFO*, *MAC\_HARQ*, "tti bundling get ulproc id is %d !\n", \*proc\_id\_p);

} **else** {

\*proc\_id\_p = sub\_fn;

}

**break**;

**case** 2:

**if** (sub\_fn == 2) \*proc\_id\_p = 0;

**else** **if** (sub\_fn == 7) \*proc\_id\_p = 1;

**else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, " Illegal UL sub-frame number!\n");

**return** -1;

}

**break**;

**case** 3:

**if** (sub\_fn == 2) \*proc\_id\_p = 0;

**else** **if** (sub\_fn == 3) \*proc\_id\_p = 1;

**else** **if** (sub\_fn == 4) \*proc\_id\_p = 2;

**else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "illegal UL sub-frame number!\n");

**return** -1;

}

**break**;

**case** 4:

**if** (sub\_fn == 2) \*proc\_id\_p = 0;

**else** **if** (sub\_fn == 3) \*proc\_id\_p = 1;

**else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, " illegal UL sub-frame number!\n");

**return** -1;

}

**break**;

**case** 5:

**if** (sub\_fn == 2) {

\*proc\_id\_p = 0;

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, " illegal UL sub-frame number!\n");

**return** -1;

}

**break**;

**case** 6:

**if** (sub\_fn >= 2 && sub\_fn <= 4) {

sub\_fn -= 2;

} **else** **if** (sub\_fn >= 7 && sub\_fn <= 8) {

sub\_fn -= 4;

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, " illegal UL sub-frame number!\n");

**return** -1;

}

**if** (ulharq\_p->tti\_bundle\_enabled == 1) {

\*proc\_id\_p = sub\_fn + sys\_fn \* 5;

\*proc\_id\_p %= 12;

\*proc\_id\_p /= 4;

\*proc\_id\_p += ulharq\_p->tti\_bundling\_proc\_num;

\*proc\_id\_p %= 3;

} **else** {

\*proc\_id\_p = (sub\_fn + sys\_fn \* 5) % 6;

}

**break**;

**default**:

**return** -1;

**break**;

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The interface provided to mac\_pdu\_rx module to do UL harq process.

\*

\* Input: rnti: the value of rnti.

\* dec\_result: the decode result, FAIL or SUCC.

\* data\_size: the byte size of the received tb.

\* phy\_p: the pointer of PHY resource.

\*

\* Output: NONE.

\*

\* Return: 0: success; -1: failure.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#if** 0

INT32 do\_ulharq(UlHarqCrcList \*harq\_crc\_list\_p)

{

UeRntiMsg \*rnti\_p = NULL;

UlHarqInfo \*ul\_harq\_p = NULL;

UlProcInfo \*ulproc\_p = NULL;

UINT16 sys\_fn = 0;

UINT8 sub\_fn = 0;

UINT8 ulproc\_id = 0;

UINT8 max\_harq\_tx = 0;

UlHarqCrcNode \*harq\_crc\_node\_p = NULL;

sys\_fn = harq\_crc\_list\_p->sys\_fn;

sub\_fn = harq\_crc\_list\_p->sub\_fn;

**if** (get\_ulproc\_id(sys\_fn, sub\_fn, &ulproc\_id)) {/\* get ul process id \*/

log\_msg(LOG\_ERR, MAC\_HARQ, "get ulproc id fail\n");

**return** ERR\_SYS\_PARAM;

}

harq\_crc\_node\_p = (UlHarqCrcList \*)get\_list((ListType \*)harq\_crc\_list\_p);

**while** (NULL != harq\_crc\_node\_p) {

rnti\_p =get\_rnti\_entity(harq\_crc\_node\_p->rnti); /\* get the pointer of RNTI entity \*/

**if** (!rnti\_p) {

log\_msg(LOG\_ERR, MAC\_HARQ, "unknown RNTI!!!rnti=%d\n",harq\_crc\_node\_p->rnti);

**return** -1;

}

**if**(C\_RNTI\_TYPE != rnti\_p->rnti\_type){

log\_msg(LOG\_INFO, MAC\_HARQ, "Rnti type is not c\_rnti!");

**return** 0;

}

ul\_harq\_p = (UlHarqInfo \*)(rnti\_p->ul\_harq\_info\_p);

**if** (!ul\_harq\_p) {

log\_msg(LOG\_ERR, MAC\_HARQ, "no harq infomation\n");

**return** ERR\_FUNC\_PARAM;

}

ulproc\_p = ul\_harq\_p ->ulproc\_p + ulproc\_id;

max\_harq\_tx = ul\_harq\_p->ul\_max\_harq\_tx;

// if (harq\_crc\_node\_p->crc\_flag == CRC\_CORRECT ||

// ulproc\_p->current\_tx\_num == max\_harq\_tx - 1)

// {/\* current starts from 0, and max\_harq\_tx starts from 1 \*/

// /\* set bsr and sr to 0 \*/

// memset(rnti\_p->ul\_sched\_param\_p, 0, sizeof(UlSchedParam));

// }

do\_ulharq\_process(harq\_crc\_node\_p->rnti, ulproc\_p, max\_harq\_tx, ulproc\_id, harq\_crc\_node\_p->crc\_flag);

lte\_free(harq\_crc\_node\_p);

harq\_crc\_node\_p = (UlHarqCrcNode \*)get\_list(harq\_crc\_list\_p);

}

**return** 0;

}

**#endif**

INT32 **do\_ulharq**(UINT16 sys\_fn, UINT8 sub\_fn, UINT16 rnti, INT8 crc\_flag)

{

UlHarqUeInfo \*ulharq\_p = NULL;

UlHarqProcInfo \*ulproc\_p = NULL;

UINT8 ulproc\_id = 0;

UINT8 max\_harq\_tx = 0;

ulharq\_p = get\_ul\_harq(rnti);

**if** (!ulharq\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "unknown RNTI !rnti = %d \n", rnti);

**return** -1;

}

/\* after explicit release, if getting ul data by sps crnti, fail and deactive sps again \*/

**if** (rnti >= MIN\_SPS\_RNTI && rnti <= MAX\_SPS\_RNTI) {

**if** (ulharq\_p->ul\_sps\_param\_p &&

ulharq\_p->ul\_sps\_param\_p->sps\_state == *MAC\_SPS\_NON\_ACTIVE*) {

log\_msg(*LOG\_ERR*, *MAC\_RX*, "deactive ul sps failed !\n");

deactive\_sps\_config(rnti, *UPLINK*);

**return** -1;

}

}

ulproc\_p = get\_ul\_harq\_proc(rnti, sys\_fn, sub\_fn);

**if** (!ulproc\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "no harq proc !\n");

**return** ERR\_FUNC\_PARAM;

}

max\_harq\_tx = ulharq\_p->ul\_max\_harq\_tx;

ulproc\_id = ulproc\_p->harq\_proc\_id;

/\*\*/

**if** (ulharq\_p->tti\_bundle\_enabled == 1) {

(ulproc\_p->tti\_bundling\_tx\_num)++;

**if** (ulproc\_p->tti\_bundling\_tx\_num < 4) {

log\_msg(*LOG\_SUMMARY*, *MAC\_HARQ*, "tti bundling data, tx\_num = %d \n",

ulproc\_p->tti\_bundling\_tx\_num );

**return** ulproc\_p->tti\_bundling\_tx\_num;

}

}

/\*\*/

do\_ulharq\_process(rnti, ulproc\_p, max\_harq\_tx, ulproc\_id, crc\_flag);

UL\_HARQ\_ACK[rnti - MIN\_C\_RNTI][ulproc\_id].rx\_sys\_fn = sys\_fn;

UL\_HARQ\_ACK[rnti - MIN\_C\_RNTI][ulproc\_id].rx\_sub\_fn = sub\_fn;

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The internel function, to do the ul harq process, record the information

\* of a process with respect to the different parameters.

\*

\* Input: ulproc\_p: the pointer which points the object process.

\* ul\_max\_harq\_tx: the max number of retransimission in UL harq process.

\* dec\_result: the decode result, FAIL or SUCC.

\* data\_size: the byte size of the received tb.

\* phy\_p: the pointer of PHY resource.

\*

\* Output: none.

\*

\* Return: 0: success; -1: fail.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **do\_ulharq\_process**(UINT16 rnti, UlHarqProcInfo \*ulproc\_p, UINT8 max\_harq\_tx,

UINT8 ulproc\_id, UINT8 crc\_flag)

{

UINT8 current\_tx\_num = 0;

**if** (!ulproc\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "The UL process is not exist !!!\n");

**return** -1;

}

current\_tx\_num = ulproc\_p->current\_tx\_num;

**if** (ulproc\_p->sched\_tran\_type == *MAC\_DYNAMIC\_NEW\_TRAN* ||

ulproc\_p->sched\_tran\_type == *MAC\_SPS\_NEW\_TRAN*) {

/\*not wait retransimission state \*/

**if** (crc\_flag == *CRC\_CORRECT*) {

ulproc\_p->tb\_size = 0;

ulproc\_p->sched\_tran\_type = *MAC\_PROC\_IDLE*;

ulproc\_p->current\_tx\_num = 0;

ulproc\_p->tti\_bundling\_tx\_num = 0;

**memset**(&(ulproc\_p->dci\_ulsch), 0, **sizeof**(DCIUlschInfo));

} **else** **if** (crc\_flag == *CRC\_ERROR*) {

current\_tx\_num ++; /\* start from 0 \*/

**if** (current\_tx\_num >= max\_harq\_tx) {

/\* current tx-num exceed the max tx-num \*/

ulproc\_p->tb\_size = 0;

ulproc\_p->sched\_tran\_type = *MAC\_PROC\_IDLE*;

ulproc\_p->current\_tx\_num = 0;

ulproc\_p->tti\_bundling\_tx\_num = 0;

**memset**(&(ulproc\_p->dci\_ulsch), 0, **sizeof**(DCIUlschInfo));

} **else** **if** (current\_tx\_num < max\_harq\_tx) {

ulproc\_p->current\_tx\_num = current\_tx\_num;

ulproc\_p->sched\_tran\_type =

(ulproc\_p->sched\_tran\_type == *MAC\_DYNAMIC\_NEW\_TRAN*) ? *MAC\_DYNAMIC\_RE\_TRAN* : *MAC\_SPS\_RE\_TRAN*;

ulproc\_p->tti\_bundling\_tx\_num = 0;

/\* need retransimission \*/

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*,"error:in current tx num!!!\n");

**return** -1;

}

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "invalid crc flag !\n");

**return** -1;

}

} **else** **if** (ulproc\_p->sched\_tran\_type == *MAC\_DYNAMIC\_RE\_TRAN* ||

ulproc\_p->sched\_tran\_type == *MAC\_SPS\_RE\_TRAN*) {

/\* wait retransimission state \*/

**if** (crc\_flag == *CRC\_CORRECT*) {

ulproc\_p->tb\_size = 0;

ulproc\_p->sched\_tran\_type = *MAC\_PROC\_IDLE*;

ulproc\_p->current\_tx\_num = 0;

ulproc\_p->tti\_bundling\_tx\_num = 0;

**memset**(&(ulproc\_p->dci\_ulsch), 0, **sizeof**(DCIUlschInfo));

} **else** **if** (crc\_flag == *CRC\_ERROR*) {

current\_tx\_num ++;

**if** (current\_tx\_num >= max\_harq\_tx) {/\* exceed the max tx-num \*/

ulproc\_p->tb\_size = 0;

ulproc\_p->sched\_tran\_type = *MAC\_PROC\_IDLE*;

ulproc\_p->current\_tx\_num = 0;

ulproc\_p->tti\_bundling\_tx\_num = 0;

**memset**(&(ulproc\_p->dci\_ulsch), 0, **sizeof**(DCIUlschInfo));

} **else** **if** (current\_tx\_num < max\_harq\_tx) {

ulproc\_p->current\_tx\_num = current\_tx\_num;

ulproc\_p->sched\_tran\_type =

(ulproc\_p->sched\_tran\_type == *MAC\_DYNAMIC\_RE\_TRAN*) ? *MAC\_DYNAMIC\_RE\_TRAN* : *MAC\_SPS\_RE\_TRAN*;

ulproc\_p->tti\_bundling\_tx\_num = 0;

/\* need retransimission \*/

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*,"error:in current tx num!!!\n");

**return** -1;

}

} **else** { /\* else DEC\_FAIL \*/

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "invalid crc flag !\n");

**return** -1;

}

} **else** { /\* else ulproc\_p->tran\_type != NEWTRAN\_TYPE \*/

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "error tran type of ulproc after recving data\n");

}

/\* build ACK/NACK list and add ACK/NACK node to the list \*/

UL\_HARQ\_ACK[rnti - MIN\_C\_RNTI][ulproc\_id].active = 1;

UL\_HARQ\_ACK[rnti - MIN\_C\_RNTI][ulproc\_id].transmissions = ulproc\_p->current\_tx\_num;

UL\_HARQ\_ACK[rnti - MIN\_C\_RNTI][ulproc\_id].ackNackResult = (crc\_flag == *CRC\_CORRECT*) ? *ACK* : *NACK*;

log\_msg(*LOG\_INFO*, *MAC\_HARQ*, "ACK/NACK result is %d \n",

UL\_HARQ\_ACK[rnti - MIN\_C\_RNTI][ulproc\_id].ackNackResult);

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* This function works when the wait ack feedback timer expires.

\*

\* Input: wait\_timer: the timer id of some process

\* \*proc\_p: the pointer to the process that the timer belongs to.

\*

\* Output: return TRUE, or FALSE if error occurs.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **do\_wait\_timer\_expiration**(DlHarqProcInfo \*dlproc\_p, UINT16 rnti)

{

DlHarqUeInfo \*dlharq\_p = NULL;

UINT16 dl\_max\_tx\_num;

**if** (dlproc\_p == NULL) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "input func param error !\n");

**return** ERR\_FUNC\_PARAM;

}

dlharq\_p = get\_dl\_harq(rnti);

**if** (NULL == dlharq\_p){

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "RNTI EROOR, rnti=%d !\n", rnti);

**return** ERR\_FUNC\_PARAM;

}

dl\_max\_tx\_num = dlharq\_p->dl\_max\_harq\_tx;

**if** (*T\_C\_RNTI\_TYPE* == get\_rnti\_type(rnti)) {

rcv\_tc\_ack\_nack(*NACK*, rnti);

**return** 0;

}

/\* deal with harq process related with ack/nack response \*/

**if** (dlproc\_p->state != *DLHARQ\_WAIT*){

log\_msg (*LOG\_ERR*, *MAC\_HARQ*, "Can't find process that ACK belongs to!\n");

**return** -1;

}**else**{

**if** (dlproc\_p->current\_tx\_num >= dl\_max\_tx\_num) {

dlproc\_p->state = *DLHARQ\_IDLE*;

dlproc\_p->current\_tx\_num = 0;

**if** (NULL != dlproc\_p->pdu\_p)

lte\_free(dlproc\_p->pdu\_p);

dlproc\_p->pdu\_p = NULL;

} **else** {

/\* set process state to be retransimission \*/

dlproc\_p->state = *DLHARQ\_RETX*;

dlproc\_p->current\_tx\_num++;

} /\* else (current\_tx\_num <= dl\_max\_tx\_num) \*/

} /\* else (i != process\_num) \*/

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* the interface provided to Pdu-TX module, to record and get the relative

\* HARQ information

\*

\* Input: rnti: the number of rnti.

\* ndi: the data would be transmitted is a new data, or just a

\* retransmitted.

\* tb\_size: the byte size of the transmitted tb.

\* \*pdu\_p: the pdu pointer,which point to mac pdu to be transmitted.

\* \*tbharqparam\_p: return some harq parameters to the tx-module.

\*

\* Output: \*tbharqparam\_p: return some harq parameters to the tx-module.

\*

\* Return: 0: success; -1: fail.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **do\_dlharq**(UINT16 rnti, UINT8 ndi, UINT32 tb\_size, UINT8 \*\*pdu\_p\_p,

DlHarqProcInfo \*dlproc\_p, DciInfo \*dci\_info\_p, UINT16 sys\_fn, UINT8 sub\_fn)

{

INT32 rt = 0;

**if** (NULL == dlproc\_p || NULL == dci\_info\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Invalid input parameter!\n");

**return** -1;

}

**if** (MAC\_NEWTX == ndi && NULL == \*pdu\_p\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Invalid input parameter pdu\_p\_p!\n");

**return** -1;

}

rt = get\_rnti\_type(rnti);

**switch**(rt) {

**case** *C\_RNTI\_TYPE*:

**case** *T\_C\_RNTI\_TYPE*:

**case** *SEMI\_C\_RNTI\_TYPE*:

**break**;

**default**:

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Rnti type is error.\n");

lte\_free(\*pdu\_p\_p);

**return** -1;

}

dlproc\_p->tx\_time = 10 \* sys\_fn + sub\_fn;

**if** (MAC\_NEWTX == ndi) {

/\* set dlproc \*/

dlproc\_p->rnti\_value = rnti;

dlproc\_p->tb\_size = tb\_size; /\*record the new transimission tb size \*/

dlproc\_p->current\_tx\_num = 0; // increase the times of tx

dci\_info\_p->rv = 0; // get new RV

dlproc\_p->state = *DLHARQ\_WAIT*; //set state to be wait ACK

dlproc\_p->pdu\_p = \*pdu\_p\_p;

**memcpy**(&dlproc\_p->dci\_info, dci\_info\_p, **sizeof**(DciInfo));

dlproc\_p->dci\_info.ndi = (dlproc\_p->pre\_ndi + 1) % 2;

dlproc\_p->pre\_ndi = dlproc\_p->dci\_info.ndi;

dlproc\_p->active = 1;

} **else** **if** (MAC\_RETX == ndi) {

dlproc\_p->rnti\_value = rnti;

dlproc\_p->current\_tx\_num++; // increase times of TX

dci\_info\_p->rv = RV\_VALUE[dlproc\_p->current\_tx\_num % 4]; //get retransimission RV

dlproc\_p->state = *DLHARQ\_WAIT*;/\* wait ack \*/

**if** (tb\_size != dlproc\_p->tb\_size) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "tb\_size is different with the process tb\_size!\n");

**return** -1;

}

**if** (!dlproc\_p->pdu\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "The process pdu is null!\n");

**return** -1;

}

\*pdu\_p\_p = dlproc\_p->pdu\_p;

**memcpy**(&dlproc\_p->dci\_info, dci\_info\_p, **sizeof**(DciInfo));

dlproc\_p->dci\_info.ndi = dlproc\_p->pre\_ndi;

dlproc\_p->active = 1;

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Harq ndi error, ndi=%d\n",ndi);

}

**return** MAC\_ALL\_RIGHT;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*do dl harq for bch

\*

\* Input:

\* \*tbharqparam\_p: return some harq parameters to the tx-module.

\*

\* Output: \*tbharqparam\_p: return some harq parameters to the tx-module.

\*

\* Return: 0: success; -1: fail.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **do\_dlharq4bch**(TbHarqParam \*tbharqparam\_p)

{

/\*need to add get rv\*/

**return** MAC\_ALL\_RIGHT;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MAC\_HARQ processing after receiving ACK/NACK.

\*

\* Input:ack\_node\_p: pointer to the ACK/NACK node.

\*

\* Output: NONE.

\*

\* Return: 0: success; -1: fail.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **process\_rcv\_ack**(DlHarqAckNode \*ack\_node\_p)

{

DlHarqUeInfo \*dlharq\_p = NULL;

DlHarqProcInfo \*dlproc\_p = NULL;

UINT32 tx\_time = 0;

UINT8 dl\_max\_tx\_num = 0;

**if** (!ack\_node\_p) {

log\_msg (*LOG\_ERR*, *MAC\_HARQ*, "invalid input parameter ack\_node\_p!\n");

**return** -1;

}

/\* get the pointer of RNTI entity \*/

dlharq\_p = get\_dl\_harq(ack\_node\_p->rnti);

**if** (NULL == dlharq\_p){

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "RNTI EROOR, rnti=%d !\n", ack\_node\_p->rnti);

**return** ERR\_FUNC\_PARAM;

}

tx\_time = ack\_node\_p->sys\_fn \* 10 + ack\_node\_p->sub\_fn;

**if** (NULL == dlharq\_p->dl\_sps\_param\_p) {

dlproc\_p = get\_dl\_harq\_proc(ack\_node\_p->rnti, ack\_node\_p->sys\_fn, ack\_node\_p->sub\_fn, *CRNTI\_DATA*);

**if** (NULL != dlproc\_p && tx\_time != dlproc\_p->tx\_time) {

dlproc\_p = NULL;

}

} **else** {

dlproc\_p = get\_dl\_harq\_proc(ack\_node\_p->rnti, ack\_node\_p->sys\_fn, ack\_node\_p->sub\_fn, *SPS\_DATA*);

**if** (NULL != dlproc\_p && tx\_time != dlproc\_p->tx\_time) {

dlproc\_p = get\_dl\_harq\_proc(ack\_node\_p->rnti, ack\_node\_p->sys\_fn, ack\_node\_p->sub\_fn, *CRNTI\_DATA*);

**if** (NULL != dlproc\_p && tx\_time != dlproc\_p->tx\_time) {

dlproc\_p = NULL;

}

}

}

**if** (NULL == dlproc\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Can not find the dl harq process, sys\_fn=%d, sub\_fn!\n",

ack\_node\_p->sys\_fn, ack\_node\_p->sub\_fn);

**return** ERR\_FUNC\_PARAM;

}

dl\_max\_tx\_num = dlharq\_p->dl\_max\_harq\_tx;

**if** (*T\_C\_RNTI\_TYPE* == get\_rnti\_type(ack\_node\_p->rnti)) {

rcv\_tc\_ack\_nack(ack\_node\_p->acktype, ack\_node\_p->rnti);

**return** 0;

}

/\* deal with harq process related with ack/nack response \*/

**if** (dlproc\_p->state != *DLHARQ\_WAIT*){

log\_msg (*LOG\_ERR*, *MAC\_HARQ*, "Can't find process that ACK belongs to!\n");

**return** -1;

}**else**{

**if** (ack\_node\_p->acktype == *ACK*) {

/\*set the process state to be IDLE after receiving ACK \*/

log\_msg(*LOG\_INFO*, *MAC\_HARQ*,"set the proc to be idle!!!\n");

dlproc\_p->state = *DLHARQ\_IDLE*;

dlproc\_p->current\_tx\_num = 0;

**if** (NULL != dlproc\_p->pdu\_p)

lte\_free(dlproc\_p->pdu\_p);

dlproc\_p->pdu\_p = NULL;

} **else** {/\*process tx faild,need retransmit\*/

**if** (dlproc\_p->current\_tx\_num >= dl\_max\_tx\_num) {

dlproc\_p->state = *DLHARQ\_IDLE*;

dlproc\_p->current\_tx\_num = 0;

**if** (NULL != dlproc\_p->pdu\_p)

lte\_free(dlproc\_p->pdu\_p);

dlproc\_p->pdu\_p = NULL;

} **else** {

/\* set process state to be retransimission \*/

dlproc\_p->state = *DLHARQ\_RETX*;

} /\* else (current\_tx\_num <= dl\_max\_tx\_num) \*/

} /\* else (receive NACK) \*/

} /\* else (i != process\_num) \*/

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MAC\_HARQ processing after receiving tc rnti ACK/NACK.

\*

\* Input:ack\_node\_p: pointer to the ACK/NACK node.

\*

\* Output: NONE.

\*

\* Return: 0: success; -1: fail.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **rcv\_tc\_ack\_nack**(AckType result, UINT16 rnti)

{

DlHarqUeInfo \*dl\_harq\_p = NULL;

DlHarqProcInfo \*dl\_proc\_p = NULL;

UeRntiMsg \*rnti\_msg\_p = NULL;

INT32 i = 0;

rnti\_msg\_p = get\_rnti\_entity(rnti);

dl\_harq\_p = get\_dl\_harq(rnti);

**if** (NULL == rnti\_msg\_p || NULL == dl\_harq\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "rnti error,rnti=%d!\n",rnti);

**return** -1;

}

**if** (*SEND\_MSG4* != rnti\_msg\_p->tc\_rnti\_state) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "TC rnti state error.rnti=%d.\n",rnti);

**return** -1;

}

**for** (i = 0; i < dl\_harq\_p->total\_proc\_num; i++) {

**if** (dl\_harq\_p->dl\_harq\_proc[i].state == *DLHARQ\_WAIT*) {

dl\_proc\_p = &dl\_harq\_p->dl\_harq\_proc[i];

**break**;

}

}

**if** (NULL == dl\_proc\_p){

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Has no tc harq.rnti=%d.\n",rnti);

**return** -1;

}

**if** (*ACK* == result) {

dl\_proc\_p->state = *DLHARQ\_IDLE*;

**if** (NULL != dl\_proc\_p->pdu\_p)

lte\_free(dl\_proc\_p->pdu\_p);

dl\_proc\_p->pdu\_p = NULL;

rnti\_msg\_p->tc\_rnti\_state = *RA\_NORMAL*;

turn\_tc2c\_rnti(rnti\_msg\_p->rnti\_value);

} **else** {

**if** (dl\_proc\_p->current\_tx\_num >= dl\_harq\_p->dl\_max\_harq\_tx) {

log\_msg(*LOG\_SUMMARY*, *MAC\_HARQ*, "TC harq retx has reached the max, RA failed.\n");

**#ifdef** TEST\_HENB\_TO\_UE

dl\_proc\_p->state = DLHARQ\_IDLE;

**if** (NULL != dl\_proc\_p->pdu\_p)

lte\_free(dl\_proc\_p->pdu\_p);

dl\_proc\_p->pdu\_p = NULL;

rnti\_msg\_p->tc\_rnti\_state = RA\_NORMAL;

turn\_tc2c\_rnti(rnti\_msg\_p->rnti\_value);

**#else**

delete\_ue\_harq(rnti);

delete\_ue\_entity(rnti);

**#endif**

**return** -1;

} **else** {

dl\_proc\_p->state = *DLHARQ\_RETX*;

rnti\_msg\_p->tc\_rnti\_state = *RE\_RCV\_MSG3*;

}

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Clean up the MAC\_HARQ modue.

\*

\* Input: rnti: the value of rnti.

\*

\* Output: NONE

\*

\* Return: 0: success; -1: fail.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **delete\_ue\_harq**(UINT16 rnti)

{

DlHarqUeInfo \*dlharq\_p = NULL;

UlHarqUeInfo \*ulharq\_p = NULL;

dlharq\_p = get\_dl\_harq(rnti);

ulharq\_p = get\_ul\_harq(rnti);

**if** (dlharq\_p == NULL || ulharq\_p == NULL){

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "RNTI EROOR, rnti=%d !\n", rnti);

**return** ERR\_FUNC\_PARAM;

}

/\*free sps\_param.\*/

**if** (ulharq\_p->ul\_sps\_param\_p) {

lte\_free(ulharq\_p->ul\_sps\_param\_p);

}

**if** (dlharq\_p->dl\_sps\_param\_p) {

lte\_free(dlharq\_p->dl\_sps\_param\_p);

}

/\*free phy cfg .\*/

**if** (ulharq\_p->phy\_cfg\_p) {

lte\_free(ulharq\_p->phy\_cfg\_p);

}

**memset**(dlharq\_p, 0, **sizeof**(DlHarqUeInfo));

**memset**(ulharq\_p, 0, **sizeof**(UlHarqUeInfo));

log\_msg(*LOG\_SUMMARY*, *MAC\_HARQ*, "delete ue harq !\n");

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The interface provided to mac\_management module, to record the relative HARQ

\* information.

\*

\* Input: rnti: value of RNTI.

\* harq\_conf\_param: configuration information of MAC\_HARQ.

\*

\* Output: NONE.

\*

\* Return: 0: success; -1: fail.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **config\_ue\_harq**(UINT16 rnti, HarqParam \*harq\_param)

{

DlHarqUeInfo \*dlharq\_p = NULL;

UlHarqUeInfo \*ulharq\_p = NULL;

UINT8 i = 0;

**if** (!harq\_param) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "input func param error !\n");

**return** ERR\_FUNC\_PARAM;

}

dlharq\_p = get\_dl\_harq(rnti);

ulharq\_p = get\_ul\_harq(rnti);

**if** (dlharq\_p == NULL || ulharq\_p == NULL){

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "RNTI EROOR, rnti=%d !\n", rnti);

**return** ERR\_FUNC\_PARAM;

}

ulharq\_p->process\_num = harq\_param->ul\_proc\_num;

dlharq\_p->total\_proc\_num = harq\_param->dl\_proc\_num;

dlharq\_p->max\_sps\_proc\_num = 0; /\* from rrc config \*/

dlharq\_p->dl\_max\_harq\_tx = harq\_param->dl\_max\_tx\_num;

ulharq\_p->ul\_max\_harq\_tx = harq\_param->ul\_max\_tx\_num;

ulharq\_p->max\_harq\_msg3\_tx = harq\_param->ul\_max\_tx\_num;

dlharq\_p->ack\_waiting\_time = harq\_param->ack\_waiting\_time;

/\* need to be read again by the designer of mac\*/

// dlharq\_p->idle\_proc\_num = harq\_param->dl\_proc\_num;

// dlharq\_p->sps\_idle\_proc\_num = 0;

// dlharq\_p->ndi = MAC\_NEWTX;

dlharq\_p->rnti\_value = rnti;

ulharq\_p->rnti\_value = rnti;

dlharq\_p->flag = 1;

ulharq\_p->flag = 1;

/\* config dl proc info \*/

**for** (i = 0; i < dlharq\_p->total\_proc\_num; i++) {

dlharq\_p->dl\_harq\_proc[i].harq\_proc\_id = i;

}

**for** (i = 0; i < ulharq\_p->process\_num; i++) {

ulharq\_p->ul\_harq\_proc[i].harq\_proc\_id = i;

}

log\_msg(*LOG\_SUMMARY*, *MAC\_HARQ*, "config ue harq !\n");

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Initial the enviroment about harq.

\*

\* Input: none.

\*

\* Output: none.

\*

\* Return: 0: success; -1: fail.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INT32 **init\_harq**()

{

**memset**(UL\_HARQ\_ACK, 0, MAX\_UE\_NUM \* MAX\_UL\_HARQ\_NUM\* **sizeof**(UlHarqAck));

**memset**(TDD\_HARQ, 0, MAX\_UE\_NUM \* **sizeof**(DlHarqUeInfo));

**memset**(TDD\_UL\_HARQ, 0, MAX\_UE\_NUM \* **sizeof**(UlHarqUeInfo));

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Get the ue dl harq.

\*

\* Input :

\* rnti : the value of ue entity.

\*

\* Output :

\* dlharq: the struct point of dlharqinfo.

\* return :

\* dlharq: the struct point of dlharqinfo.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DlHarqUeInfo\* **get\_dl\_harq**(UINT16 rnti)

{

// if (rnti < MIN\_C\_RNTI || rnti >= MIN\_C\_RNTI + MAX\_UE\_NUM) {

// log\_msg(LOG\_ERR, MAC\_HARQ, "The rnti is illegal! rnti=%d\n", rnti);

// return NULL;

// }

// return &TDD\_HARQ[rnti - MIN\_C\_RNTI];

**if** (rnti >= MIN\_C\_RNTI && rnti <= MIN\_C\_RNTI + MAX\_UE\_NUM -1) {

**return** &TDD\_HARQ[rnti - MIN\_C\_RNTI];

} **else** **if** (rnti >= MIN\_SPS\_RNTI && rnti <= MAX\_SPS\_RNTI) {

**return** dl\_sps\_harq\_table[rnti - MIN\_SPS\_RNTI];

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "The rnti is illegal! rnti = %d \n", rnti);

**return** NULL;

}

}

UlHarqUeInfo\* **get\_ul\_harq**(UINT16 rnti)

{

**if** (rnti >= MIN\_C\_RNTI && rnti <= MIN\_C\_RNTI + MAX\_UE\_NUM -1) {

**return** &TDD\_UL\_HARQ[rnti - MIN\_C\_RNTI];

} **else** **if** (rnti >= MIN\_SPS\_RNTI && rnti <= MAX\_SPS\_RNTI) {

**return** ul\_sps\_harq\_table[rnti - MIN\_SPS\_RNTI];

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "The rnti is illegal! rnti = %d \n", rnti);

**return** NULL;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Get the ue dl harq process.

\*

\* Input :

\* rnti : the value of ue entity.

\* sys\_fn: the systerm frame number

\* sub\_fn: the subframe number.

\* sps\_flag: the flag of sps. 0 is dynamic schedule ,1 is sps schedule

\* Output :

\* dl\_harq\_proc: the struct point of dlharqprocinfo.

\* return :

\* dl\_harq\_proc: the struct point of dlharqprocinfo.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DlHarqProcInfo\* **get\_dl\_harq\_proc**(UINT16 rnti, UINT16 sys\_fn, UINT8 sub\_fn, UINT8 sps\_flag)

{

DlHarqUeInfo \*dl\_harq\_p = NULL;

UINT8 proc\_id = 0;

dl\_sps\_param\_t \*dl\_sps\_param\_p = NULL;

**if** (rnti < MIN\_C\_RNTI || rnti >= MIN\_C\_RNTI + MAX\_UE\_NUM) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "The rnti is illegal! rnti=%d\n", rnti);

**return** NULL;

}

dl\_harq\_p = get\_dl\_harq(rnti);

**if** (NULL == dl\_harq\_p || (dl\_harq\_p->total\_proc\_num <= 0 ||

dl\_harq\_p->total\_proc\_num > MAX\_DL\_HARQ\_NUM)) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Get ue harq error! rnti=%d\n", rnti);

**return** NULL;

}

**if** (sps\_flag == 0) {

**if** (DL\_LOCATION[G\_CRNT\_TDD\_DLUL\_CONF][sub\_fn] < 0 ||

DL\_LOCATION[G\_CRNT\_TDD\_DLUL\_CONF][sub\_fn] >= DL\_NUM[G\_CRNT\_TDD\_DLUL\_CONF]) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "The frame info is error. sub\_fn=%d\n", sub\_fn);

**return** NULL;

}

proc\_id = (sys\_fn \* DL\_NUM[G\_CRNT\_TDD\_DLUL\_CONF] + DL\_LOCATION[G\_CRNT\_TDD\_DLUL\_CONF][sub\_fn]) %

(dl\_harq\_p->total\_proc\_num - dl\_harq\_p->max\_sps\_proc\_num) + dl\_harq\_p->max\_sps\_proc\_num;

} **else** **if** (sps\_flag == 1) {

dl\_sps\_param\_p = dl\_harq\_p->dl\_sps\_param\_p;

**if** (NULL == dl\_sps\_param\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "SPS param is null.\n");

**return** NULL;

}

**if** (dl\_sps\_param\_p->dl\_intval <=0 || dl\_sps\_param\_p->sps\_proc\_num <=0

|| dl\_sps\_param\_p->sps\_proc\_num > MAX\_DL\_HARQ\_NUM) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "SPS param error, dl-intval=%d, sps\_proc\_num=%d\n",

dl\_sps\_param\_p->dl\_intval, dl\_sps\_param\_p->sps\_proc\_num);

**return** NULL;

}

proc\_id = ((sys\_fn \* 10 + sub\_fn) / dl\_sps\_param\_p->dl\_intval) % dl\_sps\_param\_p->sps\_proc\_num;

} **else** {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Sps flag error. flag=%d\n", sps\_flag);

**return** NULL;

}

**return** &dl\_harq\_p->dl\_harq\_proc[proc\_id];

}

UlHarqProcInfo\* **get\_ul\_harq\_proc**(UINT16 rnti, UINT16 sys\_fn, UINT8 sub\_fn)

{

UlHarqUeInfo \*ul\_harq\_p = NULL;

UINT8 ulproc\_id = 0;

// if (rnti < MIN\_C\_RNTI || rnti >= MIN\_C\_RNTI + MAX\_UE\_NUM) {

// log\_msg(LOG\_ERR, MAC\_HARQ, "The rnti is illegal! rnti=%d\n", rnti);

// return NULL;

// }

// (rnti >= MIN\_SPS\_RNTI && rnti <= MAX\_SPS\_RNTI)

ul\_harq\_p = get\_ul\_harq(rnti);

**if** (NULL == ul\_harq\_p) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Get ul harq error! rnti=%d\n", rnti);

**return** NULL;

}

**if** (get\_ulproc\_id(sys\_fn, sub\_fn, ul\_harq\_p, &ulproc\_id)) {

log\_msg(*LOG\_ERR*, *MAC\_HARQ*, "Get ul proc id error! ");

**return** NULL;

}

**return** &(ul\_harq\_p->ul\_harq\_proc[ulproc\_id]);

}