**文件编号：LTE\_HeNB\_MAC\_HARQ\_XXSJ\_V1.0**

**TD-LTE HeNB协议栈软件系统**

**MAC\_HARQ模块**

**详细设计说明书**

拟制：张倩

时间：2010-10-27

**中国科学院计算技术研究所**

**无线通信技术研究中心**

**软件组**

**LTE协议栈研发项目组**

**修改记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文件编号 | 版本号 | 拟制人/  修改人 | 拟制/修改日期 | 更改理由 | 主要更改内容  （写要点即可） |
|  | 1.0 | 张倩 | 2010-10-27 |  |  |

修改列表：



本文档的程序或内容受版权法的保护，未经中科院计算所的书面许可，不得擅自泄漏、拷贝或复制本文档资料的全部或部分。

**目录**

[**TD-LTE HeNB协议栈软件系统** 1](#_Toc291493117)

[1 引言 5](#_Toc291493118)

[1.1 编写目的 5](#_Toc291493119)

[1.2 背景 5](#_Toc291493120)

[1.3 定义 5](#_Toc291493121)

[1.4 参考资料 5](#_Toc291493122)

[2 模块描述 6](#_Toc291493123)

[2.1 模块综述 6](#_Toc291493124)

[2.2 模块接口 6](#_Toc291493125)

[3 模块设计 7](#_Toc291493126)

[3.1 模块结构 7](#_Toc291493127)

[3.2 关键算法 8](#_Toc291493128)

[3.2.1 HARQ初始化 8](#_Toc291493129)

[3.2.1 上行HARQ处理流程 8](#_Toc291493130)

[3.2.2 接收ACK/NACK消息流程 9](#_Toc291493131)

[3.2.3 下行HARQ处理流程 9](#_Toc291493132)

[3.3 函数描述 10](#_Toc291493133)

[3.4 函数调用关系 11](#_Toc291493134)

[4 数据描述 12](#_Toc291493135)

[4.1数据结构说明 12](#_Toc291493136)

[4.2全局变量说明 14](#_Toc291493137)

[4.3数据库说明 14](#_Toc291493138)

[5 函数定义 15](#_Toc291493139)

[5.1引用函数描述 15](#_Toc291493140)

[5.1.1 引用帧管理模块的接口 15](#_Toc291493141)

[5.1.2 引用RNTI管理模块的接口 15](#_Toc291493142)

[5.2内部函数定义 15](#_Toc291493143)

[INT32 set\_dlproc\_state (DlProcInfo \*dlproc\_p, UINT8 state) 15](#_Toc291493144)

[INT32 do\_ulharq\_process (UlProcInfo \*ulproc\_p, UINT8 ul\_max\_harq\_tx, INT8 dec\_result, UINT32 data\_size, ResourcePhyInfo \*phy\_p 16](#_Toc291493145)

[INT32 get\_next\_rv (UINT8 current\_rv, UINT8 \*next\_rv\_p) 17](#_Toc291493146)

[INT32 get\_ulproc\_id(UINT32 sys\_fn, UINT8 sub\_fn，UINT8 \*proc\_id\_p) 17](#_Toc291493147)

[INT32 rcv\_ack\_nack (AckNode \* ack\_node\_p) 18](#_Toc291493148)

[INT32 do\_newtran\_dlharq (DlHarqInfo \*dlharq\_p, UINT32 tb\_size, UINT8 \*pdu\_p, TbHarqParam \*tbharqparam\_p) 19](#_Toc291493149)

[INT32 do\_retran\_dlharq (DlHarqInfo \*dlharq\_p, UINT8 \*pdu\_p, TbHarqParam \*tbharqparam\_p) 20](#_Toc291493150)

[INT32 get\_dlproc\_id (DlProcInfo \*dlproc\_p，UINT8 \*dlproc\_id\_p，UINT8 state) 20](#_Toc291493151)

[6 接口设计 21](#_Toc291493152)

[6.1用户接口 21](#_Toc291493153)

[6.2硬件接口 21](#_Toc291493154)

[6.3软件接口 21](#_Toc291493155)

[INT32 do\_ulharq (UINT16 rnti, INT8 dec\_result, UINT32 data\_size, ResourcePhyInfo \*phy\_p) 21](#_Toc291493156)

[INT32 process\_rcv\_ack (AckLst \*ack\_lst\_p) 22](#_Toc291493157)

[INT32 do\_dlharq (UINT16 rnti, UINT8 ndi, UINT32 tb\_size, UINT8 \*pdu\_p, TbHarqParam \*tbharqparam\_p) 23](#_Toc291493158)

[INT32 get\_tx\_acklst (AckLst \*ack\_lst) 24](#_Toc291493159)

[INT32 get\_ulharq\_sched\_info (UINT16 rnti, UINT8 \*retran\_ind, UINT32 \*tb\_size) 24](#_Toc291493160)

[INT32 get\_dlharq\_sched\_info (UINT16 rnti, UINT8 \*state, UINT32 \*tb\_size) 24](#_Toc291493161)

[INT32 config\_ue\_harq (UINT16 rnti, HarqParam harq\_param) 24](#_Toc291493162)

[INT32 init\_ue\_harq (UINT16 rnti, HarqParam harq\_param) 25](#_Toc291493163)

[INT32 delete\_ue\_harq (UINT16 rnti) 25](#_Toc291493164)

[7 功能测试设计 26](#_Toc291493165)

[7.1 接收ACK/NACK功能测试 26](#_Toc291493166)

[7.2下行HARQ处理功能测试 26](#_Toc291493167)

[7.3上行HARQ处理功能测试 27](#_Toc291493168)

[8 其他 28](#_Toc291493169)

**详细设计说明书**

# 1 引言

## 1.1 编写目的

本说明书对LTE HeNB端MAC子系统HARQ模块的主要功能以及子模块划分进行简单的说明，阐述HeNB端MAC子系统HARQ模块的设计思想以及其工作流程。本说明书的读者为LTE UE端MAC子系统HARQ模块设计、编码人员、测试人员、项目组负责人员、实验室主任及相关项目管理人员。

编写本说明书的目的在于

* 为开发人员提供依据。
* 为修改和维护本系统提供条件。
* 项目负责人将根据本文档计划和控制系统设计、开发的全过程。

## 1.2 背景

软件系统名称：TD-LTE HeNB协议栈软件系统

软件开发者：无线通信技术研究中心LTE协议栈研发组

开发语言：C语言

硬件平台：

软件平台：linux 2.6

## 1.3 定义

无

## 1.4 参考资料

1. 《3GPP TS 36.321, "Medium Access Control (MAC) protocol specification", Rel. 8, V 8.9.0》
2. 《3GPP TS 36.300, "Overall description", Rel. 8, V 8.b.0》
3. 《3GPP TS 36.331, "Radio Resource Control (RRC)", Rel. 8, V 8.9.0》

# 2 模块描述

## 2.1 模块综述

HARQ模块在MAC收发子系统中，负责上下行共享信道的HARQ信息的记录与处理。具体分为上行HARQ功能和下行HARQ功能，分别于MAC接收模块和MAC发送模块设有接口。在接收上行数据时完成HARQ的功能并负责发送ACK/NACK，上行HARQ的功能；发送下行数据时根据下行数据的ACK/NACK反馈，决定完成新传或是重传，完成下行HARQ的功能。

|  |  |
| --- | --- |
| 功能 | 功能描述 |
| HARQ初始化，重置，模块销毁。 | 得到基站上层为每个UE所配置的HARQ配置参数。负责初始化模块，模块最终的销毁。 |
| 上行HARQ功能 | 实现上行HARQ功能，并负责对上行数据的接收反馈。并维护每个UE的上行HARQ实体，提供给上行调度相关参数。完成上行同步HARQ的功能。 |
| 下行HARQ功能 | 实现下行HARQ功能，负责接收下行数据的反馈消息。并维护每个UE的下行HARQ实体，提供给下行调度相关参数。完成下行异步HARQ的功能。 |

## 2.2 模块接口



图１ HARQ模块与其它模块接口关系图

HARQ模块与其它模块的输入输出关系如图N所示。

* 提供给MAC接收模块的接口
* **do\_ulharq():** 接收到上行数据通知上行HARQ实体进行上行HARQ处理。
* 提供给lowmac模块的接口
* **process\_rcv\_ack():** 接收到ACK/NACK消息通知相应下行HARQ实体。
* 提供给MAC发送模块的接口
* **do\_dlharq()：**发送下行数据做下行HARQ处理。
* **get\_tx\_acklst( )：**得到一个在PHICH发送ACK/NACK的链表。
* 提供给MAC管理模块的接口
* **config\_ue\_harq()：**基站记录为每一个UE的HARQ配置。
* 提供给MAC调度模块的接口
* **get\_ulharq\_sched\_info()：**提供给上行调度，通知上行HARQ情况。
* **get\_dlharq\_sched\_info()：**提供给下行调度，通知下行HARQ情况。

# 3 模块设计

## 3.1 模块结构



模块结构如图，子模块划分如下。

* HARQ管理子模块

供给MAC管理模块接口，得到基站和每个UE的HARQ配置参数。负责初始化模块，模块最终的销毁，数据结构的内存分配与释放提。

* 上行HARQ处理子模块

实现上行HARQ功能，并负责对上行数据的接收反馈，即发送ACK/NACK消息。并维护每个UE的上行HARQ实体，提供给上行调度相关参数。

* 下行HARQ处理子模块

实现下行HARQ功能，负责接收ACK/NACK消息。并维护每个UE的下行HARQ实体，提供给下行调度相关参数。

## 3.2 关键算法

### 3.2.1 HARQ初始化

根据上层的配置信息，初始化UE的HARQ实体。init\_ue\_harq ()



### 3.2.1 上行HARQ处理流程

MAC接收模块收到PUSCH上行数据，调用接口函数do\_ulharq()进行上行HARQ处理。处理流程如下



### 3.2.2 接收ACK/NACK消息流程

lowmac模块收到PUSCH或PUCCH上的ACK/NACK消息行数据，调用接口函数process\_rcv\_ack（），进行下行HARQ处理。处理流程如下



### 3.2.3 下行HARQ处理流程

MAC发送模块在发送PDSCH数据时做HARQ处理。



## 3.3 函数描述

|  |  |  |
| --- | --- | --- |
| **函数名** | **功能** | **子模块** |
| init\_ue\_harq() | 为某个UE建立上下行HARQ实体，并建立上下行HARQ进程 | HARQ管理子模块 |
| delete\_ue\_harq() | 销毁UE的HARQ实体 | HARQ管理子模块 |
| config\_ue\_harq() | 提供给MAC管理模块的接口，配置UE的上下行HARQ。 | HARQ管理子模块 |
| do\_ulharq() | 提供给接收模块的接口，做上行数据的HARQ处理 | 上行HARQ子模块 |
| get\_tx\_acklst() | 得到要发送的ACK链表 | 上行HARQ子模块 |
| get\_ulharq\_sched\_info() | 提供给调度模块的接口，得到UE的上行HARQ信息 | 上行HARQ子模块 |
| do\_ulharq\_process() | 找到该HARQ进程后做HARQ处理 | 上行HARQ子模块 |
| get\_ulproc\_id() | 通过接收的子帧号和帧号得到HARQ进程号 | 上行HARQ子模块 |
| get\_next\_rv() | 根据当前rv获得下一个重传时所需的rv | 上行HARQ子模块 |
| process\_rcv\_ack() | 提供给lowmac模块的接口，做上行数据反馈的HARQ处理 | 下行HARQ子模块 |
| do\_dlharq() | 提供给MAC发送模块的接口，做下行数据的HARQ处理 | 下行HARQ子模块 |
| get\_dlharq\_sched\_info() | 提供给调度模块的接口，得到UE的下行HARQ信息 | 下行HARQ子模块 |
| rcv\_ack\_nack() | 接收到ACK/NACK的HARQ处理 | 下行HARQ子模块 |
| set\_dlproc\_state() | 将该下行进程初始化为IDLE状态或需要重传调度状态 | 下行HARQ子模块 |
| do\_newtran\_dlharq（） | 做下行新传数据的HARQ处理 | 下行HARQ子模块 |
| do\_retran\_dlharq（） | 做下行重传数据的HARQ处理 | 下行HARQ子模块 |
| get\_ulproc\_id() | 得到下行HARQ进程号 | 下行HARQ子模块 |

## 3.4 函数调用关系



# 4 数据描述

## 4.1数据结构说明

/\*HARQ初始化设置参数\*/

typedef struct {

UINT8 ul\_proc\_num; /\*该UE上行HARQ进程数\*/

UINT8 dl\_proc\_num; /\*该UE下行HARQ进程数\*/

UINT8 dl\_max\_tx\_num; /\*该UE下行HARQ进程所配置的重传最大数\*/

UINT8 ul\_max\_tx\_num; /\*该UE上行HARQ进程所配置的重传最大数\*/

INT32 ack\_waiting\_time; /\*该UE下行HARQ进程等待ACK时间\*/

} HarqParam;

/\*一个UE的上行HARQ实体\*/

typedef struct {

UlProcInfo \*ulproc\_p; /\*上行HARQ实体中的HARQ进程指针\*/

UINT8 process\_num; /\*上行HARQ实体的总进程数\*/

UINT8 ul\_max\_harq\_tx; /\*该UE上行HARQ进程所配置的重传最大数\*/

UINT8 max\_harq\_msg3\_tx; /\*该UE上行MSG3配置的重传最大数\*/

} UlHarqInfo;

/\*一个UE的上行HARQ进程\*/

typedef struct {

ResourcePhyInfo phy\_info;

UINT8 retran\_ind; /\*指示是否需要重传\*/

UINT32 tb\_size; /\*重传TB大小\*/

UINT32 current\_tx\_num; /\*失败传输的次数\*/

} UlProcInfo;

/\*物理资源PRB信息\*/

typedef struct {

UINT16 rb\_start; /\*起始RB\*/

UINT16 rb\_num; /\*RB个数\*/

UINT8 mcs;

UINT8 rv;

} ResourcePhyInfo;

/\*一个UE的下行HARQ实体\*/

typedef struct {

DlProcInfo \*dlproc\_p; /\*下行HARQ实体中的HARQ进程指针\*/

UINT8 total\_proc\_num; /\*下行HARQ实体的总进程数\*/

UINT8 retran\_tb\_size; /\*指示所有重传的下行HARQ进程的总TB大小\*/

UINT8 retran\_proc\_num; /\*指示现在重传进程的个数\*/

UINT8 idle\_proc\_num; /\*指示现在空闲进程的个数\*/

UINT8 ndi；

UINT8 dl\_max\_harq\_tx; /\*该UE下行HARQ进程所配置的重传最大数\*/

UINT8 max\_sps\_proc\_num; /\*该UE下行SPS HARQ进程最大数\*/

} DlHarqInfo;

/\*一个下行HARQ进程\*/

typedef struct {

UINT32 dl\_tx\_time; /\*下行数据发送的时间\*/

Timer\_t wait\_ack\_timer; /\*等待ACK的计时器\*/

PduInfoLst \*pdulst\_p; /\*缓存的下行数据指针\*/

UINT32 tb\_size; /\*下行数据大小\*/

UINT8 current\_tx\_num; /\*现在下行发送的次数\*/

UINT8 rv; /\*RV\*/

UINT8 state; /\*下行态指示，idle，wait，retran三种状态\*/

} DlProcInfo;

/\*ACK/NACK消息节点\*/

typedef struct {

NodeType ln;

UINT16 rnti；

AckType acktype; /\* ack类型 \*/

UINT8 sub\_fn; /\* 子帧号 \*/

UITN8 sys\_fn; /\* 系统帧号 \*/

} AckNode；

typedef enum {

NACK = 1,

ACK = 0

} AckType;

/\*ACK/NACK消息链表\*/

typedef ListType AckLst /\* ACK节点链表 \*/

typedef struct{

UINT8 process\_id; /\* process number for current TB \*/

UINT8 irv; /\* redundancy version for current TB \*/

}TbHarqParam;

## 4.2全局变量说明

UINT8 dl\_max\_tx\_num; 下行最大发送次数

UINT8 ul\_max\_tx\_num; 上行最大发送次数

INT32 ack\_waiting\_time; 发送后的ACK等待时间，超时后将认为发送失败，要求重传。

AckLst \*g\_tx\_ack\_lst\_p; 指向ack发送队列的指针

## 4.3数据库说明

无

# 5 函数定义

## 5.1引用函数描述

### 5.1.1 引用帧管理模块的接口

对时间[子帧数]的获取，对核心数据结构的操作等。

#### UINT8 get\_frame\_num(void)

**功能描述**

获得当前的系统帧号

#### UINT8 get\_subframe\_num (void)

**功能描述**

获得当前的子帧号

### 5.1.2 引用RNTI管理模块的接口

#### INT32 get\_rnti\_type (UINT16 rnti)

**功能描述**

根据rnti的值，判断当前此rnti的使用类型

#### UeRntiMsg \*get\_rnti\_entity (UINT16 rnti)

**功能描述**

根据rnti的值获得核心数据结构中该rnti节点的指针。

## 5.2内部函数定义

### INT32 set\_dlproc\_state (DlProcInfo \*dlproc\_p, UINT8 state)

**函数描述**

将该下行进程设置为需要重传调度状态，或空闲状态。

**输入**

dlproc\_p：需要置为重传调度状态的HARQ下行进程。

state: 重传调度状态，或空闲状态。

**算法与处理流程**

停止等待ACK的计时器，将该进程结构体DlProcInfo中的state置为需要重传调度的状态或空闲状态。

**输出**

无

**返回**

0/1，操作是否成功。

### INT32 do\_ulharq\_process (UlProcInfo \*ulproc\_p, UINT8 ul\_max\_harq\_tx, INT8 dec\_result, UINT32 data\_size, ResourcePhyInfo \*phy\_p

**功能描述**：

获得所需进程号

**输入**：

ulproc\_p：处理的HARQ上行进程指针；

dec\_result：解码成功标志；

data\_size：接收数据大小；

phy\_p：这次传输的物理资源信息。

**算法与处理流程**

/\*判断是否解码成功\*/

If (**dec\_result** == DEC\_SUCC) {

ulproc\_p->retran\_ind = 0;

ulproc\_p->tb\_size = 0;

(ulproc\_p->ResourcePhyInfo).rb\_start = 0;

(ulproc\_p->ResourcePhyInfo).rb\_num = 0;

(ulproc\_p->ResourcePhyInfo).mcs = 0;

(ulproc\_p->ResourcePhyInfo).rv = 0;

} else {

current\_tx\_num++;

if (current == **ul\_max\_harq\_tx** - 1){

ulproc\_p->retran\_ind = 0;

ulproc\_p->tb\_size = 0;

(ulproc\_p->ResourcePhyInfo).rb\_start = 0;

(ulproc\_p->ResourcePhyInfo).rb\_num = 0;

(ulproc\_p->ResourcePhyInfo).mcs = 0;

(ulproc\_p->ResourcePhyInfo).rv = 0;

} else if (current < ul\_max\_harq\_tx - 1) {

ulproc\_p->retran\_ind = 1;

ulproc\_p->tb\_size = **data\_size**;

(ulproc\_p->ResourcePhyInfo).rb\_start = **phy\_p**->rb\_start;

(ulproc\_p->ResourcePhyInfo).rb\_num = **phy\_p**->rb\_num;

(ulproc\_p->ResourcePhyInfo).mcs = **phy\_p**->mcs;

(ulproc\_p->ResourcePhyInfo).rv = **phy\_p**->rv;

} else {

/\*出错\*/

}

/\*建立ACK/NACK链表，并添加ACK/NACK节点\*/

g\_tx\_ack\_lst\_p = (AckLst \*) malloc (sizeof (AckLst));

init\_list ((ListType \*) g\_tx\_ack\_lst\_p);

acknode\_p->acktype = (decode\_result == DEC\_SUCC)? ACK: NACK;

acknode\_p->sub\_fn = sub\_fn;

acknode\_p->sys\_fn = sys\_fn;

add\_list ((ListType \*) g\_tx\_ack\_lst\_p, (NodeType \*) acknode\_p);

}

**输出：**

无

**返回:**

0/1，操作是否成功。

### INT32 get\_next\_rv (UINT8 current\_rv, UINT8 \*next\_rv\_p)

**功能描述**：

根据当前rv获得下一个重传时所需的rv。

**输入**：

current\_rv：当前RV

**算法与处理流程**

if (0 == current\_rv) {

\*next\_rv\_p = 2;

} else if (2 == current\_rv) {

\*next\_rv\_p = 3;

} else if (3 == current\_rv) {

\*next\_rv\_p = 1;

} else {

\*next\_rv\_p = 0;

}

**输出**：

next\_rv：下个RV

**返回**：

0/1，操作是否成功。

### INT32 get\_ulproc\_id(UINT32 sys\_fn, UINT8 sub\_fn，UINT8 \*proc\_id\_p)

**功能描述**：

获得所需进程号

**输入**：

sys\_fn: 系统帧号

sub\_fn: 系统子帧号

**算法与处理流程**

根据上行子帧号和TDD配置得到相应进程号。

**输出:**

proc\_id\_p：对应进程ID号

**返回:**

0/1，操作是否成功。

### INT32 rcv\_ack\_nack (AckNode \* ack\_node\_p)

**功能描述**：

接收到ACK/NACK的HARQ处理。

**输入**：

ack\_node\_p: ACK节点

**算法与处理流程**

/\*根据RNTI值得到对应RNTI实体并找到指向第一个进程的指针\*/

get\_rnti\_entity (ack\_node\_p->rnti);

dlharq\_p = (DlHarqInfo \*) rnti\_p->dl\_harq\_info\_p;

dlproc\_p = (DlProcInfo \*) dlharq\_p->dlproc\_p;

/\*根据node\_p中的帧号子帧号信息找到下行HARQ进程\*/

for (i = 0; i<进程总数; i++) {

if (发送数据子帧号与帧号匹配) {

break;

}

dlproc\_p++;

}

if (dlproc\_p->acktype == ACK) {

/\*收到ACK将该进程置为IDLE状态\*/

set\_dlproc\_idle();

/\*若成功，空闲进程增一\*/

dlharq\_p->idle\_proc\_num++;

}else{/\*收到NACK消息\*/

if（current\_tx\_num >= g\_max\_tx\_num）{

/\*达到重传最大次数，将该进程置为IDLE状态\*/

set\_dlproc\_idle ();

/\*若成功，空闲进程增一\*/

dlharq\_p->idle\_proc\_num++;

} else {

/\*将该进程置为需要重传的状态\*/

set\_dlproc\_state ();

/\*若成功，重传进程增一\*/

dlharq\_p->retran\_proc\_num++;

dlharq\_p->retran\_tb\_size =+ dlproc\_p->tb\_size;

}

}

**输出**：

无

**返回**：

0/1，操作是否成功。

### INT32 do\_newtran\_dlharq (DlHarqInfo \*dlharq\_p, UINT32 tb\_size, UINT8 \*pdu\_p, TbHarqParam \*tbharqparam\_p)

**函数描述**

下行新传HARQ处理。

**输入**

dlharq\_p：指向下行HARQ实体的指针；

tb\_size：数据大小；

pdulst\_p：指向新传数据的指针；

**算法与处理流程**

/\*计算空闲进程ID号\*/

get\_idle\_dlproc\_id()；

/\*若成功空闲的进程数少一\*/

dlharq\_p->idle\_proc\_num--;

/\*得到进程指针\*/

dlproc\_p = (DlProcInfo \*) harq\_p->dlproc\_p + process\_id;

dlproc\_p->tb\_size = tb\_size；//记录新传TB大小

dlproc\_p->tx\_time = 当前时间;

dlproc\_p->current\_tx\_num++;//发送次数增加

dlproc\_p->rv = 0;//得到新传RV值

dlproc\_p->state = 1;//设为等待ACK状态

开启等待ACK计时器

将新传数据缓存在HARQ buffer中

**输出**

tbharqparam\_p：HARQ信息。

**返回**

0/1，操作是否成功。

### INT32 do\_retran\_dlharq (DlHarqInfo \*dlharq\_p, UINT8 \*pdu\_p, TbHarqParam \*tbharqparam\_p)

**函数描述**

下行重传HARQ处理。

**输入**

dlharq\_p：指向下行HARQ实体的指针；

pdulst\_p：指向新传数据的指针；

**算法与处理流程**

/\*计算重传进程ID号，得到进程指针\*/

get\_dlproc\_id()；

/\*若成功重传的进程数少一\*/

dlharq\_p->retran\_proc\_num- -；

dlproc\_p = (DlProcInfo \*) harq\_p->dlproc\_p + process\_id;

dlproc\_p->tx\_time = 当前时间;

dlproc\_p->current\_tx\_num++;//发送次数增加

dlproc\_p->rv = get\_next\_rv();//得到重传RV值

dlproc\_p->state = 1;//设为等待ACK状态

开启等待ACK计时器

**输出**

pdulst\_p：指向重传数据的指针；

tbharqparam\_p：HARQ信息。

**返回**

0/1，操作是否成功。

### INT32 get\_dlproc\_id (DlProcInfo \*dlproc\_p，UINT8 \*dlproc\_id\_p，UINT8 state)

**函数描述**

得到空闲的下行HARQ进程ID，或者得到等待最久的下行HARQ重传进程ID。

**输入**

dlproc\_p：指向第一个下行HARQ进程的指针；

state: 重传调度状态，或空闲状态。

**算法与处理流程**

if (state = = 空闲的下行HARQ) {

for (i = 0; i<进程总数; i++) {

if (dlproc\_p->state = = IDLE) {

\*dlproc\_id\_p = i;

break;

}

dlproc\_p++;

}

}

if (state = = 重传的下行HARQ) {

temp\_time = ffff;

for (i = 0; i<进程总数; i++) {

if (dlproc\_p->state = = need\_retran) {

if (temp\_time >= dlproc\_p->dl\_tx\_time){

temp\_time = dlproc\_p->dl\_tx\_time;

retran\_id = i;

}

}

dlproc\_p++;

}

\*dlproc\_id\_p=retran\_id;

}

**输出**

dlproc\_id\_p：指向空闲的下行HARQ进程ID的指针,或指向等待最久的下行HARQ进程ID的指针。

**返回**

0/1，操作是否成功。

# 6 接口设计

## 6.1用户接口

## 6.2硬件接口

## 6.3软件接口

### INT32 do\_ulharq (UINT16 rnti, INT8 dec\_result, UINT32 data\_size, ResourcePhyInfo \*phy\_p)

**函数描述**

接收到上行数据通知上行HARQ实体。根据RNTI值找到该UE的上行HARQ进程后，若解码成功回复ACK，不成功回复NACK，并在该上行HARQ进程中标明等待重传。

**输入**

rnit： RNTI值；

dec\_result：解码成功标志；

data\_size：接收数据大小；

phy\_p:这次传输的物理资源信息。

**算法与处理流程**

/\*根据RNTI索引到RNTI实体rnti\_p\*/

UeRntiMsg \*get\_rnti\_entity (UINT16 rnti)

/\*根据当前子帧号和帧号得到该上行进程号process\_id\*/

get\_ulproc\_id ();

/\*根据上行进程号process\_id找到上行进程ulproc\_p\*/

ulproc\_p = (rnti\_p->ul\_harq\_info\_p) ->ulproc\_p + process\_id;

/\*对这个上行HARQ进程进行处理\*/

do\_ulharq\_process ()

**输出**

无

**返回**

0/1，操作是否成功。

### INT32 process\_rcv\_ack (AckLst \*ack\_lst\_p)

**函数描述:**

接收到ACK/NACK list通知相应下行HARQ实体。根据RNTI值找到该UE, 再根据收到ACK/NACK的帧号和子帧号找到下行HARQ进程后收到NACK和ACK做不同处理，即NACK情况时标明需重传，ACK情况说明下行发送成功。

**输入:**

AckLst \*ack\_lst\_p：是指向ACK/NACK list的指针。

**算法与处理流程**

/\*遍历AckLst中的每个节点\*/

node\_p = (UlAckNode \*) get\_list ((ListType \*) acklst\_p);

While (node\_p) {

/\*接收到ACK/NACK消息，做HARQ处理\*/

rcv\_ack\_nack();

node\_p = (AckNode \*) get\_list ((ListType \*) acklst\_p);

}

**输出:**

无

**返回:**

0/1，操作是否成功。

### INT32 do\_dlharq (UINT16 rnti, UINT8 ndi, UINT32 tb\_size, UINT8 \*pdu\_p, TbHarqParam \*tbharqparam\_p)

**函数描述**

发送下行数据做下行HARQ处理。根据RNTI值找到该UE的下行HARQ进程后，根据新传指示来判断是新传数据还是重传数据。在记录HARQ相关信息。

**输入**

rnit： RNTI值；

ndi：新传标志；

tb\_size：数据大小；

pdu\_p：新传数据。

**算法与处理流程**

/\*根据RNTI索引到RNTI实体rnti\_p\*/

get\_rnti\_entity()

/\*得到下行HARQ进程指针\*/

dlproc\_p = (DlHarqInfo \*)rnti\_p->dl\_harq\_info\_p;

/\*根据下行数据是新传还是旧传来找到相应的进程ID号\*/

If (ndi == NEW\_TRAN) {

do\_newtran\_dlharq ();

} elseif (ndi == RE\_TRAN) {

do\_retran\_dlharq ();

}

**输出**

pdu\_p：返回重传数据

tbharqparam\_p：HARQ信息

**返回**

0/1，操作是否成功。

### INT32 get\_tx\_acklst (AckLst \*ack\_lst)

**函数描述**

得到一个在PHICH发送ACK/NACK的链表。

**输入**

无

**输出**

得到一个在PHICH发送ACK/NACK的链表。第一层是RNTI，第二层是HARQ进程。

**返回**

0/1，操作是否成功。

### INT32 get\_ulharq\_sched\_info (UINT16 rnti, UINT8 \*retran\_ind, UINT32 \*tb\_size)

**功能描述**：

提供给上行调度，某一RNTI，当前上行HARQ情况。

**输入**：

rnti,：RNTI值。

**输出**：

retran\_ind：重传请求标志

tb\_size：重传tb大小

**返回**：

-1，0

### INT32 get\_dlharq\_sched\_info (UINT16 rnti, UINT8 \*state, UINT32 \*tb\_size)

**功能描述**：

提供给上行调度，某一RNTI，当前上行HARQ情况。

**输入**：

rnti,：RNTI值。

**输出**：

state：下行HARQ进程状态

tb\_size：重传tb大小

**返回**：

-1，0

### INT32 config\_ue\_harq (UINT16 rnti, HarqParam harq\_param)

**函数描述**

基站记录为每一个UE的HARQ配置。

**输入**

rnit： RNTI值；

harq\_param：HARQ配置信息（上下行最大重传次数，计时器等）。

**输出**

无

**返回**

0/1，操作是否成功。

### INT32 init\_ue\_harq (UINT16 rnti, HarqParam harq\_param)

**函数描述**

初始化UE的HARQ实体。

**输入**

rnit： RNTI值；

harq\_param：HARQ配置信息（上下行最大重传次数，计时器等）。

**输出**

无

**返回**

0/1，操作是否成功。

### INT32 delete\_ue\_harq (UINT16 rnti)

**函数描述**

销毁UE的HARQ实体。

**输入**

rnit： RNTI值；

**输出**

无

**返回**

0/1，操作是否成功。

# 7 功能测试设计

## 7.1 接收ACK/NACK功能测试

**功能描述：**

接收到ACK/NACK list通知相应下行HARQ实体。根据RNTI值找到该UE, 再根据收到ACK/NACK的帧号和子帧号找到下行HARQ进程后收到NACK和ACK做不同处理，即NACK情况时标明需重传，ACK情况说明下行发送成功。

**所涉及函数及调用关系**



**测试设计**

|  |  |  |
| --- | --- | --- |
| **测试编号** | **测试内容** | |
| 测试1 | 测试说明： | 收到ack节点，通过子帧数和帧数找到相应进程。设置相应dl harq进程为空闲状态。 |
| 测试输入： | rnti = 0x003d，acktype=ACK |
| 预期输出： | Ack节点对应进程成功设为空闲进程。 |
| 测试2 | 测试说明： | 收到nack节点，通过子帧数和帧数找到相应进程。未超最大重传次数时，设置相应dl harq进程为重传状态 |
| 测试输入： | rnti = 0x003d，acktype=NACK |
| 预期输出： | nack节点对应进程成功设为重传进程。 |
| 测试3 | 测试说明： | 收到nack节点，通过子帧数和帧数找到相应进程。超过最大重传次数时，设置相应dl harq进程为空闲状态 |
| 测试输入： | rnti = 0x003d，acktype=NACK |
| 预期输出： | Nack节点对应进程设为空闲进程。 |
| 测试4 | 测试说明： | 收到nack节点，通过子帧数和帧数不能找到相应进程 |
| 测试输入： | rnti = 0x003d，acktype=NACK |
| 预期输出： | 显示出错信息。 |

## 7.2下行HARQ处理功能测试

**功能描述：**

MAC发送模块在发送PDSCH数据时做HARQ处理。根据RNTI值找到该UE的下行HARQ进程后，根据新传指示来判断是新传数据还是重传数据，新传则进行下行新传处理，重传则进行下行重传处理。

**所涉及函数及调用关系**



**测试设计**

|  |  |  |
| --- | --- | --- |
| **测试编号** | **测试内容** | |
| 测试1 | 测试说明： | 收到新传标志，选择空闲进程进行新传处理。 |
| 测试输入： | ndi=1，rnti=00x003d对应五个进程：两个空闲，三个重传。 |
| 预期输出： | 找到空闲进程进行新传处理。 |
| 测试2 | 测试说明： | 收到重传标志，选择重传进程进行重传处理。 |
| 测试输入： | ndi=0，rnti=00x003d对应五个进程：两个空闲，三个重传。 |
| 预期输出： | 找到等待时间最久的重传进程进行重传处理。 |

## 7.3上行HARQ处理功能测试

**功能描述：**

MAC接收模块收到PUSCH上行数据，调用接口函数do\_ulharq()进行上行HARQ处理。根据RNTI值找到该UE的上行HARQ进程后，若解码成功回复ACK，不成功回复NACK，并在该上行HARQ进程中标明等待重传。

**所涉及函数及调用关系**



**测试设计**

|  |  |  |
| --- | --- | --- |
| **测试编号** | **测试内容** | |
| 测试1 | 测试说明： | TB解码成功未标注重传 |
| 测试输入： | dec\_result=DEC\_SUCC，四个不同情况进程。 |
| 预期输出： | 回复ACK。 |
| 测试2 | 测试说明： | TB解码失败未标注重传且传输次数大于最大重传次数情况。 |
| 测试输入： | dec\_result=DEC\_FAIL，四个不同情况进程。 |
| 预期输出： | 回复NACK，设进程为等待重传状态。 |
| 测试3 | 测试说明： | TB解码成功标注重传。 |
| 测试输入： | dec\_result=DEC\_SUCC，四个不同情况进程。 |
| 预期输出： | 回复ACK。 |

# 8 其他