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

**TD-LTE HeNB协议栈软件系统**

**MAC\_HARQ模块**

**单元测试报告**

拟制：李亚楠

时间：2011-02-22

**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 | 李亚楠 | 2011.2.22 | 无 | 无 |

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。

[1 引言 1](#_Toc291494752)

[1.1 背景 1](#_Toc291494753)

[1.2 定义 1](#_Toc291494754)

[1.3 参考资料 1](#_Toc291494755)

[2 测试模块 2](#_Toc291494756)

[2.1 模块名 2](#_Toc291494757)

[2.2 文件名 2](#_Toc291494758)

[2.3 测试人 2](#_Toc291494759)

[2.4 测试用例列表 2](#_Toc291494760)

[3 函数测试 4](#_Toc291494761)

[3.1 被测函数：INT32 get\_next\_rv (UINT8 current\_rv, UINT8 \*next\_rv\_p) 4](#_Toc291494762)

[3.1.1 函数描述： 4](#_Toc291494763)

[3.1.2 测试用例： 5](#_Toc291494764)

[3.2 被测函数：INT32 get\_ulproc\_id(UINT32 sys\_fn, UINT8 sub\_fn，UINT8 \*proc\_id\_p) 6](#_Toc291494765)

[3.2.1 函数描述： 6](#_Toc291494766)

[3.2.2 测试用例： 6](#_Toc291494767)

[3.3 被测函数：INT32 get\_dlproc\_id (DlProcInfo \*dlproc\_p，UINT8 \*dlproc\_id\_p，UINT8 state) 7](#_Toc291494768)

[3.3.1 函数描述： 7](#_Toc291494769)

[3.3.2 测试用例： 8](#_Toc291494770)

[3.4 被测函数：INT32 set\_dlproc\_state (DlProcInfo \*dlproc\_p, UINT8 state) 12](#_Toc291494771)

[3.4.1 函数描述： 12](#_Toc291494772)

[3.4.2 测试用例： 12](#_Toc291494773)

[3.5 被测函数：INT32 process\_rcv\_ack(AckLst \*ack\_lst\_p) 15](#_Toc291494774)

[3.5.1 函数描述： 15](#_Toc291494775)

[3.5.2 测试用例： 16](#_Toc291494776)

[3.6 被测函数：INT32 do\_dlharq (UINT16 rnti, UINT8 ndi, UINT32 tb\_size, PduInfo4HarqLst \*pdu\_p, TbHarqParam \*tbharqparam\_p) 17](#_Toc291494777)

[3.6.1 函数描述： 17](#_Toc291494778)

[3.6.2 测试用例： 18](#_Toc291494779)

[3.7被测函数：INT32 do\_ulharq (UINT16 rnti, INT8 dec\_result, UINT32 data\_size, ResourcePhyInfo \*phy\_p) 19](#_Toc291494780)

[3.7.1 函数描述： 19](#_Toc291494781)

[3.7.2试用例： 19](#_Toc291494782)

[3.8测函数：INT32 do\_ulharq\_process (UlProcInfo \*ulproc\_p, UINT8 ul\_max\_harq\_tx, INT8 dec\_result, UINT32 data\_size, ResourcePhyInfo \*phy\_p 20](#_Toc291494783)

[3.8.1 函数描述： 20](#_Toc291494784)

[3.8.2 测试用例： 21](#_Toc291494785)

[3.9被测函数： INT32 do\_newtran\_dlharq (DlHarqInfo \*dlharq\_p, UINT32 tb\_size, PduInfo4HarqLst \*pdulst\_p, TbHarqParam \*tbharqparam\_p) 32](#_Toc291494786)

[3.9.1 函数描述： 32](#_Toc291494787)

[3.9.2 测试用例： 33](#_Toc291494788)

[3.10被测函数：INT32 do\_retran\_dlharq (DlHarqInfo \*dlharq\_p, PduInfo4HarqLst \*pdulst\_p, TbHarqParam \*tbharqparam\_p) 34](#_Toc291494789)

[3.10.2 测试用例： 35](#_Toc291494790)

[3.11被测函数：INT32 delete\_ue\_harq (UINT16 rnti) 35](#_Toc291494791)

[3.11.1 函数描述： 35](#_Toc291494792)

[3.11.2 测试用例： 36](#_Toc291494793)

[3.12 被测函数：INT32 rcv\_ack\_nack (AckNode \* ack\_node\_p) 37](#_Toc291494794)

[3.12.1 函数描述： 37](#_Toc291494795)

[3.12.2 测试用例： 38](#_Toc291494796)

[4 功能测试 1](#_Toc291494797)

[4.1 接收ACK/NACK功能测试 1](#_Toc291494798)

[4.1.1 功能描述 1](#_Toc291494799)

[4.1.2 所涉及函数及调用关系 1](#_Toc291494800)

[4.1.3 测试内容 1](#_Toc291494801)

[4.1.4 测试用例： 1](#_Toc291494802)

[4.2下行HARQ处理功能测试 4](#_Toc291494803)

[4.2.1 功能描述 4](#_Toc291494804)

[4.2.2 所涉及函数及调用关系 4](#_Toc291494805)

[4.2.3 测试内容 4](#_Toc291494806)

[4.2.4 测试用例： 4](#_Toc291494807)

[4.3上行HARQ处理功能测试 7](#_Toc291494808)

[4.3.1 功能描述 7](#_Toc291494809)

[4.3.2 所涉及函数及调用关系 7](#_Toc291494810)

[4.3.3 测试内容 7](#_Toc291494811)

[4.3.4 测试用例： 7](#_Toc291494812)

[5 其他测试 1](#_Toc291494813)

[6 附录1 测试类型 1](#_Toc291494814)

[1. 等价类划分 1](#_Toc291494815)

[2. 边值分析法 2](#_Toc291494816)

[3. 语句覆盖 3](#_Toc291494817)

[4. 判定覆盖 3](#_Toc291494818)

[5. 条件覆盖 4](#_Toc291494819)

[6. 判定／条件覆盖 4](#_Toc291494820)

[7. 条件组合覆盖 5](#_Toc291494821)

**单元测试报告**

# 1 引言

## 1.1 背景

1. 软件系统：TD-LTE HeNB协议栈软件系统
2. 开发环境：
   1. 开发工具： Vim
   2. 编程语言： C语言
   3. 编译器： GCC
   4. 调试器： GDB
3. 操作系统：目标模块所运行的操作系统Linux
4. 测试环境：
5. 运行平台： PC-Linux
6. CPU类型：AMD Athlon（tm）Processor，2.71G
7. 内存大小：960M

## 1.2 定义

无

## 1.3 参考资料

1. 《LTE-HeNB协议栈软件系统MAC HARQ模块概要设计说明书\_v1.0》；
2. 《LTE-HeNB协议栈软件系统MAC HARQ模块详细设计说明书\_v1.0》。

# 2 测试模块

## 2.1 模块名

MAC\_HARQ模块

## 2.2 文件名

harq.c

harq.h

## 2.3 测试人

李亚楠

[liyanan01@ict.ac.cn](mailto:liyanan01@ict.ac.cn)。

## 2.4 测试用例列表

|  |  |
| --- | --- |
| **测试用例** | **测试说明** |
| HeNB\_HARQ\_1 | get\_next\_rv（） |
| HeNB\_HARQ\_2 | get\_ulproc\_id（） |
| HeNB\_HARQ\_3 | get\_dlproc\_id（） |
| HeNB\_HARQ\_4 | get\_dlproc\_id（） |
| HeNB\_HARQ\_5 | get\_dlproc\_id（） |
| HeNB\_HARQ\_6 | set\_dlproc\_state () |
| HeNB\_HARQ\_7 | set\_dlproc\_state（） |
| HeNB\_HARQ\_8 | set\_dlproc\_state（） |
| HeNB\_HARQ\_9 | process\_rcv\_ack（） |
| HeNB\_HARQ\_10 | do\_dlharq（） |
| HeNB\_HARQ\_11 | do\_ulharq（） |
| HeNB\_HARQ\_12 | do\_ulharq\_process（） |
| HeNB\_HARQ\_13 | do\_ulharq\_process（） |
| HeNB\_HARQ\_14 | do\_ulharq\_process（） |
| HeNB\_HARQ\_15 | do\_ulharq\_process（） |
| HeNB\_HARQ\_16 | do\_ulharq\_process（） |
| HeNB\_HARQ\_17 | do\_ulharq\_process（） |
| HeNB\_HARQ\_18 | do\_ulharq\_process（） |
| HeNB\_HARQ\_19 | do\_ulharq\_process（） |
| HeNB\_HARQ\_20 | do\_ulharq\_process（） |
| HeNB\_HARQ\_21 | do\_newtran\_dlharq（） |
| HeNB\_HARQ\_22 | do\_retran\_dlharq（） |
| HeNB\_HARQ\_23 | delete\_ue\_harq（） |
| HeNB\_HARQ\_24 | rcv\_ack\_nack（） |
| HeNB\_HARQ\_25 | 功能测试 接收ACK/NACK |
| HeNB\_HARQ\_26 | 功能测试DL HARQ 处理 |
| HeNB\_HARQ\_27 | 功能测试 UL HARQ 处理 |

# 3 函数测试

|  |  |
| --- | --- |
| **被测函数** | **函数说明** |
| get\_next\_rv（） | 根据当前rv获得下一个重传时所需的rv。 |
| get\_ulproc\_id（） | 获得所需进程号。 |
| get\_dlproc\_id（） | 得到空闲的下行进程ID，或者得到等待最久的下行重传进程ID。 |
| set\_dlproc\_state () | 将该下行进程设置为需要重传调度状态，或空闲状态。 |
| process\_rcv\_ack（） | 根据ack/nack链表，找到相应进程并根据收到的反馈进行相应处理。 |
| do\_ulharq（） | 收到上行数据找到相应进程，根据解码结果进行相应反馈。 |
| do\_ulharq\_process（） | 根据解码结果进行反馈处理。 |
| do\_newtran\_dlharq（） | 下行新传HARQ处理 |
| do\_retran\_dlharq（） | 下行重传HARQ处理 |
| delete\_ue\_harq（） | 销毁UE的HARQ实体。 |
| rcv\_ack\_nack（） | 接收到ACK/NACK的HARQ处理。 |

## 3.1 被测函数：INT32 get\_next\_rv (UINT8 current\_rv, UINT8 \*next\_rv\_p)

### 3.1.1 函数描述：

**功能**

根据当前rv获得下一个重传时所需的rv。

**输入**：

current\_rv：当前RV。

**输出**

next\_rv：下一rv值。

**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;

}

### 3.1.2 测试用例：

#### 测试用例：HeNB\_HARQ\_1

|  |  |
| --- | --- |
| **说明** | current\_rv分别为0，1,2,3时，获得正确next\_rv值。 |
| **输入** | current\_rv =0，current\_rv =1，current\_rv =2，current\_rv =3。 |
| **预期输出** | next\_rv = 2，next\_rv = 0，next\_rv =3，next\_rv = 1. |

**测试修改**

无

**测试方法**

get\_next\_rv (0, &next\_rv);

get\_next\_rv (1, &next\_rv);

get\_next\_rv (2, &next\_rv);

get\_next\_rv (3, &next\_rv);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 1  
when current\_rv=0, the next rv is 2  
when current\_rv=1, the next rv is 0  
when current\_rv=2, the next rv is 3  
when current\_rv=3, the next rv is 1  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.2 被测函数：INT32 get\_ulproc\_id(UINT32 sys\_fn, UINT8 sub\_fn，UINT8 \*proc\_id\_p)

### 3.2.1 函数描述：

**函数描述**

**描述**

获得所需进程号

**输入**：

sys\_fn: 系统帧号

sub\_fn: 系统子帧号

**输出**

proc\_id\_p：对应进程ID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。

### 3.2.2 测试用例：

#### 测试用例：HeNB\_HARQ\_2

|  |  |
| --- | --- |
| **说明** | TDD配置为2时，输入子帧0,1,2 ～，9。获得可能的ulproc\_id. |
| **输入** | sub\_fn = 0,1,2,3,4,5,6,7,8,9. sys\_fn = 1. |
| **预期输出** | sub\_fn = 2时，ulproc\_id=0；sub\_fn =7时，ulproc\_id=1； |

**测试修改**

无

**测试方法**

For (i=0, i<10, i++) {

get\_ulproc\_id (1, i, &ulproc\_id);

Printf ("when sub\_fn=%d, the ulproc\_id is %d\n", i, ulproc\_id);

}

**测试输出**

**##################################  
#### HARQ.c test start ####  
##################################  
the test case is: 2  
Illegal UL sub-frame number!  
Illegal UL sub-frame number!  
When sub\_fn=2, the ulproc\_id is 0  
Illegal UL sub-frame number!  
Illegal UL sub-frame number!  
Illegal UL sub-frame number!  
Illegal UL sub-frame number!  
When sub\_fn=7, the ulproc\_id is 1  
Illegal UL sub-frame number!  
Illegal UL sub-frame number!  
##################################  
#### HARQ.c test end ####  
##################################**

**BUG描述**

无

## 3.3 被测函数：INT32 get\_dlproc\_id (DlProcInfo \*dlproc\_p，UINT8 \*dlproc\_id\_p，UINT8 state)

### 3.3.1 函数描述：

**函数描述**

**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。

**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。

**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;

}

### 3.3.2 测试用例：

|  |  |  |
| --- | --- | --- |
| **测试用例** | **测试类型** | **测试说明** |
| HeNB\_HARQ\_3 | 等价类划分 | 输入参数错误情况 |
| HeNB\_HARQ\_4 | 等价类划分 | State=0，寻找空闲进程ID |
| HeNB\_HARQ\_5 | 等价类划分 | State=2，寻找重传进程ID |

#### 测试用例：HeNB\_HARQ\_3

|  |  |
| --- | --- |
| **说明** | 输入参数错误情况 |
| **输入** | dlharq\_p=NULL |
| **预期输出** | 显示信息“Invalid input parameter dlharq\_p” |

**测试修改**

无

**测试方法**

dlharq\_p = NULL;

state = 0;

get\_dlproc\_id (dlharq\_p, &dlproc\_id, state);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 3  
Invalid input parameter dlharq\_p!  
##################################  
####     HARQ.c test end      ####  
##################################**

#### 测试用例：HeNB\_HARQ\_4

|  |  |
| --- | --- |
| **说明** | 当输入State = 0时，寻找空闲进程ID |
| **输入** | State = 0 |
| **预期输出** | 空闲进程ID |

**测试修改**

无

**测试方法**

dlharq\_p= (DlHarqInfo \*) lte\_malloc (sizeof (DlHarqInfo));

if (dlharq\_p == NULL) {

printf("Not Enough Memory!\n");

return -1;

}

Memset (dlharq\_p, 0, sizeof (DlHarqInfo));

dlharq\_p->total\_proc\_num = 5;

dlharq\_p->dlproc\_p =

(DlProcInfo \*) lte\_malloc (5 \* sizeof (DlProcInfo));

if (dlharq\_p->dlproc\_p == NULL) {

printf("Not Enough Memory!\n");

return -1;

}

Memset (dlharq\_p->dlproc\_p, 0, 5 \* sizeof(DlProcInfo));

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

for (m = 0; m < 5; m++) {

dlproc\_p->dl\_tx\_time = 2;

dlproc\_p->state = (m % 2 == 0)? 0: 2;

dlproc\_p++;

} /\* for \*/

state = 0;

get\_dlproc\_id (dlharq\_p, &dlproc\_id, state);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 4  
the total process num is 5  
proc 0:,dlproc\_p->state is 2  
proc 1:,dlproc\_p->state is 2  
proc 2:,dlproc\_p->state is 2  
proc 3:,dlproc\_p->state is 0  
proc 4:,dlproc\_p->state is 2  
the idle dlproc\_id is 3  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_5

|  |  |
| --- | --- |
| **说明** | 当输入State = 2时，寻找重传进程ID |
| **输入** | State = 2 |
| **预期输出** | 重传进程ID |

**测试修改**

无

**测试方法**

dlharq\_p= (DlHarqInfo \*) lte\_malloc (sizeof (DlHarqInfo));

if (dlharq\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

Memset (dlharq\_p, 0, sizeof (DlHarqInfo));

dlharq\_p->total\_proc\_num = 3;

dlharq\_p->dlproc\_p =

(DlProcInfo \*) lte\_malloc (3 \* sizeof (DlProcInfo));

if (dlharq\_p->dlproc\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

Memset (dlharq\_p->dlproc\_p, 0, 3 \* sizeof (DlProcInfo));

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

for (m = 0; m < 3; m++) {

dlproc\_p->dl\_tx\_time = m + 2;

dlproc\_p->state = (m % 2 == 0)? 0: 2;

dlproc\_p++;

} /\* for \*/

state = 2;

get\_dlproc\_id (dlharq\_p, &dlproc\_id, state);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 5  
the total process num is 5  
proc 0:,state is 2,dl\_tx\_time is 55  
proc 1:,state is 2,dl\_tx\_time is 49  
proc 2:,state is 2,dl\_tx\_time is 50  
proc 3:,state is 2,dl\_tx\_time is 52  
proc 4:,state is 2,dl\_tx\_time is 48  
the retran dlproc\_id is 4  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.4 被测函数：INT32 set\_dlproc\_state (DlProcInfo \*dlproc\_p, UINT8 state)

### 3.4.1 函数描述：

**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。

**输出**

无

**算法与处理流程**

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。

### 3.4.2 测试用例：

|  |  |  |
| --- | --- | --- |
| **测试用例** | **测试类型** | **测试说明** |
| HeNB\_HARQ\_6 | 等价类划分 | 输入参数错误情况 |
| HeNB\_HARQ\_7 | 等价类划分 | state =IDLE |
| HeNB\_HARQ\_8 | 等价类划分 | state =Retransmission |

#### 测试用例：HeNB\_HARQ\_6

|  |  |
| --- | --- |
| **说明** | 输入参数错误时，显示错误信息。 |
| **输入** | dlproc\_p=NULL |
| **预期输出** | 显示错误信息“invalid input parameter dlproc\_p” |

**测试修改**

无

**测试方法**

proc\_p = NULL;

state = 0;

set\_dlproc\_state (proc\_p, state);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 6  
invalid input parameter dlproc\_p!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_7

|  |  |
| --- | --- |
| **说明** | 当收到置idle状态输入时，可成功将进程置空。 |
| **输入** | State = 0。 |
| **预期输出** | 置空的进程信息。 |

**测试修改**

无

**测试方法**

proc\_p = (DlProcInfo \*) lte\_malloc (3 \* sizeof (DlProcInfo));

if (proc\_p == NULL) {

printf("Not Enough Memory!\n");

return -1;

}

Memset (proc\_p, 0, 3 \* sizeof (DlProcInfo));

for (m = 0; m < 3; m++) {

proc\_p->dl\_tx\_time = m + 2;

proc\_p->state = (m % 2 == 0)? 0: 2;

proc\_p->tb\_size = 128;

proc\_p->current\_tx\_num = 1;

proc\_p->rv = 2;

proc\_p->pdulst\_p = construct\_Pdulst (&pdu\_lst\_p);

proc\_p++;

} /\* for \*/

state = 0;

set\_dlproc\_state (proc\_p, state);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 7  
proc\_state:2, current\_tx\_num:1  
\*\*\*\*start to set proc state\*\*\*\*  
dlproc\_p->state:0  
current\_tx\_num:0  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_8

|  |  |
| --- | --- |
| **说明** | 当收到置重传状态输入时，可将进程置为重传状态。 |
| **输入** | State = 2 |
| **预期输出** | 置为重传的进程信息。 |

**测试修改**

无

**测试方法**

proc\_p = (DlProcInfo \*) lte\_malloc (2 \* sizeof (DlProcInfo));

if (proc\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

Memset (proc\_p, 0, 2 \* sizeof (DlProcInfo));

for (m = 0; m < 2; m++) {

proc\_p->dl\_tx\_time = m + 2;

proc\_p->state = (m % 2 == 0)? 0: 2;

proc\_p->tb\_size = 128;

proc\_p->current\_tx\_num = 1;

proc\_p->rv = 2;

proc\_p->pdulst\_p = construct\_Pdulst (&pdu\_lst\_p);

Proc\_p++;

} /\* for \*/

state = 2;

set\_dlproc\_state (proc\_p, state);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 8  
proc\_state:0, rv: 2  
\*\*\*\*start to set proc state\*\*\*\*  
next\_rv:3, dlproc\_p->state: 2  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.5 被测函数：INT32 process\_rcv\_ack(AckLst \*ack\_lst\_p)

### 3.5.1 函数描述：

**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。

**输出**

无

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

}

### 3.5.2 测试用例：

#### 测试用例：HeNB\_HARQ\_9

|  |  |
| --- | --- |
| **说明** | 当输入ack\_lst 为空 |
| **输入** | ack\_lst\_p=NULL |
| **预期输出** | 不能成功接收反馈进行HARQ处理。 |

**测试修改**

无

**测试方法**

ack\_lst\_p = NULL;

process\_rcv\_ack (ack\_lst\_p);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 9  
invalid input parameter ack\_lst\_p!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.6 被测函数：INT32 do\_dlharq (UINT16 rnti, UINT8 ndi, UINT32 tb\_size, PduInfo4HarqLst \*pdu\_p, TbHarqParam \*tbharqparam\_p)

### 3.6.1 函数描述：

**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。

**输出**

pdu\_p：返回重传数据

tbharqparam\_p：HARQ信息

**算法与处理流程**

/\*根据RNTI索引到RNTI实体rnti\_p\*/

get\_rnti\_entity(UINT16 rnti)；

/\*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 ();

}

### 3.6.2 测试用例：

#### 测试用例：HeNB\_HARQ\_10

|  |  |
| --- | --- |
| **说明** | 当输入错误参数时，显示错误信息。 |
| **输入** | rnti = 0x002A，Incorrect rnti。pdu\_p = NULL; tbharqparam\_p = NULL; |
| **预期输出** | 不能进行dl harq。 |

**测试修改**

无

**测试方法**

rnti = 0x002A;

ndi = 0;

tb\_size = 0;

pdu\_p = NULL;

tbharqparam\_p = NULL;

do\_dlharq (rnti, ndi, tb\_size, pdu\_p, tbharqparam\_p);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 10  
Invalid input parameter pdu\_p!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.7被测函数：INT32 do\_ulharq (UINT16 rnti, INT8 dec\_result, UINT32 data\_size, ResourcePhyInfo \*phy\_p)

### 3.7.1 函数描述：

**功能**

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

**输出**

无

**算法与处理流程**

/\*根据RNTI索引到RNTI实体rnti\_p\*/

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 ()

### 3.7.2试用例：

#### 测试用例：HeNB\_HARQ\_11

|  |  |
| --- | --- |
| **说明** | 当输入错误值时，显示错误信息。 |
| **输入** | rnti = 0x002A; dec\_result = 0; data\_size = 0;phy\_p = NULL; |
| **预期输出** | 不能为该rnti建立rnti实体。 |

**测试修改**

无

**测试方法**

rnti = 0x002A;

dec\_result = 0;

data\_size = 0;

phy\_p = NULL;

do\_ulharq (rnti, dec\_result, data\_size, phy\_p);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 11  
Invalid input parameter phy\_p!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.8测函数：INT32 do\_ulharq\_process (UlProcInfo \*ulproc\_p, UINT8 ul\_max\_harq\_tx, INT8 dec\_result, UINT32 data\_size, ResourcePhyInfo \*phy\_p

### 3.8.1 函数描述：

**功能**

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。

**输出**

无

**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);

}

### 3.8.2 测试用例：

|  |  |  |
| --- | --- | --- |
| **测试用例** | **测试类型** | **测试说明** |
| HeNB\_HARQ\_12 | 等价类划分 | 输入参数错误情况。 |
| HeNB\_HARQ\_13 | 等价类划分 | 当前进程重传时，解码失败且进程数超出最大进程数情况。 |
| HeNB\_HARQ\_14 | 语句覆盖 | 当前进程重传时，解码失败且进程数等于最大进程数情况。 |
| HeNB\_HARQ\_15 | 语句覆盖 | 当前进程重传时，解码失败且进程数小于最大进程数情况。 |
| HeNB\_HARQ\_16 | 语句覆盖 | 当前进程重传时，解码成功情况。 |
| HeNB\_HARQ\_17 | 语句覆盖 | 当前进程空闲，解码成功情况。 |
| HeNB\_HARQ\_18 | 语句覆盖 | 当前进程空闲，进程数大于最大进程数且解码失败情况。 |
| HeNB\_HARQ\_19 | 语句覆盖 | 当前进程空闲，进程数等于最大进程数且解码失败情况。 |
| HeNB\_HARQ\_20 | 语句覆盖 | 当前进程空闲，进程数小于最大进程数且解码失败情况。 |

#### 测试用例：HeNB\_HARQ\_12

|  |  |
| --- | --- |
| **说明** | 当输入错误参数时，显示错误信息。 |
| **输入** | ulproc\_p = NULL; |
| **预期输出** | 错误信息显示 |

**测试修改**

无

**测试方法**

ulproc\_p = NULL;

dec\_result = DEC\_SUCC;

ul\_max\_harq\_tx = 3;

data\_size = 128;

do\_ulharq\_process (ulproc\_p, ul\_max\_harq\_tx, dec\_result, data\_size,

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 12  
\*\*\*start to do ulharq process\*\*\*  
The UL process is not exist!!!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_13

|  |  |
| --- | --- |
| **说明** | 当前进程重传时，解码失败且进程数超出最大进程数情况。 |
| **输入** | current\_tx\_num =2, dec\_result= DEC\_FAIL |
| **预期输出** |  |

**测试修改**

无

**测试方法**

ulproc\_p = (UlProcInfo \*) lte\_malloc (sizeof (UlProcInfo));

if (ulproc\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

Memset (ulproc\_p, 0, sizeof (UlProcInfo));

ulproc\_p->current\_tx\_num = 2;

ulproc\_p->retran\_ind = 1;

ulproc\_p->tb\_size = 128;

(ulproc\_p->phy\_info).rb\_start = 2;

(ulproc\_p->phy\_info).rb\_num = 2;

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

(ulproc\_p->phy\_info).rv = 3;

dec\_result = DEC\_FAIL;

ul\_max\_harq\_tx = 3;

data\_size = 128;

do\_ulharq\_process (ulproc\_p, ul\_max\_harq\_tx, dec\_result, data\_size,

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 13  
retran\_ind:1, dec\_result:1.  
ulproc\_p->current\_tx\_num:2  
\*\*\*start to do ulharq process\*\*\*  
error: in current tx num!!!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_14

|  |  |
| --- | --- |
| **说明** | 当前进程重传时，解码失败且进程数等于最大进程数情况。 |
| **输入** | current\_tx\_num =1, dec\_result= DEC\_FAIL |
| **预期输出** | ACK链表 |

**测试修改**

无

**测试方法**

ulproc\_p = (UlProcInfo \*) lte\_malloc (sizeof (UlProcInfo));

if (ulproc\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

Memset (ulproc\_p, 0, sizeof (UlProcInfo));

ulproc\_p->current\_tx\_num = 1;

ulproc\_p->retran\_ind = 1;

ulproc\_p->tb\_size = 128;

(ulproc\_p->phy\_info).rb\_start = 2;

(ulproc\_p->phy\_info).rb\_num = 2;

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

(ulproc\_p->phy\_info).rv = 3;

dec\_result = DEC\_FAIL;

do\_ulharq\_process (ulproc\_p, ul\_max\_harq\_tx, dec\_result, data\_size,

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 14  
retran\_ind:1, dec\_result:1.  
ulproc\_p->current\_tx\_num:1  
\*\*\*start to do ulharq process\*\*\*  
ulproc\_p->retran\_ind:0  
ack\_type:1  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_15

|  |  |
| --- | --- |
| **说明** | 当前进程重传时，解码失败且进程数小于最大进程数情况。 |
| **输入** | current\_tx\_num =0, dec\_result= DEC\_FAIL |
| **预期输出** | NACK链表 |

**测试修改**

无

**测试方法**

ulproc\_p = (UlProcInfo \*) lte\_malloc (sizeof (UlProcInfo));

if (ulproc\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

Memset (ulproc\_p, 0, sizeof (UlProcInfo));

ulproc\_p->current\_tx\_num = 0;

ulproc\_p->retran\_ind = 1;

ulproc\_p->tb\_size = 128;

(ulproc\_p->phy\_info).rb\_start = 2;

(ulproc\_p->phy\_info).rb\_num = 2;

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

(ulproc\_p->phy\_info).rv = 3;

dec\_result = DEC\_FAIL;

do\_ulharq\_process (ulproc\_p, ul\_max\_harq\_tx, dec\_result, data\_size,

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 15  
retran\_ind:1, dec\_result:1.  
ulproc\_p->current\_tx\_num:0  
\*\*\*start to do ulharq process\*\*\*  
ulproc\_p->retran\_ind:1  
ack\_type:1  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_16

|  |  |
| --- | --- |
| **说明** | 当前进程重传时，解码成功情况。 |
| **输入** | current\_tx\_num =1, dec\_result= DEC\_SUCC |
| **预期输出** | NACK链表 |

**测试修改**

无

**测试方法**

ulproc\_p = (UlProcInfo \*) lte\_malloc (sizeof (UlProcInfo));

if (ulproc\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

Memset (ulproc\_p, 0, sizeof (UlProcInfo));

ulproc\_p->current\_tx\_num = 1;

ulproc\_p->retran\_ind = 1;

ulproc\_p->tb\_size = 128;

(ulproc\_p->phy\_info).rb\_start = 2;

(ulproc\_p->phy\_info).rb\_num = 2;

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

(ulproc\_p->phy\_info).rv = 3;

dec\_result = DEC\_SUCC;

do\_ulharq\_process (ulproc\_p, ul\_max\_harq\_tx, dec\_result, data\_size,

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 16  
retran\_ind:1, dec\_result:0.  
ulproc\_p->current\_tx\_num:1  
\*\*\*start to do ulharq process\*\*\*  
ulproc\_p->retran\_ind:0  
ack\_type:0  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_17

|  |  |
| --- | --- |
| **说明** | 当前进程空闲，解码成功情况。 |
| **输入** | current\_tx\_num =1, dec\_result= DEC\_SUCC |
| **预期输出** | NACK链表 |

**测试修改**

无

**测试方法**

ulproc\_p = (UlProcInfo \*) lte\_malloc (sizeof (UlProcInfo));

if (ulproc\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

memset (ulproc\_p, 0, sizeof(UlProcInfo));

ulproc\_p->current\_tx\_num = 2;

ulproc\_p->retran\_ind = 0;

ulproc\_p->tb\_size = 128;

(ulproc\_p->phy\_info).rb\_start = 2;

(ulproc\_p->phy\_info).rb\_num = 2;

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

(ulproc\_p->phy\_info).rv = 3;

dec\_result = DEC\_SUCC;

do\_ulharq\_process (ulproc\_p, ul\_max\_harq\_tx, dec\_result, data\_size,

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 17  
retran\_ind:0,dec\_result:0.  
ulproc\_p->current\_tx\_num:2  
\*\*\*start to do ulharq process\*\*\*  
ulproc\_p->retran\_ind:0  
ack\_type:0  
##################################  
####     HARQ.c test end      ####  
##################################  
BUG描述**

无

#### 测试用例：HeNB\_HARQ\_18

|  |  |
| --- | --- |
| **说明** | 当前进程空闲，进程数大于最大进程数且解码失败情况。 |
| **输入** | current\_tx\_num =2, dec\_result= DEC\_FAIL |
| **预期输出** | NACK链表 |

**测试修改**

无

**测试方法**

ulproc\_p = (UlProcInfo \*) lte\_malloc (sizeof (UlProcInfo));

if (ulproc\_p == NULL) {

printf("Not Enough Memory!\n");

return -1;

}

memset (ulproc\_p, 0, sizeof(UlProcInfo));

ulproc\_p->current\_tx\_num = 2;

ulproc\_p->retran\_ind = 0;

ulproc\_p->tb\_size = 128;

(ulproc\_p->phy\_info).rb\_start = 2;

(ulproc\_p->phy\_info).rb\_num = 2;

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

(ulproc\_p->phy\_info).rv = 3;

dec\_result = DEC\_FAIL;

do\_ulharq\_process (ulproc\_p, ul\_max\_harq\_tx, dec\_result, data\_size,

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 18  
retran\_ind:0, dec\_result:1.  
ulproc\_p->current\_tx\_num:2  
\*\*\*start to do ulharq process\*\*\*  
error: in current tx num!!!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_19

|  |  |
| --- | --- |
| **说明** | 当前进程空闲，进程数等于最大进程数且解码失败情况。 |
| **输入** | current\_tx\_num =1, dec\_result= DEC\_FAIL |
| **预期输出** | NACK链表 |

**测试修改**

无

**测试方法**

ulproc\_p = (UlProcInfo \*) lte\_malloc (sizeof (UlProcInfo));

if (ulproc\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

memset (ulproc\_p, 0, sizeof(UlProcInfo));

ulproc\_p->current\_tx\_num = 1;

ulproc\_p->retran\_ind = 0;

ulproc\_p->tb\_size = 128;

(ulproc\_p->phy\_info).rb\_start = 2;

(ulproc\_p->phy\_info).rb\_num = 2;

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

(ulproc\_p->phy\_info).rv = 3;

dec\_result = DEC\_FAIL;

do\_ulharq\_process (ulproc\_p, ul\_max\_harq\_tx, dec\_result, data\_size,

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 19  
retran\_ind:0, dec\_result:1.  
ulproc\_p->current\_tx\_num:1  
\*\*\*start to do ulharq process\*\*\*  
ulproc\_p->retran\_ind:0  
ack\_type:1  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

#### 测试用例：HeNB\_HARQ\_20

|  |  |
| --- | --- |
| **说明** | 当前进程空闲，进程数小于最大进程数且解码失败情况。 |
| **输入** | current\_tx\_num =0, dec\_result= DEC\_FAIL |
| **预期输出** | NACK链表 |

**测试修改**

无

**测试方法**

ulproc\_p = (UlProcInfo \*) lte\_malloc (sizeof (UlProcInfo));

if (ulproc\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

memset (ulproc\_p, 0, sizeof(UlProcInfo));

ulproc\_p->current\_tx\_num = 0;

ulproc\_p->retran\_ind = 0;

ulproc\_p->tb\_size = 128;

(ulproc\_p->phy\_info).rb\_start = 2;

(ulproc\_p->phy\_info).rb\_num = 2;

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

(ulproc\_p->phy\_info).rv = 3;

dec\_result = DEC\_FAIL;

do\_ulharq\_process (ulproc\_p, ul\_max\_harq\_tx, dec\_result, data\_size,

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 20  
retran\_ind:0, dec\_result:1.  
ulproc\_p->current\_tx\_num:0  
\*\*\*start to do ulharq process\*\*\*  
ulproc\_p->retran\_ind:1  
ack\_type:1  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.9被测函数： INT32 do\_newtran\_dlharq (DlHarqInfo \*dlharq\_p, UINT32 tb\_size, PduInfo4HarqLst \*pdulst\_p, TbHarqParam \*tbharqparam\_p)

### 3.9.1 函数描述：

**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；

**输出**

tbharqparam\_p：HARQ信息。

**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

### 3.9.2 测试用例：

#### 测试用例：HeNB\_HARQ\_21

|  |  |
| --- | --- |
| **说明** | 输入参数错误情况 |
| **输入** | dlharq\_p=NULL, pdulst\_p=NULL, |
| **预期输出** | 不能为该rnti建立rnti实体。 |

**测试修改**

无

**测试方法**

dlharq\_p = NULL;

tb\_size = 128;

pdulst\_p = NULL;

pdulst\_p = NULL;

do\_newtran\_dlharq (dlharq\_p, tb\_size, pdulst\_p, pdulst\_p);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 21  
Invalid input parameter pdulst\_p!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.10被测函数：INT32 do\_retran\_dlharq (DlHarqInfo \*dlharq\_p, PduInfo4HarqLst \*pdulst\_p, TbHarqParam \*tbharqparam\_p)

#### 3.10.1 函数描述：

**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；

**输出**

pdulst\_p：指向重传数据的指针；

tbharqparam\_p：HARQ信息。

**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

### 3.10.2 测试用例：

#### 测试用例：HeNB\_HARQ\_22

|  |  |
| --- | --- |
| **说明** | 输入参数错误情况 |
| **输入** | dlharq\_p=NULL, pdulst\_p=NULL, |
| **预期输出** | 不能为该rnti建立rnti实体。 |

**测试修改**

无

**测试方法**

DlHarqInfo \*dlharq\_p = NULL;

PduInfo4HarqLst \*pdulst\_p = NULL;

TbHarqParam \*tbharqparam\_p = NULL;

do\_retran\_dlharq (dlharq\_p, pdulst\_p, tbharqparam\_p);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 22  
Invalid input parameter pdulst\_p!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.11被测函数：INT32 delete\_ue\_harq (UINT16 rnti)

### 3.11.1 函数描述：

**功能**

销毁UE的HARQ实体。

**输入**

rnit： RNTI值；

**输出**

无

**算法与处理流程**

销毁UE的HARQ实体。

### 3.11.2 测试用例：

#### 测试用例：HeNB\_HARQ\_23

|  |  |
| --- | --- |
| **说明** | 当输入错误RNTI值时，显示错误信息。 |
| **输入** | rnti：0x002A; ，Incorrect rnti。 |
| **预期输出** | 错误信息 |

**测试修改**

无

**测试方法**

rnti = 0x002A;

delete\_ue\_harq (rnti);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 23  
the entity of this rnti is not exist!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 3.12 被测函数：INT32 rcv\_ack\_nack (AckNode \* ack\_node\_p)

### 3.12.1 函数描述：

**函数描述**

接收到ACK/NACK的HARQ处理。

**输入**：

ack\_node\_p: ACK节点

**输出**

无

**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\_state ();

/\*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\_state ();

/\*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;

}

}

### 3.12.2 测试用例：

#### 测试用例：HeNB\_HARQ\_24

|  |  |
| --- | --- |
| **说明** | 当输入参数错误情况 |
| **输入** | ack\_node\_p=NULL |
| **预期输出** | 显示错误信息 |

**测试修改**

无

**测试方法**

ack\_node\_p = NULL;

rcv\_ack\_nack (ack\_node\_p);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 25  
invalid input parameter ack\_node\_p!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

# 4 功能测试

## 4.1 接收ACK/NACK功能测试

### 4.1.1 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。

### 4.1.2 所涉及函数及调用关系



### 4.1.3 测试内容

### 4.1.4 测试用例：

#### 测试用例：HeNB\_HARQ\_25

|  |  |
| --- | --- |
| **说明** | 根据五种不同情况的ack节点，分别找到对应的五种不同情况进程并进行相应处理。 |
| **输入** | Acknode0：ack，sub\_fn2，sys\_fn1；  Acknode1：nack，sub\_fn3，sys\_fn1；  Acknode2：ack，sub\_fn4，sys\_fn1；  Acknode3：nack，sub\_fn5，sys\_fn1；  Acknode4：ack，sub\_fn6，sys\_fn1；  Proc0：dl\_tx\_time=11；current\_tx\_num=1;  Proc1：dl\_tx\_time=12；current\_tx\_num=1;  Proc2：dl\_tx\_time=13；current\_tx\_num=3;  Proc3：dl\_tx\_time=14；current\_tx\_num=1;  Proc4：dl\_tx\_time=15；current\_tx\_num=1; |
| **预期输出** | 收到ack/nack后，进行相应进程的置空或重传处理。 |

**测试修改**

无

**测试方法**

construct\_ack\_lst（）；

process\_rcv\_ack(ack\_lst\_p)；

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 25  
 This ACK list has 5 nodes.  
node 0:rnti:ff,acktype:0  
node 1:rnti:3d,acktype:1  
node 2:rnti:3e,acktype:0  
node 3:rnti:3f,acktype:1  
node 4:rnti:3f,acktype:0  
\*\*\*\*\*\*\*ACK node\*\*\*\*\*\*\*  
dlproc0:dl\_tx\_time is b  
dlproc1:dl\_tx\_time is c  
dlproc2:dl\_tx\_time is d  
dlproc3:dl\_tx\_time is e  
dlproc4:dl\_tx\_time is f  
tx\_time is c!  
the corresponding proc:1  
set the proc to be idle!!!  
dlproc\_p->state: 0  
current\_tx\_num:0  
\*\*\*\*\*\*\*ACK node\*\*\*\*\*\*\*  
dlproc0:dl\_tx\_time is b  
dlproc1:dl\_tx\_time is c  
dlproc2:dl\_tx\_time is d  
dlproc3:dl\_tx\_time is e  
dlproc4:dl\_tx\_time is f  
tx\_time is d!  
the corresponding proc:2  
dlproc\_p->state:0  
current\_tx\_num:0  
\*\*\*\*\*\*\*ACK node\*\*\*\*\*\*\*  
dlproc0:dl\_tx\_time is b  
dlproc1:dl\_tx\_time is c  
dlproc2:dl\_tx\_time is d  
dlproc3:dl\_tx\_time is e  
dlproc4:dl\_tx\_time is f  
tx\_time is e!  
the corresponding proc:3  
set the proc to be idle!!!  
dlproc\_p->state: 0  
current\_tx\_num:0  
\*\*\*\*\*\*\*ACK node\*\*\*\*\*\*\*  
dlproc0:dl\_tx\_time is b  
dlproc1:dl\_tx\_time is c  
dlproc2:dl\_tx\_time is d  
dlproc3:dl\_tx\_time is e  
dlproc4:dl\_tx\_time is f  
tx\_time is f!  
the corresponding proc:4  
set the proc to be retran!!!  
next\_rv:3, dlproc\_p->state: 2  
\*\*\*\*\*\*\*ACK node\*\*\*\*\*\*\*  
dlproc0:dl\_tx\_time is b  
dlproc1:dl\_tx\_time is c  
dlproc2:dl\_tx\_time is d  
dlproc3:dl\_tx\_time is e  
dlproc4:dl\_tx\_time is f  
tx\_time is 10!  
Can't find process that ACK belongs to!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

## 4.2下行HARQ处理功能测试

### 4.2.1 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。

### 4.2.2 所涉及函数及调用关系



### 4.2.3 测试内容

### 4.2.4 测试用例：

#### 测试用例： HeNB\_HARQ\_26

|  |  |
| --- | --- |
| **说明** | 测试五种不同进程情况。 |
| **输入** | Proc0：state1，dl\_tx\_time=11，rv=2；  Proc1：state1，dl\_tx\_time=12，rv=2； Proc2：state0，dl\_tx\_time=13，rv=2；  Proc3：state0，dl\_tx\_time=14，rv=2；  Proc4：state1，dl\_tx\_time=15，rv=2； |
| **预期输出** | 新传时找到空闲进程id进行新传处理，重传时找到重传进程id进行重传处理。 |

**测试修改**

无

**测试方法**

PduInfo4HarqLst \*pdu\_p;

TbHarqParam \*tbharqparam\_p;

UINT32 tb\_size = 128;

UINT16 rnti = 0x003d;

UINT8 ndi;

UINT8 i, j = 0;

pdu\_p = construct\_Pdulst ();

tbharqparam\_p = (TbHarqParam \*) lte\_malloc (sizeof (TbHarqParam));

tbharqparam\_p->irv = 0;

for (i=0; i<4; i++) {

tbharqparam\_p->process\_id = j++;

switch (i) {

case 0:

case 2:

ndi = 1; /\* test case of newtransimission \*/

break;

case 1:

case 3:

ndi = 0; /\* test case of retransimission \*/

break;

} /\* switch \*/

do\_dlharq (rnti, ndi, tb\_size, pdu\_p, tbharqparam\_p);

printf ("\*\*\*\*the next test\*\*\*\*\n");

}**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is:26  
proc 0:state 1,rv 2  
proc 1:state 1,rv 2  
proc 2:state 0,rv 2  
proc 3:state 0,rv 2  
proc 4:state 1,rv 2  
the idle dlproc\_id is 2  
proc:2,rv 0  
\*\*\*\*the next test\*\*\*\*  
proc 0:state 1,rv 2  
proc 1:state 1,rv 2  
proc 2:state 0,rv 2  
proc 3:state 0,rv 2  
proc 4:state 1,rv 2  
the retran dlproc\_id is 0  
proc:0,retran\_rv 3  
\*\*\*\*the next test\*\*\*\*  
proc 0:state 1,rv 2  
proc 1:state 1,rv 2  
proc 2:state 0,rv 2  
proc 3:state 0,rv 2  
proc 4:state 1,rv 2  
the idle dlproc\_id is 2  
proc:2,rv 0  
\*\*\*\*the next test\*\*\*\*  
proc 0:state 1,rv 2  
proc 1:state 1,rv 2  
proc 2:state 0,rv 2  
proc 3:state 0,rv 2  
proc 4:state 1,rv 2  
the retran dlproc\_id is 0  
proc:0,retran\_rv 3  
\*\*\*\*the next test\*\*\*\*  
##################################  
####     HARQ.c test end      ####  
##################################  
BUG描述**

无

## 4.3上行HARQ处理功能测试

### 4.3.1 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。

### 4.3.2 所涉及函数及调用关系



### 4.3.3 测试内容

### 4.3.4 测试用例：

|  |  |  |
| --- | --- | --- |
| **测试用例** | **测试类型** | **测试说明** |
| HeNB\_HARQ\_27 | 等价类划分 | TB解码成功未标注重传。 |
| HeNB\_HARQ\_27 | 等价类划分 | TB解码失败未标注重传。 |
| HeNB\_HARQ\_27 | 等价类划分 | TB解码成功标注重传。 |
| HeNB\_HARQ\_27 | 等价类划分 | TB解码失败标注重传。 |

#### 测试用例：HeNB\_HARQ\_27

|  |  |
| --- | --- |
| **说明** | 该测试用例处理四种不同情况的进程。 |
| **输入** | Proc0：DEC\_SUCC, current\_tx\_num=1，retran\_ind=0；  Proc1：DEC\_FAIL, current\_tx\_num=3，retran\_ind=0；  Proc2：DEC\_SUCC, current\_tx\_num=2，retran\_ind=1；  Proc3：DEC\_FAIL, current\_tx\_num=3，retran\_ind=1； |
| **预期输出** | 反馈结果。 |

**测试修改**

无

**测试方法**

ResourcePhyInfo \*phy\_p = NULL;

UINT32 data\_size = 128;

UINT16 rnti = 0x003d;

UINT8 j = 0;

UINT8 dec\_result;

phy\_p = (ResourcePhyInfo \*) lte\_malloc (sizeof (ResourcePhyInfo));

if (phy\_p == NULL) {

printf ("Not Enough Memory!\n");

return -1;

}

phy\_p->rb\_start = 0;

phy\_p->rb\_num = 3;

phy\_p->mcs = 2;

phy\_p->rv = 2;

for (j=0;j<4;j++) {

switch (j) {

case 0:

case 2:

dec\_result = DEC\_SUCC;

break;

case 1:

case 3:

dec\_result = DEC\_FAIL;

break;

}

do\_ulharq (rnti, dec\_result, data\_size, phy\_p);

**测试输出**

**##################################  
####     HARQ.c test start    ####  
##################################  
the test case is: 27  
proc0:current\_tx\_num 1, retran\_ind 0  
\*\*\*after ulharq process\*\*\*  
ulproc\_p->retran\_ind:0  
ack\_type:0  
  
\*\*\*\*the next test\*\*\*\*  
proc1:current\_tx\_num 3,retran\_ind 0  
error: in current tx num!!!  
  
\*\*\*\*the next test\*\*\*\*  
proc2:current\_tx\_num 2, retran\_ind 1  
\*\*\*after ulharq process\*\*\*  
ulproc\_p->retran\_ind:0  
ack\_type:0  
  
\*\*\*\*the next test\*\*\*\*  
proc3:current\_tx\_num 3, retran\_ind 1  
error: in current tx num!!!  
##################################  
####     HARQ.c test end      ####  
##################################**

**BUG描述**

无

# 5 其他测试

# 6 附录1 测试类型

## 1. 等价类划分

等价类划分是一种典型的黑盒测试方法。等价类是指某个输入域的集合。它表示对揭露程序中的错误来说，集合中的每个输入条件是等效的。因此我们只要在一个集合中选取一个测试数据即可。等价类划分的办法是把程序的输入域划分成若干等价类，然后从每个部分中选取少数代表性数据当作测试用例。这样就可使用少数测试用例检验程序在一大类情况下的反映。

在考虑等价类时，应该注意区别以下两种不同的情况：

有效等价类：有效等价类指的是对程序的规范是有意义的、合理的输入数据所构成的集合。在具体问题中，有效等价类可以是一个，也可以是多个。

无效等价类：无效等价类指对程序的规范是不合理的或无意义的输入数据所构成的集合。对于具体的问题，无效等价类至少应有一个，也可能有多个。

确定等价类有以下几条原则：

如果输入条件规定了取值范围或值的个数，则可确定一个有效等价类和两个无效等价类。例如，程序的规范中提到的输入条包括“……项数可以从1到999……”，则可取有效等价类为“l考项数＜999”，无效等价类为“项数＜l，，及“项数＞999”。

**输入条件规定了输入值的集合，或是规定了“必须如何”的条件，则可确定一个有效等价类和一个无效等价类。如某程序涉及标识符，其输入条件规定“标识符应以字母开头……”则“以字母开头者”作为有效等价类，“以非字母开头”作为无效等价类。**

如果我们确知，已划分的等价类中各元素在程序中的处理方式是不同的，则应将此等价类进一步划分成更小等价类。

|  |  |  |
| --- | --- | --- |
| 输入条件 | 有效等价类 | 无效等价类 |
| 。。。。。。  。。。。。。 | 。。。。。。  。。。。。。 | 。。。。。。  。。。。。。 |

根据已列出的等价类表，按以下步骤确定测试用例：

为每个等价类规定一个唯一的编号；

设计一个测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类。重复这一步，最后使得所有有效等价类均被测试用例所覆盖；

设计一个新的测试用例，使其只覆盖一个无效等价类。重复这一步，使所有无效等价类均被覆盖。这里强调每次只覆盖一个无效等价类。这是因为一个测试用例中如果含有多个缺陷，有可能在测试中只发现其中的一个，另一些被忽视。等价类划分法能够全面、系统地考虑黑盒测试的测试用例设计问题，但是没有注意选用一些“高效的”、“有针对性的”测试用例。后面介绍的边值分析法可以弥补这一缺点。

## 2. 边值分析法

边值分析法是列出单元功能、输入、状态及控制的合法边界值和非法边界值，设计测试用例，包含全部边界值的方法。典型地包括IF语句中的判别值，定义域、值域边界，空或畸形输入，末受控状态等。边值分析法不是一类找一个例子的方法，而是以边界情况的处理作为主要目标专门设计测试用例的方法。另外，边值分析不仅考查输入的边值，也要考虑输出的边值。这是从人们的经验得出的一种有效方法。人们发现许多软件错误只是在下标、数据结构和标量值的边界值及其上、下出现，运行这个区域的测试用例发现错误的概率很高。

用边值分析法设计测试用例时，有以下几条原则：

如果输入条件规定了取值范围，或是规定了值的个数，则应以该范围的边界内及刚刚超出范围的边界外的值，或是分别对最大、最小及稍小于最小、稍大于最大个数作为测试用例。如有规范“某文件可包含l至255”个记录……“，则测试用例可选1和255及0和256等。

针对规范的每个输出条件使用原则〔a〕。

如果程序规范中提到的输入或输出域是个有序的集合(如顺序文件、表格等)就应注意选取有序集的第一个和最后一个元素作为测试用例。

分析规范，尽可能找出可能的边界条件。一个典型的边值分析例子是三角形分类程序。选取a，b，c构成三角形三边，“任意两边之和大于第三边”为边界条件。边值分析相等价类划分侧重不同，对等价类划分是一个补充。如上述三角形问题，选取a＝3，b＝4，c＝5，a＝2，b＝4，c＝7则覆盖有效和无效等价类。如果能在等价类划分中注入边值分析的思想。在每个等价类中不只选取一个覆盖用例，而是进而选取该等价类的边界值等价类划分法将更有效，最后可以用边值分析法再补充一些测试用例。

## 3. 语句覆盖

程序的某次运行一般并不能执行到其中的每一个语句，因此，如果某语句含有一个错误，而它在测试中没执行，这个错误就不可能被发现。为了提高发现错误的可能性，应该在测试时至少要执行程序中的每一个语句。

所谓“语句覆盖”测试标准，它的含义是：选择足够的测试用例，使得程序中每个语句至少都能执行一次。

例子：

Procedure Example(Var A,B,C:real)

begin

if(A>1)and(B=0)

then x:=x/A；

if(A=2)or(x>1)

then x:=x+l

end;

为了使程序中每个语句至少执行一次，只需设计一个能通过路径ace的例子就可以了。例如选择输入数据为：

A=2，B=0，x=3

就可达到“语句覆盖”标准。

显然，语句覆盖是一个比较弱的覆盖标准。如果第一个条件语句中的and错误地写成or，上面的测试用例是不能发现这个错误的，或者是第二个条件语句中x>1误写成x>0，这个测试用例也不能暴露它。我们还可以举出许多错误情况是上述测试数据不能发现的。所以，一般认为“语句覆盖”是很不充分的最低的一种覆盖标准。

## 4. 判定覆盖

比“语句覆盖”稍强的覆盖标准是“判定覆盖”(或称分支覆盖)。这个标准是：执行足够的测试用例，使得程序中每个判定至少都获得一次“真”值和“假”值，即使得程序中的每一个分文至少都通过一次。

对上面那个例子，如果设计两个测试用例，就可以达到“判定覆盖”的标难。为此，我们可以选择输人数据为：

(1)A=3，B=0，x=l

(2)A=2，B=1，x=3

“判定覆盖”比“语句覆盖”严格，因为如果每个分支都执行过了，自然每个语句也就执行了。

## 5. 条件覆盖

它的含义是：执行足够的测试用例，使得判定中每个条件获得各种可能的结果。

对于例子程序,我们只需设计以下两个测试用例就可满足这标准：

(1)A＝2，B＝o，x＝4(沿路径ace执行)

(2)A＝1，B＝l，x＝l(沿路径aN执行)

虽然同样只要两个测试用例，但它比判定覆盖中两个测试用例更有效。一般来说，“条件覆盖”比“判定覆盖”强，但是，并不总是如此，满足“条件覆盖”不一定满足“判定覆盖”。例如对语句。

IF(A AND B)THEN S

设计两个测试用例：A“真”B“假”和A“假”B“真”。对于上例我们设计两个测试用例为：

(1)A＝1，B＝o，x＝3

(2)A＝2，B＝l，x＝1

亦是如此，它们能满足“条件覆盖”但不满足“判定覆盖”。

## 6. 判定／条件覆盖

针对上面的问题引出了另一种覆盖标准，这就是“判定／条件覆盖”，它的含义是：执行足够的测试用例，同时满足判定覆盖和条件覆盖的要求。显然，它比“判定覆盖”和“条件覆盖”都强。

对于例子程序，我们选取测试用例：

(1)A=2，B=0，x=4

(2)A=1，B=l，x=l

它满足判定／条件覆盖标准。

值得指出，看起来“判定／条件覆盖”似乎是比较合理的，应成为我们的目标，但是事实并非如此，因为大多数计算机不能用一条指令对多个条件作出判定，而必须将源程序中对多个条件的判定分解成几个简单判定。这个讨论说明了，尽管“判定／条件覆盖”看起来能使各种条件取到所有可能的值，但实际上并不一定能检查到这样的程度。针对这种情况，有下面的条件组合覆盖标准。

## 7. 条件组合覆盖

“条件组合覆盖”的含义是：执行足够的测试用例，使得每个判定中条件的各种可能组合都至少执行一次。这是一个最强的逻辑覆盖标准。

再看例子程序，必须使测试用例覆盖八种组合结果

(1)A>1，B=0 (5)A=2，x>1

(2)A>1，B<>0 (6)A=2，x<1

(3)A<l，B=0 (7)A<>2，x>1

(4)A<1，B<>0 (8)A<>2，x<1

必须注意到，(5)、(6)、(7)、(8)四种情况是第二个条件语句的条件组合，而x的值在该语句之前是要经过计算的，所以我们还必须根据程序的逻辑推算出在程序的人口点x的输入值应是什么。

要测试八个组合结果并不是意味着需要八种测试用例，事实上，我们能用四种测试用例来覆盖它们：

(1)A＝2，B＝o，x＝4；

(2)A＝2，B＝1，x＝l；

(3)A＝l，B＝o，x＝2；

(4)A＝1，B＝1，x＝l。

上面四个例子虽然满足条件组合覆盖，但并不能覆盖程序中的每一条路径，可以看出条件组合覆盖仍然是不彻底的，在白盒测试时，要设法弥补这个缺陷。