# 当前方案的实现框图



图 1 当前无线方案

DMA及PCIE的接口驱动采用DINI的IP核，并与子卡物理连接，CPU通过AHB总线通信。无线802.11通过在内部设计802.11的逻辑驱动实现链路层，再经物理层连接外部射频部分，采用RW的驱动程序实现MAC层，同样通过AHB总线通信。两个模块之间通过双口RAM共享数据。

## Platform DMA

该DMA是设计为PCIE服务，具有2个DMA通道，每个通道都具有收发能力，通过REQ，ACK进行通信同步，是DINI系列的专用设计，可在FPGA和如MV78200的SDRAM之间实现8Gbps的数据传输速率。

## 共享内存

在IP核内部实现了内存仲裁器，共享内存宽度64位，深度32768，16位地址线。

## DMA的全双工模式

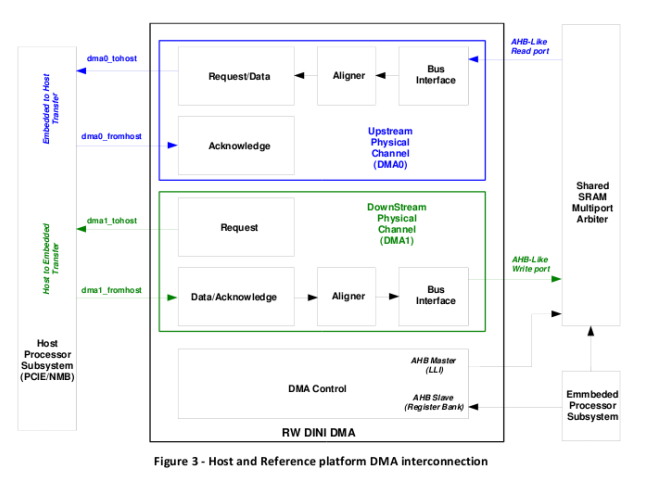


图 2 DINI子卡的DMA全双工模式

分为上行UpStream和下行DowmStream通道，一个通道读，一个通道写，通过DMA Control进行控制，操作时序参考手册。

# 在ZYNQ上的移植

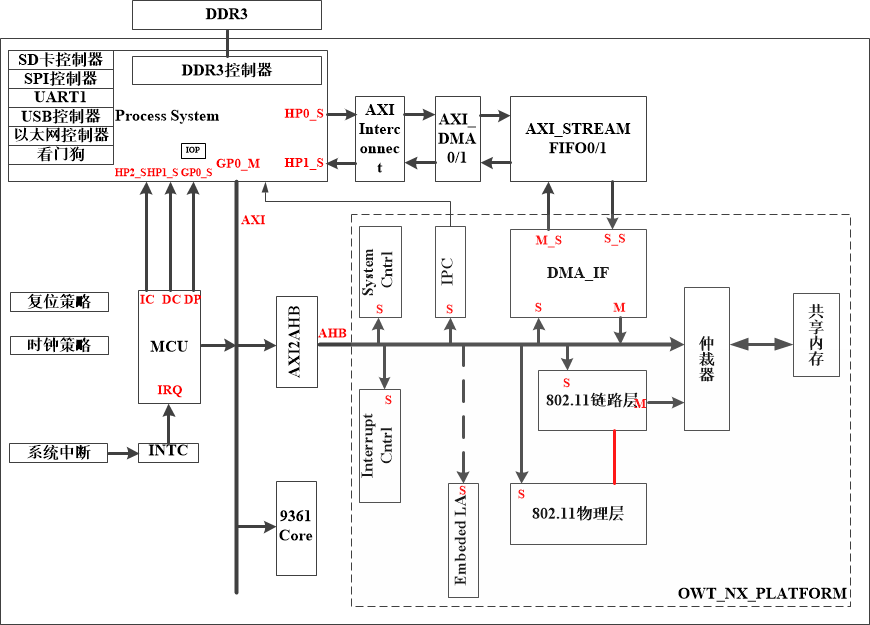


图 3 ZYNQ的无线方案初步设计

将IP核移植到ZYNQ平台上，需要处理AXI，AHB总线的转换，去掉DINI接口和Platform DMA，修改系统控制、中断控制、内核通信模块，添加共享内存和DDR3之间的DMA通信。

## DINI接口和Platform DMA

DINI接口是外接扩展存储板，由于ZYNQ平台有DDR3存储，因此去掉，连带Platform DMA也不需要。在ZYNQ内部设计DMA通道接通DDR3和共享内存即可。

## DMA和DDR3的通信

### 数据搬移方式

#### DMA方式

在ZYNQ内，有DMAC和IP核DMA两种方式，DMAC是PS片上DMA，驱动复杂，使用IP核的DMA有AXI\_DMA，AXI\_VDMA，AXI\_CDMA三种，除此之外的数据搬移IP还有AXI\_DATAMOVER，适用于DMA和DDR3进行通信的IP核是AXI\_DMA和AXI\_DATAMOVER，通过HP接口可在PL和DDR间建立数据传输通道。

#### 直接例化7系列存储器控制器

在vivado中，提供了7系列的存储器控制器IP，可直接在PL中例化使用，但是控制驱动复杂。

### 共享内存写入

在vivado中，分为bram和dram，bram是片上资源，dram是由逻辑生成的，xilinx均提供了IP核进行例化，能够自由分配深度和宽度。在当前方案中，使用了64\*32768=256KBytes的共享内存，因此在ZYNQ的方案中，也应该使用64位宽度的bram。

在ZYNQ中的bram生成IP和bram控制器IP都不是AXI\_STREAM总线形式，不能直接使用AXI\_DMA，AXI\_DMA是AXI\_STREAM总线，因此无论何种方案，如果使用AXI\_DMA，必定有总线转换过程。

FCS：Frame Check Sequence，帧的CRC校验值，帧检验序列。

## 数据验证及bd搭建

### 无线模式

有802.11a，802.11b，802.11g三种调制模式，由psel\_modema，psel\_modemb，pesl\_modemg进行选择，物理层均是通过DSSS+OFDM实现的。

## 测试平台文件的编写

测试平台文件编写首先例化RivieraWaves的物理层接口IP核rw\_wlanbb\_11g\_maxim，将输入时钟，输入复位信号，输出时钟、输出复位信号、输出时钟门控信号、看门狗控制信号，AHB总线信号、APB总线信号，模拟端口进行信号映射。

之后生成240Mhz、33Mhz、32Khz的测试时钟给测试IP，完成后进行复位信号产生、调制模式选择、门控信号产生。

完成后testbench通过M\_APB总线模拟APB主机向IP发送控制指令进行bup、modema、modemg、radio、frontend的寄存器的配置，总线信号主要是地址、数据、使能、写数据、读数据、功能选择。测试平台通过任务模拟APB总线时序通过接口控制IP核的内部动作，配置任务是pwbupreg、pwmdmareg、pwmdmgreg、pwradioreg、pwfrontendreg。

## AXI2APB的使用记录

由APB端的ready信号进行控制，如果APB从机没有准备好，数据收发不会发生。AXI发送的数据及地址和APB是同步的，

## IP核驱动设计

对MAC-PHY进行测试需要给出时钟接口和门控接口。

## 仿真记录

直接使用vivado 的仿真器报错：在viterb.vhd中存在Minmun已定义的问题，导致仿真无法进行。

使用modelsim的仿真器，库编译不对，使用现有库则报错。

### 仿真的库错误

分别是peak\_detect.v，equalizer.v，dsss\_demod.v，phase\_estimation.v，txrxcntl.v五个模块中的Library找不到导致错误，禁用这五个文件后能正常就进行仿真。

### VHDL版本问题

PHY模块编写使用的是VHDL93，需注意后继版本中对一些Minimum进行了声明，如使用VHDL2008的语言分析则会因重复声明报错。

### 时钟复位问题

在时钟复位模块中，PHY使用了复位管理电路，输入的全局复位在输入时钟的控制下进行逻辑门操作转移到输出端，因此当全局复位脉冲到来之时，必须有输入时钟，否则全局复位时钟不会转移到输出端。

#### 在ZYNQ中使用时钟的问题

在vivado中使用时钟模块会有输出时钟延时的问题，当该模块收到复位脉冲后会延时一段时间才有时钟输出，因此该复位脉冲和输出时钟不能同时接入PHY模块，否则时钟复位模块的输出全局复位不能启动。

## 发现的问题

从原工程导入时，MODEM802\_11b文件夹内的peak\_detect，phase\_eatimation，equalizer，dsss\_demod均找不到库文件。

### 库的识别问题

工程内能识别到pkg，但是识别不到global\_pkg。原工程中一个文件夹下的文件被vivado分成了rtl和xil\_default两个库，导入IP时分为了两部分。

### 对global\_pkg不能识别的处理

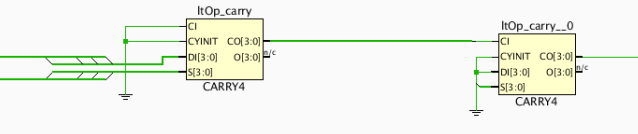
反复修改路径和文件夹不能处理，使用modelsim仿真会报错但能继续仿真，使用vivado仿真则不能进行，解决办法是复制global中内容到引用文件粘贴。

### 实现错误

ERROR DRC23-20 Rule violation (PDCY-4) CARRY4 unconnected input- CARRY input pin(s) ......cca\_fsm\_maxim\_1/lt0p\_carry\_0/S[0] unconnected. These pins must be connected to a signal or tied to vcc or GND when output pin .....cca\_fsm\_maxim\_1/lt0p\_carry\_0/C0[0] is used..

禁用opt\_design之后再进行实现出现：Rule violation(NDRV-1) Driverless Nets - Net top\_i/top\_rw\_wlan/inst/.......cca\_maxim\_1/cca\_fsm\_maxim\_1/lto0p\_carry\_0\_n\_18 are undriven。

分析后是进位链逻辑布局错误，主要和CCA中的时钟延时部分相关。



### Cca\_fsm\_maxim中的输入接口

Reg\_agccca\_disb，Reg\_ccarffoffen，Reg\_forceagcrst，Reg\_act，Reg\_modemabg，Reg\_agcwaitdc，Reg\_ccastatbdgen，Reg\_thragcoff，Agc\_lock，Agc\_rise，Agc\_fall，Pantpack\_dbm，Cs\_a\_high，Cs\_a\_low，Cs\_b\_high，Cs\_b\_low，Cs\_flag\_nb，Cs\_flag\_valid，Phy\_rxstartend\_ind，Sdf\_found，Cp2\_detected，B\_psdu\_duration，Rxv\_length，Rxv\_datarate，Phy\_txonoff\_req，Phy\_txonoff\_conf，Phy\_rxonoff\_stat，Phy\_ccarst\_req，Rxv\_macaddr\_match，Phy\_txstartend\_req，Fcs\_ok\_pulse，In\_sifs\_pulse。

## 实现错误的处理

选中-flatten\_hierarchy选项，设为none，由于层次结构太大，不展开层次结构综合。实现后的时序不收敛，仍有警告处理。

## 时序收敛与时序约束

Worst negetive slack (WNS ) ：-2.262ns

Total negative Slack (TNS) ：-37.908ns

Number of Failing Endpoints：17

No\_clock，unconstrained\_internal\_endpoints问题，时钟产生模块的建立时间2.262ns不符合。

## 内核配置细节问题

需要配置ARM核时钟，内存时钟(需注意应为时钟周期整数倍)，内存(低压)型号，MT41K256M16-re125。CPU时钟660MHz，DDR时钟528MHz，输入时钟33MHz。

## AXI转APB调试结果

在SDK上设计上位机测试软件进行测试，结果BUP读写正常，MODEMA读写不正常。

Config The Phy Registers.....

the bup reg at 43C00000 value is 307

the bup reg at 43C00004 value is 201

the bup reg at 43C00008 value is 0

the bup reg at 43C0000C value is 39

the bup reg at 43C00010 value is 53

the bup reg at 43C00014 value is 0

the bup reg at 43C00018 value is 0

the bup reg at 43C0001C value is F

the bup reg at 43C00020 value is 0

the bup reg at 43C00024 value is 0

the bup reg at 43C00028 value is 0

the bup reg at 43C0002C value is 400E0D0C

the bup reg at 43C00030 value is 0

the bup reg at 43C00034 value is 10080706

the bup reg at 43C00038 value is B0A

the bup reg at 43C0003C value is 110

the bup reg at 43C00040 value is 0

the bup reg at 43C00044 value is 0

the bup reg at 43C00048 value is 800000

the bup reg at 43C0004C value is 0

the bup reg at 43C00050 value is 0

the bup reg at 43C00054 value is 0

the bup reg at 43C00058 value is 0

the bup reg at 43C0005C value is 0

the bup reg at 43C00060 value is 0

the bup reg at 43C00064 value is 0

the bup reg at 43C00068 value is 0

the bup reg at 43C0006C value is 0

the bup reg at 43C00070 value is 0

the bup reg at 43C00074 value is 0

the bup reg at 43C00078 value is 0

BUP的读写寄存器符合程序的设计预期，而MODEMA的结果不符合程序预期，经查是front-end被禁导致部分bit位无法写入。

## RW\_WLAN\_PLATFORM的综合问题

存在一些未定义网络类型的输入输出端口，综合器默认为nonetype报错，解决办法删除预定义类型`define defaulttype none。

## AXI\_Stream到Dinidma的转换

### Rw\_nx\_platform出错

时钟模糊错误：maccontroller源代码不规范，添加else if描述修正错误。

### Rw\_nx\_platform宏定义未定义

宏定义未定义错误，经查是由于在自定义IP中，Global\_include是无效的，要想使宏定义生效，需要在设计工程的最顶层加入宏定义的引用描述。

### Rw\_nx\_platform的AHB总线

生成IP的时候强行使hready\_in和hsel置位，但内部并没有使用这两个信号，是否有关系？

### MicroBlaze在DDR上启动

MicroBlaze在DDR上启动需要设置两个HP接口和一个AXI从接口，将DDR的高256M字节空间分配给MicroBlaze。

### 中断的设置

将AXI\_DMA的中断分别给ARM和MicroBlaze，出现问题，ARM的中断号会被改变。

### ARM的cache问题

ARM默认开启cache，需禁止cache，microblaze和ARM的数据才能一致。

### 设备树不能生成的问题

1. 将AXI\_DMA连入MicroBlaze不连入ARM，发现生成设备树报错，提示找不到中断输入。
2. 将AXI\_DMA中断同时连入MB和ARM，设备树生成报错，仍然提示没有中断输入。
3. 将AXI\_DMA中断只连入ARM，成功生成设备树。
4. 将AXI\_DMA中断只连入MB，并且使AXI\_DMA不再挂载在ARM的设备总线上，成功生成设备树。

从现象看，ARM对自己的设备生成设备树时，同时也会生成该设备中断号，如果该设备中断没有连入ARM，生成设备树的时候就会报错。

### Dinidma的非对齐传输问题

由于Dinidma数据接口是64位，内部Shared SRAM写入是也是按64位，因此AXI\_DMA，HP接口都是64位。而MB通过Malloc动态内存分配内存可能不按字节对齐，因此会出现非对齐传输的问题。由于AXI\_DMA有非对齐传输的选项，因此勾选该选项进行测试结果如下：

1、不勾选，第一次DMA传输不错位，后面四次DMA传输发生错位。

2、勾选，第一次DMA传输错位，后面四次DMA传输不错位。

#### 非对齐解决方法

1. 使用GNU的\_\_attribute\_\_ (aligned(8))或直接设定分配地址 \_\_attribute\_\_ (address)。
2. 在有效数据的开头添加冗余数据。
3. 将数据宽度设为32位，进行串并转换。

4、数据宽度仍为64位，控制传输的字节码。

### MAC更新的问题

MAC调试发生问题，MAC的时钟没有接入。

### IPC的中断ARM侧不正确

后继测试发现ARM侧有中断，但是中断没有复位。

### ARM的启动问题

由于SDO的MIO管脚被占用，因此使用SD1启动。而UBOOT设置为SD0，因此需要改，先从QSPI FLASH启动，后从SD1启动。

在ZYNQ中使能了QSPI FLASH但是并不能从FLASH中启动。

由于MB有一部分数据段设置从Rw\_nw\_platform中的SRAM启动，并且是在FSBL中配置，而FSBL启动后，bit文件并没有烧写到PL部分，因此当FSBL阶段是找不到Rw\_nw\_platform中的SRAM，从而导致MB无法启动，解决方法修改FSBL，使MB在Bit文件烧录后再从Rw\_nw\_platform启动。

#### MB启动不稳定问题

带系统调试MB时，发现MB的启动不稳定，怀疑是因为MB的一部分数据段放在Shard MEM导致的问题。

##### MB启动设置

MB的qspi flash启动需注意设置FSBL的BSP，不能新建BSP，而应使用当前启动elf对应的bsp。对程序进行编译的时候应使用release版本进行下载。

#### MB中断不正常

发现MAC\_TOMB的中断一直为1，经查MACHW内部中断正常，内部中断到外部时设计的反相语句不起作用。于是将Rw\_nx\_platform中的代码进行了优化，使中断的网线不优化(\* syn\_keep = “true” \*)，并对反相的部分用括号括起来。

#### MB与ARM的双核调度问题

MB通过ARM的S\_AXI接口进行IOP的共享，使用ARM的片上资源如UART等。

# 资源分配

### 片上外设分配

表 1

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 端口号 | 配置 | 备注 |
| Quad SPI Flash | MIO 1-6 | 单FLASH |  |
| Ethernet 0 | MIO16-27 MIO52，53 |  |  |
| SD1 | MIO10-15 MIO0 | MIO0用于SD卡检测 |  |
| UART1 | EMIO W20,Y20 |  |  |
| USB1 | MIO28-39 |  |  |
| USB2 | MIO40-51 |  |  |

### CPU地址分配

#### ARM实际地址分配

表 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 地址开始 | 地址结束 | 大小 | 设备 |  |
| 0x0000\_0000 | 0x3FFF\_FFFF | 1GB | DDR |  |
| 0x4040\_0000 | 0x4040\_FFFF | 64KB | AXI\_DMA |  |
| 0x4041\_0000 | 0x4041\_FFFF | 64KB | AXI\_DMA |  |
| 0x6000\_0000 | 0x60FF\_FFFF | 16MB | RW\_WLAN PLF |  |
| 0x6000\_0000 | 0x607F\_FFFF | 8MB | Shard RAM |  |
| 0x6080\_0000 | 0x608F\_FFFF | 1MB | IPC |  |
| 0x6090\_0000 | 0x6090\_FFFF | 1MB | System Controller |  |
| 0x6091\_0000 | 0x6091\_FFFF | 1MB | Interrupt Controller |  |
| 0x60A0\_0000 | 0x60AF\_FFFF | 1MB | DMA Contorller |  |
| 0x60B0\_0000 | 0x60BF\_FFFF | 1MB | MAC |  |
| 0x60C0\_0000 | 0x60FF\_FFFF | 1MB | 保留 |  |

#### MicroBlaze实际地址分配

表 3

| 地址开始 | 地址结束 | 大小 | 设备 |  |
| --- | --- | --- | --- | --- |
| 0x3000\_0000 | 0x3FFF\_FFFF | 256MB | DDR |  |
| 0x4040\_0000 | 0x4040\_FFFF | 64KB | AXI\_DMA |  |
| 0x4041\_0000 | 0x4041\_FFFF | 64KB | AXI\_DMA |  |
| 0x6000\_0000 | 0x60FF\_FFFF | 16MB | RW\_WLAN PLF |  |
| 0x6000\_0000 | 0x607F\_FFFF | 8MB | Shard RAM |  |
| 0x6080\_0000 | 0x608F\_FFFF | 1MB | IPC |  |
| 0x6090\_0000 | 0x6090\_FFFF | 1MB | System Controller |  |
| 0x6091\_0000 | 0x6091\_FFFF | 1MB | Interrupt Controller |  |
| 0x60A0\_0000 | 0x60AF\_FFFF | 1MB | DMA Contorller |  |
| 0x60B0\_0000 | 0x60BF\_FFFF | 1MB | MAC |  |
| 0x60C0\_0000 | 0x60FF\_FFFF | 1MB | 保留 |  |

#### 中断设计

##### ARM中断设计

在ARM端采用一个中断口，使用CPU共享中断IRQ\_F2P，响应IPC的EMB2APP中断，在BSP中查找该中断进行配置。

表 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 中断源 | 中断矢量名称 | 中断源名称 | 复位空闲状态 | 中断触发 |
| IRQ\_F2P | Host\_irq | IPC中断 | 低电平 | 上升沿 |

##### MicroBlaze中断设计

在MicroBlaze侧采用32个中断口，分别使用AXI\_DMA收发中断，MAC传输中断，IPC中断，DMA中断，LLI中断和保留中断。

### 系统复位

系统使用上电复位信号PS\_POR\_B，上电后进行一次复位。JTAG的复位只能复位PS的调试部分和一个系统级复位。当BIT流文件下载后，会自动形成一次“软复位”，如果需要PL复位，可以操作PS的寄存器实现。首先打开写保护寄存器写入，再向PL寄存器写复位和复位清零指令。

表 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 中断源 | 中断矢量名称 | 中断源名称 | 复位空闲状态 | 中断触发 |
| AXI\_DMA中断 | axi\_dma\_m2ss | Axi\_dma数据发送中断 | 低电平 | 上升沿 |
| axi\_dma\_s2mm | Axi\_dma数据收取中断 | 低电平 | 上升沿 |
| MAC中断 | mac\_irq\_tomb[5] | ~mac\_int\_prot\_trigger\_n | 低电平 | 上升沿 |
| mac\_irq\_tomb[4] | ~mac\_int\_gen\_n | 低电平 | 上升沿 |
| mac\_irq\_tomb[3] | ~mac\_int\_tx\_trigger\_n | 低电平 | 上升沿 |
| mac\_irq\_tomb[2] | ~mac\_int\_rx\_trigger\_n | 低电平 | 上升沿 |
| mac\_irq\_tomb[1] | ~mac\_int\_tx\_rx\_misc\_n | 低电平 | 上升沿 |
| mac\_irq\_tomb[0] | ~mac\_int\_tx\_rx\_timer\_n | 低电平 | 上升沿 |
| IPC中断 | Ipc\_irq\_tomb[3] | IPC中断3 | 低电平 | 上升沿 |
| Ipc\_irq\_tomb[2] | IPC中断2 | 低电平 | 上升沿 |
| Ipc\_irq\_tomb[1] | IPC中断1 | 低电平 | 上升沿 |
| Ipc\_irq\_tomb[0] | IPC中断0 | 低电平 | 上升沿 |
| DMA中断 | dma\_channel\_irq\_tomb[3] | Dma中断3 | 低电平 | 上升沿 |
| dma\_channel\_irq\_tomb[2] | Dma中断2 | 低电平 | 上升沿 |

表 6

| 中断源 | 中断矢量名称 | 中断源名称 | 复位状态 | 中断触发 |
| --- | --- | --- | --- | --- |
| DMA中断 | dma\_channel\_irq\_tomb[1] | Dma中断1 | 低电平 | 上升沿 |
| dma\_channel\_irq\_tomb[0] | Dma中断0 | 低电平 | 上升沿 |
| LLI中断 | lli\_irq\_tomb[12] | LLI中断12 | 低电平 | 上升沿 |
| lli\_irq\_tomb[11] | LLI中断11 | 低电平 | 上升沿 |
| lli\_irq\_tomb[10] | LLI中断10 | 低电平 | 上升沿 |
| lli\_irq\_tomb[9] | LLI中断9 | 低电平 | 上升沿 |
| lli\_irq\_tomb[8] | LLI中断8 | 低电平 | 上升沿 |
| lli\_irq\_tomb[7] | LLI中断7 | 低电平 | 上升沿 |
| lli\_irq\_tomb[6] | LLI中断6 | 低电平 | 上升沿 |
| lli\_irq\_tomb[5] | LLI中断5 | 低电平 | 上升沿 |
| lli\_irq\_tomb[4] | LLI中断4 | 低电平 | 上升沿 |
| lli\_irq\_tomb[3] | LLI中断3 | 低电平 | 上升沿 |
| lli\_irq\_tomb[2] | LLI中断2 | 低电平 | 上升沿 |
| lli\_irq\_tomb[1] | LLI中断1 | 低电平 | 上升沿 |
| lli\_irq\_tomb[0] | LLI中断0 | 低电平 | 上升沿 |
| 保留中断 | Reserve\_irq\_tomb[1] | 预留空闲 |  |  |
| Reserve\_irq\_tomb[0] | 预留空闲 |  |  |

## 存储器

Axi\_dma的数据传输，如果不是8对齐的，则会在64位数据后面补零。如果没有将高双字写入到SRAM中，则tlast信号给不出来。

## 时钟树

# 优化

跨时钟报告：Report Clock Interaction.

内部时钟路径：Intra-Clock Paths

单时钟约束：intra-clock paths

同步时钟的跨时钟路径：inter-clocks paths

异步时钟的跨时钟路径：Other path group(async\_default)。

# MAC层的调试

## MAC协议笔记

802.11中的MAC层访问机制最常用的是分布式协调功能DCF。DCF信道访问机制是载波监听多路访问/避免CSMA/CA。由于无线通信检测冲突比较困难，因此使用冲突避免检测，分布式协调机制通过二进制随机退避实现，根据随机过程而来，是一种概率退避的算法。

802.11使用信道空闲评估(CCA)算法决定信道是否空闲，通过接收信号强度(RSSI)和载波侦听(CS)来完成，其中，载波侦听主要基于t1-t10这个部分的自相关或者互相关来完成的。

MAC层分为两层，即逻辑链路控制子层(LLI)和媒体访问控制子层(MAC)；物理层也分为两层：物理汇聚子层PLCP和物理媒体子层PMD。802.11的LLC子层和802.3的LLC子层完全相同。不同的是媒体访问控制子层，无线信道的特点决定了WLAN具体独特的信道访问控制(MAC)机制。

虚拟载波侦听(NAV)是一种预约信道的方案，可以降低隐曝终端的问题。

## 802.11的MAC层描述框图



图 4 MAC简单框图

## MAC层寄存器

### MAC控制寄存器(macCntrl1Reg)

地址0x60B0004C。

表 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特域名 | 读写控制 | 比特位 | 复位值 | 功能 |
| BssType | 读/写 | 0 | 1 | 服务类型，1-结构式BSS，0-IBSS |
| Ap | 读/写 | 1 | 0 | 1-AP模式 |
| pwrMgt | 读/写 | 2 | 0 | 功率管理 |
| cfgAware | 读/写 | 3 | 0 | 保留，仅用于AP模式 |
| Reserved |  | 6：4 | 000 |  |
| activeClkGating | 读/写 | 7 | 1 | 1-开启硬件门控，0-关闭硬件门控 |
| disableACKResp | 读/写 | 8 | 0 | 1-关闭自动ACK，0-使能硬件ACK |
| disableCTSResp | 读/写 | 9 | 0 | 1-关闭硬件CTS，0-使能硬件CTS |
| disableBAResp | 读/写 | 10 | 0 | 1-关闭硬件BA，0-使能硬件BA |
| rateControlledMPIF | 读/写 | 11 | 1 | MPIF接口速率控制 |
| mibTableReset |  | 12 | 0 | MIB内容清零 |
| keyStoRAMReset |  | 13 | 0 | KeyStorRAMReset清零 |
| abgnMode | 读写 | 16：14 | 011 | ABGN模式选择 |
| Reserved | 仅读 | 23：17 | 0000000 |  |
| tsfUpdateBySW |  | 24 | 0 | Tsf更新 |
| tsfMgtDisable | 读/写 | 25 | 0 |  |
| rxRIFSEn | 读/写 | 26 | 0 |  |
| Reserved | 仅读 | 31：27 | 00000 | 保留 |

## DMA的状态机转换



图 5

在WAIT\_TXRX\_TRIG状态时，如果MAC Core赢得竞争，则MAC Core状态机转换到ACTIVE\_TX状态触发DMA和MAC的发送状态机。

## MAC发送路径模块



图 6

## MAC层回退模块

backoffctrl具有6种状态：空闲IDLE，载入BK计数LD\_BKOFF\_CNT,检测无线媒介状态CHK\_MEDIUM，等待AIFS包WAIT\_AIFS，开始回退计数DEC\_BCKOFF\_CNT，完成回退计数BACKOFF\_DONE。

backoffCtrlCs **------>**回退状态**机**

IDLE**(**0**)** **------>**空闲

LD\_BKOFF\_CNT**(**1**)** **------>载**入BK计数

CHK\_MEDIUM**(**2**)** **------>**检查无线媒介状态

WAIT\_ATFS**(**3**)** **------>**等待AIFS包

DEC\_BCKOFF\_CNT**(**4**)** **------>**开始回退计数

BACKOFF\_DONE**(**5**)** **------>**完成回退计数

如果!(macPhyIfRxCca||channelBusy||txInPress||(!acHasData && (txACState != 4'b0010)))则等待无线媒介，否则进入下一个状态。

逻辑通过(macPhyIfRxCca,channelBusy,txInProgress)对信道情况进行判断

如果PhyRdy就会进行rx\_vector的计数。

//----如果PhyRdy,就会计数，开始检测无线媒介状态，进行rxstate的状态转移

// rx vector counter

**always** **@** **(posedge** mpIFClk **or** **negedge** mpIFClkHardRst\_n**)**

**begin**

**if** **(**mpIFClkHardRst\_n **==** 1'b0**)** // Asynchronous Reset

rx\_vector\_cnt **<=** 5'b0000**;**

**else** **if** **(**mpIFClkSoftRst\_n **==** 1'b0**)** // Synchronous Reset

rx\_vector\_cnt **<=** 5'b0000**;**

// increment vector count each time a vector is sampled

// byt the rx FIFO

**else** **if** **(** rxstateCs **==** RXVECTOR **)**

**begin**

**if** **(**phyRdy**)**

rx\_vector\_cnt **<=** rx\_vector\_cnt **+** 5'd1**;**

**end**

**else**

rx\_vector\_cnt **<=** 5'b0000**;**

**end**

## MAC接收过程

### 帧聚合与FCS校验

1、接收数据首先通过帧解聚合模块，再通过FCS模块，FCS不论通不通过，都会向Rx\_Fifo写数据，如果不通过，会写丢弃描述符到TAG。

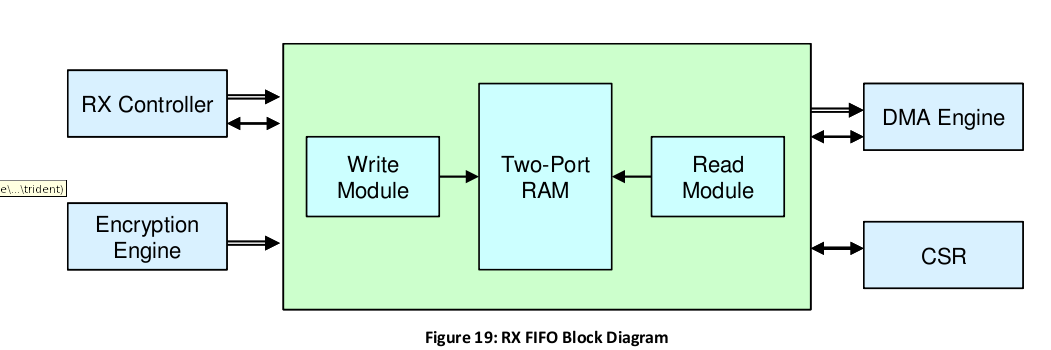
2、如果通过了，在传送TSF和MPDU信息后，Rx\_Vector就会写入到RX FIFO。

3、如果FCS通过后，同时还会进行ACK的更新，NAV根据收到的包类型进行计数更新。

4、Rx Controller会写入TSF，Rx Vector，MPDU Status等到TAG FIFO中。

### RX FIFO

RX FIFO的写入过程受RX Controller和Encryption Engine控制，读出受DMA Engine和CSR控制。RX Controller从MPIF\_FIFO提取物理层传输的数据，根据数据的校验结果写入RX\_FIFO中，同时写入随路控制信号TAG。



### RX CONTROLLER

RX CONTROLLER负责从物理层-链路层接口提取数据，识别MAC头部，数据并分解，将信息写入RX FIFO的TAG中。

#### RX FIFO随路控制

RX TAG FIFO的数据控制DMA Engine的搬移，定义如下：

表 8

|  |  |
| --- | --- |
| 标志意义 | 比特位b3-b0 |
| 存储当前帧 | 0000 |
| MPDU的MAC头部开始 | 0001 |
| MPDU的MAC头部结束 | 0010 |
| MPDU的payload开始 | 0011 |
| MPDU的payload结束 | 0100 |
| MPDU的MAC头部描述符开始 | 0101 |
| MPDU的MAC头部描述符结束 | 0110 |
| 保留 | 0111-1110 |
| 丢弃当前帧 | 1111 |

#### 接收状态机

DMA回到WAIT\_TXRX\_TRIG state进行接收，无论帧是被传送到SRAM还是被丢弃掉，DMA都会按照802.11协议要求工作。

### DMA engine

DMA使用两种DMA描述符进行数据搬移，一种是MAC头部描述符，一种是MPDU描述符。如果DMA发现在RX TAG FIFO 中存在丢弃描述符，则不会更新RX完成比特位，继续从当前地址传输。

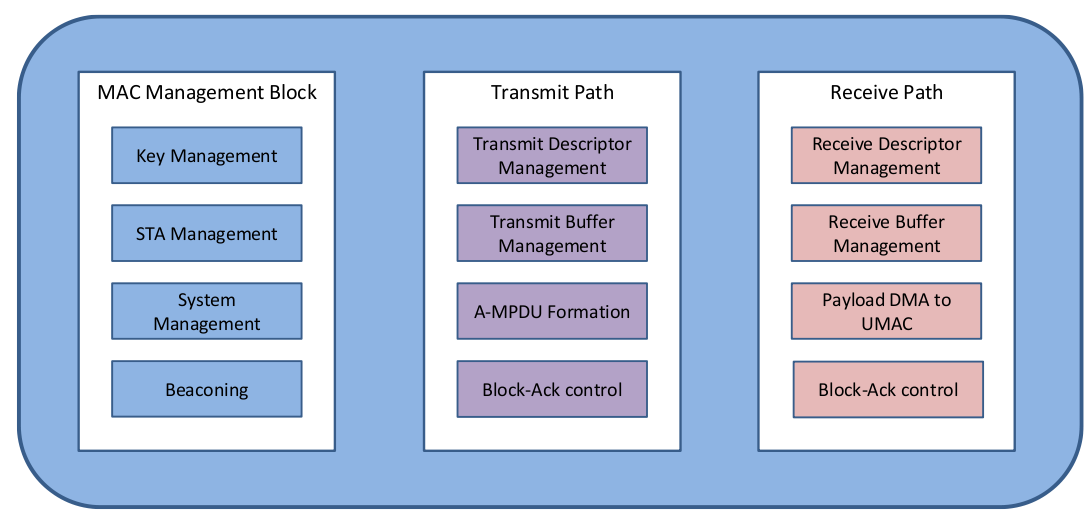
DMA engine在发送过程中通过newHead和newTail进行状态和过程的更新，txListProc是受控制过程。而rxListProc则不是。rxListProc是一直在进行中，它使用statusPointer和currentPointer维护数据接收过程。

#### 调试问题

1. RxFIFOThreshold值从寄存器模块出来没有连接

## LMAC软件

LMAC软件主要包括MAC管理模块，发送通道模块，接受通道模块以及调试模块

。

### MAC管理模块

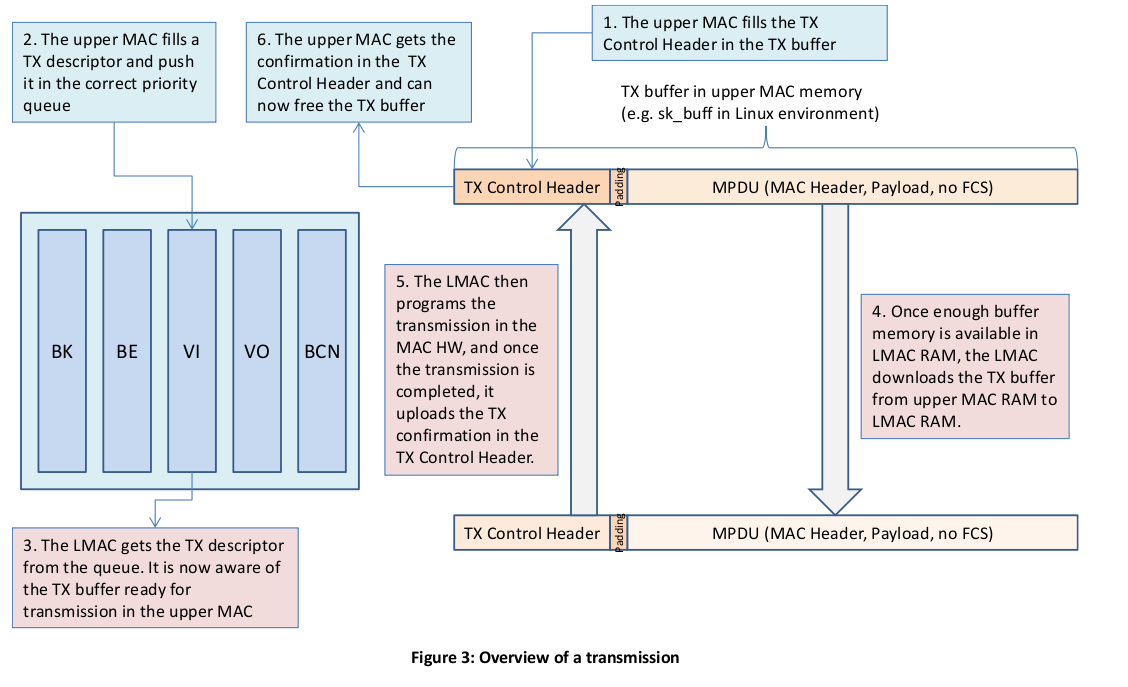
虚拟接口管理，静态参数配置，STA数据集管理。

### 发送通道

创建A-MPDU帧，每次接入种类时请求/释放数据Buffer和HW描述符，通过Dinidma从UMAC收取payload，将HW的描述符链表化，当A-MPDU传输后处理接收到的BA，生成TX的确认到UMAC。

#### 发送过程

1. 从UMAC得到发送描述符；
2. 当应用要求时进行帧聚合A-MPDU；
3. 申请传输缓冲区；
4. 从4个接入种类中得到payload和beacon队列；
5. 更新某些MAC头部和传输头部域；
6. 将传输头描述符队列发送到MACHW
7. 确认发送包到UMAC；



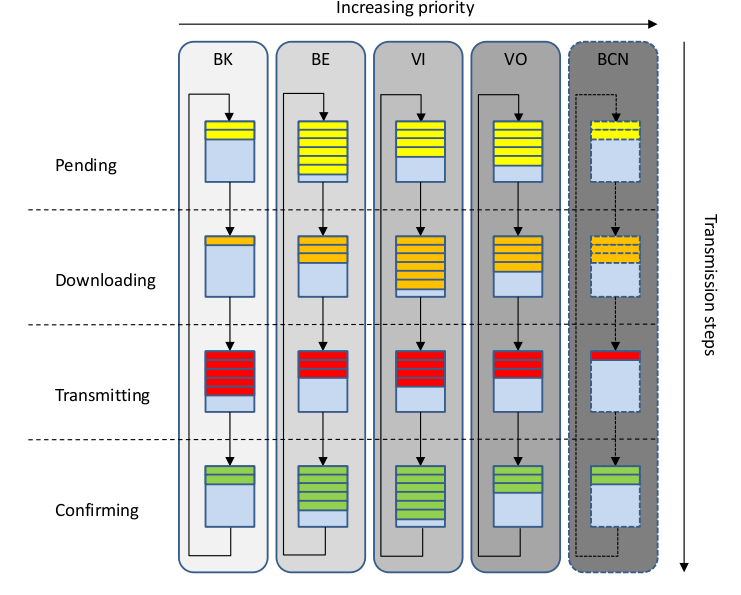
#### 发送数据结构

##### TX 描述符

TX描述符是描述MPDU传输的数据结构，通过UMAC发送到LMAC，然后LMAC在传输过程中使用该描述符，仅受SW控制。包含如下信息：

1. UMAC中的MPDU物理地址；
2. MPDU长度
3. 目的位置序号；
4. 优先级
5. MAC和PHY的传输标志；
6. 序号
7. 申请的数据缓存指针；
8. A-MPUD描述符指针；
9. TX-确认描述符指针；

##### 发送描述符列表

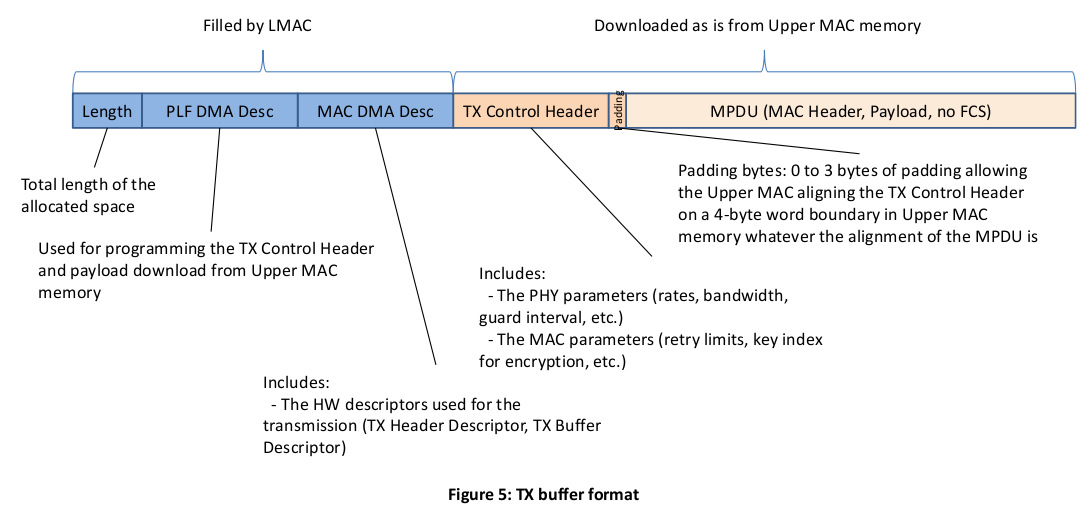


##### A-MPDU描述符

包含如下信息：

##### TX确认描述符

##### TX数据缓冲



#### 发送主要步骤

##### 从UMAC取发送描述符

##### 传输准备事件

##### Payload句柄

##### 传输触发句柄

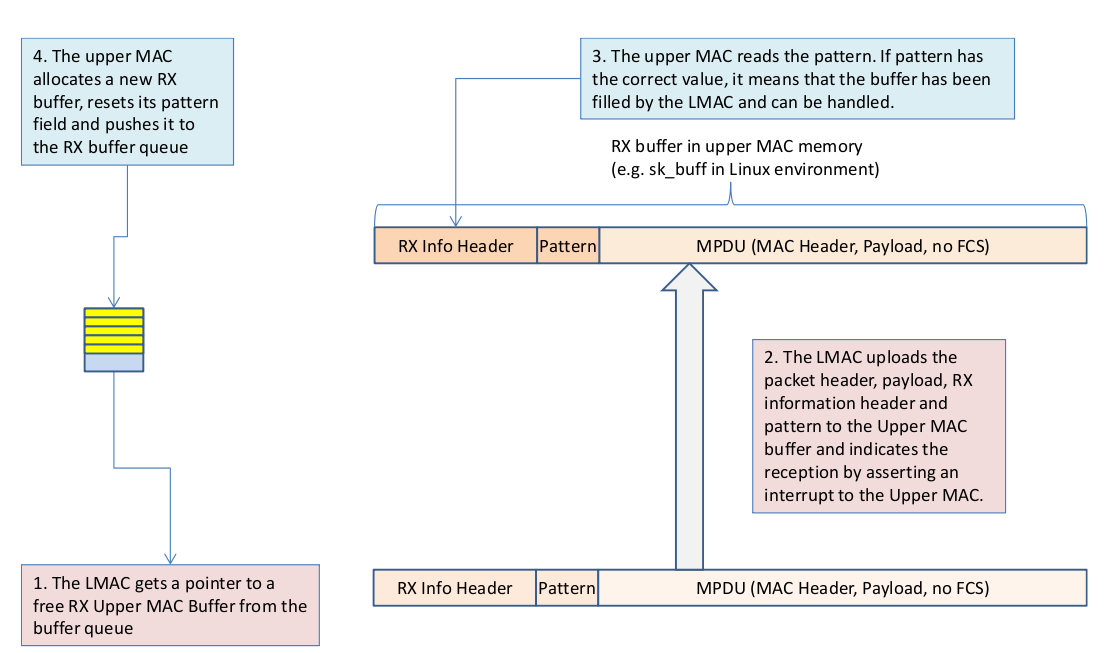
##### 传输确认事件

##### 传输缓冲区管理

### 接收通道

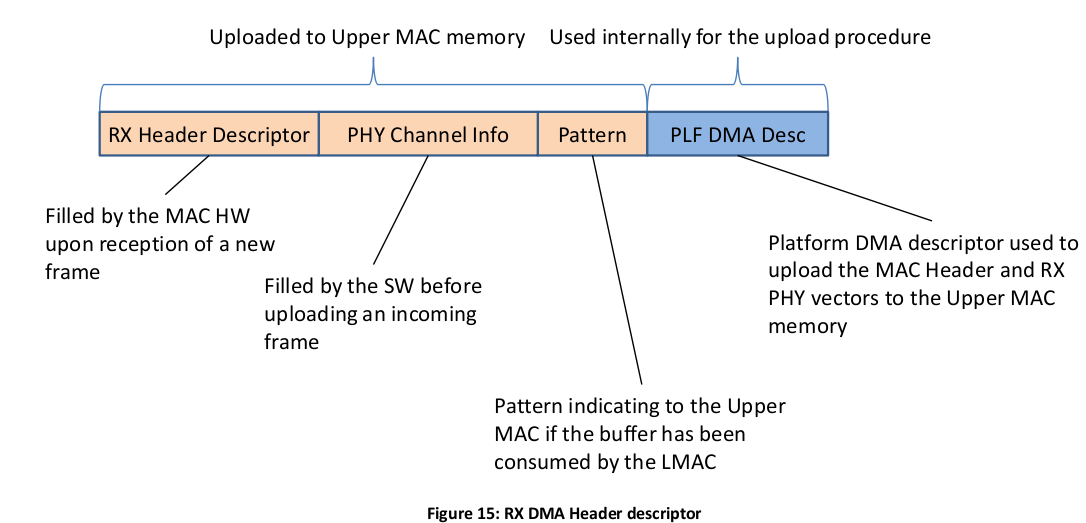
#### 接收概述

该模块响应从MAC HW接受到的包，并通过IPC向UMAC通知。操作流程是：1、LMAC首先得到一个指向UMAC可用Buffer的指针；2、LMAC上传数据到UMAC中并给UMAC一个中断。3、UMAC读取，确认正确后发送句柄处理。4、UMAC释放空间，接收数据。



#### 接收数据结构

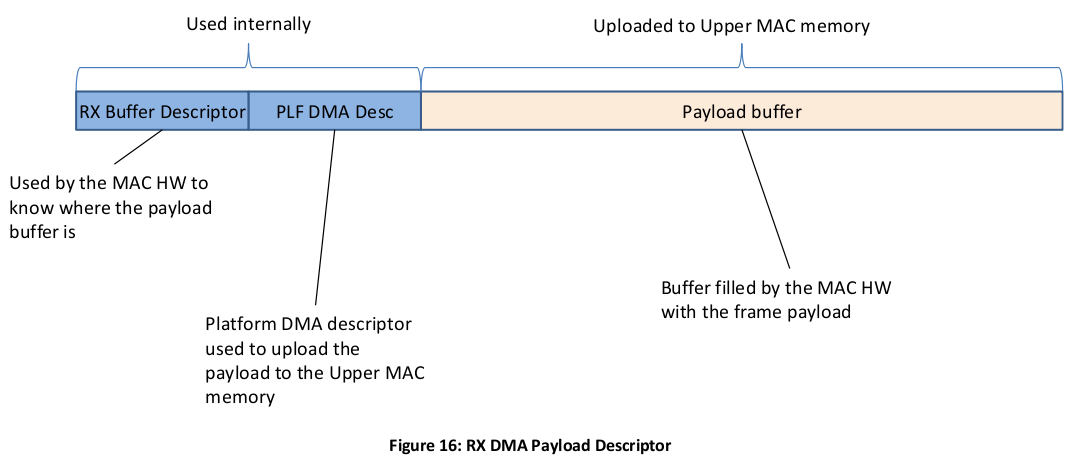
##### 接收DMA头部描述符



RX DMA头部描述符包含所有关于接收的信息，每一包传输只使用一个描述符。一部分RX DMA的描述符即RX Header Descriptor链接成列表并被发送到硬件，当一帧接收到后，该值域被硬件填写。另一部分Rx DMA descriptor(物理层信道信息)被软件填写。

接收头部描述符和物理层通道信息都上传到UMAC缓冲区。Pattern字段为32bit双子用来同步LMAC和UMAC。

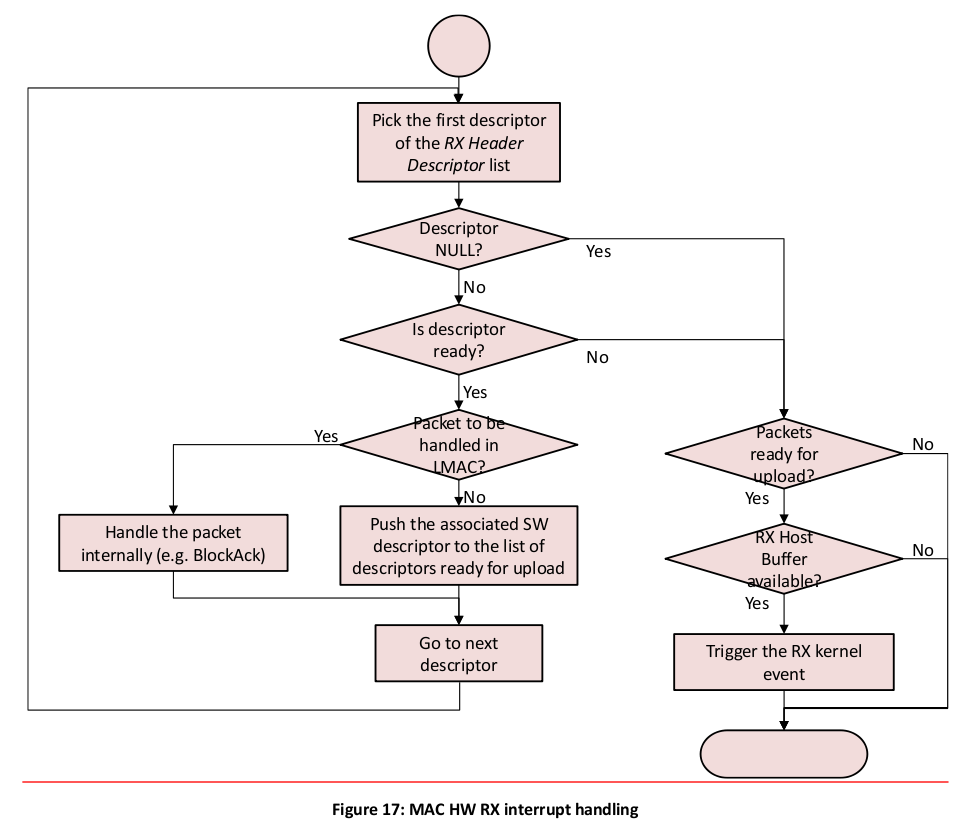
##### 接收DMA payload描述符



RX Buffer Descriptor用来告诉MAC硬件payload缓冲区的位置。该描述符被链接成链表送到MACHW。PLF MDA描述符用来上传payload到UMAC的RX buffer中。Payload buffer被MACHW所填充。一旦帧被确认送到LMAC SW，数据就已经被包含在缓冲区中并被送到UMAC的RX buffer。

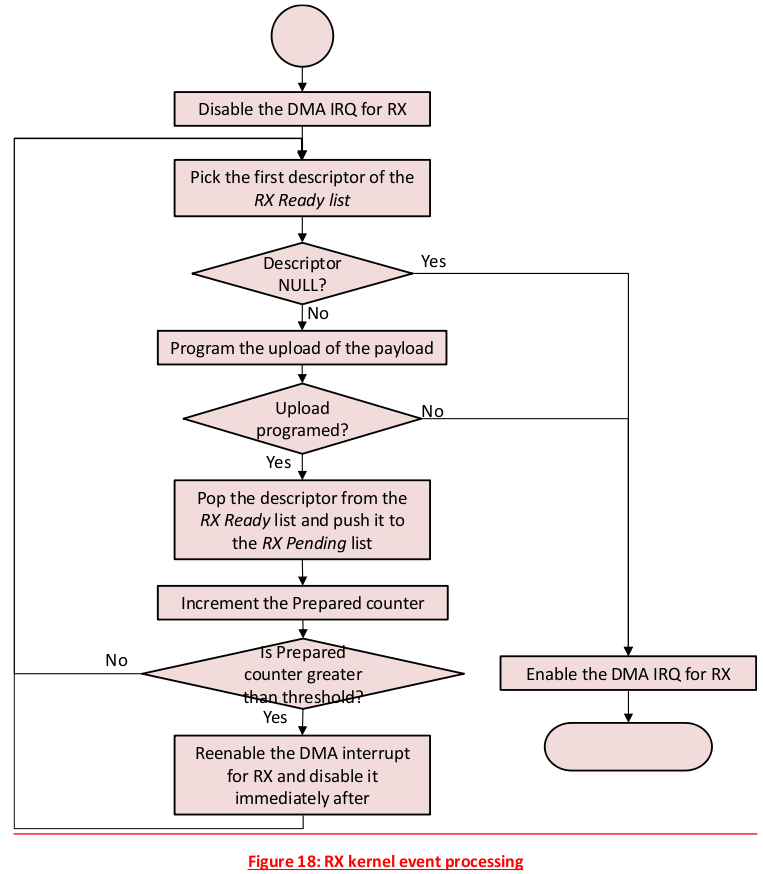
#### 接收主要过程

##### MAC硬件接收中断

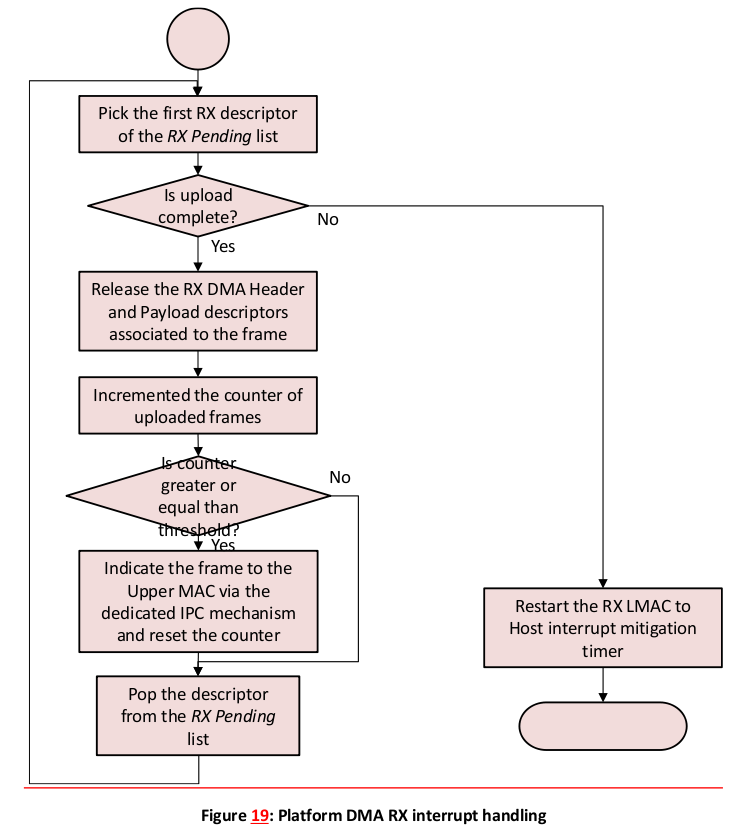


1. MAC硬件中断
2. 接收内核事件
3. Dinidna中断；

##### 接收内核事件



##### DiniDma中断



### 调试模块

# 物理层的调试

对fpga\_clk，clk\_wiz\_1，clk\_wiz\_2，clk\_wiz\_3分别设了异步时钟组，由于clk\_wiz\_1是30M，实际没有使用，因此综合被优化，因此在布局布线约束时候报告找不到clk\_wiz\_1。

## 外部时钟

表 9

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 来源 | 作用 | 驱动 |
| 全局80M时钟 | ARM硬核自带PLL | 供给MAC基时钟，供给软核，供给所有总线，现设计为MPIF接口。 | MAC\_ROOT，MAC\_PI，MB，IOP，  MPIF。 |
| 30M时钟 | DCM IP核 | 原设计为MPIF接口时钟和PCIE的时钟，现未用。 | 未驱动 |
| 240M时钟 | DCM IP核 | PHY的分频源时钟 | 驱动PHY中的div80\_clk，div44\_clk，div60\_clk。 |
| 20.32M时钟 | DCM IP核 | 射频接口时钟 | 驱动回环rx\_tx\_fifo。 |

## 生成时钟

表 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 来源 | 频率 | 作用 | 说明 |
| Div80\_clk | 外部240M时钟 | 80M | 基带A时钟 | 移位分频时钟，占空比1：2 |
| Div44\_clk | 外部240M时钟 | 44M | 基带B时钟 | 移位分频时钟，占空比1：1 |
| Div60\_clk | 外部240M时钟 | 60M | 射频时钟 | 移位分频时钟，占空比1：1 |

# 物理层笔记

* 1. OFDM笔记

MIMO和OFDM是LTE的两大核心技术。信道估计算法和信号检测算法是MIMO-OFDM系统的关键技术。其中信道估计算法对MIMO-OFDM系统接收端的相干解调和空时检测起着至关重要的作用。

* + 1. OFDM的IFFT公式

OFDM是把子载波符号调制到对应子载波频率上的方法，可写为：



子载波个数为N，子载波频率间隔为。为了采用FFT进行变换，令采样率fs为：



则，带入得到：



与标准IFFT公式对比，如果要使用IFFT进行OFDM调制，只需令且=1 ，国标划分20MHz带宽传输64路子载波，则=312.5KHz，=3.2us，则采样率为：



为了对抗多径，添加了循环前缀，因此在802.11a中，OFDM的调制参数应设为：

表 12

|  |  |
| --- | --- |
| 参数 | 数值 |
| 基本采样频率/带宽 | 20MHz |
| 数据符号时间 | 3.2us |
| 循环前缀时间 | 0.8us |
| OFDM符号时间 | 4us |
| 数据子载波数 | 52 |
| 导频子载波数 | 4 |
| 总子载波数 | 64 |
| 子载波间隔 | 312.5KHz |
| 数据子载波占用带宽 | 16.16MHz |
| 数据速率 | 6，9，12，18，24，36，48，54 |
| 调制方式 | BPSK，QPSK，16QAM，64QAM |
| 编码速率 | 1/2，2/3，3/4 |

## 发射机概述

### 扰码

在扰码前已做并转串，扰码模块串入串出，无码率变换。生成多项式为：

-- Input scrambled value for the pseudo-noise generator.

pn\_scrambled **<=** pn\_shift**(**6**)** xor pn\_shift**(**3**);**

-- Scramble the incomming data with the pseudo noise.

data\_o **<=** **(** data\_i xor pn\_scrambled **)**

**when** **(** scram\_enable **=** '1' and tail\_ctrl **=** '0' **)**

**else** data\_i**;**

### 卷积码

802.11a协议中规定卷积编码生成多项式为133和171，码率(有用比特/总比特)为1/2.

生成多项式为：

每输入一比特数据，会输出2比特数据，因此卷积部分固定码率为1/2。

-- For encoding, use industry-standard generator polynomials.

x\_o **<=** data\_i xor shiftreg**(**1**)** xor shiftreg**(**2**)** xor shiftreg**(**4**)** xor shiftreg**(**5**);**

y\_o **<=** data\_i xor shiftreg**(**0**)** xor shiftreg**(**1**)** xor shiftreg**(**2**)** xor shiftreg**(**5**);**

### 打孔

根据总码率决定打孔方式，码率1/2则不打孔，码率3/4则每6bit打2孔(1/(2\*4/6))，码率2/3则每4bit打一孔(1/(2\*3/4))。

发射机总码率由txvector中的调试参数决定，逻辑根据参数决定卷积码方式和打孔方式，使用16QAM码率为3/4。由于使用3/4码率每6比特打2孔(2/3码率只在y数据打一孔)，因此在punct\_dpath中设计了移位寄存器，mux\_sel\_i由上级模块根据码率决定选择输出，根据选择标志进行数据的打孔输出。

-- Y mux is controlled by mux\_sel\_i(0).

y\_mux **<=** y\_i **when** mux\_sel\_i**(**0**)** **=** '0' **else**

x\_i**;**

-- X mux2 is controlled by mux\_sel\_i(1).

x\_mux2 **<=** x\_mux1 **when** mux\_sel\_i**(**1**)** **=** '0' **else**

x\_mux1\_ff**;**

### 交织

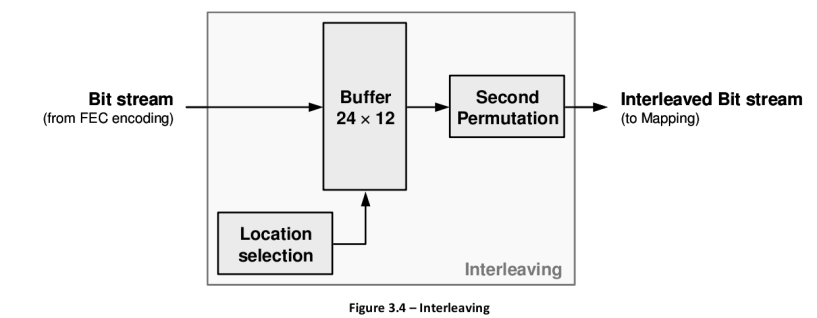


图 7

打孔后的数据进入交织模块，交织模块设计为宽度12bit，深度为24，其中12bit分为两部分，各自占用6bit，这样总共产生2\*24=48组IQ数据进入映射。根据调制方式不同，6bit的选择也不同，以16QAM为例，使用0，1，3，4位bit组合到星座映射IQ数据。

## 接收机概述

### 时域处理

物理层接收机分为时域处理和频域处理两部分。时域处理在FFT之前，首先在前导同步模块中搜索短训练符号并计算第一次频率估计。然后前导同步模块标记长训练序列的开始位置，并计算粗频偏估计。

当计算完粗频偏估计后，从buffer中取数据，校正长训练序列。校正完成后进入精频偏估计模块，然后计算出精频偏估计。精频偏估计送入NCO生成精确频率校正结果。然后将长训练序列再一次送入精频偏校正模块。校正后结果进入符号模块，根据长训练序列移除循环前缀。

当长训练序列第二次进入频偏校正模块时，此刻数据已在SIGNAL域，当长训练序列校正完毕后，再把数据送入频偏校正。

### 频域处理

移除完CP并且通过FFT后，数据转为频域处理，此时从64载波减为48载波，包含4导频和1直流。数据(SIGNAL+DATA)首先进入前导分离模块，当start\_of\_burst信号有效后，进入维纳滤波器，维纳滤波器生成第一次信号估计，送入后级均衡器和软bit生成器。后继的符号均为数据符号。

均衡器和软bit生成器生成软bit。该算法需要知道速率，由SIGNAL域的速率字段给出。

随后软bit被送入信道解码器，完成解交织，解打孔和维特比译码。完成维特比译码后，速率字段可解出，并送入前级均衡和软bit生成器。最后数据送入解扰模块后进入MAC。

同时，导频用来跟踪残余相位偏差，用来校正星座图的映射和补偿频率偏移。

## 同步

802.11a中的同步主要包括定时同步、载波频率同步和采样时钟同步。

定时同步主要包括帧(分组)同步和符号同步两种，其中帧(分组)同步用于确定数据分组的起始位置，而符号同步目的在于正确的定出OFDM符号数据部分的开始位置，以进行正确的FFT操作。

载波频率同步首先是要检测出频率偏移，然后加以补偿。频偏检测按精度要求可分为粗同步和细同步两个部分。

采样时钟同步的目的在于消除接收端A/D采样的频率、相位与D/A时钟频率，以及相位的偏差对系统性能造成的影响。

### 分组检测

分组检测是寻找数据分组起始地近似估算，是接收机工作的第一步。

常用分组检测方法：

1. 接收信号能量检测

通过滑动窗口求取共轭相乘再窗口内累加结果达到，监控一段时隙内的无线信号能量。

1. 双滑动窗口分组检测
2. 采用前导结构进行分组检测

采用延时相关算法进行相关检测简单有效，方法是通过利用短训练符号的周期性。



图 8 延迟相关一般框图

利用延迟相关算法，可以有效区别噪声和OFDM有效信号，注意除法和绝对值的简化处理。

#### 延迟自相关算法

延时相关算法基本公式：

(8-1-1)

(8-1-2)

(8-1-3)

其中，为接收信号样值，L为短训练符号长度，滑动窗口C求得互相关系数，D为接收信号延时采样点数(16)，滑动窗口P计算接收信号能量，为帧检测的判决函数。

为了避免除法运算，而且滑动窗口求和困难，因此将延迟相关值做如下推导：

(8-1-4)

将接收信号能量做如下推导：

(8-1-5)

得到移动递归公式，不仅简化了延迟相关自相关算法的乘加运算，而且更容易硬件实现。采用该算法的电路框图如下：



图 9 延迟相关值计算框图



图 10 接收信号能量计算框图

在延迟相关算法里，采用量化后的复数数据根据正负选择原数据的加减组合得到复数乘法结果，取反是取复数数据的共轭，利用共轭进行复数相关。关键代码如下：

--IQ加减组合

iq\_add **<=** signed**(**i\_inbd\_sextd**)** **+** signed**(**q\_inbd\_sextd**);**

iq\_add\_comp **<=** "000000000000" **-** iq\_add\_int**;**

iq\_sub **<=** signed**(**i\_inbd\_sextd**)** **-** signed**(**q\_inbd\_sextd**);**

iq\_sub\_comp **<=** "000000000000" **-** iq\_sub\_int**;**

--取符号位量化

iq\_ff\_mux\_sel **<=** i\_ff\_sel**(**i\_ff\_sel'**high)** **&** q\_ff\_sel**(**q\_ff\_sel'**high);**

**case** iq\_ff\_mux\_sel **is**

--根据符号位选择加减组合合成IQ

**when** "00" **=>**

i\_add\_in1\_int **<=** iq\_ff\_add**;** --new R = i+q

q\_add\_in1\_int **<=** iq\_ff\_sub\_comp**;** --new I = -i+q

**when** "01" **=>**

i\_add\_in1\_int **<=** iq\_ff\_sub**;** --new R = i-q

q\_add\_in1\_int **<=** iq\_ff\_add**;** --new I = i+q

**when** "10" **=>**

i\_add\_in1\_int **<=** iq\_ff\_sub\_comp**;** --new R = -i+q

q\_add\_in1\_int **<=** iq\_ff\_add\_comp**;** --new I = -i-q

**when** **others** **=>**

i\_add\_in1\_int **<=** iq\_ff\_add\_comp**;** --new R = -i-q

q\_add\_in1\_int **<=** iq\_ff\_sub**;** --new I = i-q

**end** **case;**

这样分别得到接收信号能量ca\_rl和延迟相关值ca\_ac。得到ca\_rl和ca\_ac之后还需要进行阈值判决，该处理过程在AGC模块中的ofdm\_det，该模块对输入的ca\_rl和ca\_ac进行阈值相乘计算，并进行比较大小，以此判决帧同步。为了避免判决过程中的除法运算，将公式做如下变换：

(8-1-6)

即当相关值大于能量值乘以阈值时，检测到OFDM帧，否则没有OFDM数据帧到达。逻辑实现关键代码如下：

--阈值相乘

ca\_rl\_ac\_mult\_i **<=** rnd\_unsigned\_slv**((**unsigned**(**ca\_rl**)** **\*** unsigned**(**thr\_ac\_i**)),** 5**);**

--比较判决

**if** **(**en\_20m **=** '1' and **(**unsigned**(**ext**(**ca\_ac**,** ca\_rl\_ac\_mult\_i'**length))** **>** unsigned**(**ca\_rl\_ac\_mult\_i**)))** **then**

ac\_det\_d **<=** '1'**;**

**else**

ac\_det\_d **<=** '0'**;**

**end** **if;**

#### 互相关运算

为了正确检测数据段起始位置，仅仅帧同步显然不够，还需完成符号同步算法，当完成帧同步延迟相关算法后，采用互相关算法得到相关峰位置。实际做法是将量化后接收信号存入移位寄存器中，由移位寄存器来实现接收信号的延时。然后分别与本地存储的16个短训练符号的共轭值一一相乘，因为已完成量化，因此复数乘法已可以简化。电路实现框图是：



图 11 互相关算法框图

在ofdm\_cross\_corr模块中完成复数相乘累加，取模，按16时钟输出。关键代码如下：

mod\_approx\_1 **:** mod\_approx

**generic** **map** **(**

data\_size\_g **=>** 16**,**

num\_ct\_g **=>** 4

**)**

**port** **map** **(**

data\_in\_i **=>** add\_out\_i**,**

data\_in\_q **=>** add\_out\_q**,**

--

data\_out **=>** ca\_cc\_int

**);**

-- Mux output

ca\_cc\_mux **<=** **(others** **=>** '0'**)** **when** **(**first16\_sample **=** '1'**)** **else**

ca\_cc\_int**;**

同样在ofdm\_det模块中进行阈值判定，由于帧同步时ca\_rl为一台阶信号，因此用互相关值ca\_cc首先乘以阈值，再和ca\_rl比较，若大于ca\_rl则判定为互相关峰，得到最后一个互相关峰时输出到后级进行数据段符号同步。关键逻辑代码如下：

**if(**en\_20m**=**'1'and**(**unsigned**(**ext**(**ca\_cc**,**ca\_rl\_cc\_mult\_i'**length))>**unsigned**(**ca\_rl\_cc\_mult\_i**)))** **then**

cc\_det\_d **<=** '1'**;**

**else**

cc\_det\_d **<=** '0'**;**

**end** **if;**

#### 分组检测判决

得到延迟自相关结果和互相关结果后，软件根据实际情况设置帧同步的结果，帧同步分为混合模式和单一模式，单一模式只根据延迟自相关或者互相关结果，混合模式综合考虑自相关结果和互相关结果。

-- OFDM Detection output mixing setting

-- OFDM 检测模式选择，根据寄存器选择只用延迟相关算法还是混合算法

**if** **(**det\_mix\_i **=** '0'**)** **then**

**if** **(**ac\_det\_q**=**'1' or cc\_peak\_i**=**'1'**)** **then** -- Detection

det\_i **<=** '1'**;**

**else**

det\_i **<=** '0'**;**

**end** **if;**

**else**

**if** **(**ac\_det\_q**=**'1' and cc\_peak\_i**=**'1'**)** **then** -- Detection

det\_i **<=** '1'**;**

**else**

det\_i **<=** '0'**;**

**end** **if;**

**end** **if;**

电路框图如下：

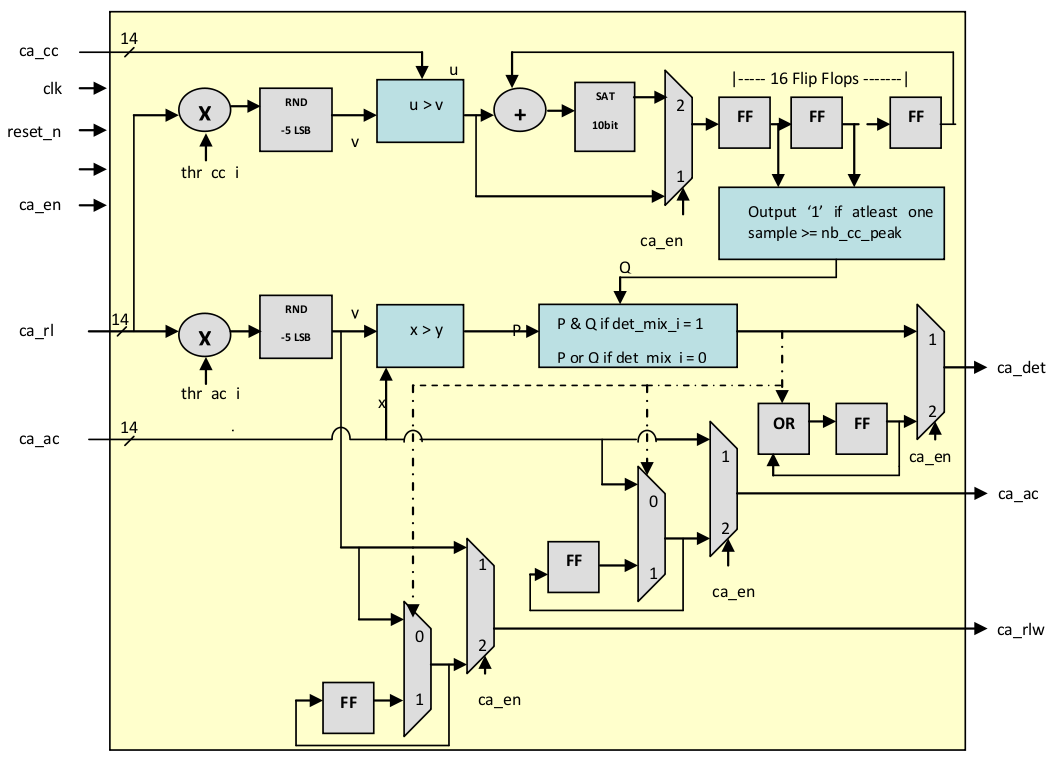


图 12

当前模式不使用快速模式，ca\_en为1。

### 载波同步

载波同步采用粗细结合的方式。首先利用短训练符号进行粗略估计频率误差，然后采用该估算对长训练符号进行修正，修正后再采用长训练符号进行改进。

#### 粗同步

在init\_sync中，已经做了基带数据与短训练序列B，循环前缀CP1，循环前缀CP2的相关计算，在载波同步的时候，根据短训练序列，CP1，CP2的符号间频偏进行载波粗同步。具体做法是：

1. 找到最后三次的短训练相关峰并使用cordic算法计算角度。
2. 找到CP1的相关峰位置并计算角度。
3. 将结果放在xp\_buffer中。

**type** GET\_CORDIC\_TYPE **is** **(**idle**,** -- wait for a f\_position

calc\_cp1\_cordic**,** -- calculate cordic for CP1

calc\_b\_cordic**,** -- calculate cordic for B

wait\_for\_init**);** -- wait for new reception

1. 得到相位后，首先进行相位解缠恢复相位到正负无穷，然后进行根据相位拟合斜率估计出频率均值，最终由斜率换算出频率得到粗频偏估计。

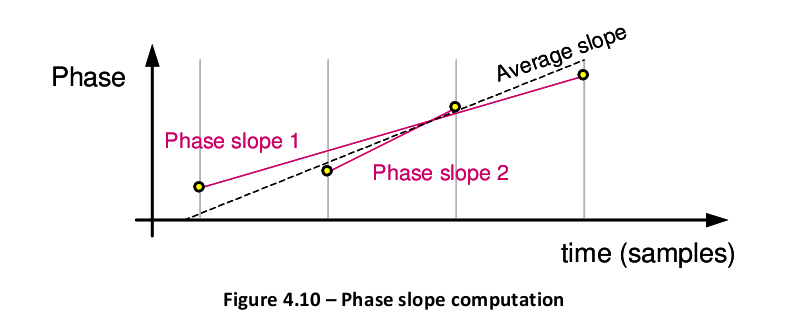


图 13

1. 得到粗频偏估计值后，将输入的I，Q数据分别与旋转因子相乘，得到校正后的粗频偏补偿值。

#### 频偏校正与细频偏校正

粗频偏校正值首先会进入频率校正模块，一旦有效后，模块就会对两个长序列进行校正，当start\_of\_burst信号有效后，64个采样值就会处理完，一次start\_of\_burst接收一次粗频偏校正值，start\_of\_busrt由采样FIFO产生。

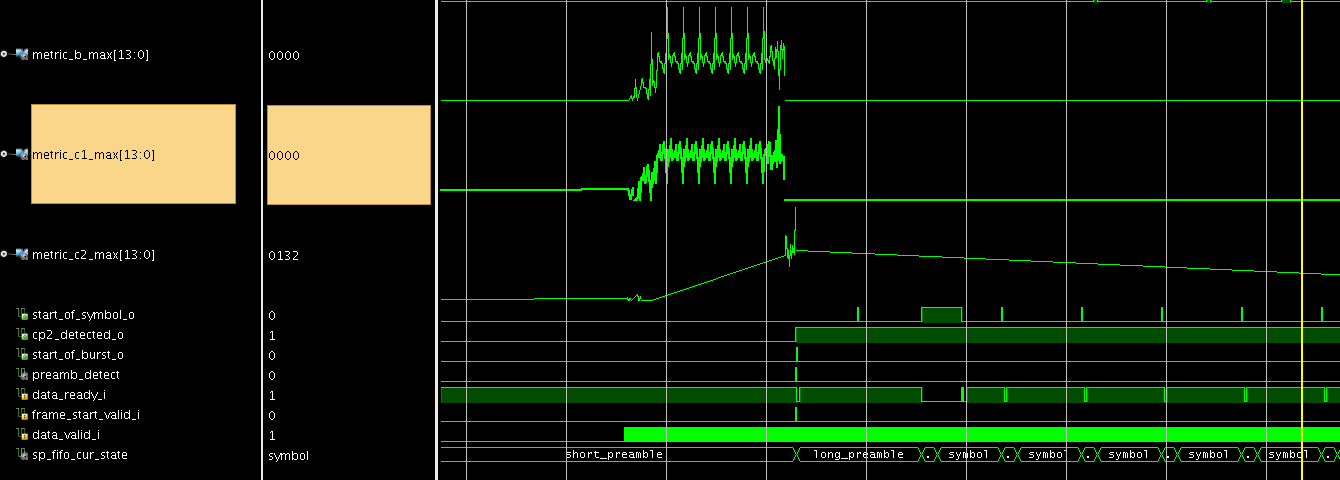


图 14

长序列在粗频偏校正完之后，会进入到细频偏估计模块。细频偏估计会计算出细频偏估计值并反馈到频率校正模块。该估计值是对粗频偏估计的一次再修正。修正的时候通过移位寄存器跳过保护间隔的16个采样点。

在细频偏校正后，长序列被送入信道估计模块，随后的符号都直接由细频偏校正模块校正。

### 符号同步

符号同步的做法是将载波同步模块的输出与本地已知的短训练符号做互相关，从而将分组检测模块对数据分组的初略估算进行进一步的精确估算，确定短训练符号的结束点或者长训练符号的起始点。

在逻辑代码里，是确定长训练符号的起始点preamb\_detect，是一个脉冲信号，cp2\_detected与之同时出现，为一个电平信号，指示前端CCA可以停止帧搜索。

#### 长训练符号preamb\_detect起始点的获取

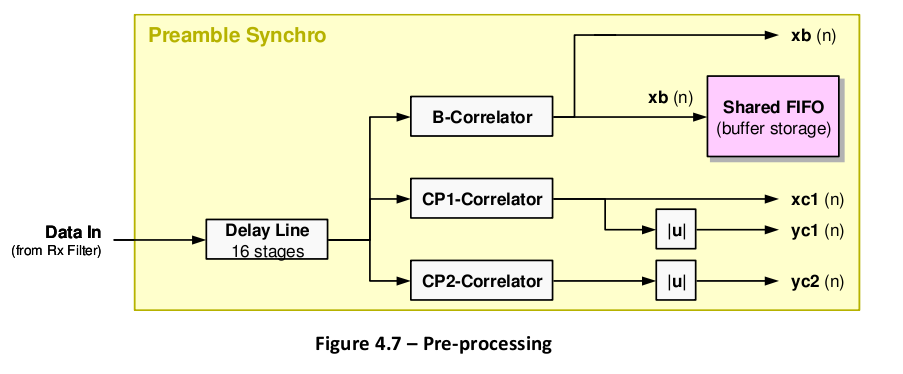
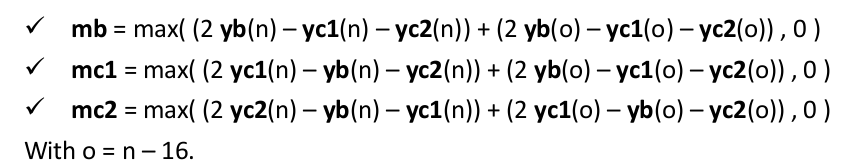


图 15

电路上的做法是在Pre-processing模块中，将基带数据按16进行滑动分别与短训练、循环前缀CP1，循环前缀CP2进行互相关，得到xb(n)，yc1(n)，yc2(n)，这样出现的结果是按时间进行下去依次出现9个短训练相关峰，1个CP1相关峰，1个CP2相关峰。

为了提高判决准确性稳定性，算法进行了相互的差值计算：



从判决相关值，转为判决相关值的差值最大值。

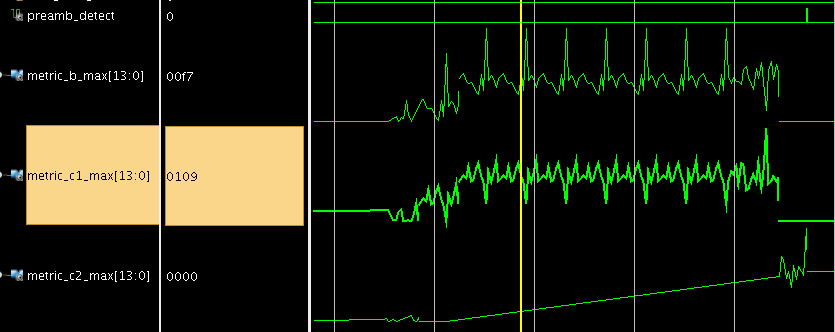
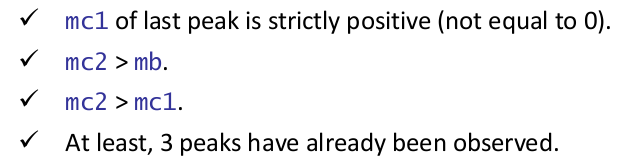


图 16

显然，当到达CP2末尾时，mb，mc1，mc2的大小关系应是mc2最大，此时判决preamb\_detect有效，preamb\_detect即是帧开始有效标志。但光有该条件可能存在误判，因此综合mc1和mb的结果，当cp2完结的时候，条件应为：1、mc2大于mc1和mb，且上一次的mc1(n-1)为正值。



#### 传输开始和符号开始标志

Start\_of\_burst和start\_symbol分别标志标识传输开始和符号开始标识，由sample\_fifo模块产生，该模块主要根据preamb\_detect信号去移除保护间隔，产生传输开始和符号开始两个标志。

模块内含状态机，设计有为移除循环前缀的缓冲和输出模块。状态机内，根据计数器计数决定状态的转移。关键代码如下：

**when** short\_preamble **=>**

**if** frame\_start\_valid\_i **=** '1' **then**

sp\_fifo\_next\_state **<=** long\_preamble**;**

**end** **if;**

-- count 128 (T1-T2 are sent)

**when** long\_preamble **=>**

**if** data\_valid\_i **=** '1' and cnt\_rs **=** T1\_T2\_LENGTH\_CT **then**

sp\_fifo\_next\_state **<=** guard\_interval**;**

**end** **if;**

-- GUARD INTERVAL : ignore 16 data

**when** guard\_interval **=>**

**if** data\_valid\_i **=** '1' and cnt\_rs **=** GI\_LENGTH\_CT **then**

sp\_fifo\_next\_state **<=** symbol**;**

**end** **if;**

-- Symbol is sent : data are sent transparently

**when** symbol **=>**

**if** data\_valid\_i **=** '1' and cnt\_rs **=** SYMBOL\_LENGTH\_CT **then**

sp\_fifo\_next\_state **<=** guard\_interval**;**

**end** **if;**

其中frame\_start\_valid即是preamb\_detect，T1\_T2\_LENGTH\_CT，GI\_LENGTH\_CT，SYMBOL\_LENGTH\_CT分别为127，15，63，data\_valid\_i由IQ补偿模块产生。

当数据传输初始化后init\_i=1，一旦数据有效，下一状态立刻转移到symbol，若上一状态为空闲，则可判断进入burst，使start\_of\_burst信号有效，开始载波同步和解码。计数值cnt64每64个输出一次start\_of\_symbol信号提示OFDM符号有效。

**if** init\_i **=** '1' **then**

**elsif** out\_modes\_cur\_state **=** idle and out\_modes\_next\_state **=** symbol **then**

-- start\_of\_burst = '1' when leaving idle state

start\_of\_burst **<=** '1'**;**

**elsif** data\_ready\_i **=** '1' **then**

start\_of\_symbol **<=** '0'**;**

**if** data\_ready **=** '1' and data\_valid **=** '1' and cnt64 **=** CNT\_MAX63\_CT **then**

-- start\_of\_symbol = '1' when the 64 symbols have been sent

start\_of\_symbol **<=** '1'**;**

data\_valid **<=** '0'**;** -- new symbol is arriving

* + - 1. 符号同步偏移

由于多径会导致定时偏差，因此设置了符号同步偏移，使FFT取序列窗口往前移动，只要定时起点在保护间隔中就不会引起严重干扰。

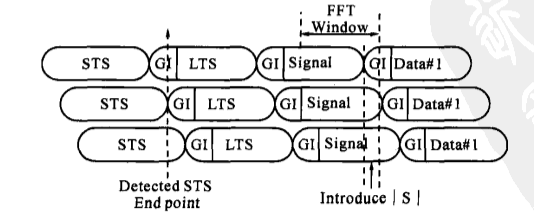


图 18 符号同步偏移影响

在逻辑代码中设置了符号同步偏移量timoffst\_i，初始化值为5个点。为了增加CP长度，减小多径影响，在寄存器配置时设为了2个点。

initsync\_timoffst **<=** "101"**;**

### 采样时钟同步

## 信道估计算法

* + 1. 信道估计概述

图 19

信道估计的目的就是得到信道的响应特征，在接收机中恢复估计出一个逆信道，以此再估计出原始信号。

实用的信道估计方法往往采用数据辅助方式即训练序列和导频进行信道估计，基于训练序列的估计算法主要有LS，MMSE，奇异值分解，维纳滤波算法。基于导频的信道估计算法主要有一维和二维两种方式，一维主要有线性，二阶内插，Cubic算法，DFT插值算法，DCT插值。二维主要使用维纳滤波插值。其中对导频处子载波信道响应的计算主要使用最小二乘(LS)，最小均方误差(MMSE)等。

在802.11A中，短序列和长序列可以被认为是既是梳状导频也是块状导频，由于在每个子载波上已知频域值，因此用来完成信道估计的初始化。而协议在±21，±7子载波处插入了随数据帧的导频，可以看作随路信号，用来跟踪信道的变化，完成相位校正(CPE)和采样时钟同步即符号偏移校正(STO)。

802.11A协议中，导频的分布方式是梳状导频，梳状导频间隔满足下式，则可恢复其余信道。是最大多径时延。



* + 1. 信道初步估计

根据长序列进行信道估计的初始化。在时域首先分离两个长序列，并进行平均，通过FFT后得到长序列的频域表达 ，由于已知理想长序列的频域表示，因此求出信道的频域表达作为信道估计的初始值。

* + - 1. 最小平方LS算法

最小平方算法计算量最小，但是受噪声影响较大，信道模型为：



根据最小平方算法，可得信道估计为：



 是长序列频域值，根据802.11A协议52个子载波和直流处频域值为：[1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,-1,-1,1,1,-1,1,-1,1,-1,-1,-1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,1,1]

在逻辑代码中对正负进行判断，如果是负则对Y(Pn)取反则完成信道H的计算，如果是正值则不变：

**if** DEMOD\_CT**(**conv\_integer**(**count\_data**))** **=** '0' **then**

tmpi\_v **:=** not**(**tmpi\_v**)** **+** '1'**;**

tmpq\_v **:=** not**(**tmpq\_v**)** **+** '1'**;**

**end** **if;**

其中DEMOD\_CT是预定义的长序列频域值由高到低排序并对-1取0值得到的，tmpi\_v则是频域输入数据：

--长序列频域值

**constant** DEMOD\_CT **:** std\_logic\_vector**(**52 **downto** 0**)** **:=**

"11110101001100000101011001011110101100111111010110011"**;**

这样只需根据长序列的正负对FFT后的数据进行取反操作即可得到信道的初始估计值。

* + - 1. 最小均方误差LMMSE算法与维纳滤波

利用LS算法进行估计的时候，由于是忽略了噪声系数的影响，因此实际信道估计值是有偏的，如果系统信噪比低则恢复的信号会带有较大噪声，而维纳滤波是均方意义下的最佳滤波器，因此在代码实现中设计维纳滤波器满足Q矩阵的要求，在信道估计初始值后进行第二次滤波，完成去噪过程

最小均方误差算法性能比LS算法好，根据最小均方差准则算法复杂，表达式为：



基于MMSE准则的算法可以看作是在LS估计结果的基础上再利用相关矩阵W进行滤波处理。



在式中，主要未知量是信道相关矩阵，可进行推导，最终可以由预设矩阵进行替代节省大量计算，信道模型为：



时间，频率上相隔， 的两点信道响应之间的相关性为：



因此只需单独计算时频相关函数：





在对频域相关函数做简化变换，在实际应用中，都是预先设定信道的相关矩阵，就不必计算信道相关矩阵，虽然实际信道可能不同，影响估计效果，但有研究表明，实际的相关性失配是非常小的。

然后再用 代替避免每次计算，损失一定性能换取较小计算量，设计出次优的MMSE估计器：



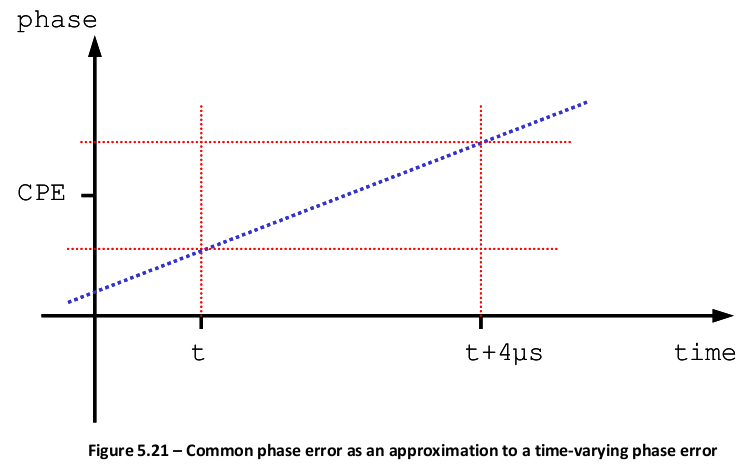
为调制系数，为平均信噪比，I为单位矩阵。 当使用QAM16调制时， 为16/9，如果是QPSK则为1。

### 信道跟踪

### 卡尔曼滤波

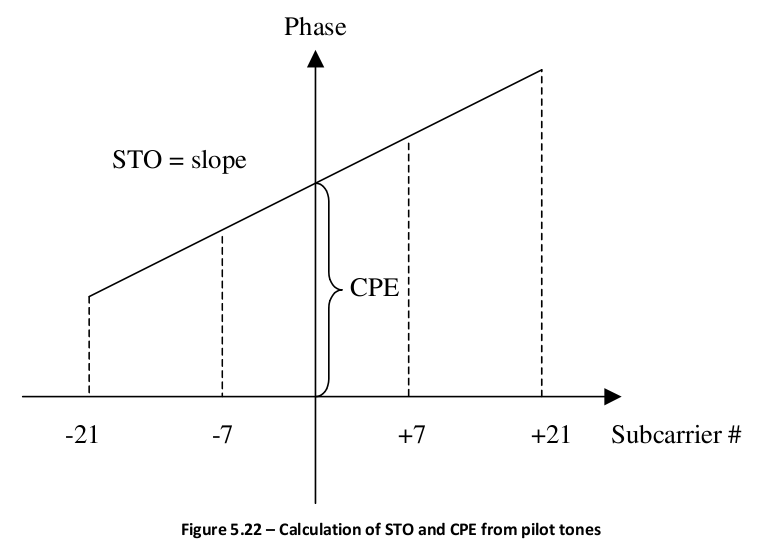
#### 载波剩余相位跟踪

载波剩余频偏会导致子载波的相位旋转，并且具有累计效应，也就会导致相位无法正确映射到星座图上。这种相位误差对所有子载波都是一样的，需要估计出来进行补偿。



#### 符号定时偏差

收发本振振荡器的振荡频率不同将会导致在收发过程中符号周期的偏移，也就是符号定时偏差(STO)。当一帧开始的时候，不会有累计误差出现，但是如果OFMD符号较多，那么时间向后持续时就会出现由于时钟偏移导致的采样定时点偏移，如果起始点超过保护间隔到了数据段中，就会产生严重的符号间干扰同时也会导致信道估计的错误。



#### STO和CPE的跟踪

如上所述，STO和CPE的跟踪每一个符号跟踪一次，计算步骤是：

1. 根据信道估计的相位计算导频点所在的相位值。
2. 根据导频的相位值提取出STO和CPE。
3. 使用卡尔曼滤波器对STO和CPE进行估计滤波。

##### 导频计算

##### STO和CPE的提取

根据导频对应相位进行直线拟合，



估计的直线表达式为：，其中，，，分别为-21，-7，+7，+21，STO和CPE为两未知量。

根据均方最小原则：



为了达到均方最小，对X求导得到：



为了再减小估计误差，令W为相位绝对值权值矩阵，



对估计的X进行一次修正为：



令，则，由于：





M的逆矩阵为：



参数为：



因此估计的STO和CPE为：



##### STO和CPE的跟踪

以卡尔曼滤波进行估计，经典卡尔曼滤波公式为：

测量值 

卡尔曼增益 

测量值与观测值差 

新估计值 

滤波均方误差更新矩阵 

即以观测值与估计值差值作为调整系数进行递推，观测值为Z，估计值为X，则新估计值为X`=X+K(Z-X)=X(1-K)+KZ，即估计值权重1-K，观测值权重K，K的取值看方差情况。

其中状态转移方程由于没有经验，因此认定上一个OFDM符号和当前的符号STO和CPE是一样的。





##### 卡尔曼实现

定义测量值为：



是STO的当前估计，是每一个符号的STO估计变化。是CPE的当前估计，是CPE的每符号的估计变化。

增益矩阵K为：



其中，都表示协方差矩阵。

增益矩阵可简化为：



因此每个符号的卡尔曼公式为：



I为误差更新，Z为根据加权卡尔曼增益最后得到X滤波输出。

## IQ信号

IQ信号是为了解决双边带调制带来的频谱浪费，通过IQ调制达到了单边带的作用。

一般IQ信号为：

写为复数形式为：

## IQ不平衡

IQ不平衡是指I，Q两路由于NCO或者传输信道的影响，造成两路并不是90°相交的。

## AGC-CCA模块

AGC-CCA模块根据AGC的状态进行状态机转移，帧同步输入控制信号为：

flag\_vector **<=** cs\_a\_high **&** cs\_a\_low **&** cs\_b\_high **&** cs\_b\_low**;**

根据帧同步向量值flag\_vector进行状态的转移，当cca\_fsm处在ofdm帧搜索状态时，如果vector高二位为1则说明分组检测有效，AGC中的延迟相关算法或者互相关算法有效，ofdm帧已到达，关键代码如下：

-- OFDM signal search 1

**when** ofdm\_search1\_st **=>**

**elsif** cs\_flag\_valid **=** '1' and cs\_flag\_nb **=** "10" **then**

-- Flag 2 = cs\_a\_high | cs\_a\_low -> continue OFDM

**if** flag\_vector**(**3**)** **=** '1' or flag\_vector**(**2**)** **=** '1' **then**

next\_cca\_state **<=** ofdm\_search2\_st**;**

**end** **if;**

随后进入ofdm搜索状态二，启动接收机中的peak\_detect过程，搜索CP2的结束位置，如果检测到CP2，进入wait\_ofdm\_header\_st状态，关键代码如下：

-- OFDM signal search 2

**when** ofdm\_search2\_st **=>**

-- Modem has found the sync

**elsif** cp2\_detected **=** '1' **then**

next\_cca\_state **<=** wait\_ofdm\_header\_st**;**

该状态根据是OFDM还是DSSS会有不同的状态跳转。如果是OFDM则进入OFDM头部(速率、长度)等待状态，如果phy\_rxstartend\_ind有效则说明OFDM头部译码完成状态进入到数据开始收取状态。

-- The modem is decoding the header

**when** wait\_ofdm\_header\_st **=>**

-- Header is received

**if** phy\_rxstartend\_ind **=** '1' **then**

next\_cca\_state **<=** start\_reception\_st**;**

其中phy\_rxstartend\_ind信号由接收机中的状态机控制，当接收机完成SIG域的数据解码时，根据signal\_field\_valid信号切换到数据接收状态。

**if** signal\_field\_valid\_i **=** '1' **then** -- signal field valid

rx\_start\_end\_ind **<=** '1'**;**

signal\_field\_valid是由channel\_decoder\_control产生，SIG域和DATA域均在频域进行处理，其中SIG域为BPSK解码，DATA域是QAM解码，根据信道解码控制状态机产生SIG域有效信号。

**type** CONTROL\_STATE\_T **is** **(**IDLE**,**

SIGNAL\_START**,**

SIGNAL\_DECODE**,**

DATA\_START**,**

DATA\_DECODE**);**

首先根据在init\_sync模块中产生Cp2\_detected，当计算时间已经过长训练序列后，给出start\_of\_burst\_i信号，启动signal和data域的解算，使能signal域解算模块。

**when** **others** **=>**

**if** start\_of\_burst\_i **=** '1' **then**

control\_next\_state **<=** SIGNAL\_START**;**

**else**

control\_next\_state **<=** IDLE**;**

**end** **if;**

**if** start\_of\_burst\_i **=** '1' **then**

control\_next\_state **<=** SIGNAL\_START**;**

**else**

control\_next\_state **<=** SIGNAL\_DECODE**;**

**end** **if;**

使能signal域解算模块后，需要等待解算完成，如果完成，信号解算模块会给出signal\_filed\_valid信号，提示解算完成，状态机收到后转入data段的解算。

--如果已将18个SIG域数据接收解算完成则进入DATA段解算

**elsif** signal\_field\_valid\_i **=** '1' **then**

control\_next\_state **<=** DATA\_START**;**

## OFDM笔记

MIMO和OFDM是LTE的两大核心技术。信道估计算法和信号检测算法是MIMO-OFDM系统的关键技术。其中信道估计算法对MIMO-OFDM系统接收端的相干解调和空时检测起着至关重要的作用。

MIMO技术在平坦衰落信号环境下会增加系统的容量和频谱利用率。但是在频率选择性多径衰落信道环境下，多输入多输出技术 的实现需要更复杂的信道均衡技术，使得接收机的复杂度大大增加。而正交频分复用将信号分布到若干并行窄带子信道上，在子信道上可以认为是平坦衰落，这样就有效地对抗了频率选择性衰落。

## 循环前缀

FFT作为OFDM的核心部件使用，在这种用法下，通常需要将FFT结果的一部分复制放到最前面，形成循环前缀(CP)。FFT结果和CP共同组成一个OFDM符号。

在IEEE802.11a/g标准中，复制1/4到

# UVM笔记

## UVM中的Phase

在不同的时间做不同的事情，这就是UVM中phase的设计哲学。UVM中常用的phase如下：

表 12

|  |
| --- |
| UVM常用phase |
| Build\_phase |
| Connect\_phase |
| End\_of\_elaboration\_phase |
| Start\_of\_simulation\_phase |
| Run\_phase |
| Extract\_phase |
| Check\_phase |
| Report\_phase |
| Final\_phase |

## Task phase和function phase

Uvm中的phase，按照是否耗费仿真时间，分为function\_phase和build\_phase，function\_phase不耗费仿真时间，通过function实现，而task\_phase耗费仿真时间，通过task实现。

# 其他

## Vivado中的FFT核使用

Vivado中的FFT核主要配置FFT点数NFFT，循环前缀长度CP，FFT模式(FFT还是IFFT)，伸缩因子。

## Vivado中的DDS核使用

DDS核可直接配置为频率输出，也可配置为可编程的相位形式。

## 物理层参数

|  |  |
| --- | --- |
| 参数 | 数值 |
| 基本采样频率 |  |
| 数据符号时间 |  |
| 循环前缀时间 |  |
| OFDM符号时间 |  |
| 数据子载波数 |  |
| 导频子载波数 |  |
| 总子载波数 |  |
| 子载波间隔 |  |
| 总子载波占用带宽 |  |
| 数据速率 |  |
| 调制方式 |  |
| 编码速率 |  |

## I/Q误差

I/Q误差是指IQ两路传输不满足正交，这将会导致数据的解码错误。

零中频的通信接收机有明显的直流偏置误差和I/Q失调误差问题，需要在逻辑部分对这两种误差进行实时补偿和校准。AD9361自身就提供三种IQ校准手段，分别是正交误差自校准，设置ENSM寄存器位即可，第二种是正交误差连续跟踪模式，通过SPI总线能够将误差值都出来进行后继正交校准。第三种是正交误差校准和跟踪模式。

# 缩写

FCS：Frame Check Sequence，帧的CRC校验值，帧检验序列，确保发送和接收数据一致。

DSSS：扩频通信

CCK：补码键控

CCA：空闲信道评估，它的作用是PHY根据某种条件来判断当前无线介质是处于忙还是空闲状态，并向MAC通报。

BuP：Burst Processor

FCS：帧校验序列

ACP：Access Priority，接入优先级。

CCMP：Counter CBC—MAC Protocal：计数器模式密码块链消息完整码协议。

TKIP：Temporal Key Integrity Protocal：临时密钥完整性协议。

WEP：Wired Equivalent Privacy：有线等效保密协议。

WPA：Wi-Fi Protected Access：Wi-Fi网络安全接入。

MCS：Modulation and Coding Scheme：调制与编码策略----WLAN通信速率表