DFS IP设计需求

|  |  |  |  |
| --- | --- | --- | --- |
| Document Number: |  | Document Version: |  |
| Owner: |  | Date: |  |
| Document Type: |  | | |
| NOTE: | ALL MATERIALS INCLUDED HEREIN ARE COPYRIGHTED AND CONFIDENTIAL UNLESS OTHERWISE INDICATED. The information is intended only for the person or entity to which it is addressed and may contain confidential and/or privileged material. Any review, retransmission, dissemination, or other use of or taking of any action in reliance upon this information by persons or entities other than the intended recipient is prohibited.  This document is subject to change without notice. Please verify that your company has the most recent specification.  Copyright © 2019 Spreadtrum Communications Inc. | | |



Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Description |
| 0.0 | 2019/3/27 | Mingmin Ling | first version |
| 1.0 | 2019/4/9 | Mingmin Ling | Make freq/bandwidth to different module  Add latency module |
| 1.1 | 2019/4/9 | Mingmin Ling | Add dfs hold time request  Add request for register rw when light sleep |
| 1.2 | 2019/4/12 | Mingmin Ling | For group review |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[DFS IP设计需求 1](#_Toc5714220)

[Revision History 2](#_Toc5714221)

[一 . 总体逻辑 4](#_Toc5714222)

[1.1 图示 4](#_Toc5714223)

[1.2 全局需求 5](#_Toc5714224)

[二 . 发生器与参数设置模块 6](#_Toc5714225)

[2.1参数设置模块 7](#_Toc5714226)

[2.2模式选择与开关 7](#_Toc5714227)

[2.3 sync logic同步模块 8](#_Toc5714228)

[2.4 频点选择模块 8](#_Toc5714229)

[三 . SW mode 10](#_Toc5714230)

[四 . bandwidth模块（hw mode requester） 11](#_Toc5714231)

[4.1概述 11](#_Toc5714232)

[4.2 统计者/采集器 12](#_Toc5714233)

[4.3 预测者/投票机 12](#_Toc5714234)

[4.4 threshold配置 13](#_Toc5714235)

[4.5 逻辑与计算单元 14](#_Toc5714236)

[五 . latency模块（hw mode requester） 15](#_Toc5714237)

[5.1概述 15](#_Toc5714238)

[5.2 统计者/采集器 15](#_Toc5714239)

[5.3 总体参数配置与逻辑模块 16](#_Toc5714240)

[六 . freq模块（hw mode requester） 17](#_Toc5714241)

[6.1概述 17](#_Toc5714242)

[6.2 预测者/投票机 17](#_Toc5714243)

[6.3逻辑与计算单元 18](#_Toc5714244)

[七 . Mixer 19](#_Toc5714245)

[八 . 记录器/debugger 20](#_Toc5714246)

[8.1 Current status 20](#_Toc5714247)

[8.2 History stack 21](#_Toc5714248)

[8.3 Interrupt generator 21](#_Toc5714249)

[九 . DFS Timer 22](#_Toc5714250)

### 一 . 总体逻辑

#### 1.1 图示



#### 1.2 全局需求

a dfs IP需要考虑掉电恢复。在功耗允许的情况下应该处于不掉电的区域，如果处于掉电区域，则需要在不掉电的寄存器上保存掉电前所在的频点信息和掉电前所在的模式信息

DFS\_LAST\_FREQ（ro） //掉电前所在的频点

DFS\_LAST\_MODE （ro）//掉电前在sw还是hw模式。

b 建议删除pure sw mode，仅保留sw mode与hw mode，默认处于sw mode。

c 新的设计要求能自动完成dvfs，如果变频时需要升压/降压，应该能自动完成。

d dfs hold timer应该达到2us以内

e dfs相关寄存器的访问应该不受到pub light的影响

f dfs IP能读出ASIC version便于软件兼容

DFS\_VERSION（ro）//IP版本号

g dfs过程本身支持timing配置，比如mrw后可以加入可变的delay等。

### 二 . 发生器与参数设置模块



#### 2.1参数设置模块

参数设置模块用于在寄存器中配置与具体频点有关的参数（clk mode/raio/d2/adjs等）。

a 参数应该有多组，与IP支持的的dfs sel总数相等。

b 如果能做到，应该使得sw模式与hw模式使用一套寄存器。

c 参数可能包括

DFS\_F(0~n)\_CLK\_MODE（rw）

DFS\_F(0~n)\_RATIO（rw）

DFS\_F(0~n)\_ RATIO \_D2（rw）

DFS\_F(0~n)\_DDL\_ADJS（rw）

DFS\_F(0~n)\_HALF\_MODE （rw）

……

d 参数可以配置对应频点对应的电压，当变频前后频点电压不同时，需要能自动完成升压/降压。

DFS\_F(0~n)\_VOL（rw）

#### 2.2模式选择与开关

a request发生器上应该包含dfs的总开关。

DFS\_ENABLE（rw） //enable为0时应该没有任何dfs能够产生。

b request发生器上应该包含sw mode与hw mode切换功能。

DFS\_MODE （rw）//用1bit代表sw或者hw模式。

c hw start/stop 以往的设计中，hw模式需要额外的start/stop信号，此信号是否必需？如果在新的设计中，仍然需要，也应该位于request发生器模式选择模块上。

HW\_DFS\_START（rw）

HW\_DFS\_STOP（rw）

HW\_DFS\_STOP\_ACK（ro）//软件通过这个bit等待stop完成。

d sw模式相关比较简单，只有request/ack与状态寄存器，也可能合并到这里。

#### 2.3 sync logic同步模块

用于处理dfs信号与sleep/urgent等信号的同步

a sleep同步

DFS\_SLEEP\_SYNC（rw）//用1bit决定在sleep时dfs是采取wait还是giveup

DFS\_SLEEP\_WAIT\_TIME（rw）//当DFS\_SLEEP\_SYNC为wait时，配置wait时间，在timeout后采取giveup。用某个特殊值代表无限等待。

DFS\_SLEEP\_OUT（rw）//置位时放弃出sleep后第一次降频请求

b 在以往的设计中，sw模式需要软件关闭light sleep来做sync。在新的设计中，如果可能，请设计为a中所提到的寄存器对sw mode/hw mode同样有效

c urgent同步

DFS\_URGENTn\_EN（rw）//打开channel n的urgent信号与dfs的同步。Dfs模块应该提供多个urgent通道用于连接多个有需求的master。

DFS\_URGENTn\_SYNC （rw）//决定某个通道在urgent时dfs是采取wait还是giveup。

DFS\_URGENTn\_WAIT\_TIME（rw）//当DFS\_URGENTn\_SYNC为wait时，配置wait时间，在timeout后采取giveup。用某个特殊值代表无限等待。

d c中的寄存器应该尽量设计为对sw mode/hw mode同样有效。

#### 2.4 频点选择模块

因为软件可能希望暂时屏蔽一些频点最终request发出的频点选择与最初升频/降频的频点可能不一致。

a DFS\_AVAIL\_FREQ（rw）//选择enable的频点。

对于升频/降频请求，应该能跳过没有enable的频点，如果跳过后没有频点可以选择，应该放弃dfs。此寄存器应该只对hw mode有效，sw mode仍能过变到任何请求的频点。

b DFS\_ AVAIL\_PLL（rw）//应该能选择跳过没有enable的pll上的所有频点。

c 老的设计中有min\_freq\_up/min\_freq\_dn/min\_en\_up/min\_en\_dn/min\_load的机制。目前看来不需要这么复杂的设计。软件用不到min\_freq\_up与min\_freq\_dn不相等的case。建议这些都不保留，使用a中的DFS\_AVAIL\_FREQ就足够了。但是DFS\_AVAIL\_FREQ没有动态load的机制，所以在添加动态load

DFS\_AVAIL\_FREQ\_LOAD（rw） //与之前min\_load类似的机制，通过写1写0动态更新、生效DFS\_AVAIL\_FREQ的配置。

### 三 . SW mode

a Sw mode比较简单，也可能被集成到（二）中的模式选择中。主要包含request与ack。

SW\_DFS\_SEL（rw）//选择要变到的频点（0～n）。

SW\_DFS\_REQUEST（rw） //dfs request

SW\_DFS\_ACK（ro）//用于等待sw dfs结束

b 其他相关参数设计上尽量和hw模式使用一套

c 与sw dfs当前状态机相关的可读值应该整合到记录器（debugger）模块中

d sw模式的dvfs升压/降压是自动完成还是软件手动操作，目前来看都可以。从一致性的角度建议都做成自动完成

e sw模式与light的同步做到与hw一致，不再需要软件关/开light。

### 四 . bandwidth模块（hw mode requester）



#### 4.1概述

a在新的设计中，我们希望将原dfi monitor和原vote模块整合为bandwidth模块。新的bandwidth模块将综合busmonitor带宽统计结果与各个模块的投票，综合得出是否需要变频的判断。我们在逻辑上将与带宽统计相关的部分称作统计者，其实现称为采集器；将与投票相关的部分称作预测者，其实现称为投票机。

b我们将原来dfi busmonitor的overflow/underflow设置与原vote模块的threshold设置也整合到一组寄存器中。用作采集器和投票机综合计算后对比的标准。

c 原来的设计中vote模块包括bandwidth投票与freq投票，但是两种投票的逻辑完全不同。现在的bandwidth模块中只包含bandwidth投票，而freq投票则被分割，放到freq模块中。

#### 4.2 统计者/采集器

a 统计者对于芯片的各个channel/master，都应该能够配置该channel/master是否参与统计。这是因为当某些模块选择带宽投票时，该模块的带宽已经通过投票体现，不应该再记录在带宽的统计中。软件需要能够配置该模块不参与带宽采集。

DFI\_MASTERn\_EN（rw）//配置对应某个channel/master是否参与dfi的采集。

b 采集器可能需要支持一个timer sync。这是因为采集器采集到的值是与采集时间窗口有关的，窗口的大小软件可配。但是各个master在投票的时候软件层面没有假设其能够知道当前的采集窗口。所以这里需要通过一个简单的计算使得采集器、投票机、threshold配置模块三者的数值基于同样的时间长度。可能的方法是在这里增加一个寄存器，配置当前采集器的窗口大小。采集器的输出值需要根据窗口大小换算成bps再进行下一步的计算。

DFI\_TIMER\_VALUE（rw）//采集窗口的大小

c 需要有一组寄存器读出采集到的带宽数值

DFI\_MASTERn\_VALUE（ro）//当前对应某个channel/master采集到的值，每个窗口更新

DFI\_VALUE（ro）//当前采集总值，不包括disable的channel/master

DFI\_VALUE\_OUTPUT（ro）//经过换算后对应的bps（输出到加法器的值）

#### 4.3 预测者/投票机

投票在于原来sharkl5/roc1/orca的vote模块基础上实现。在这里只包含bandwidth投票。

a 对于所有master， Bandwidth投票代表master对自己当前状态下的带宽做出预测，这种模式应用于master不希望使用dfi采集带宽的情况。这种情况往往是因为dfi的统计窗口与master的带宽需求周期不匹配导致的采集值不可用。

b 投票机对每个matser都有自己的enable位

BW\_VOTE\_MATSERn\_EN（rw）//enable

c bandwidth投票值使用bps作为单位，注意采集器的值也会换算成bps再与投票器的值一起参与计算。

BW\_VOTE\_MASTERn\_\_VALUE（rw）//bandwidth投票value

d Hw mode的逻辑

某些master应该能支持以特殊的硬件信号来触发投票是否有效，我们需要将HW mode置位来使这个机制生效。

BW\_VOTE\_MASTERn\_HW\_MODE（rw）//bandwidth投票使用HW mode。

每个master的enable位与HW mode的逻辑关系应该如下图所示。

|  |  |  |  |
| --- | --- | --- | --- |
| VOTE\_EN | HW\_MODE | Hw sigal | output |
| 0 | X | X | no |
| 1 | 0 | X | yes |
| 1 | 1 | 1 | yes |
| 1 | 1 | 0 | no |

当HW mode打开时，根据对应硬件信号的动态变化来决定投票是否生效。

e 投票机需要支持动态改变投票值，也就是不reset enable位和HW mode的前提下更新投票值。如果必要的话，可以增加一个load的寄存器用于动态更新投票值。

VOTE\_MASTERn\_BW\_VALUE\_LOAD（rw）//bandwidth投票值动态更新

VOTE\_MASTERn\_MF\_VALUE\_LOAD（rw）//minfrq投票值动态更新

#### 4.4 threshold配置

a 将原来dfi模块的overflow/underflow配置与vote模块threshold配置整合到一起。这里应该统一按照bps作为单位。

FREQn\_OVERFLOW（rw）//设置频点n的overflow

FREQn\_UNDERFLOW（rw）//设置频点n的underflow

b bandwidth模块应该随时能知道当前频点（即使上一次变频是由其他模块发起的，比如laterncy模块）。在这里需要有一个用于debug目的的寄存器可以读取整个bandwidth模块看到的当前频点，它应该随时于系统的当前频点一致

DFI\_DBG\_CUR\_FREQ（ro）//确保与系统当前频点一致

#### 4.5 逻辑与计算单元

a 用于bandwidth计算的加法单元。加法的输入包括采集器输出的值（这个值只包括使能了采集的master且经过了换算，详见4.2.a）和所有使能了bangwidth投票模式的master所投的value（详见4.3）。

b 用于比较加法单元输出与threshold的比较单元。当带宽加和大于当前频点的overflow时，变频到高一个频点；当带宽加和小于当前频点的underflow时，变频到低一频点

c 此处应有一个总开关，用于控制采集器、投票机以及所有运算单元。

DFS\_BANDWIDTH\_EN（rw）

d 需要一些debug寄存器，用于读出一些中间状态。

DBG\_BW\_ALL（ro）//加法器的最终结果

DBG\_BW\_REQ（ro）//0比较单元没有输出变频请求；1比较单元有升频请求；2比较单元有降频请求

DBG\_BW\_REQ\_FREQ（ro）//最终请求变到的频点

### 五 . latency模块（hw mode requester）



#### 5.1概述

Latency模块参考sharkl3/roc1 latency模块实现。Latency模块只有统计者，没有预测者，其统计者的实现同样叫做采集器，实际设计沿用latency monitor的方式。但是原latency monitor的设计有一些复杂的地方，目前看来可以简化，仅采集每个周期的latency时间即可。

#### 5.2 统计者/采集器

a 统计者对于芯片的各个channel/master，都应该能够配置该channel/master是否参与统计。这是因为在某个项目中，某个master是否关注latency，是各个驱动的软件同事来决定的。

LATMON\_MASTERn\_EN（rw）//配置对应某个channel/master是否参与latency monitor的采集。

b 采集器需要支持一个timer sync。用于配置latency monitor的窗口。这里所有master/channel使用同样长度的窗口应该就可以满足需求。

LATMON\_TIMER（rw）//采集窗口的大小

c 每个master/channel都需要一组参数配置

LATMON\_MASTERn \_URGENT\_TIME（rw）//当该master的latency时间超过这个设置值时计算一次urgent

LATMON\_MASTERn \_URGENT\_UP（rw）//当连续几个urgent后发出一次升频请求

d 需要有一组寄存器读出采集到的latency

LATMON\_MASTERn\_VALUE（ro）//当前对应某个channel/master采集到的值，每个窗口更新

#### 5.3 总体参数配置与逻辑模块

a 由于1/2的原因某些高频点在latency上反而弱于地频点，所以在latency模块需要配置升频时跳过一些频点。

LATMON\_JUMP\_FREQ（rw）//配置升频时跳过的频点

b 此处应有一个总开关，用于控制采集器、投票机以及所有运算单元。

LATMON \_EN（rw）

c 需要一些debug寄存器，用于读出一些中间状态。

DBG\_BW\_ALL（ro）//加法器的最终结果

DBG\_ LATMON \_REQS（ro）//哪些master发出了升频请求

DBG\_ LATMON \_REQ\_FREQ（ro）//最终请求变到的频点

### 六 . freq模块（hw mode requester）



#### 6.1概述

a在新的设计中，我们希望将原vote中freq投票单独做出来，因为这个逻辑虽然简单，但是目的与原则都与bandwidth有很大的差别。

b freq模块只有预测者/投票机，各个master直接投出自己想要的频点。

#### 6.2 预测者/投票机

投票在于原来sharkl5/roc1/orca的vote模块基础上实现。在这里只包含freq投票。

a 投票机对每个matser都有自己的enable位

FV\_VOTE\_MATSERn\_EN（rw）//enable

b freq投票值使用3bit或者4bit的freq sel。

FV \_VOTE\_MASTERn\_\_VALUE（rw）//freq投票value

c某些master应该能支持以特殊的硬件信号来触发投票是否有效，我们需要将HW mode置位来使这个机制生效。

FV\_VOTE\_MASTERn\_HW\_MODE（rw）//freq投票使用HW mode。

每个master的enable位与HW mode的逻辑关系应该如下图所示。

|  |  |  |  |
| --- | --- | --- | --- |
| VOTE\_EN | HW\_MODE | Hw sigal | output |
| 0 | X | X | no |
| 1 | 0 | X | yes |
| 1 | 1 | 1 | yes |
| 1 | 1 | 0 | no |

当HW mode打开时，根据对应硬件信号的动态变化来决定投票是否生效。

d 投票机需要支持动态改变投票值，也就是不reset enable位和HW mode的前提下更新投票值。如果必要的话，可以增加一个load的寄存器用于动态更新投票值。

VOTE\_MASTERn\_FV\_VALUE\_LOAD（rw）//freq投票值动态更新

#### 6.3逻辑与计算单元

a 一个取max值的模块，在多个投票的master中，只有投票值最高的master投票生效。

b 用于比较max与当前频点的模块，当投票值更新时，决定是否产生升频信号且升频到的频点。Freq模块目前看来不需要产生降频信号。

c 此处应有一个总开关，用于控制采集器、投票机以及所有运算单元。

DFS\_FREQ\_EN（rw）

d 需要一些debug寄存器，用于读出一些中间状态。

DBG\_MAX\_FREQ（ro）//加法器的最终结果

DBG\_FV\_REQ（ro）//0没有输出变频请求；1有升频请求；

DBG\_FV\_REQ\_FREQ（ro）//最终请求变到的频点

### 七 . Mixer

Mixer是将多个hw mode下的多个request发生模块（bandwidth模块、latency模块、etc）综合在一起的一个小单元。规则是1既有升频也有降频时保留升频，2 有多个升频时保留目标频点最高。这个单元需要一些只读寄存器。

REQn\_STATUS（ro）//某个request发生模块当前状态，0：没有请求；1：请求升频；2：请求降频

REQn\_FREQ（ro）//某个request发生模块请求的频点。

REQ\_FIN（ro）//读出最终哪个request发生模块的请求生效了

REQ\_FIN\_STATUS（ro）//最终是否发出请求，0：没有请求；1：请求升频；2：请求降频

REQ\_FIN\_FREQ（ro）//最终请求的频点。

### 八 . 记录器/debugger



#### 8.1 Current status

保存当前/最近一次变频的状态/结果。

DEBUGGER\_CUR\_MODE（ro）//当前处于sw还是hw模式

DEBUGGER\_DFS\_STATUS（ro）//当前dfs状态。变频中or idel。

DEBUGGER\_REQER（ro）//当前/最近一次变频的请求者（sw、bandwidth、latency、etc）

DEBUGGER\_REQ（ro）//当前/最近一次变频的类型 升频or降频

DEBUGGER\_REQ\_FREQ（ro）//当前/最近一次变频的目标频点

DEBUGGER\_ACK\_STATUS（ro）//当前/最近一次变频的结果。成功或者失败

DEBUGGER\_ERR（ro）//当前/最近一次变频失败时，记录失败原因

DEBUGGER\_HOLD\_TIME（ro）//当前/最近一次变频的hold time

DEBUGGER\_SLEEP\_STATUS（ro）//当前是否light

DEBUGGER\_URGENT\_STATUS（ro）//当前各个master是否发出urgent

#### 8.2 History stack

以栈的形式记录每次的dfs status。

a 通过buffer on/off来控制。置位时开始记录，每次dfs结束将current status中的值保存在栈中，超过buffer size的时候较早的记录应该被覆盖。清零时停止记录。记录时需要同时记录时间戳。

HIS\_ON（rw）//打开history stack

b 保证最新的的记录在stack的顶部。或者提供一个标记表明最新记录在buffer中的位置也可以。

c history中每条记录的内容与current status中的一致且增加一个时间戳。

d history 支持一个timer counter，当buffer on时，计算在各个频点的累计时间以及进入light sleep的累计时间。

#### 8.3 Interrupt generator

提供一个可配置的中断模块用于调试，在异常时给出中断到对应的模块（cm4/arm）。

a支持的中断类型可能包括：收到ack； Ack Timeout；其他支持的err状态。

b提供一个enable位开关整个interrupt generator

DFS\_INT\_EN（rw）

### 九 . DFS Timer

在较老的设计中，每颗芯片为dfi或者其他monitor提供时序的timer都是各不相同的，建议在新的设计中明确dfs IP对timer的需求，做出统一的设计。

a dfs timer不需要像系统中大多数timer一样配置high level/low level时间，只需要配置发出tick的周期就可以。同时应该提供多个不同的tick周期供给bandwidth、latency以及其他多个模块。应该保留多个周期间成倍数关系的设计。

DFS\_TICKn（rw）//设置tick

b 需要考虑DFS Timer在deep sleep与light sleep下的状态。建议保留出sleep后可以忽略一次变频请求的设计（在2.3.a中已经提及）。并保证可配置。