文件编号：

模板版本：

Wrap\_rd\_logic详细说明书（FPGA）

制定：邢海涛

审核：

批准：

中国大恒（集团）有限公司北京图像视觉技术分公司

2016年09月14日

|  |
| --- |
| 文件状态： [ ]草稿 [√ ]正式发布[ ]正在修改 |

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 日期 | 撰写人 | 变更内容 |
| V1.0.0 | 2016-09-14 | 邢海涛 | 初稿 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1. Wap\_rd\_logic模块规格指标（Module Features） 4](#_Toc462421742)

[1.1. 概要设计指标 4](#_Toc462421743)

[1.2. 后级模块约束指标 5](#_Toc462421744)

[2. Wrap\_rd\_logic模块方案设计（Module Designs） 6](#_Toc462421745)

[2.1. Wrap\_rd\_logic结构 6](#_Toc462421746)

[2.2. back\_buf设计 6](#_Toc462421747)

[3. 模块测试项（Module Testings） 6](#_Toc462421748)

[4. 系统测试传递测试项（可选） 6](#_Toc462421749)

[5. 重点测试项（可选） 6](#_Toc462421750)

[6. 单元测试结果（Module Testings Result） 6](#_Toc462421751)

[7. 附录（可选） 6](#_Toc462421752)

# Wap\_rd\_logic模块规格指标（Module Features）

## 概要设计指标

| 指标 | 指标分解 |
| --- | --- |
| 1. SE=0设置完0.5ms之内，帧存模块要停下来 | 当se=0时，wrap\_wr\_logic立即停止写MCB WR FIFO。发出WR CMD命令，BL=64。这样会将MCB WR FIFO清空。但是会造成MCB WR FIFO underrun的现象，目前认为underrun不会影响到后面的传输，且一次发出64个数据量一定能够将MCB WR FIFO清空，这样做是比较安全的。 |
| 1. 设置SE=1后，场信号到来后，便允许图像数据进入DDR | 当se上升沿时，wrap\_wr\_logic的状态机不能立即启动，要看到一个fval上升沿才能认为是一帧开始 |
| 1. 传输阻塞时，FPGA抛弃当前曝光产生的所有ROI的图像，而不是抛弃其中某个ROI的图像 | 当写指针追上读指针的时候，写指针要跳过读指针，这样就会覆盖掉一帧数据。 |
| 1. 一次触发所有窗口都输出完整 | 一次触发，sensor输出一帧图像，一帧图像中可能包括多个ROI，帧存模块要将所有的ROI都读出 |
| 1. 切换采集模式，采集过程中，由连续切换触发模式，触发模式最多滞后n帧生效，生效后发送触发信号可以得到图像；触发切换连续模式，如果正在采集图像，连续模式最多滞后n帧生效，生效后得到连续图像，如果处于触发等待期间，连续模式可以立即生效；n表示当前设置的ROI的个数。 | 一次触发，sensor输出一帧图像，一帧图像中可能包括多个ROI，帧存模块要将所有的ROI都读出 |
| 1. FPGA后端传输帧率是Sensor输出帧率的n倍，n表示当前ROI的个数 | 一次触发，sensor输出一帧图像，一帧图像中可能包括多个ROI，帧存模块要将所有的ROI都读出 |

## 后级模块约束指标

| 信号 | 指标 |
| --- | --- |
| ov\_rd\_ptr | 当 i\_wr\_ptr\_change =0的时候改变  由于 i\_wr\_ptr\_change是2拍的宽度，且 iv\_wr\_ptr 在 i\_wr\_ptr\_change第二拍的时候改变，因此 ov\_rd\_ptr和 iv\_wr\_ptr不会在同一时间改变。 |
| ov\_rd\_addr | 读地址，在 o\_rd\_cmd\_en =1时改变 |
| o\_rd\_cmd\_en | 发出o\_rd\_cmd\_en的条件   1. 处于 RD\_CMD 状态 2. 后端FIFO有足够的数据量，prog\_full=0 3. Mcb rd cmd 不满 |
| ov\_rd\_cmd\_bl | 1. 当前ROI的宽度-已经读走的数据量>256byte时，bl=32 2. 当前ROI的宽度-已经读走的数据量<256byte时，bl=剩余数据量 |
| ov\_rd\_cmd\_instr | 固定为3’b001，不加precharge的读指令 |
| ov\_rd\_cmd\_byte\_addr | 读地址，在o\_rd\_cmd\_en 之后改变 |
| o\_rd\_en | Mcb rd fifo读使能，读使能为1的条件为   1. 处于读状态 2. Mcb rd fifo不空 3. 后端fifo不满 |
| ov\_rd\_data | 后端FIFO32bit数据，后端fifo读时钟域 |
| o\_emtpy | 后端FIFO空标志，后端fifo读时钟域 |
| o\_reading | 正在读标志   1. 当开始读的时候，设为1 2. 当一帧读完的时候，设为0 |

# Wrap\_rd\_logic模块方案设计（Module Designs）

## Wrap\_rd\_logic结构



Wrap\_rd\_logic由两部分组成，rd\_logic和back\_bufc。rd\_logic完成写逻辑的操作，工作在clk\_frame\_buf时钟域。back\_buf是一个fifo，完成缓存帧存数据，隔离时钟域的作用。

## back\_buf设计

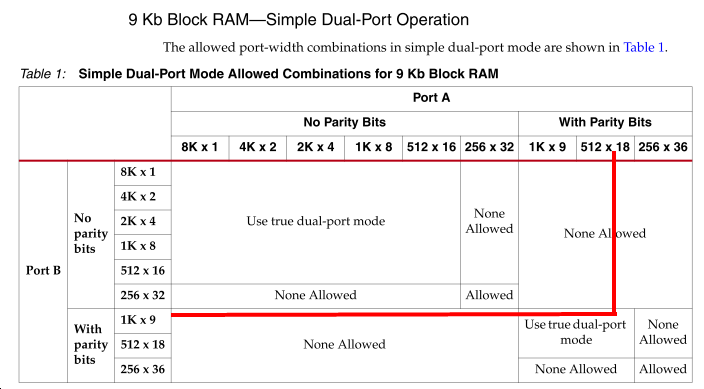
### fifo选择

MCB的数据位宽是64bit，GPIF传输的位宽是32bit。当ROI的行宽是4的倍数的时候，一行的宽度有可能不能被8整除。比如ROI0的行宽是100，像素格式是8bit，此时ROI0的每一行的数据量是100byte，不能被8整除。由于MCB的数据位宽是8byte，因此在每一行的末尾会多读出4byte的数据。因此将后端fifo的写入的数据位宽定为66bit，读出的数据位宽定位33bit。高bit作为判断标记，如果读出端看到bit32=1，则说明是多余的byte，需要丢弃。

同理，在leader、trailer中，也存在长度不是8的整数倍的情况，需要用bit32标记。

 上图是后端fifo读写端口的情况

在写入端看来，fifo的深度是512。在读出端看来，fifo的深度是1024。此时的fifo占用了4个M18K。



通过上图可以看到，如果写入宽度是66，读出宽度是33，则必须占用4个M9K。因此把写深度设定为512，最大限度的利用了blockRam的存储资源。

Fifo的最大存储容量是4Kbyte，MCB 一次读的数据量是256byte，考虑到有可能升级为512byte，因此后端fifo program full的条件是还差512byte就要满。因此将program full的阈值设为440。

### fifo复位

在以下条件下，fifo需要复位：（或操作）

* reset有效(帧存时钟域)
* se=0

在例化fifo的时候，full信号在复位时要设置为0。因为在se=0时，还要继续从mcb rd fifo中读取数据。

### fifo写

fifo写的条件是：（与操作）

* mcb rd fifo不空
* 后端fifo不满
* 处于读mcb的状态
* 开采信号有效

从mcb rd fifo读出来的数据是8byte的倍数，如果当前的数据块的长度是4的倍数，而不是8的倍数，需要写标志位。

### fifo读

fifo读的逻辑由后端模块控制，后端fifo空信号会输出，因此建议后端读模块：

* 后端fifo不空

如果bit32=1，则说明是dummy byte，需要丢弃。

### fifo读写波形

## 读逻辑设计

### 状态机



**状态机状态说明**

* IDLE 空闲状态
* PTR 读指针改变状态
* CMD 发出命令状态
* RD 读一帧数据状态
* LINE 切换行状态
* FLAG 切换flag状态
* ROI 切换roi状态

**状态机跳转**

* IDLE-> PTR

1. se=1，且
2. Calibration完成

* PTR -> IDLE

1. 当wr\_ptr\_change=0时，且
2. 允许读=0
   1. 如果帧存深度为1，即单帧，只有当写刷新过之后，才能读
   2. 如果帧存深度大于1，只有当读指针不等于写指针时才能读

* PTR -> CMD

1. 当wr\_ptr\_change=0时，且
2. 允许读=1
   1. 如果帧存深度为1，即单帧，只有当写刷新过之后，才能读
   2. 如果帧存深度大于1，只有当读指针不等于写指针时才能读

* CMD -> RD

1. 当mcb rd cmd fifo 不满的时候，且
2. 读地址允许=1
   1. 如果读写指针相等，且此时正在写，那么读地址要小于写地址且mcb wr cmd fifo是空的。或
   2. 如果读写指针相等，且此时不在写。或
   3. 如果读写指针不相等

* RD -> IDLE

1. 从mcb rd fifo中读出的数据量是 burst\_cnt-1 且 正在读mcb rd fifo。且
2. Se=0

* RD -> CMD

1. 从mcb rd fifo中读出的数据量是 burst\_cnt-1 且 正在读mcb rd fifo。且
2. 没有满足一行的数据量。且
3. Se=1

* RD -> LINE
  + 当正在读image\_flag的阶段

1. 从mcb rd fifo中读出的数据量是 burst\_cnt -1 且 正在读mcb rd fifo。且
2. 满足一行的数据量。且
3. Se=1
   * 当正在非image\_flag的阶段
     1. 不会进入到LINE状态

* RD -> FLAG
  + 当正在读image\_flag的阶段

1. 从mcb rd fifo中读出的数据量是 burst\_cnt -1 且 正在读mcb rd fifo。且
2. 满足一个flag的数据量。且
3. 当前ROI的行宽与总行宽相等
4. Se=1
   * 当正在读非image\_flag的阶段
     1. 从mcb rd fifo中读出的数据量是 burst\_cnt -1 且 正在读mcb rd fifo。且
     2. 满足一个flag的数据量。且
     3. Se=1

* LINE -> CMD

1. 不满足一个flag的数据量

* LINE -> FLAG

1. 满足一个flag的数据量

* FLAG -> CMD

1. 当前flag不是最后一个flag

* FLAG -> ROI

1. 当前flag是最后一个flag

* ROI -> CMD

1. 当前ROI不是最后一个ROI

* ROI -> IDLE

1. 当前flag是最后一个flag

### mcb读模块整体波形图

略

### mcb读逻辑

mcb rd fifo读的条件是：（与操作）

* mcb rd fifo不空
* 后端fifo不满
* 处于读mcb的状态

### mcb读命令逻辑

只有在CMD状态会发出写命令

* CMD状态：

当mcb rd cmd fifo不满发出读命令

### mcb读burst\_length

读burst length的最大长度是32，由于MCB RD FIFO的数据位宽是64bit，因此每次读操作的数据量是32\*8=256byte。读的数据量是固定的。

当flag末尾或者image阶段行尾的时候，有效数据量不足一个burst，此时读的数据量还是一个完整的burst，只不过写到后端fifo中的数据只有有效的数据量而已。

### mcb读每个flag的大小

#### 获取ROI NUM

下图是u3v\_format模块中leader包中最后一个8byte数据示意图。Byte0-byte3都是无效数据，帧存模块要解析出byte0和byte1。

Byte0是ROI NUM，帧存读逻辑根据ROI NUM选择对应的image\_size。Byte1是最后一个ROI的标志位，如果不是最后一个ROI，trailer从TRAILER地址段中读取。如果是最后一个ROI，trailer从TRAILER FINAL地址段中读取。



* + BYTE3:0
  + BYTE2:0
  + BYTE1:如果是最后一个ROI，则为1，其他为0
  + BYTE0：当前ROI的num号码

#### 各个flag的数据分布

下面以4个ROI为例说明：



每个FLAG都是分别存储的，地址从小到大分别是，所有的leader、所有的trailer、所有的chunk（就算没有chunk，地址也是保留的）、图像、最后一个ROI的trailer（由于图像大小是可变的，因此放在最下方）。

其中所有的leader、tailer、chunk（可选）都是连续分布的，如果上层打开的ROI不是连续的，在帧存中存储时也是连续的。

#### leader的读取方法

每个leader的大小默认是56byte，其中52byte是有效数据，剩余4byte是多余数据，byte0和byte1中包含ROI信息。

首先，要解析剩余4byte中的ROI信息，选择trailer地址和相应的image size。其次，还要标记出高4byte是多余数据，方便后级模块处理。

#### trailer的读取方法

每个trailer的大小会随着chunk打开和关闭改变，当chunk打开时，trailer是36byte，当chunk关闭时，trailer是32byte。

当chunk打开时，帧存读逻辑会读出40byte，高4byte是多余的数据，不包含任何信息，需要标记出来。

如果是最后一个ROI，那么需要从TRAILER FINAL这个地址读数据。

#### chunk的读取方法

Chunk大小与总开关和分开关有关，chunk大小的计算方法为 roi0\_payload\_size – roi0\_image\_size。

如果chunk总开关打开，每个ROI都需要传输chunk数据。根据ROI顺序，依次读取即可。

#### image的读取方法

读取每个ROI的图像，需要获取如下的信息：

* + 1. 当前的pixel\_format
    2. 当前sensor输出的宽度
    3. 每个ROI的宽度
    4. 每个ROI的image\_size
    5. 每个ROI相对于sensor输出图像的起点

### mcb读地址

此处的写地址是寄存器wr\_addr，不是mcb的byte addr。

#### 切换读地址的时机

写地址分为三种情况改变

* IDLE，优先级最高

在IDLE状态，地址复位为0

* CMD，优先级次之

地址累加

* LINE，优先级次之

地址切换为下一行的首地址

* FLAG，优先级次之

地址切换为下一个flag的首地址

#### 读地址的排布

目前共有5个flag，每个flag个需要一个起始地址。在frame\_buffer中已经根据每个flag数据量的大小，分配了每个flag的首地址。

### mcb读指针

读指针就是指当前正在读DDR中的第几帧，与读指针相关的有一个参数和一个寄存器。

现在指针的位宽是8bit，即缓存深度是1帧-255帧。

## 停采的处理方法

停采时，地址、指针、正在写状态立即归零。

停采信号不做完整帧控制，立即生效。

## 寄存器生效时机

在开采信号上升沿采样multi\_roi\_global\_en、roi\_pic\_width\_mroi和，当multi\_roi\_global\_en=1是，总行宽是multi\_roi\_width，否则是roi\_pic\_width\_mroi[0]

# 模块测试项（Module Testings）

系统测试传递测试项（可选）

无

重点测试项（可选）

单元测试结果（Module Testings Result）

附录（可选）