多通道数据整形器设计

多通道数据整形器MCDF（Multi-Channel Data Formatter），将多个通道的上行(umlink)数据经过内部的FIFO以数据包(data packet)的形式送出。其结构如图1所示。

图1 MCDF的设计结构(fmt\_length应该是6位，cmd\_addr只需要6位)

1. MCDF外部接口

1.1 系统信号接口

* clk ：时钟信号
* rst\_n ：复位信号，低电平有效

1.2通道从端(slave)接口（x=0, 1, 2）

* chx\_data[31:0] ：通道数据输入
* chx\_valid ：通道数据有效指示信号，高电平有效
* chx\_ready ：通道数据接收信号，高电平表示接收成功

1.3整形器(formatter)接口信号

* fmt\_chid[1:0] ：整形数据包的通道ID号
* fmt\_length[5 5:0]：整形数据包长度信号
* fmt\_req ：整形数据包发送请求
* fmt\_grant ：整形数据包被允许发送的接收指示
* fmt\_data[31:0] ：数据输出信号
* fmt\_start ：数据包起始指示信号
* fmt\_end ：数据包结束指示信号

1.4控制寄存器(control register)接口

* cmd[1:0] ：寄存器读写命令
* cmd\_addr[5:0] ：寄存器地址
* cmd\_data\_i[31:0] ：寄存器写入数据
* cmd\_data\_o[31:0] ：寄存器读出数据

2、 MCDF各模块接口信号及时序

2.1 通道从端(slave)

一个通道从端包括图1中的slaveX及FIFOX（X=0，1，2）。其中FIFO深度为64。

通道从端从外部接口接收数据，当接收到一个完整的数据包（到底多大？怎么知道fmt要多大？默认32，由2.3.3中所述的通道X的控制寄存器bit[5:3]决定

0，1，2，3：4，8，16，32）后，则向arbiter发出发送请求。若请求信号得到响应，则开始发送，直至整个数据包全部发送完成后，再根据情况确定是否发出发送数据请求（是否数据多余32，因为这时无法读入）。

（一直能写，满了不ready）

2.1.1通道从端接口信号

通道从端接口信号如下表所示同，包括系统信号、MCDF外部接口信号、register模块接口信号、arbiter模块接口信号。

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名称 | 方向 | 功能 | 备注 |
| clk\_i | input | Clock input | 系统信号 |
| rstn\_i | input | low level effective reset |
| chx\_data\_i[31:0] | input | Data input | 外部接口 |
| chx\_valid\_i | input | Data is valid From outside |
| chx\_ready\_o | output | Ready to accept data |
| slvx\_en\_i | input | Write enable To FIFO（通道使能信号） | register接口 |
| margin\_o[5:0] | output | Data margin（通道FIFO空余信号） |
| ‍slvx\_pkglen\_i[2:0] | input | 来自寄存器设定的通道数据包长度 |
| slvx\_data\_o[31:0] | output | Data Output to arbiter | Arbiter |
| slvx\_val\_o | output | data valid to Arbiter |
| slvx\_req\_o | output | required send data to arbiter（接收到完整数据包后） |
| a2sx\_ack\_i | input | Read acknowledge |

2.1.2通道从端接口时序

如图2所示。当chx\_valid信号为高时，表示写入数据有效。此周期chx\_data应给出要写入的数据。若该时钟周期chx\_ready为高，则表示已经将数据写入。若此时钟周期chx\_ready信号为低，则表示数据还未写入，需要等待chx\_ready为高时才将数据写入。

注意：后面其它模块的xx\_valid及xx\_ready时序均与图2中相同。



图2通道从端接口时序

2.2 arbiter

2.2.1 接口信号

如果formatter的发送数据请求信号f2a\_id\_req\_i为高，则arbiter根据slave0\_FIFO的发送请求信号slv0\_req\_i、slave1\_FIFO发送请求信号slv1\_req\_i、slave2\_FIFO发送请求信号slv2\_req\_i，按优先级确定响应通道的发送请求，并根据通道的编号（id=0, 1, 2）X产生以下信号送到formatter：

a2f\_id\_o = X(通道编号)

a2f\_data\_o = slvX\_data\_i

a2f\_pkglen\_sel\_o = slvX\_pkglen\_i

a2f\_val\_o = slvX\_val\_i

将formatter的响应信号送往对应的slave通道：

a2sX\_ack\_o = f2a\_ack\_i

如果各通道的优先级相同，则按通道编号从低到高轮流发送。

arbiter模块的接口信号如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名称 | 方向 | 功能 | 备注 |
| clk\_i | input | Clock input | 系统信号 |
| rstn\_i | input | low level effective reset |
| slv0\_prio\_i[1:0] | input | slave 0 priority | 控制寄存器的信号 |
| slv0\_pkglen\_i[2:0] | input | slave 0 package length |
| slv1\_prio\_i[1:0] | input | slave 1 priority |
| slv1\_pkglen\_i[2:0] | input | slave 1 package length |
| slv2\_prio\_i[1:0] | input | slave 2 priority |
| slv2\_pkglen\_i[2:0] | input | slave 2 package length |
| slv0\_data\_i[31:0] | input | slave 0 data | slave接口 |
| slv0\_req\_i | input | slave 0 required |
| slv0\_val\_i | input | slave 0 data valid |
| a2s0\_ack\_o | output | slave 0 Read acknowledge |
| slv1\_data\_i[31:0] | input | slave 1 data |
| slv1\_req\_i | input | slave 1 required |
| slv1\_val\_i | input | slave 1 data valid |
| a2s1\_ack\_o | output | slave 1 Read acknowledge |
| slv2\_data\_i[31:0] | input | slave 2 data |
| slv2\_req\_i | input | slave 3 required |
| slv2\_val\_i | input | slave 3 data valid |
| a2s2\_ack\_o | output | slave 2 Read acknowledge |
| f2a\_id\_req\_i | input | formatter read required | formatter接口 |
| f2a\_ack\_i | input | formatter acknowledge |
| a2f\_val\_o | output | arbiter data valid |
| a2f\_id\_o[1:0] | output | slave id(0,1,2) |
| a2f\_data\_o[31:0] | output | data output to formatter |
| a2f\_pkglen\_sel\_o[2:0] | output | data package length |

2.3整形器(formatter)

2.3.1整形器接口信号

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名称 | 方向 | 功能 | 备注 |
| clk\_i | input | Clock input | 系统信号 |
| rstn\_i | input | low level effective reset |
| f2a\_ack\_o | output | formatter acknowledge | arbiter接口 |
| fmt\_id\_req\_o | output | formatter read required |
| a2f\_val\_i | input | arbiter data valid |
| a2f\_id\_i[ 1:0] | input | slave id(0,1,2) |
| a2f\_data\_i[31:0] | input | data input from arbiter |
| a2f\_pkglen\_sel\_i[ 2:0] | input | data package length |
| **fmt\_grant\_i** | input | Read acknowledge | 外部接口 |
| fmt\_chid\_o[ 1:0] | output | slave id(0,1,2) |
| fmt\_length\_o[ 5:0] | output | data package length |
| **fmt\_req\_o** | output | data output required |
| fmt\_data\_o[31:0] | output | data output |
| fmt\_start\_o | output | first data indicate |
| fmt\_end\_o | output | last data indicate |

2.2.2整形器接口时序

如图3所示。整形器按照数据包的形式发送数据。数据包的长度可以是4、8、16和32。整形器必须完整的发送某一个通道的数据包后，才可以准备发送下一个数据包。在发送数据包期间，fmt\_chid和fmt\_length应保持不变，直到数据包发送结束。



图3整形器接口时序

整形器准备发送数据包时**，先将fmt\_req置高，等待接收端的fmt\_grant信号。当fmt\_grant信号变为高时，**则在下一个周期将：

1. **fmt\_req置低**
2. fmt\_start输出一个时钟周期的高电平脉冲
3. fmt\_data开始送出第一个数据

数据开始发送后应连接发送，中间不允许有空闲周期，直到发送完所有数据。在发送最后一个数据时，fmt\_end信号应置高并保持一个时钟周期。

相邻的数据包之间应至少有一个时钟周期的空闲，即fmt\_end变为低电平后，至少经过一个时钟周期fmt\_req才可以再次置高。

2.3控制寄存器(control register) //完成设计与初步验证

2.3.1接口时序

控制寄存器接口时序如图4所示。在每个时钟周期根据cmd命令完成指定操作。当cmd为写指令时，将数据cmd\_data\_i写入cmd\_addr指定的寄存器中。当cmd为读指令时，从cmd\_addr指定的寄存器中的数据读出，并在下一个时钟周期送到cmd\_data\_o端口。当cmd为其它指令时不进行任何操作。



图4控制寄存器接口时序

2.3.2接口信号

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名称 | 方向 | 功能 | 备注 |
| clk\_i | input | Clock input | 系统信号 |
| rstn\_i | input | low level effective reset |
| cmd\_i[1:0] | input | read or write control？读写各代表什么，我这里是01读10写，onehot | 外部接口 |
| cmd\_addr\_i[5:0] | input | command address |
| cmd\_data\_i[31:0] | input | command data input |
| cmd\_data\_o[31:0] | output | command data output |
| slv0\_margin\_i[5:0] | input | slave0 channel margin通道余量 | slave接口 |
| slv0\_en\_o | output | slave 0 enable（bit0） |
| slv1\_margin\_i[5:0] | input | slave 1 channel margin |
| slv0\_en\_o | output | slave 1 enable |
| slv2\_margin\_i[5:0] | input | slave 2 channel margin |
| slv0\_en\_o | output | slave 2 enable |
| slv0\_pkglen\_o[2:0] | output | slave 0 package lenth(bit[5:3]) | arbiter接口 |
| slv0\_prio\_o[1:0] | output | slave 0 priority（bit[2:1]） |
| slv1\_pkglen\_o[2:0] | output | slave 1 package lenth |
| slv1\_prio\_o[1:0] | output | slave 1 priority |
| slv2\_pkglen\_o[2:0] | output | slave 2 package lenth |
| Slv2\_prio\_o[1:0] | output | slave 2 priority |

2.3.3控制寄存器模块中的寄存器

* cmd\_addr地址0x00：通道slave0的控制寄存器，32bit，可读写，位定义为：
* bit[0] ：通道使能信号。1为打开，0为关闭。**复位值为1。**
* bit[2:1] ：优先级。0为最高，3为最低。复位值为3。
* bit[5:3] ：数据包长度。0对应长度4，1对应8，2对应16，3对应32。其它数值均暂时对应32。复位值为0。
* bit[31:6] ：保留位，不能写入。复位值为0。
* cmd\_addr地址0x04：通道slave1的控制寄存器，32bit，可读写，位定义同slave0。
* cmd\_addr地址0x08：通道slave2的控制寄存器，32bit，可读写，位定义同slave0。
* cmd\_addr地址0x12（不是0x0c吗？）：通道slave0的状态寄存器，32bit，只读，位定义为：
  + bit[ 7: 0] ：从端FIFO0的可写余量，实时同步FIFO0的余量，复位值为FIFO深度值。
  + bit[31:8] ：保留位，复位值为0。
* cmd\_addr地址0x16：通道slave1的状态寄存器，位定义同slave0。
* cmd\_addr地址0x20：通道slave2的状态寄存器，位定义同sl

00>>2=0

100>>2=1

1000>>2=10=2

1100>>2=11=3

10000>>2=100=4