之前文档回顾（文档损害，没有缺失关于策略的描述）

1. 设计了基于冲突内容回压的方案，及使用2个fifo分别接收外部输入和冲突内容，发现16入16出情况下，会导致FIFO中内容堆积，将输入分摊到多份（2/4）时，能够改善，但是仍然无法满足FIFO数据不堆积的情况，为此将数据输出改为32出，则当分摊到2份时可以保证数据不堆积
2. 设计了原始的策略2，采用多级级联的方式，从每个输入通道中选择一个有效的（与前面不冲突的）数据，这样来实现16入16出，数据存储在RAM中，输出使用FIFO作多通道对齐，发现后级RAM需要横大，但即使如此也会数据堆积
3. 原始策略2改为多路同时开始选择，本希望能够降低Ram的压力，但是时间长了以后，仍然无法满足需求
4. 接下来设计了双向数据流处理，即从第一个和最后一个同时开始任务，希望可以平本一下ram中的数据堆积量，本文档即从该设计开始

采用策略二方式，面临的最大问题是，后级Ram容量需求加大，当随机数据在通道间分布不均匀时，会导致较多数据被集中放置在后级存储器中，其结果是后级存储器被堆满，无法接收新的数，从而使得整体处理性能下降

解决方案一：

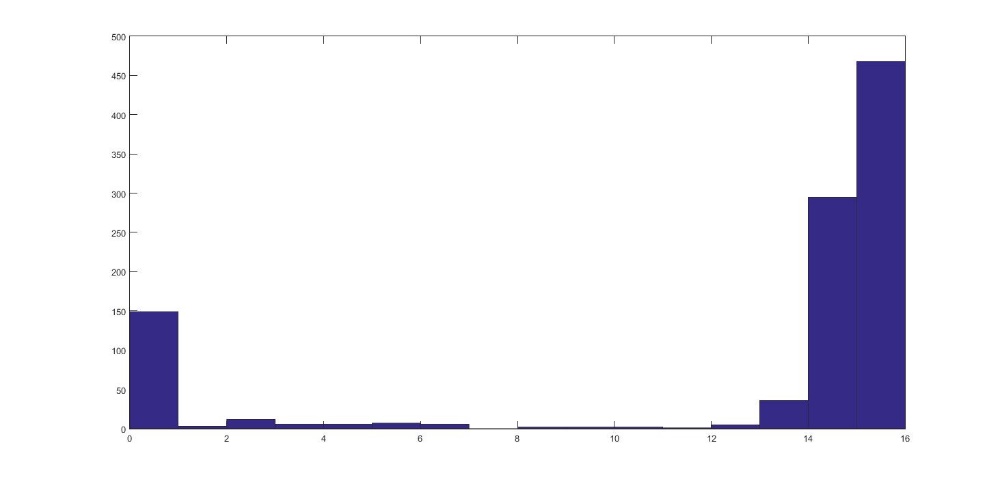
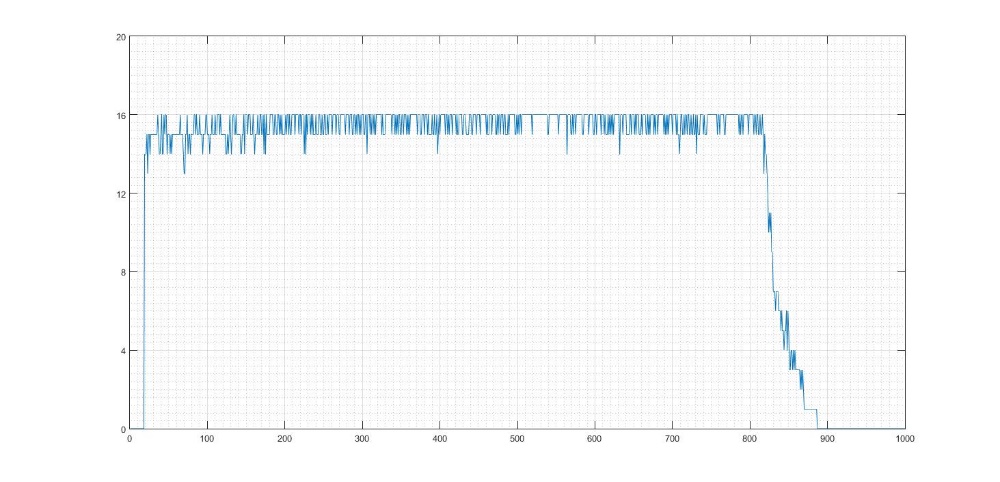
通过观测堆积在存储器中的数，发现其中存在数据缺项，为了改善该情况，拟设计双向计算方式来弥补该情况，通过双向启动流水处理，理论上可以将堆积的数据每2clk取出来，降低堆积的数据量。

一下为处理通道时序排列，前向后向数据流交错处理

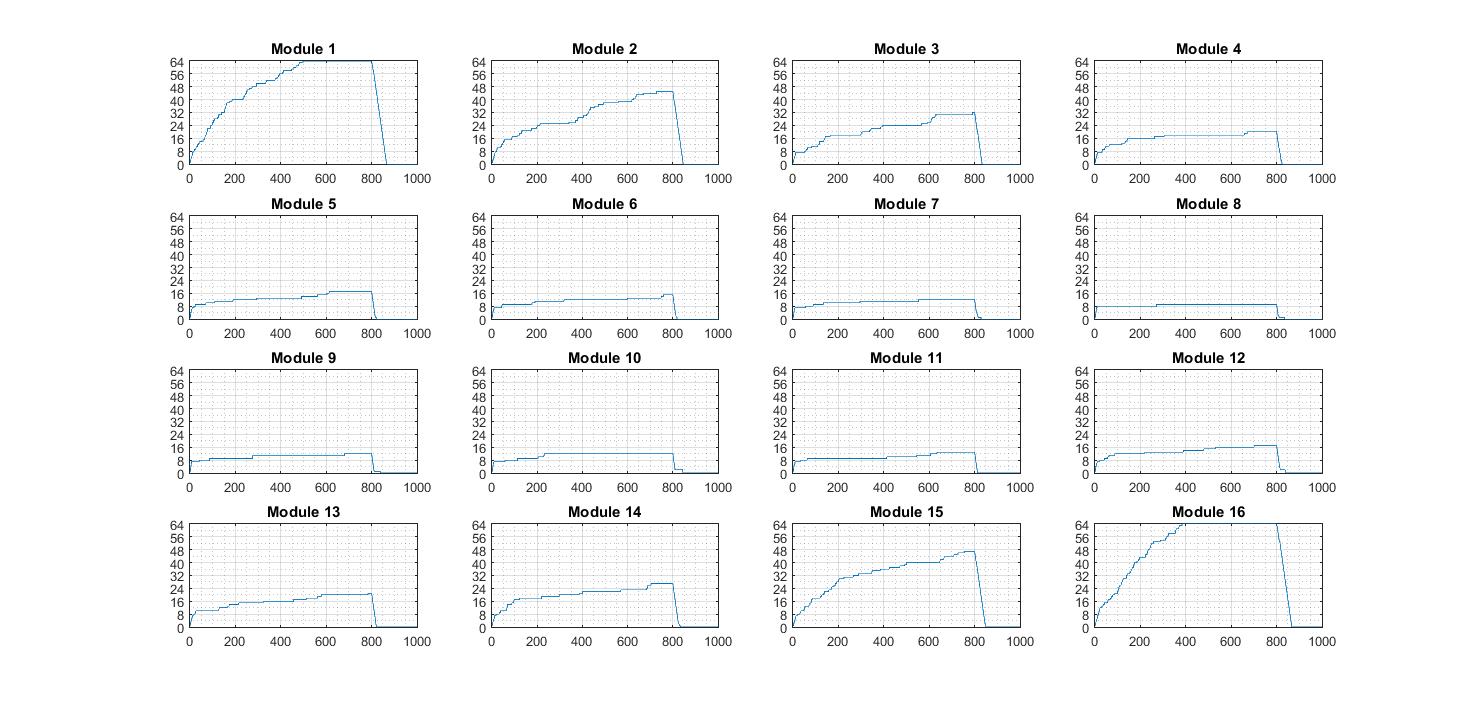


从时序设计中可以看到，所有的计算单元分时刻在计算前向和后向数据，那么每一次输出的数据在存储的时候需要做一个地址上的调整，以确保连续新，当然也可以在输出的时候进行调整

以上方案仍然无法解决存储器满问题，放弃之。



输出通道个数 输出通道个数直方图



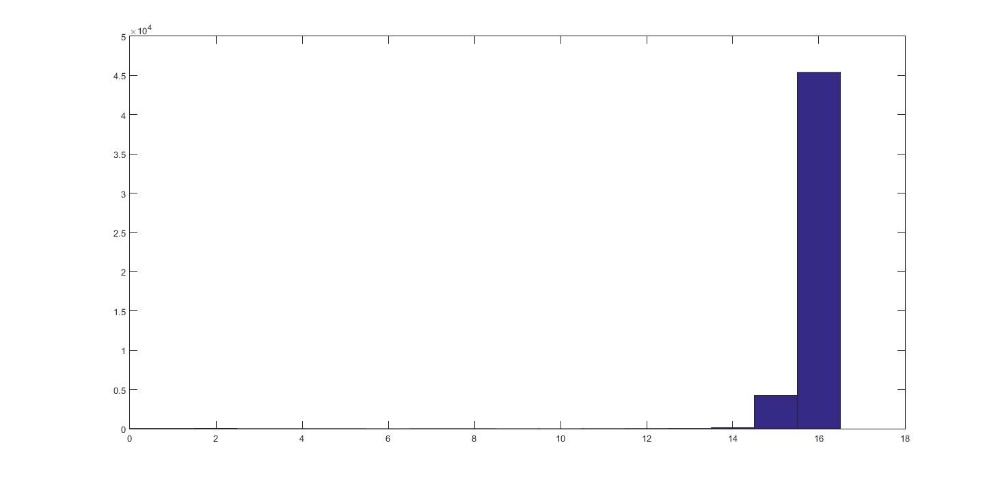
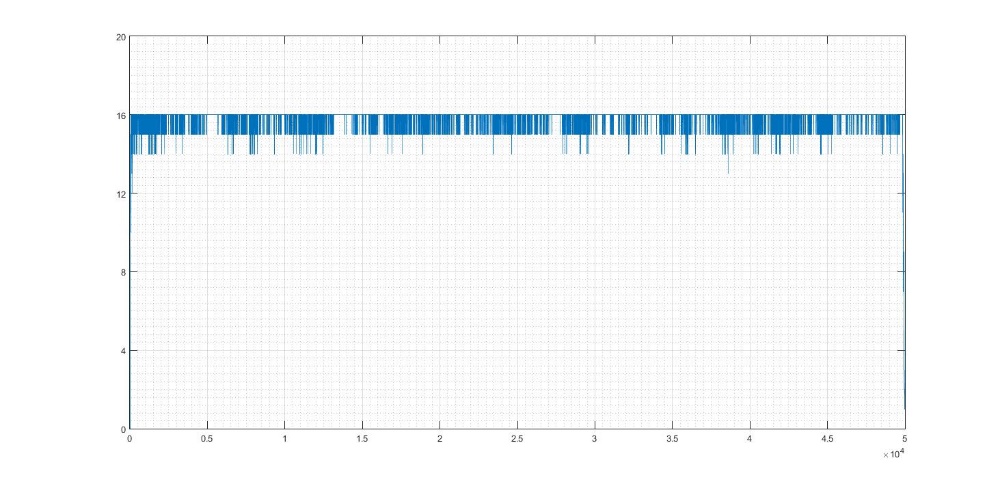
Ram中数据个数

解决方案二：

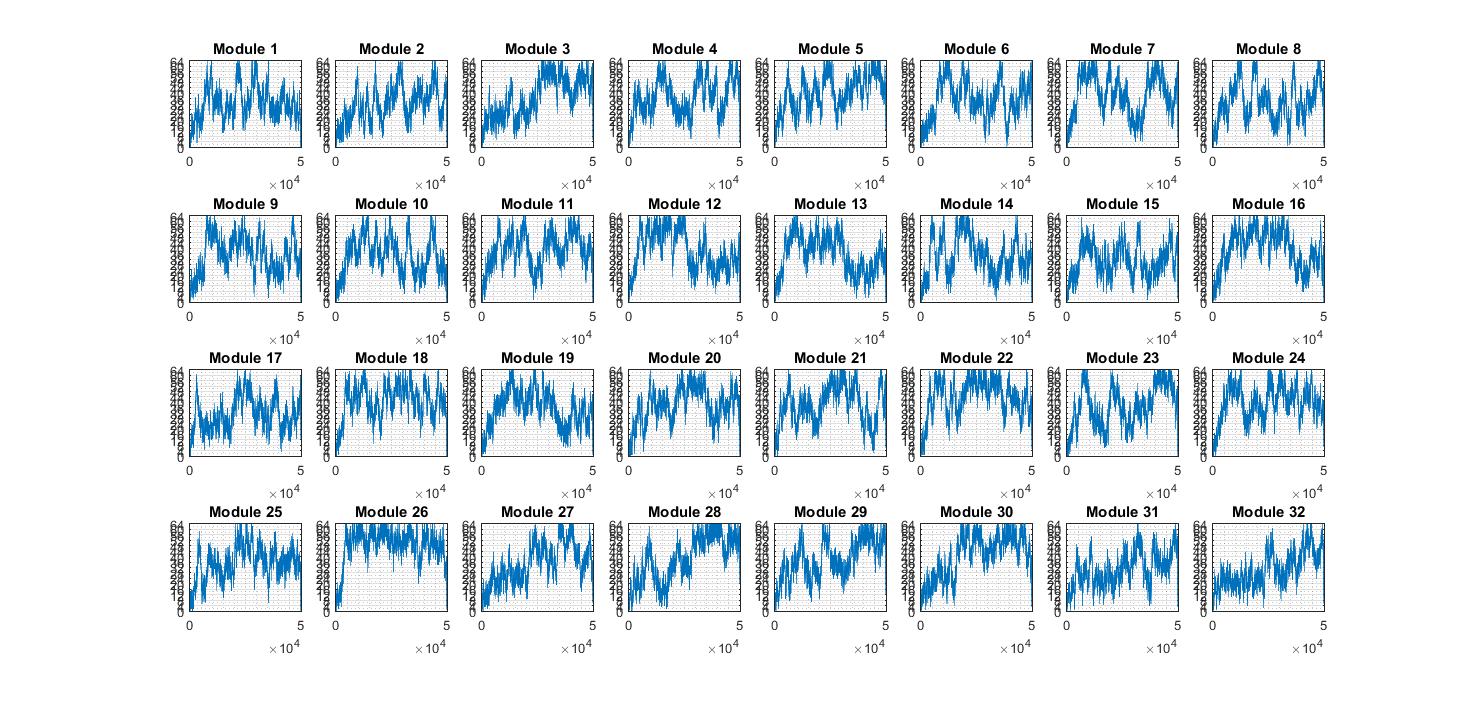
1）采用多通道同时启动工作方式，进行连续计算

2）同时输入通道为32个，每次数据输入为随机个数，输出通道限制在16个

采用该方案，可以将吞吐效率保持在99.5%以上（存储器64个，对于128个提高到99.8%，意义不大），输出个数直方图可以看到个数为16的几率大于90%，每个通道的存储器会发生偶然充满的情况，但是存在退回的可能。实际情况中，输入的数据在长时间内是保持16个通道输出数是相同的，这个有利于提高效率

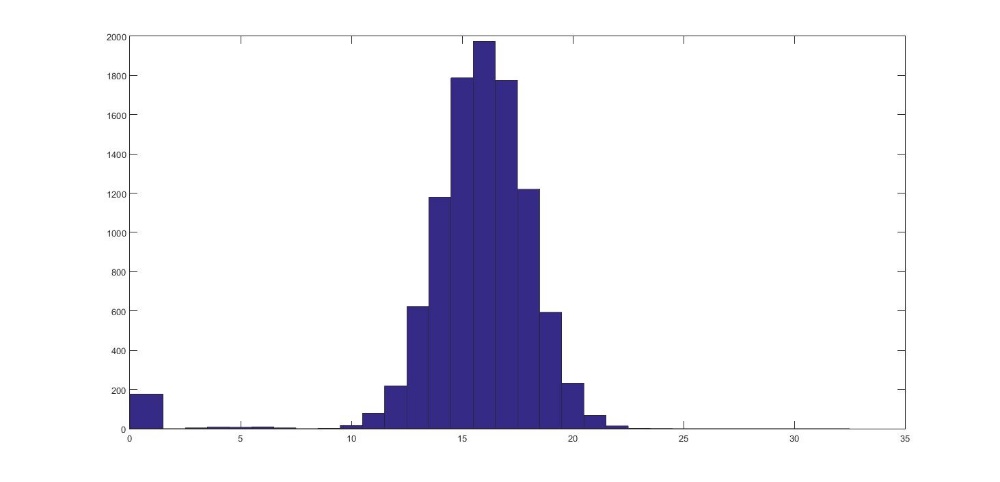
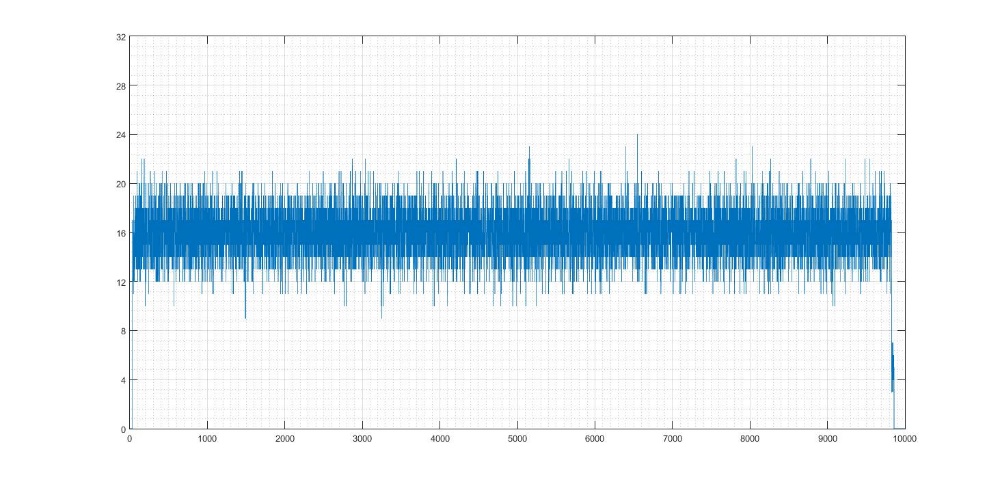


输出通道个数 输出通道个数直方图

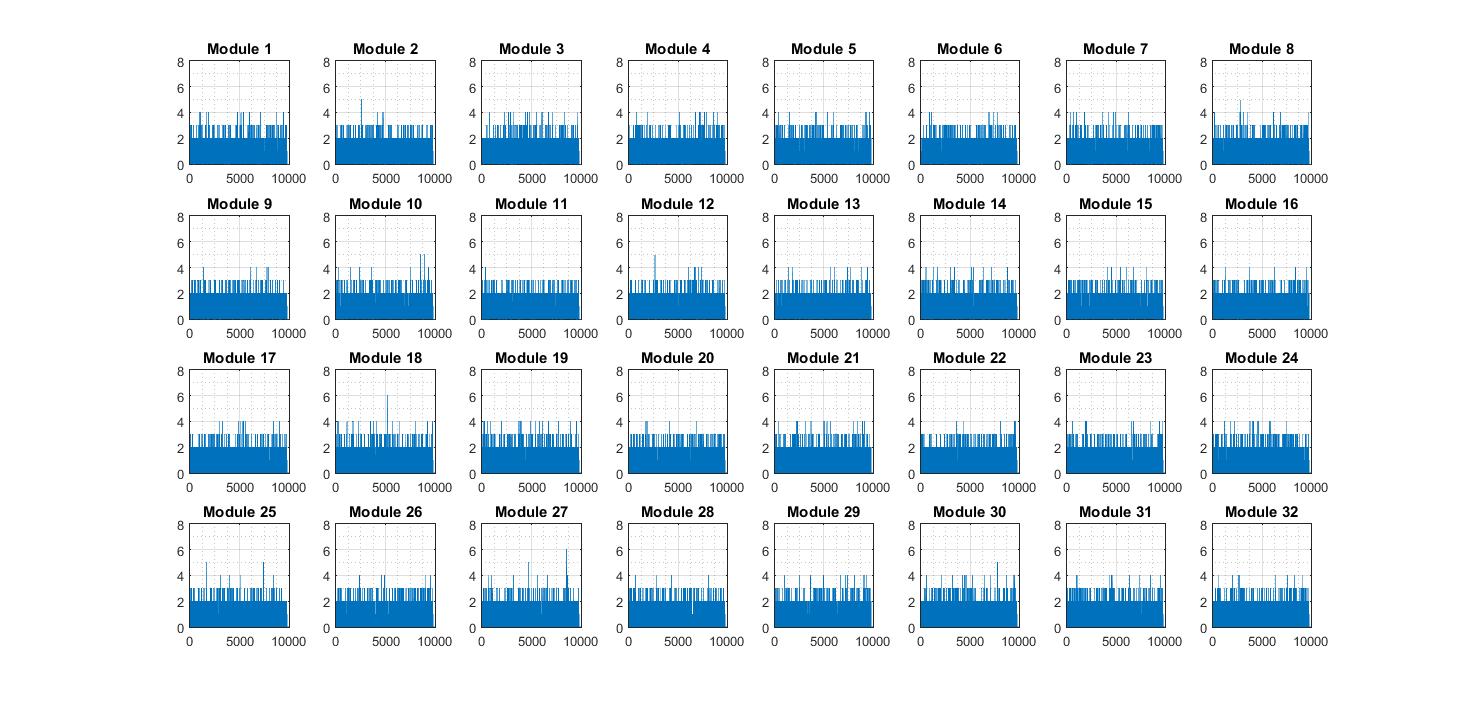


Ram中数据个数

该方案的另外一个衍生引用是输入32通道输出32通道，但是输入的数据每次仅进入16个（奇偶随机进入，数值是1~32随机），则在该模型下进行了仿真，发现数据不会堆积（堆积数不大于8个，可以极大减小对RAM深度的依赖），即数据可以得到及其快速的有效处理，效率能够达到100%



输出通道个数 输出通道个数直方图



Ram中数据个数

FPGA实现



设计要点：

1. 通道选择器采用Msk级联方式，Msk从输入到输出经过3clk，主要逻辑风险在两个地方，包括logic处理和xor处理（包括mux）
2. 数据存储到oRam相对msk输出多一个延迟
3. 由于选择过程使用3clk（以msk处理传递周期为计数），需要连续启动3次才能保证正确流水
4. 由3）导出输出数据每3\*32（假设通道数32个）次为一个处理周期，oRam的深度不能小于2\*3\*32
5. 写入oRam地址采用分段方式，即连续3次输出分别存在3个地址段上，则oRam输出地址可以按32\*32的数据块进行读取
6. 由于输入需要取地址，会导致输入有2clk延迟，占用较多寄存器
7. 数据Mix网络需要较多资源
8. 访问延迟为32\*3\*2+4



一次msk\_clr保持3clk

oRam写入地址延迟4clk

oRam输出地址延迟3\*32+4clk

以上信号要一起复位和启动，保持时序一致

经过仿真，随机输入16通道，输出16通道，在10k次仿真中无数据冲突