# 技术路线

将视频中的8K图像（4320\*7680）分割成4个4K图像（2160\*3840），然后根据时间相邻、位置对应的前后2帧4K图像，在其间进行4K插帧。这样可以得到4个4K帧，再组合成一个8K图像并输出，完成视频流的插帧过程。

针对单个4K图像的插帧，可以采用运动估计的方法。运动估计主要包括2个过程：计算运动向量、根据运动向量进行插帧。

计算运动向量：将前后相邻的4K帧A、B分割成相等大小的矩形宏块。对于前帧A的单个宏块a，在后帧B中划定一块搜索窗口，在搜索窗口内找出与宏块a的绝对差和（SAD）最小的宏块b。此时从a到b的位置偏移即为宏块的运动向量V。由于划分出了多个宏块，在A帧和B帧之间就会存在多个不同的运动向量。

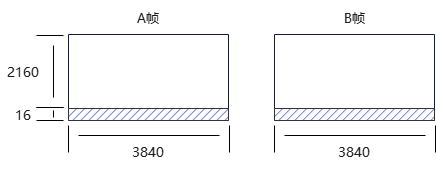
根据运动向量进行插帧：假设待插的帧为C，先将C初始化为A帧的信息。然后，将A帧中的宏块a和B帧中的宏块b进行平均，得到新的宏块c。将运动向量V减半为V2，以a为起点，以V2为运动偏移，确定c的位置，将c覆盖到C帧中的相应区域。对于C帧中的其他宏块，都可以通过上述方法进行覆盖与填充。

1. **运动估计过程的理论分析**

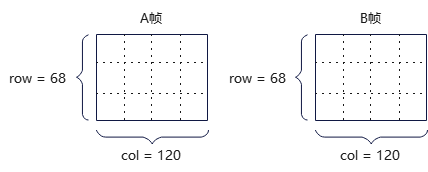
假设前后相邻的4K帧分别为A、B（3840\*2160），待插的帧为C（3840\*2160）。

* 1. **计算运动向量**

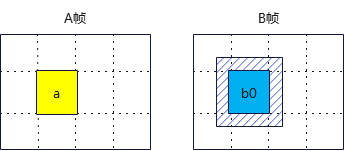
1. 将A帧、B帧的后16行像素信息，作为新的16行追加到图像的末尾，得到3840\*2176的两帧图像。



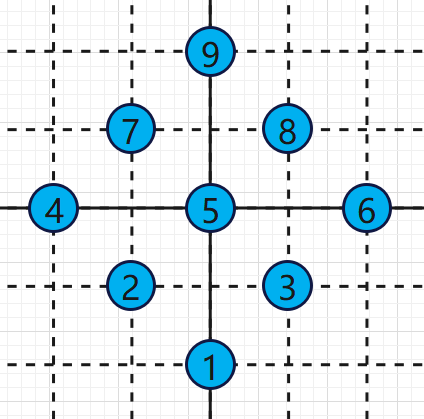
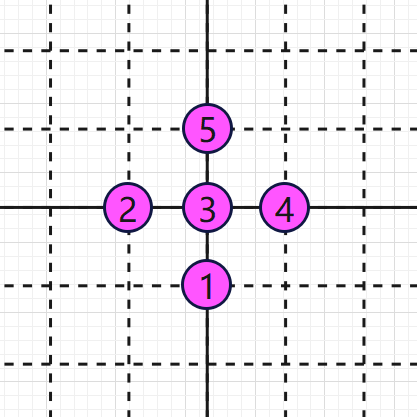
1. 分割A帧、B帧，得到相等大小（32\*32像素）的矩形宏块。每帧宏块行数row = 68、列数col = 120。



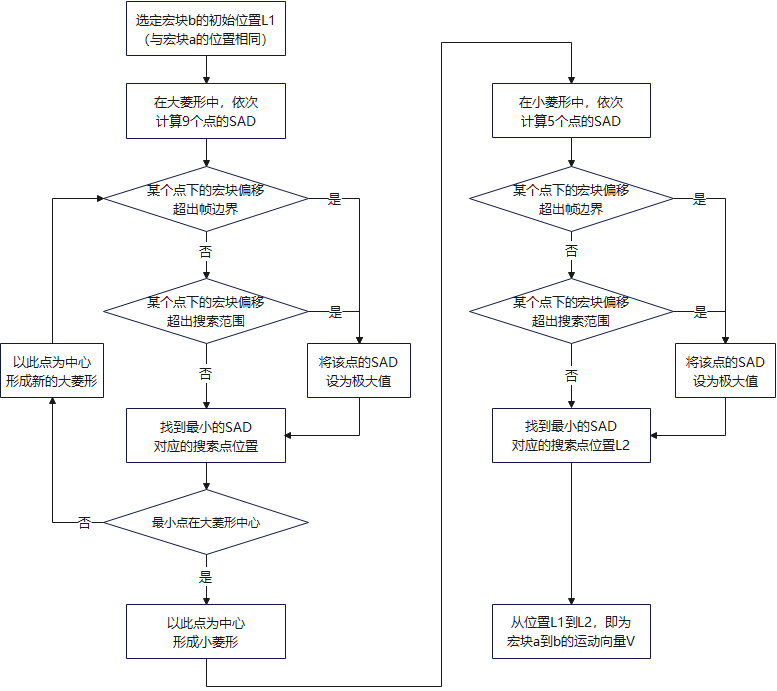
1. 针对A帧的某个宏块a（32\*32），在B帧中的相应位置划定一块64\*64的搜索范围，用来寻找此搜索范围内最匹配的块b。b的初始位置设为b0。



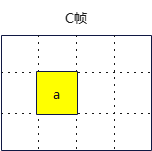
1. 在搜索范围内，按照菱形搜索算法移动b（大菱形9个点，小菱形5个点），并计算新的b与a的绝对差和SAD。

针对宏块a的菱形搜索算法的流程如下：



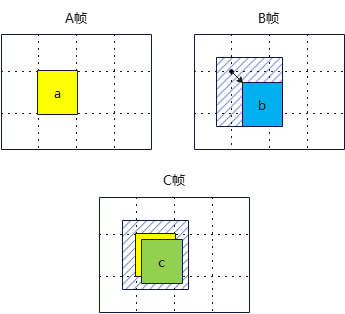
1. 对于其他的宏块，计算运动向量的流程与上图相同。最终可得到68 \* 120个运动向量。
   1. **根据运动向量进行插帧**
2. 将待插的C帧初始化为A帧。



1. 将宏块a和宏块b对应位置的每一对像素进行算术平均，得到宏块c。

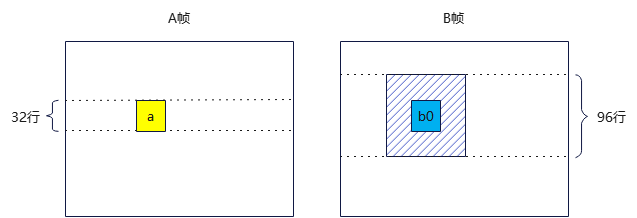
IMG_256

1. 确定c的位置：将运动向量V折半为V2。然后以a为起点，经过V2后，得到c的位置。再将宏块c覆盖到此位置处。



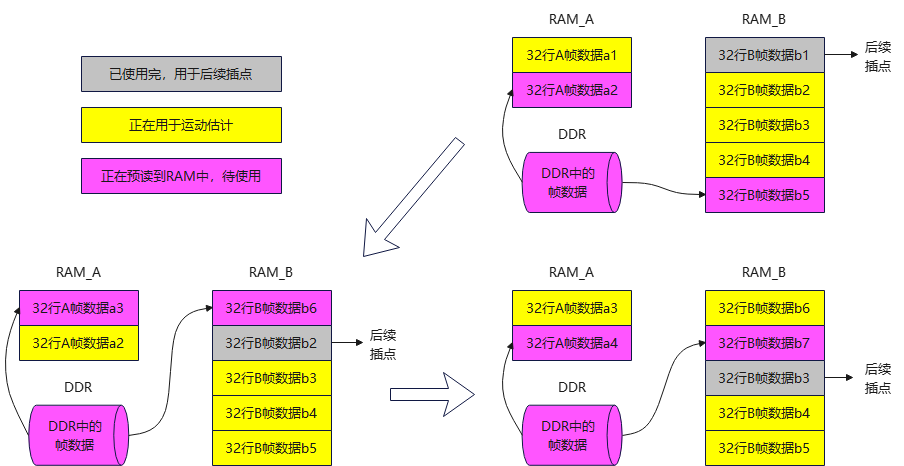
1. 对于其他的宏块及其运动向量，也按照上述流程进行处理。
2. **RAM的使用方式设计**

需要例化一系列的片内RAM，事先从DDR读取并存储图像帧A、B的Y、U、V信息（每个像素的Y、U、V信息各为8bit），用于后续的计算和插帧操作。



在运动估计时，A帧的32\*32宏块需要在B帧的96\*96搜索范围内进行运动与计算。可以例化实现：

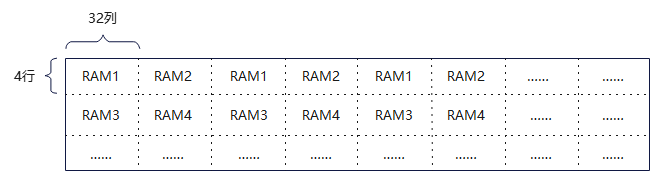
1. RAM的集合RAM\_A：用于存储A帧的64行像素数据。
   1. 存储A帧的32行像素数据，进行当前的运动估计处理。
   2. 对32行数据之后的32行进行预读和存储。
2. RAM的集合RAM\_B：用于存储B帧的160行像素数据。
   1. 存储B帧的96行像素数据，进行当前的运动估计处理。
   2. 对96行数据之前的32行进行存储，由于此32行已在插帧过程中使用完毕，因此可用于输出到后级，进行后续的插点模块处理。这样，原始图像点就不再通过读DDR获取了，减少一次DDR读操作。
   3. 对96行数据之后的32行进行预读和存储。



如上图所示，每次从RAM\_A、RAM\_B分别取32行、96行的数据的同时，从DDR预读A帧、B帧的后32行数据存储到RAM\_A、RAM\_B中。另外，在RAM\_B中，每次处理完运动估计过程，就取96行的前32行数据，输出给后续模块作为插点处理。将RAM\_A和RAM\_B分别当成环形FIFO，存储和读取的过程循环往复。

# RAM的容量设计

RAM\_A、RAM\_B集合均是由一系列的RAM组成。其中，每个RAM只存储Y、U、V信息中的其中一类。多个RAM对一个帧的存储如下示意图：



从RAM存储的视角看，每一帧被划分为4\*32的子块。然后，需要将每行里间隔的子块，同时存储到一个RAM中。

1. 一个RAM的大小计算：

一行中的子块数量为：3840 / 32 = 120

所以一个RAM存储的子块数量为：120 / 2 = 60

又因为一个子块的大小为（若只考虑Y信息）：32 \* 4 \* 8 = 1024 = 1k

所以一个RAM的容量大小为：60 \* 1k = 60k

1. RAM数量的计算：

RAM\_A集合存储A帧的64行所有Y、U、V信息。

所以，RAM\_A集合需要的RAM数量为：64 / 4 \* 2 \* 3 = 96个

RAM\_B集合存储B帧的160行所有Y、U、V信息。

所以，RAM\_B集合需要的RAM数量为：160 / 4 \* 2 \* 3 = 240个

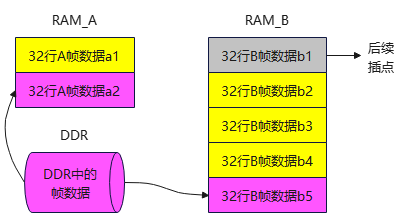
一共需要RAM：96 + 240 = 336个

1. 每个RAM读写位宽1024位，对应一次可以读写4行、32列的Y、U或V信息。

# 流水线设计

## 整体思路

插帧的实现采取“大流水+小流水”的思想来完成。



如上图所示，每次处理A帧的32行、B帧的96行数据过程中，将两个图像帧都划分成32\*32像素的宏块。在上述行的内部，块和块之间进行并行流水操作，即“大流水”。由于32行中的子块数量为120，因此最多会有120个任务进行“大流水”并行处理。

对一个块进行菱形搜索时，多个菱形搜索点的计算也实施并行流水操作，即“小流水”。每个块的菱形搜索点数不固定，具体有如下情况：

1. 对一个块进行第一次大菱形搜索时，需要计算9个点。
2. 若上一轮搜索的最小值在大菱形的顶点处，则下次的大菱形搜索只需计算5个点。
3. 若上一轮搜索的最小值在大菱形的边上，则下次的大菱形搜索只需计算3个点。
4. 若上一轮搜索的最小值在大菱形的中心，则转为小菱形搜索，只需计算4个点。

## 初始化阶段

每次针对新的32行120个宏块开始运动估计之前，需对每个宏块初始化以下数据：

1. 当前计算出的运动估计误差SAD的最小值。

一对像素的Y值（或U、V值）求绝对差后，占：8bit

因此对一个宏块的32\*32对像素求SAD，占：8 + log232 + log232 = 18bit

1. 当前块的运动估计过程是否结束的标志位。

占1bit

1. 运动估计过程中，当前菱形的中心点位置。此位置是为了与初始位置作比较。

由于一个块的搜索范围是96\*96，因此，中心点的横、纵坐标各只需：log296 < 7bit

一共需要：7 \* 2 = 14bit

1. 运动估计过程中，当前菱形相对于上次菱形位置的偏移方向。此位置为了与上次位置作比较，确定下次需要搜索的点的位置。

菱形一共有8个运动方向，因此需要：log28 = 3bit

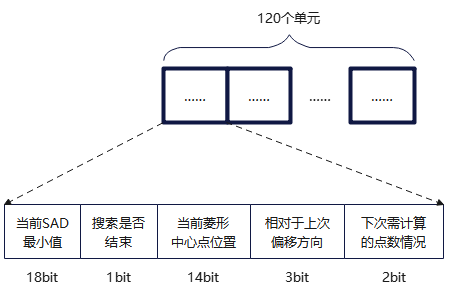
1. 下一次需要搜索和计算的点数。

由之前的分析可知，下一次可能需要搜索的点数为9个、5个、3个或4个点。一共有4种状态，因此需要：log24 = 2bit

针对一个宏块，上述数据共占用比特数为：18 + 1 + 14 + 3 + 2 = 38bit

针对一行的所有120个宏块，则共需占用：38 \* 120 = 4560bit

可以例化一个数组block\_states，存储上述数据。示意图如下：



另外，还需要例化2个FIFO，用于“大流水”与“小流水”的实现：

1. 第一个FIFO（FIFO\_1）记录120个块中还没有结束搜索的块的编号（0到119），若某个块已搜索完毕，则从FIFO中移出。

一个块编号所占位宽为：log2119 < 7bit

因此FIFO\_1最大容量为：120 \* 7 = 840bit

1. 第二个FIFO（FIFO\_2）记录每个块在本轮中具体要搜索哪几个点。

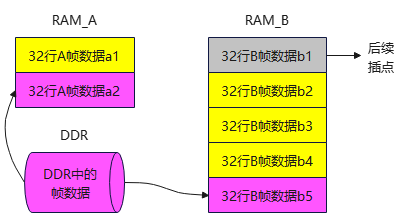
在记录每个点的横、纵坐标时，只需记录其相对于菱形中心点的偏移。

横坐标（纵坐标）偏移的范围为-2 ~ 2，因此需要占用位宽：3bit

一个点的位置需要占用位宽：3 \* 2 = 6bit

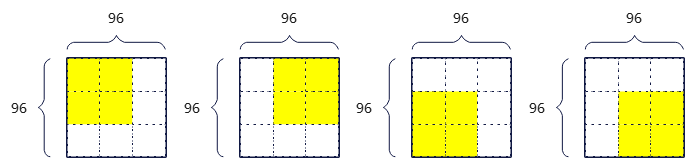
由于最多需要搜索9个点，因此FIFO\_2最大容量为：6 \* 9 \* 120 = 6480bit

## 运动估计的流水线分析



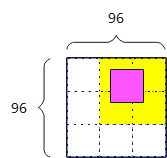
针对一个32\*32宏块的运动估计过程，其具体步骤如下（以Y、U、V中的Y进行大菱形搜索为例）：

1. “读取”：根据数组block\_states中的“当前菱形中心点位置”、搜索点在菱形中的位置这两个信息，在RAM\_B中，判断宏块相对于搜索区域中心的偏移方向。然后在96\*96搜索范围内，读取64\*64大小的像素，数据读取的情况为下列4种中的1种。



“读取”阶段需要1拍。

1. “定块”：根据数组block\_states中的“当前菱形中心点位置”、“相对于上次偏移方向”，以及FIFO\_2中的点位置信息，确定在64\*64的范围中，具体是哪个32\*32的块需要参与计算。示意图如下：

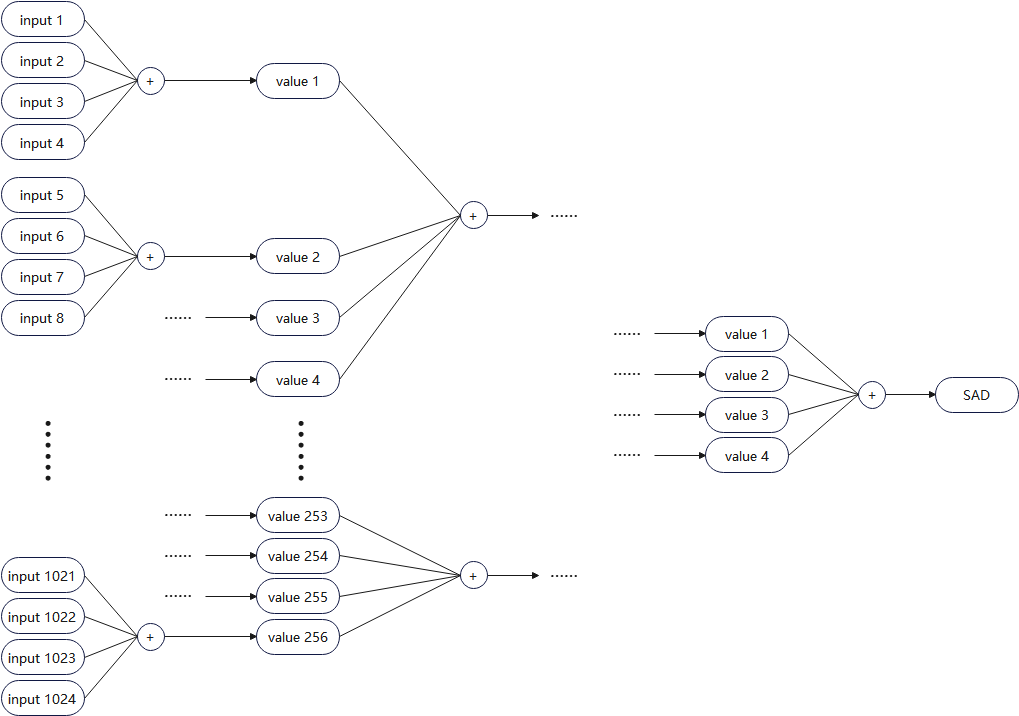


“定块”阶段需要1拍。

1. “求绝对差”：将2个32\*32的块的每一对像素点作绝对差。

对于所有的32\*32个像素，可以并行求绝对差值。因此，“求绝对差”阶段需要1拍。

1. “求和”：对32\*32个（1024个）绝对差值进行求和，得到SAD的值。



将1024个绝对差值，以4个为一组进行求和，如上图。需要节拍数：log41024 = 5拍

1. “比较”：对大菱形搜索中的各个SAD值进行比较，找出最小值及其对应的点位置。

“比较”阶段需要1拍。

1. “更新”：根据SAD最小值及其对应的点位置，前者更新到数组block\_states中的“当前SAD最小值”中，后者更新到数组block\_states中的“当前菱形中心点位置”中。同时，要判断并更新block\_states的“相对于上次偏移方向”、“下次需计算的点数情况”。
2. “判断结束”：若某个SAD小于阈值（提前终止条件），或SAD的最小值位于小菱形搜索中的中心点，则此32\*32宏块的运动估计过程结束。

由于整帧图像的运动估计，要针对多个块的多个搜索点进行计算，因此在上述过程中，对于“读取”、“定块”、“求绝对差”、“求和”、“比较”阶段可以设计流水线操作：



## “大流水”与“小流水”设计

“大流水”：在32行中，对所有的120个块进行流水处理。

“小流水”：在每个块中，对同一个菱形下的所有搜索点进行流水处理。

可用前文所述的2个FIFO（FIFO\_1、FIFO\_2）进行辅助实现。

1. 初始时，如下图所示：



将120块的块号按照依次存入FIFO\_1，同时，需要根据FIFO\_1中块号的顺序，依次将各个块的9个大菱形搜索点位置存入FIFO\_2中。

1. 每次根据FIFO\_1的队首所指示的块号，以及数组block\_states中的“下次需计算的点数情况”信息，从FIFO\_2的队首取出相应数量的点，依次对每个点进行流水操作，求各个点下的SAD的值。
2. 执行“更新”与“判断结束”过程。以0号块举例：
   1. 若此宏块的搜索未结束，下次需要搜索的点数为5（其他点数的情况类似），则更新block\_states中的“下次需计算的点数情况”信息，并将FIFO\_1中的0号块出队、再入队，将5个需要新计算的点存入FIFO\_2，如下图所示：



* 1. 若此宏块的搜索结束，则将此块号移出FIFO\_1，如下图所示：



1. 对于其他的宏块，搜索过程与上面所述类似，并且不同宏块的多个点也可以进行连续的流水计算，宏块之间无需停顿。
2. 最后当FIFO\_1和FIFO\_2为空时，代表此32行120块的大流水、小流水计算结束，然后开启新32行的计算过程。