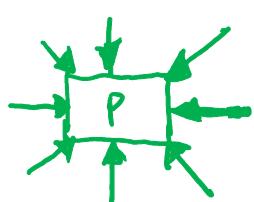
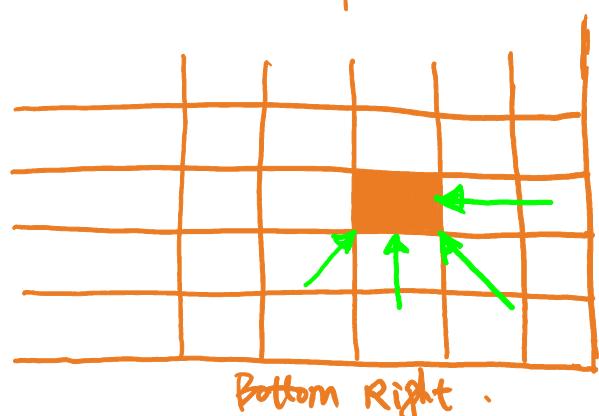
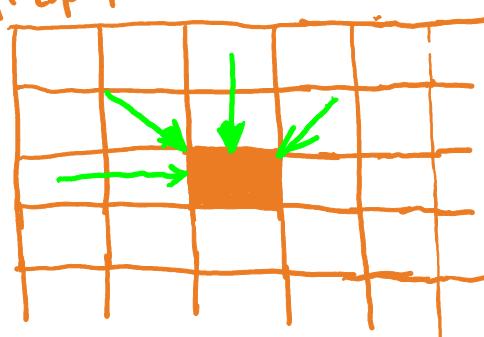


pixelwise cost + cost Aggregation

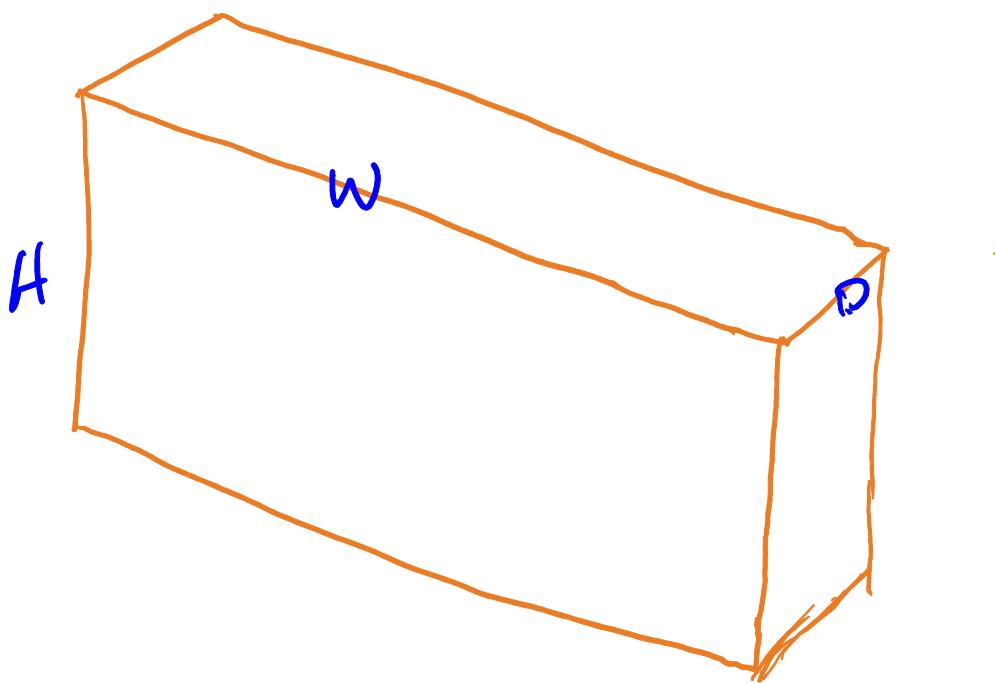
$$L_r(p_r, D_i) = C(c_p, D_i) + \min(L_r(p_r, D_i), L_r(p_r, D_i+1) + p_1, L_r(p_r, D_i-1) + p_1, \min_{k>2} L_r(p_r, D_i \pm k)) - \min_k (L_r(p_r, D_k))$$



Left top path



- memory for pixel-wise cost



- memory for aggregation (8 direction)

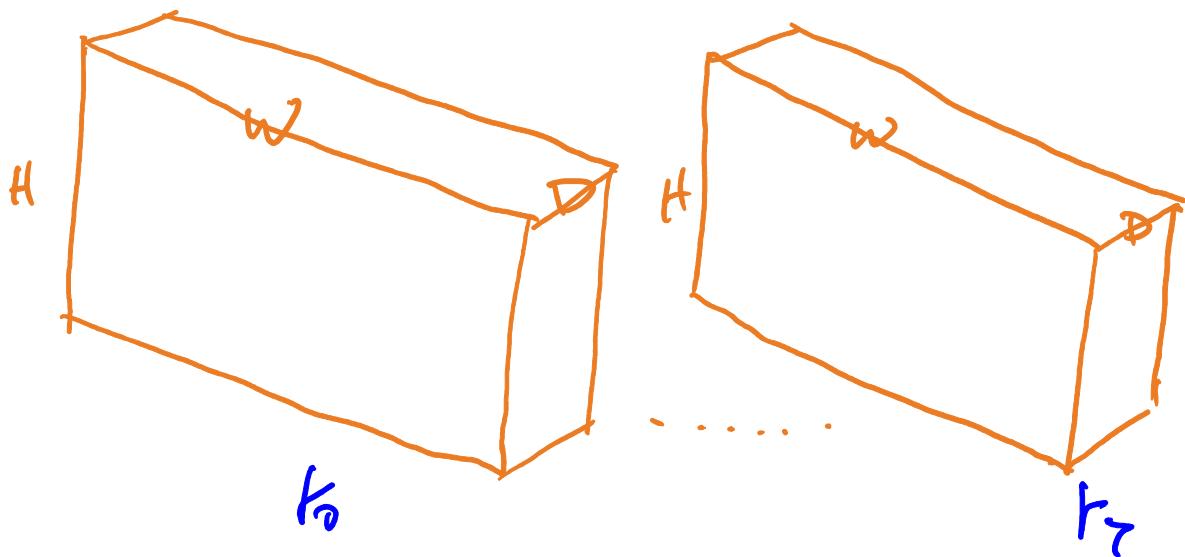


image pixel

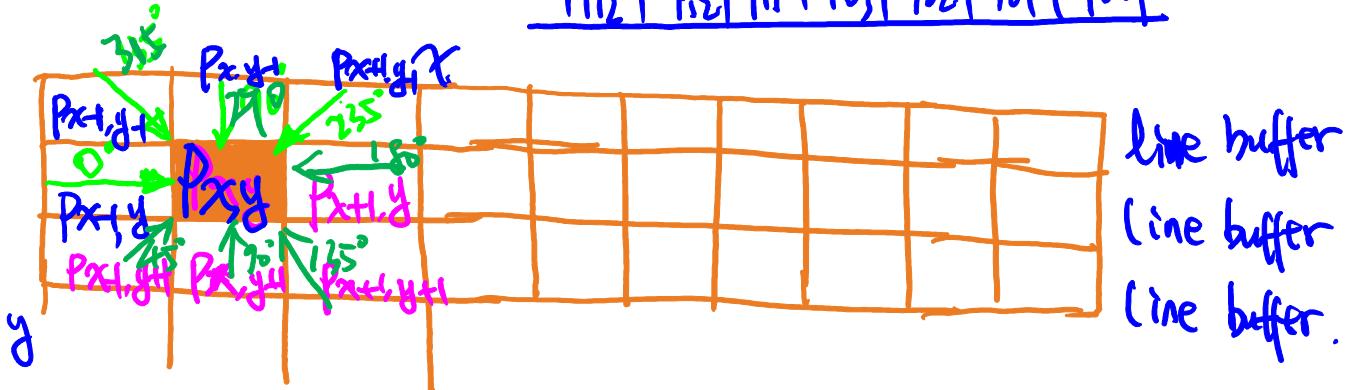
P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}

$\downarrow Y$ $\rightarrow C$

• Data flow.

假设是 census 算法过滤的图像块 \rightarrow Bitstream

$\dots | P_{12} | P_{12} | P_{11} | P_{03} | P_{02} | P_{01} | P_{00} |$



用 pixel stream 的方式代替空间寻址，即只要达到等效图像流完，所有的计算也就做完！

- 用 true dual port Block Ram 代替 line buffer 的移位！
- 简单的流(path)演算。从 0° 方向为例， \rightarrow \rightarrow 实现 Aggregation。

①

P ₀₀	P ₀₁	P ₀₂	P ₀₃
P ₁₀	P ₁₁	P ₁₂	P ₁₃
\rightarrow P ₂₀	P ₂₁	P ₂₂	P ₂₃

②

P ₃₀	P ₃₁	P ₃₂	P ₃₃
P ₁₀	P ₁₁	P ₁₂	P ₁₃
P ₂₀	P ₂₁	P ₂₂	P ₂₃

③

P ₄₀	P ₄₁	P ₄₂	P ₄₃
\rightarrow P ₃₀	P ₃₁	P ₃₂	P ₃₃
P ₂₀	P ₂₁	P ₂₂	P ₂₃

④

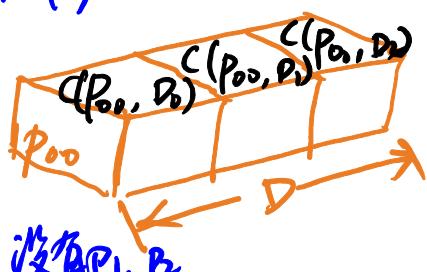
P ₅₀	P ₅₁	P ₅₂	P ₅₃
P ₄₀	P ₄₁	P ₄₂	P ₄₃
\rightarrow P ₃₀	P ₃₁	P ₃₂	P ₃₃

$$L_r(p, D_i) = C(p, D_i) + \min(L_r(p_{i-1}, D_i), L_r(p_{i-1}, D_{i-1}) + p_1, \\ L_r(p_{i-1}, D_{i+1}) + p_1, \min_{k=2}^D \{ L_r(p_{i-1}, D_{i+k}) \} \\ - \min_{k=0}^1 L_r(p_{i-1}, D_{i+k}))$$

$$C(P_{00}, D_0), C(P_{00}, D_1) - C(P_{00}, D_2)$$

第一行 pixel 没有前一行 pixel 的深度约束；但可以

假设前一个 pixel 跟当前 pixel 是一样的，也就是没有 p₁, p₂.



把 P_0 的 $Lr(P_{00}, D_i)$ 保存到一个寄存器组中。

根据 $Lr(P_{00}, D_i)$ 计算出 $Lr(P_{01}, D_i)$

计算完 $Lr(P_{01}, D_i)$, 把 $Lr(P_{00}, D_i)$

存储到 DDR 中, $Lr(P_{01}, D_i)$ 继续

留在 TempRegArray 中用于计算 $Lr(P_{02}, D_i)$
中。

漏了步, 计算 $C(P_{01}, D_i)$, 这一项不是这样算的。

计算 $Lr(P_{01}, D_i)$ 之前要到双端口 Ram 里去

取相关的数然后面计算 census 的距离。这个值每个 $Lr(P_{01}, D_i)$
只对应一个, 因此不需要另外开辟 Register, 直接存储在 TempRegArray
里面。

以此类推, 口方向的 Aggregation 很好计算。

- ↓ 90° 方向也可以算。

- ← 180° 方向也可以算, 只要 line buffer 够长。

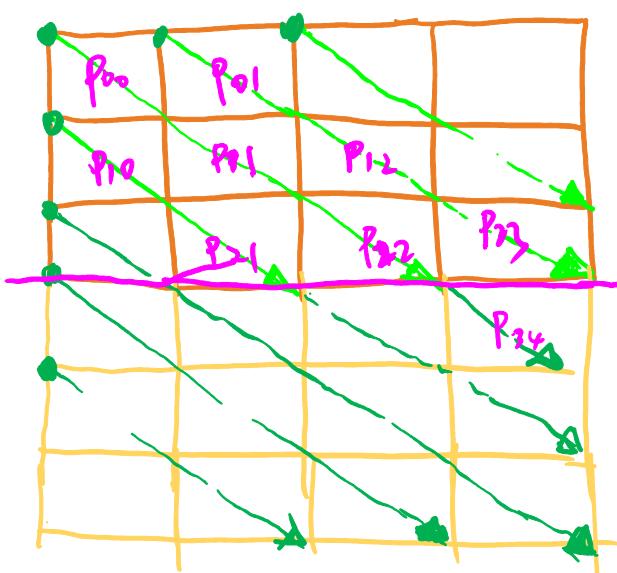
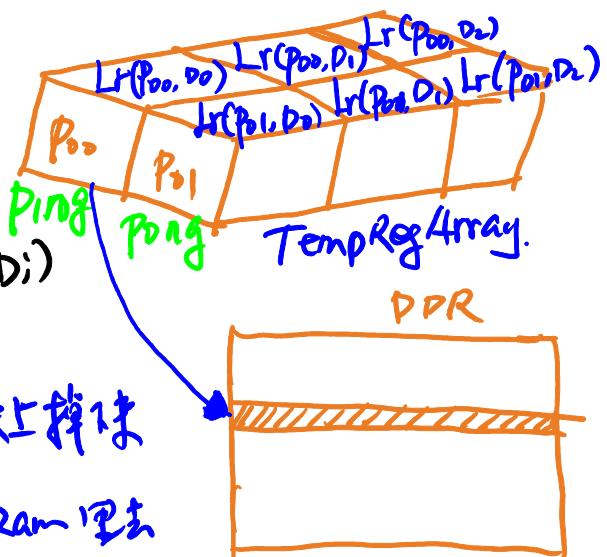
- ↙ 225° ↘ 315° ?

有多少条 ↓ 和 ↘ 的 scan
线路, 共 $(W-2) + (H-2)$?

由图可知, 边线路也要处理。

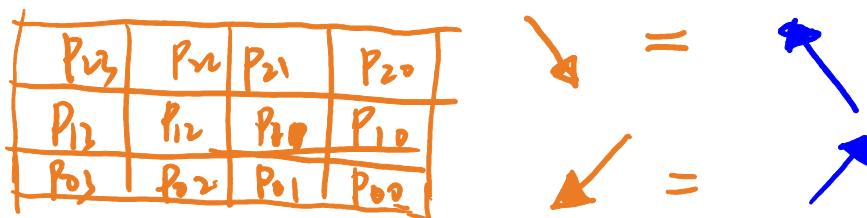
$P_{00} \rightarrow P_{11} \rightarrow P_{22} \dots \rightarrow P_{44}$ (从左到右)
每条线路要记住什么时候完成。在循环的 line buffer 里, 只有有限的距离才能
完成。

而且每一次都有新的线路会开始或结束。

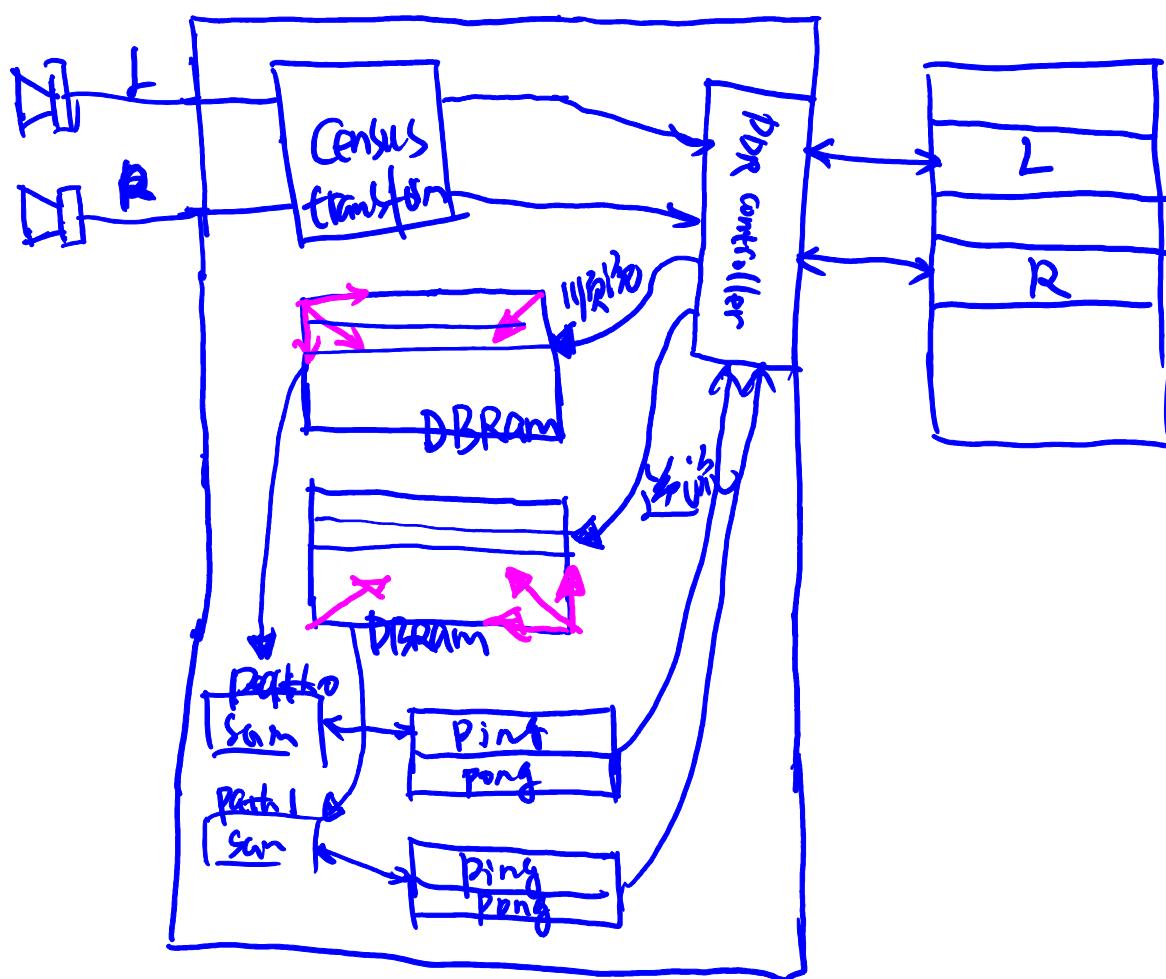


- \rightarrow \leftarrow \rightarrow \leftarrow 这两个方向只能是把图像数据流到过来往的方向。

答：



- 实现同时实现在上角和右下角 patch 的方案如下：



- 若再加上一个片外 SRAM 补充片内 Blockram 的不足，用伴映缓存的 Buffer/Cache 不是很妙？成本会上升多少？

• 第一步先根据你的优势确定你的第一款产品！

- 3D机器人导航项目，你还是尽早放一边，过隧道2D的都还没稳定，你做3D，你有方案可用于量产么？3D还处于研发阶段！等成熟了，等别人试验过了，再进场不迟，市场那么大你还有分不完！
- AR/VR 定位的可以考虑，这个反正是游戏类的，对稳定性要求不高。
- 所以做产品一定要脚踏实地，先用比较成熟的方法 ~~sgm~~ 做一款，虽然不完美，但是能用于 AR/VR，能卖的产品！
- 等需要了，你看到了这市场，但你却没自信，到处乱说，这个项目说实话投入不需要多少钱！需什么风投，天使，十万美元搞起来的，到处谈风投！！
- ~~一些~~ ^{应用} Demo 是要做的，比如同双目做机械人导航，人物跟踪，不要花多少时间占弄，主要是同炫酷的功能告诉客户，给他信心！
 - 不要小看为同这个产品，能量产的还真不多，最主要，也在也估计只有 AR/VR 市场可用。你就做好这件事情就够了！

• 还要做深度 refinement 吗？ 先不用了，~~sgm~~ 目前这个效果已经不错

能卖了！要优化，可以尝试以下几点：

- ① 在做 coarse-scan 之前将图像做 segmentation，颜色相近的一块，给予比较一致的权重。

权重

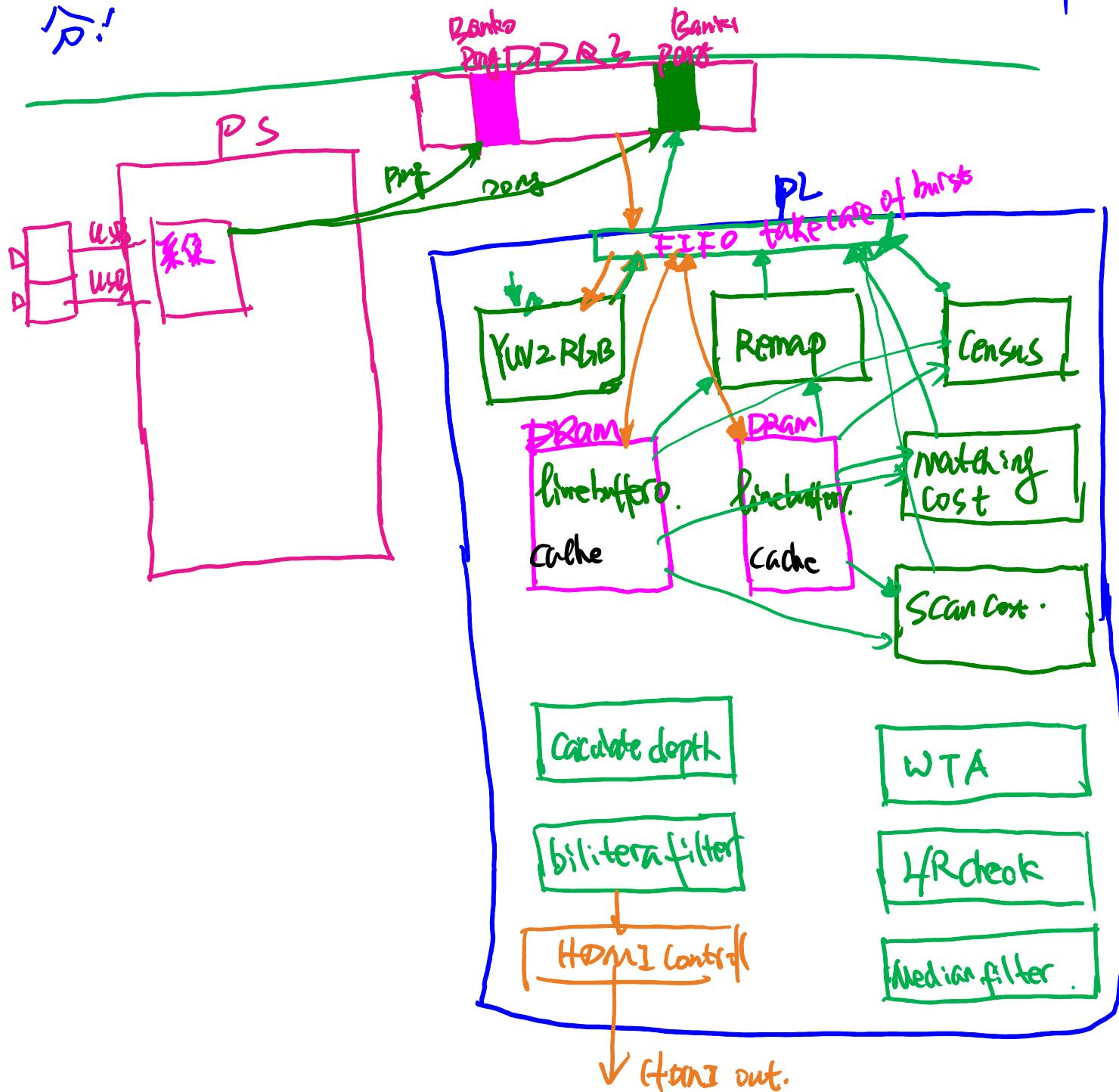
$$S = \sum_{\text{all}} (C(p, p_i) + \sum_{p \in \text{NP}} (\max(D_p - D_{p_i}, 0) \times p_1 + T[\max(D_p - D_{p_i}, 0)] \times p_2))$$

在实际测试中发现，白墙对取“全局”最优时影响很大，因为可以先标注出特征点不明显的区域。在加 $C(p, p_i)$ 时要乘以权值，这个权值跟 feature 的强弱有关，纹理越强，权重越大，越小权重越小，如白墙一点特征点都没有的区域直接丢弃。这样在做最优传递的时候无操作点的地方全局最优值影响就大了。

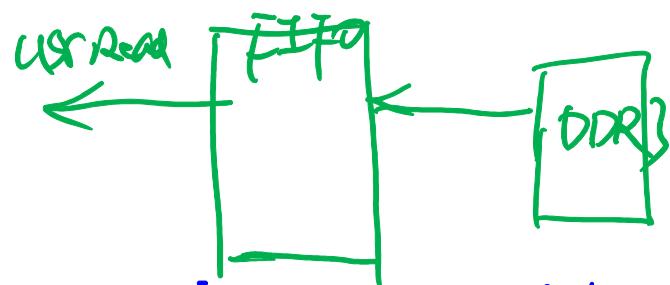
- 另外，在最局 Refinement 的时候，对同一区域内的深度做处理。假设同一区域的深度不跳跃。不然，已经包含在这个公式里了！当然也有可能会发现跳跃，有跳跃就丢掉！
- 另外一个最靠谱的 Refinement 方法就是用 Vslam 的思路，用每个角被来估计最后的 3D坐标，也就是包含深度了！

● 双向Q PS 和 PL 任务划分。

考虑到 FPGA 视频采集的复杂性，要跟 PL 端共享数据交换速率，可以先考虑将 PS 端采集的数据在 DDR3 中开辟 2 对 Ping/pong buffer 用于存储 YUV 数据。因为只有双目的参数，可以将 Remap、像素模块放到 arm 上运行，这样方案改动相对较小，实现起来比较快！只优化耗时部分！



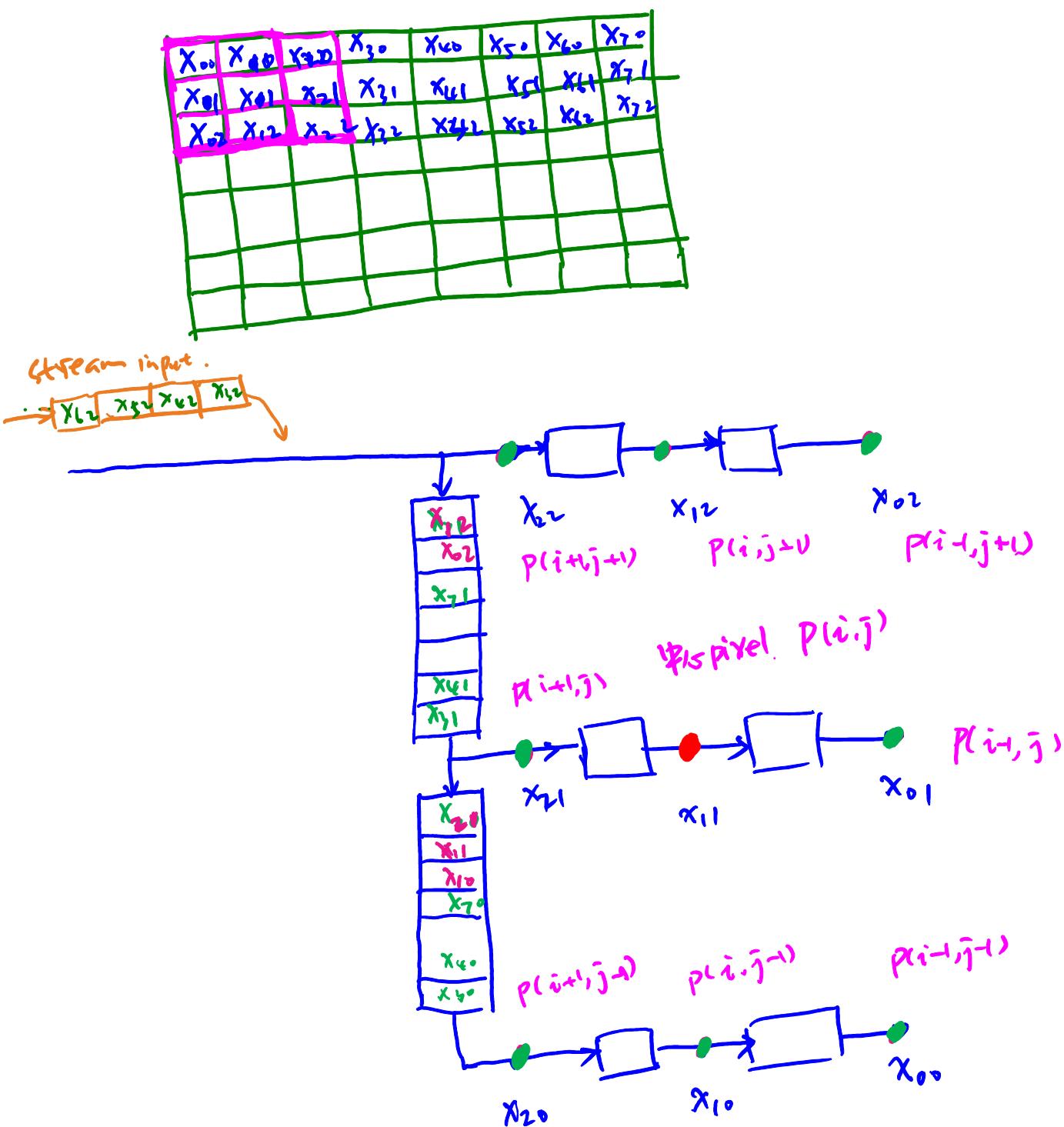
- 能允许多 FIFO 用于 line buffers。总共 Block Ram 有 4.9 mb, LUT 53,200, flip-flop: 106,400.
- 缓存 5 行 in line buffer 需要 $480 \times 5 \times 8 \times 2 = 38400$ bit
 \dots 7 行 $\dots = 480 \times 7 \times 8 \times 2 = 53760$ bit.
- 每个模块都顺序执行，意味着要往返读写 DDR 很多次，如果有大很大的 SRAM 作为帧缓存就完美了！相当于有一个 Cache！
- Anyway 因为配置就如此，能否尽量减小对 DDR 的读取？
如果尽量做到 DDR 的 Burst 最优化，反复读写的情况有哪么糟吗？
- 如何尽量做到 streaming, pipeline，如何保证在 FIFO 中每帧数据能完整传输，是否多打入数据。因为如果 burst 是 128*8b. n，这个良率，DDR 是不写入或读出的???
- 如果不用担心。拿读来说，FIFO 就是可读空的。



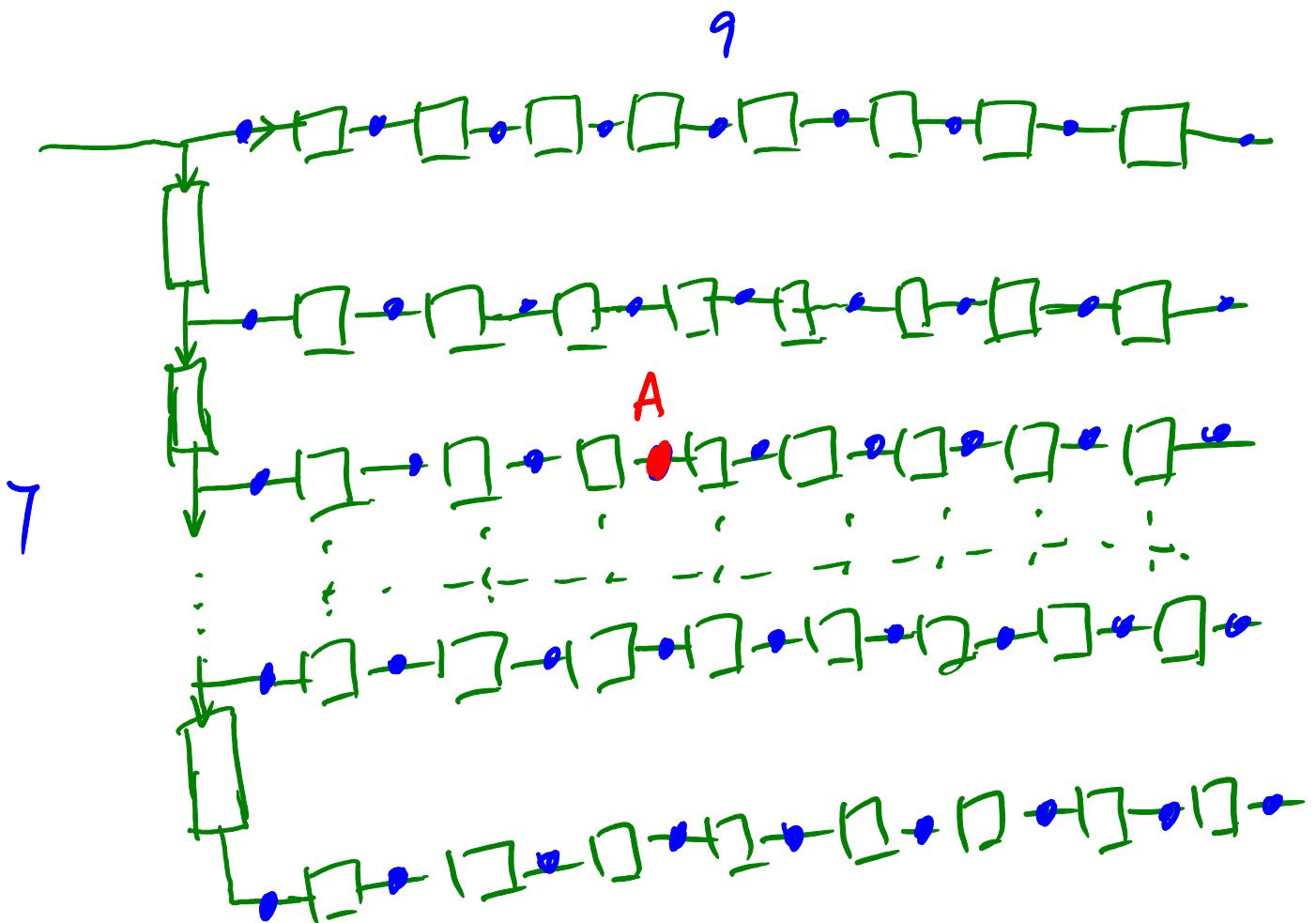
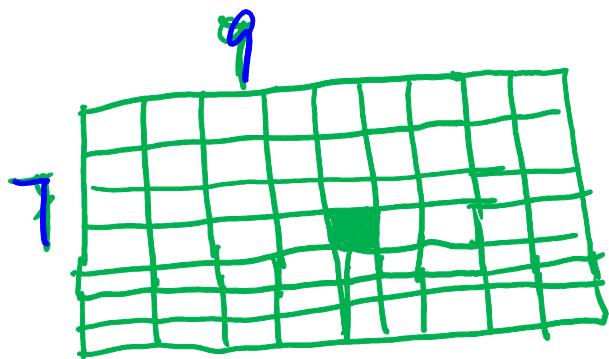
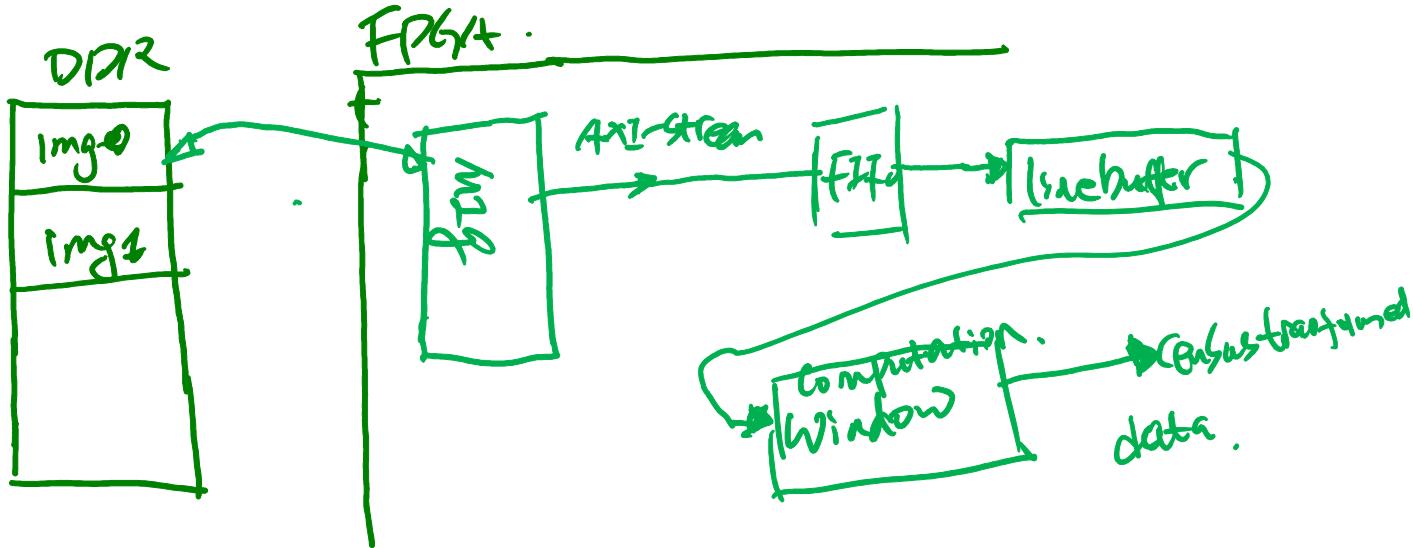
如何保证读完整一帧，必须保证 image width 是 burst 的整倍数！！这样 FIFO 里才不会有残留的非 burst 填充的数据。
每次读完一帧数据 FIFO 空了，一帧数据从 DDR 传入 FIFO 中写入！

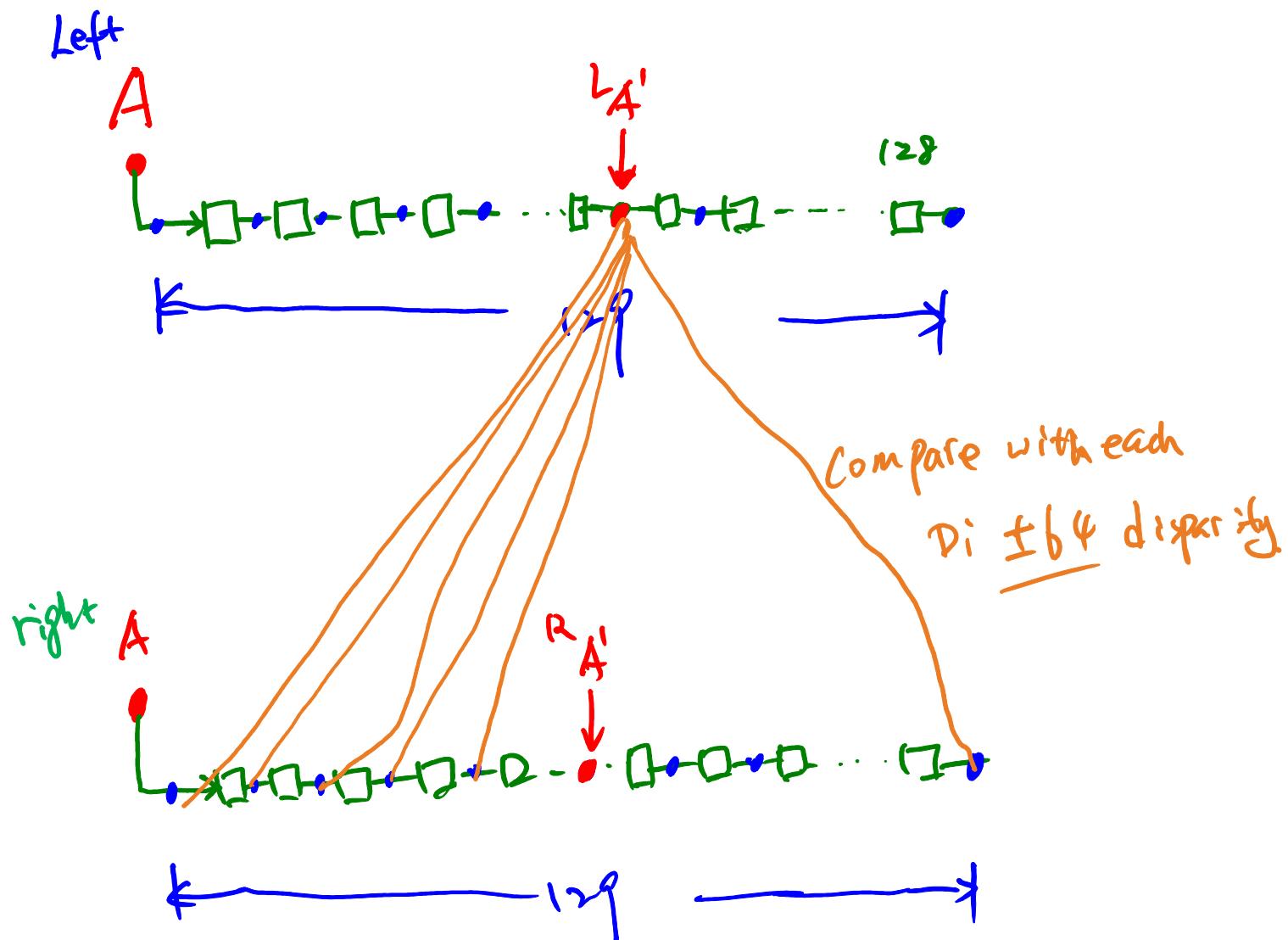
- ARM 能否分担一些任务？
- 如何利用双核？

从线缓冲区 (linebuffer) 的公共部分是错的。

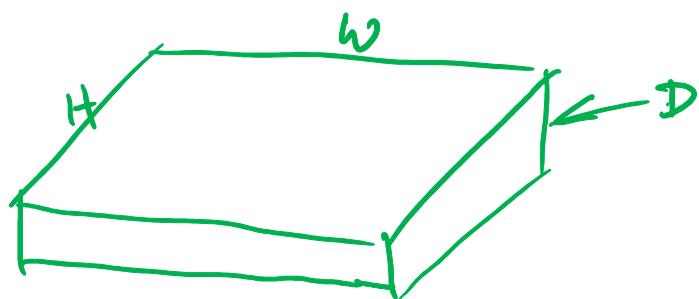


• census transform 模型.





- 这些都可用FIFO代替，进位模块的数据率为 $1:128$ ，用不同的采样时钟解决。
- 最终得到的是 matching cost 为 3 维数据块。

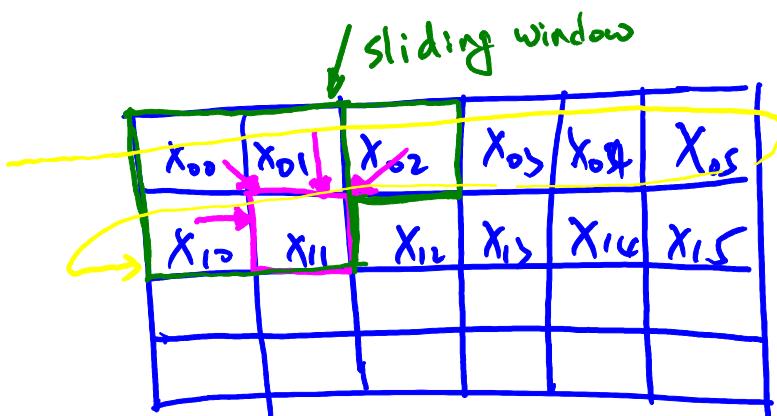


- GPU的算法流程为 $8bit \rightarrow \text{Census} \rightarrow 64bit \rightarrow \text{DDR} \rightarrow \text{Matching cost}$. 对 DDR 有 3 次读和写操作。FPGA 可以把 census 和

Match cost 放在一起做！减小对 DDR 的操作：

- 另外，应该把 line buffer 部分独立出来，给多个模块共用！

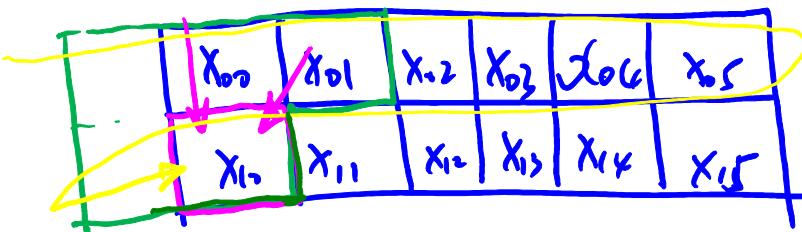
- Scan cost 在 FPGA 流实现。



数据流

新方向都要缓存(最后一行)。

数据流滑出 line buffer $\leq \frac{1}{2}$.
用 WTA 找出每个 P 点的

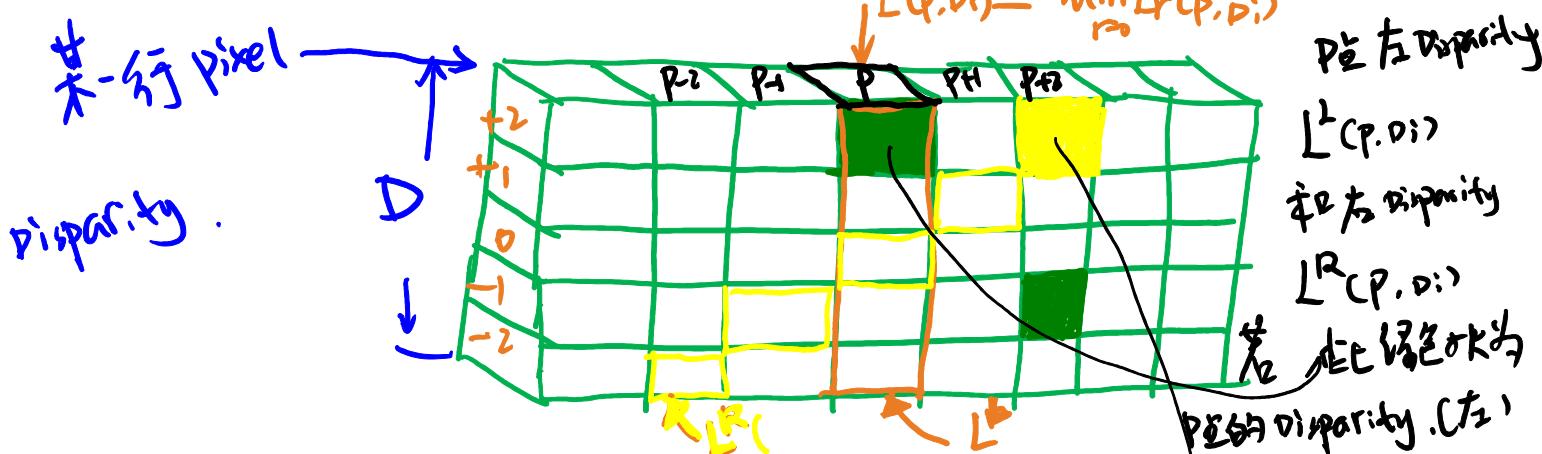


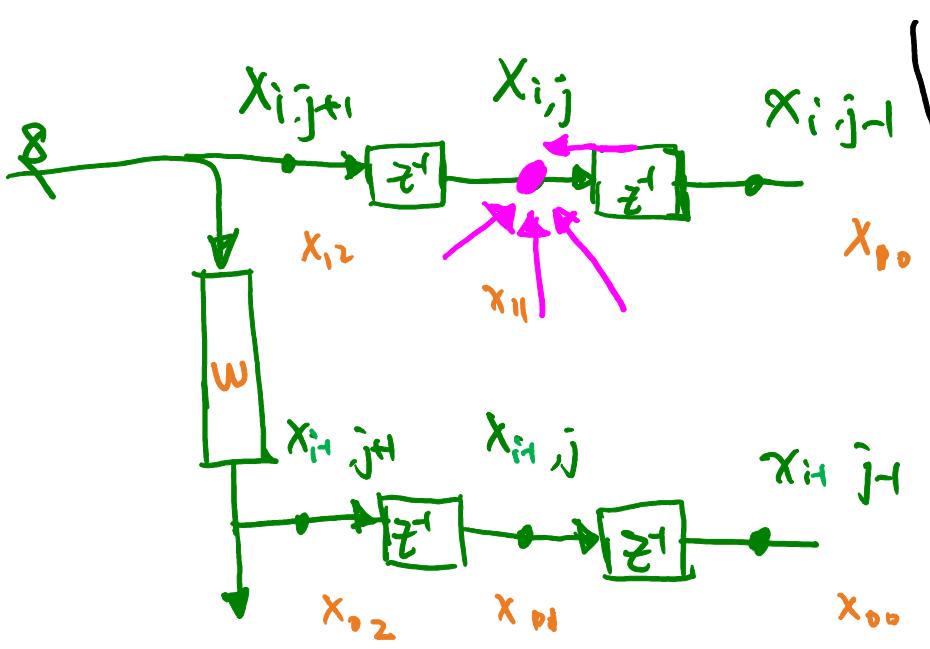
$\min_{d_i} L(p, d_i)$

由左向右check disparity.

由此可知，要求 $P(i, j)$ 四个方向的 $L_r(p; j, d_i)$ ，只需要缓存上一行的 $L_r(p_{i-1}, j, d_i)$ 的值，sliding window 一直滑行就可以求出各个 $P(i, j)$ 的值了。

- 知道了上述规律如何用数据流方式实现，即纵向加权的 L

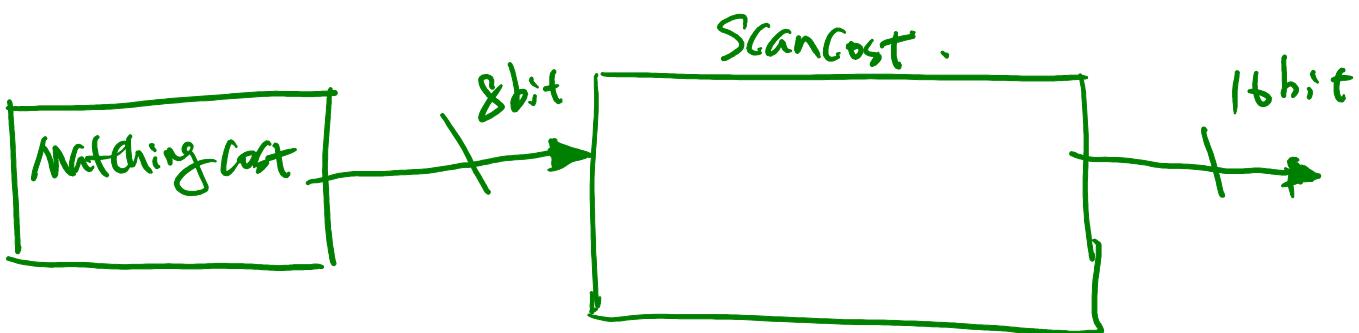




那么，其对应的(D_i) Dispaly
应该是白色的黄色块，应该
不等于 P+2 号那一列的
最小值 L(P,D_i) 即绿色块。

所以，如果左 Dispaly 对的
该绿色块和黄色块
应该值是一样的。这样
就计算出了右
Dispaly

- 注意边界情况，用小的数据来验证想法。



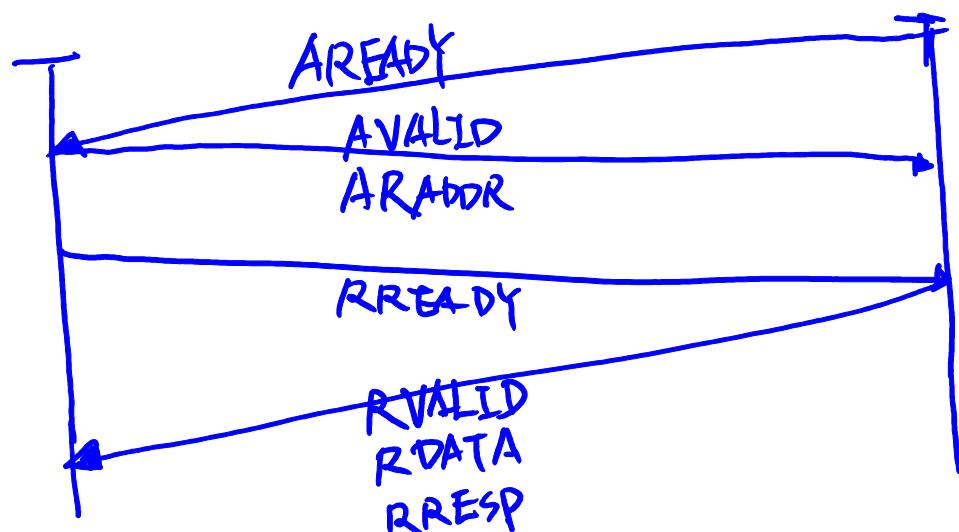
- 模块化设计。先定义好接口，在验证阶段，还是假设 Scan_cost 模块的数据从 DDR 读出来，通过 AXI-stream 流的方式获得。这样做的好处是，前面的模块可以通用 PS 芯片实现，省去模块一个模块地实现！

• AXI 总线.

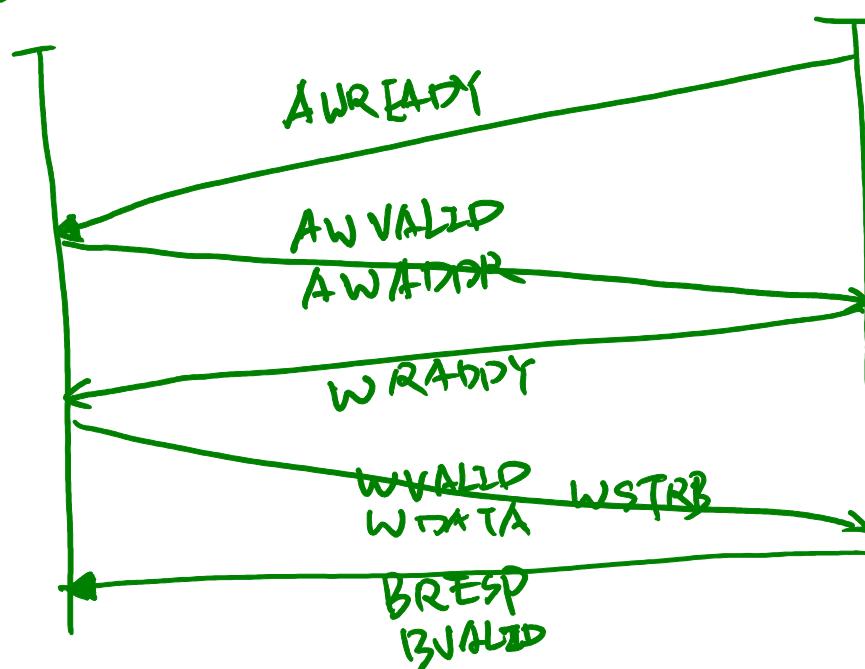
分为5个通道信号

- 读地址通道: ARVALID, ARADDR, ARREADY
- 写地址通道: AWVALID, AWADDR, AWREADY,
- 读数据通道: RVALID, RDATA, RRESP
- 写数据通道: WVALID, WDATA, WSTRB, WRREADY,
- 写应答通道: BVALID, BRESP, BREADY.

主 从



主 从

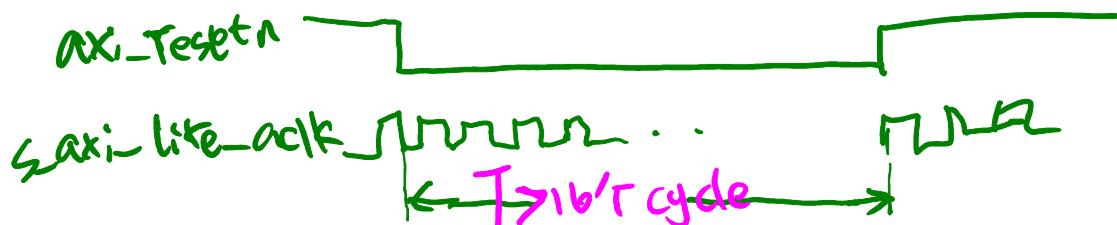


Tuser = 用于纹理信号。

Tstrb : 为1的bit为TJL+data有效字节，宽度为 tdata/8

● VDMA

- Reset : global reset , power on Reset .



- Soft reset (only affect the corresponding channel)

● programming sequence .

- ① Write VDMACR (offset 0x00 for mm2s, and 0x30 for S2mm) , set VDMACR.RS=1 to start the AXI VDMA channel Running
- ② Write a valid Video frame buffer start address to the channel START_ADDRESS register from 1-N. (offset 0x5C - 0x98 for mm2s and 0x1C - 0x18 for S2mm)

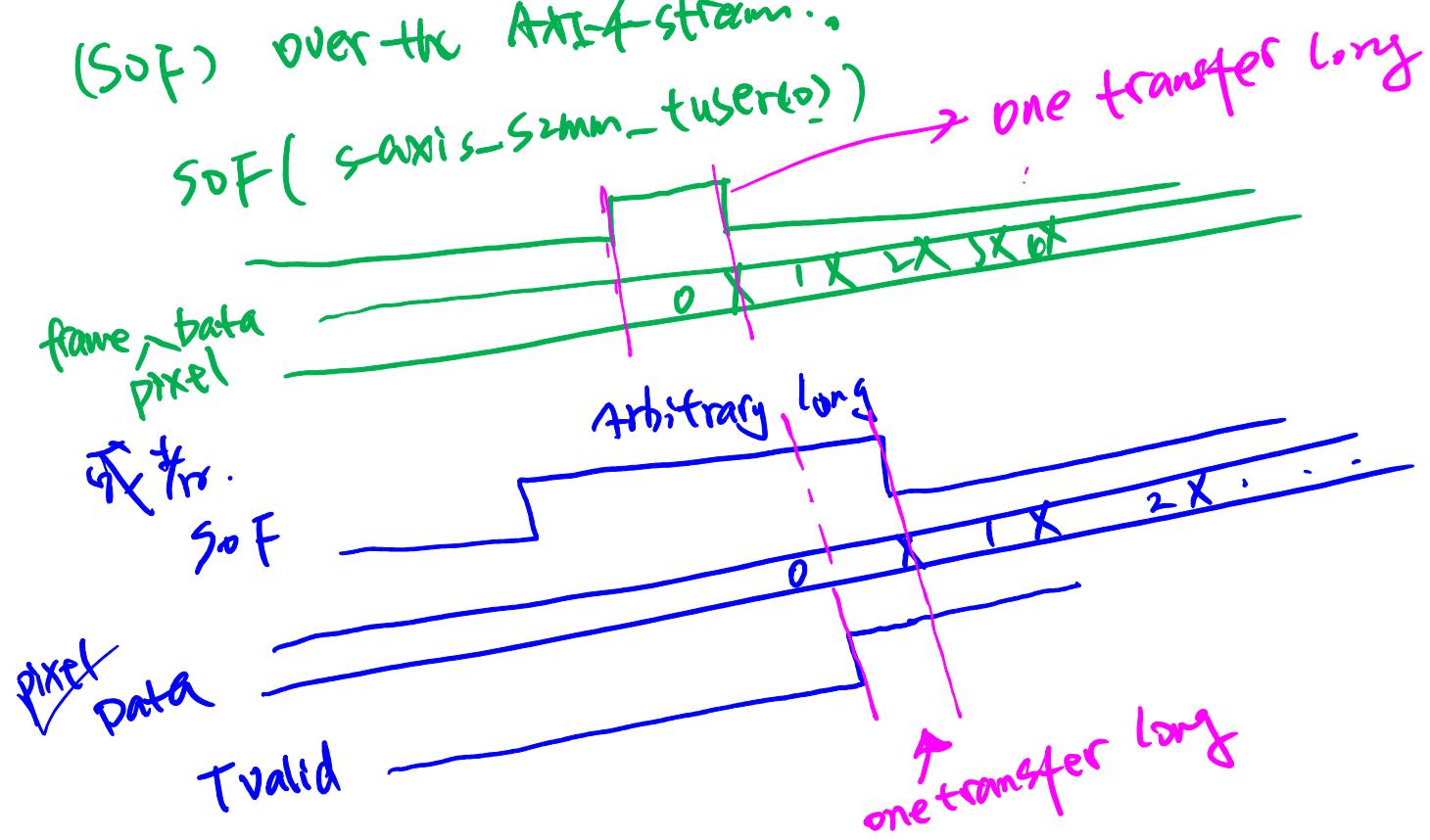
Set REG_INDEX ?

- ③ Write a valid frame Delay ? (Valid only for GoniLock slave) and stride to the channel FRMDLY STRIDE register offset 0x58 for mm2s, and 0x18 for S2mm .
- ④ Write a valid horizontal size to the channel HSIZE register (offset 0x54 for mm2s, and 0x14 for S2mm)
- ⑤ Write a valid vertical size to VSIZE register
see page 50 for detailed process .

- The newly written video config parameters takes effect on the next frame boundary after you write to vertical register for the respective channel. 写入 VSIZE 后要触发它.

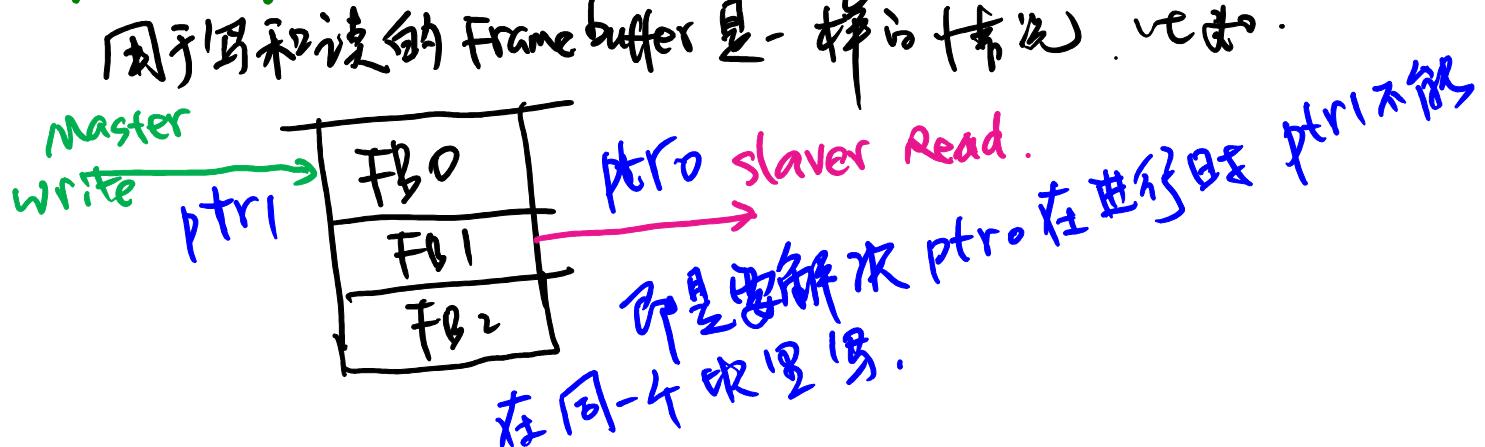
写入立显大小后，记住先写其之register设置，最后写VSIZE。下一个Frame 自动生效。

- S2mm-tuser :
When selected, the AXI VDMA expects the start-of-frame (SOF) over the AXI-stream..



- Genlock mode .

用于写和读的 Framebuffer 是一样的办法 . vdo .



这里是 master, 当前写操作的块 index 出现在 sum-frame_ptr_out 上。
 &假设置为 slave mode, 则 index 跳过 master 从 index 0.
 由 frame delay 次之保持多个 frame 间隔.

- 另外注意 Data alignment.. 32: 每个 Frame 的起始地址必须
需要是读写宽度的倍数. 例如 32-bit., 则必须是 4 字节
对齐. stride 也必须是这样

• FSync option: 帧同步信号.

双目的图像采集可能需同步, 同时同步图像?

when unchecked, it runs freely as quickly as possible. without
waiting for external trigger signal.

• Mm23 fSync: m_axis_mm2s_tuser[0] 为门控 trigger. (50F)



左下降沿(即开始处理, -Mm23 trigger).

Rectification:

$$p' = (R_{rect}^T)^{-1} p$$

$\equiv G$

$p' = M_r^T D_{rect} ((R_{rect})^T p')$

↑
5x1 distortion coefficients.

↑
Camera matrix. 3x3

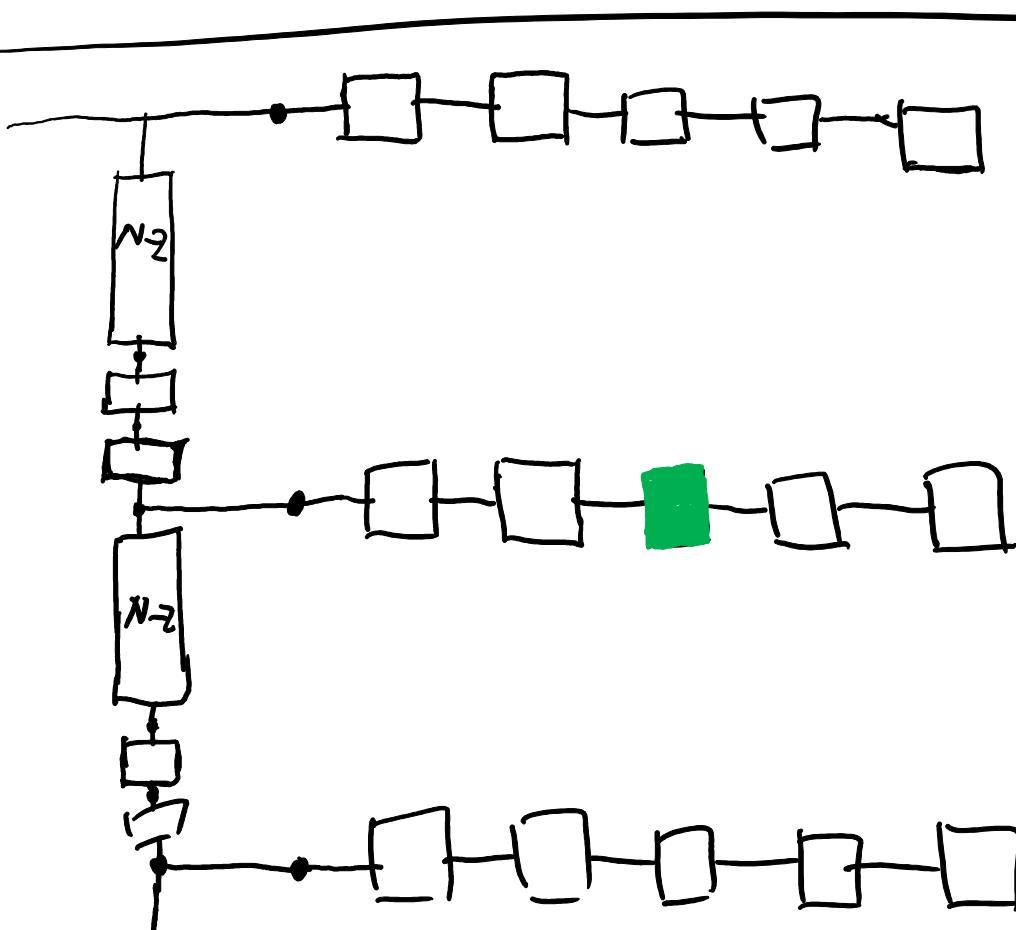
↑
Rectified camera matrix.

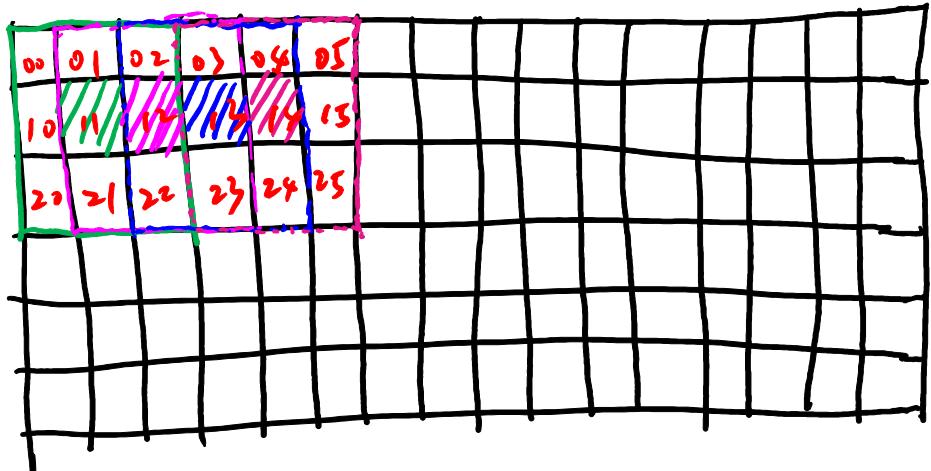
3x3
3x3
rotation matrix.

$$P' = \begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} \xrightarrow{\text{prest}} P'' = \begin{pmatrix} x'' \\ y'' \\ w'' \end{pmatrix} \xrightarrow{\text{middle}} P = \begin{pmatrix} x \\ y \\ w \end{pmatrix} \xrightarrow{\text{source}}$$

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = G \cdot P' = \begin{bmatrix} ir[0] & ir[1] & ir[2] \\ ir[3] & ir[4] & ir[5] \\ ir[6] & ir[7] & ir[8] \end{bmatrix} \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \xrightarrow{\text{j}} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \xrightarrow{\text{i}} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$P = M_r^{-1} \xrightarrow{\substack{\uparrow \\ 3 \times 3}} \xrightarrow{\text{Distortion } (P'')} \xrightarrow{3 \times 1}$$





$$\frac{640 \times 360 \text{ Byte}}{140 \text{ MBps}} = 0.01646 \text{ s/frame} \Rightarrow 60.6 \text{ frame/sec}$$

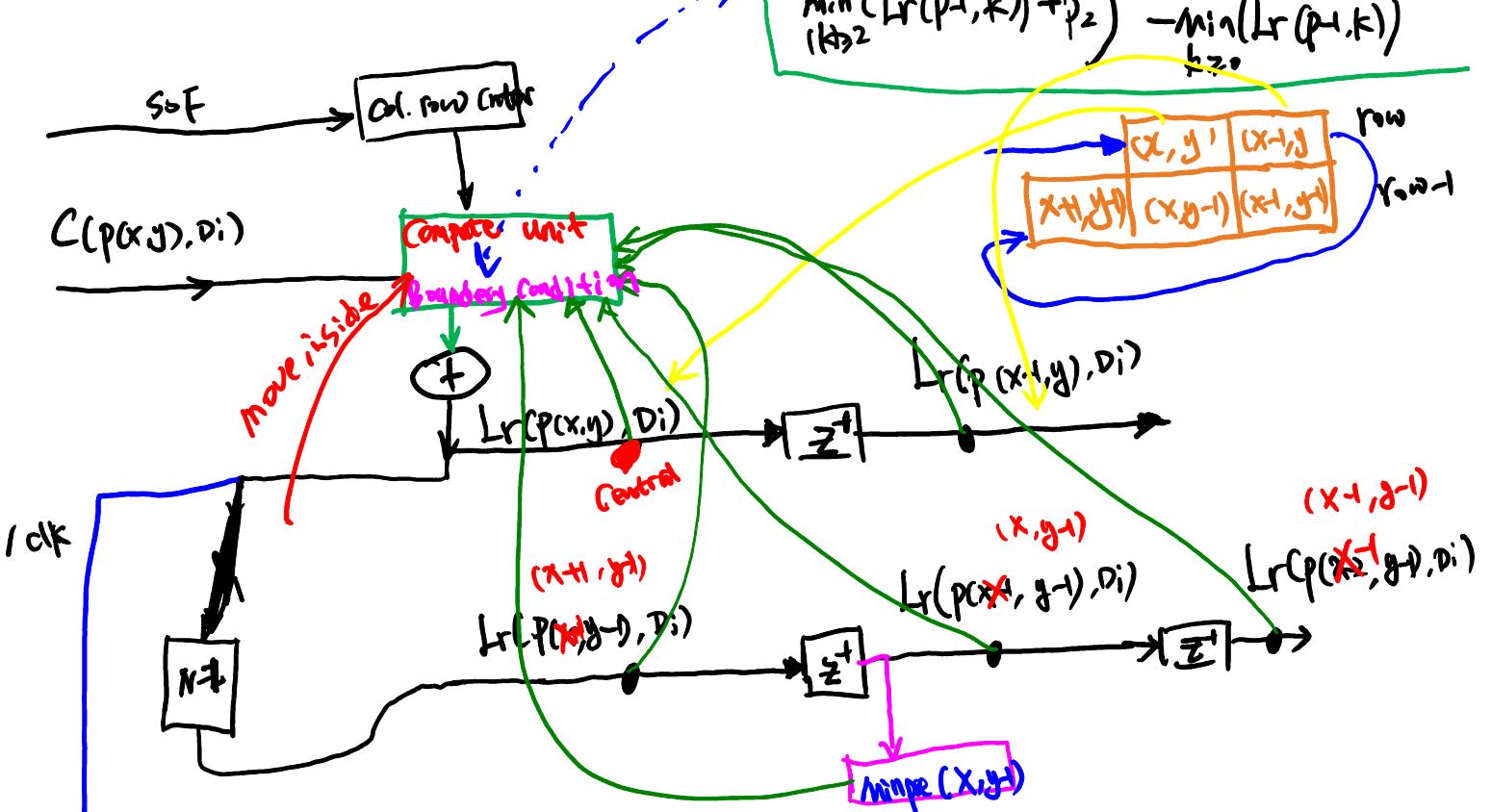
$$\frac{1920 \times 1080 \text{ Byte}}{140 \text{ MBps}} = 0.0148 \text{ s/frame} \Rightarrow 67.5 \text{ frame/sec.}$$

$$\frac{1280 \times 720 \text{ Byte}}{140 \text{ MBps}} = 0.00658 \text{ s/frame} \Rightarrow 151 \text{ frame/s.}$$

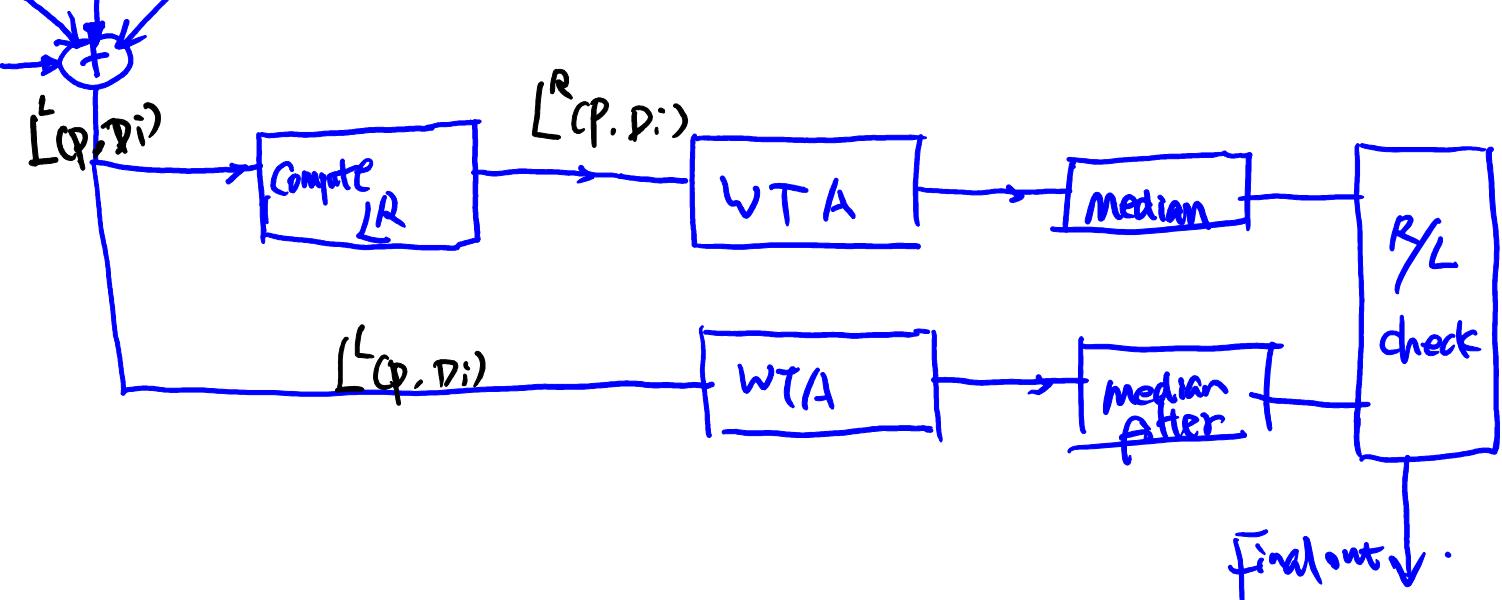
一定要速率，没必要做的太高，够用就行。

$$Lr(p(x,y), D_i) = C(p(x,y), D_i) + \min(Lr(p-1, D_i), Lr(p1, D_i-1) + p_1, Lr(p1, D_i+1) + p_1,$$

$$\min_{1 \leq k \leq 2} (Lr(p+k, D_i) + p_k) - \min_{k \geq 0} (Lr(p+k, D_i))$$

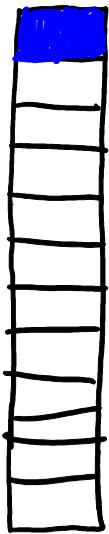


注意反馈电路的时延
要在上图的基础上加上.



Compute unit

parallel Compute



minIndex
 $L_{r_0}(p_i) \text{ minIndex}$

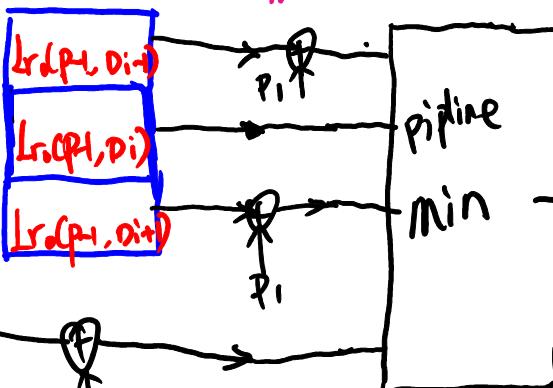
$L_{r_0}(p, D_i)$

$L_{r_0}(p_i, D_i)$
 $L_{r_0}(p_i, D_i+1)$

$\min L_{r_0} + p_i$ 小于 $\min L_{r_0} + p_1$.

$L_{r_0}(p, D_i)$

minfre



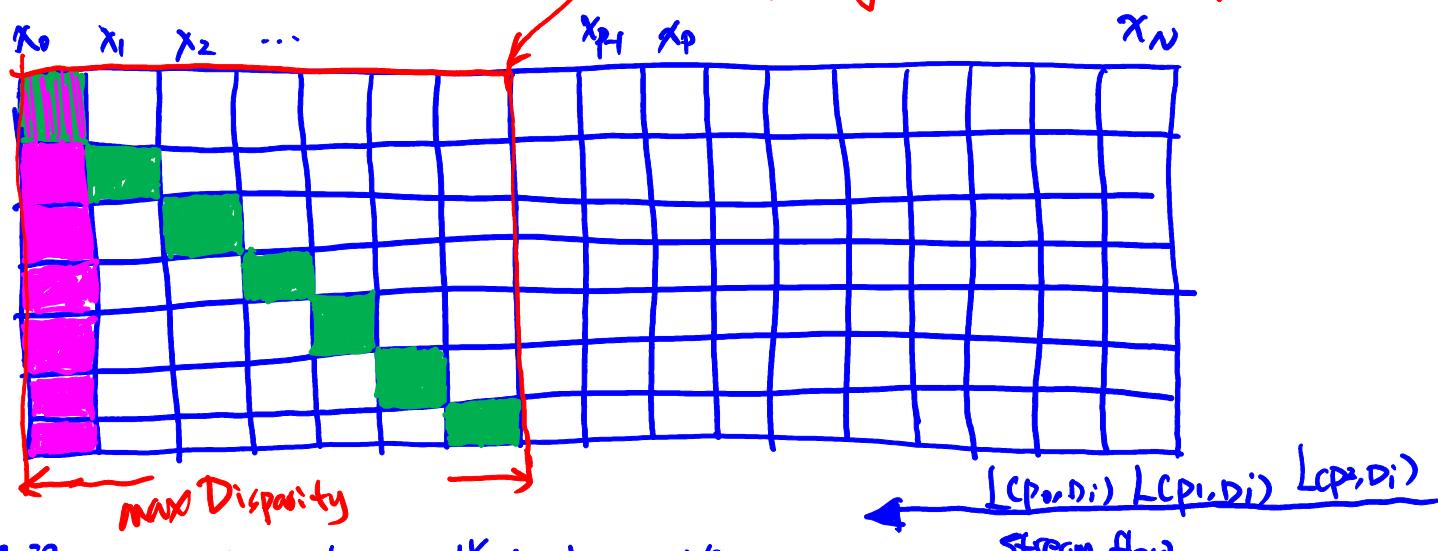
边缘情况处理:

$D_i = 0$ 时 (即最顶部的单元格)
特殊处理?

$L_{r_0}(p_1, D_i)$ 这个模块的输入是 $L_{r_0}(p_1, D_i+1)$ 的值。
 $D_i = D-1$ 时, 同样 $L_{r_0}(p_1, D_i)$ 的值。

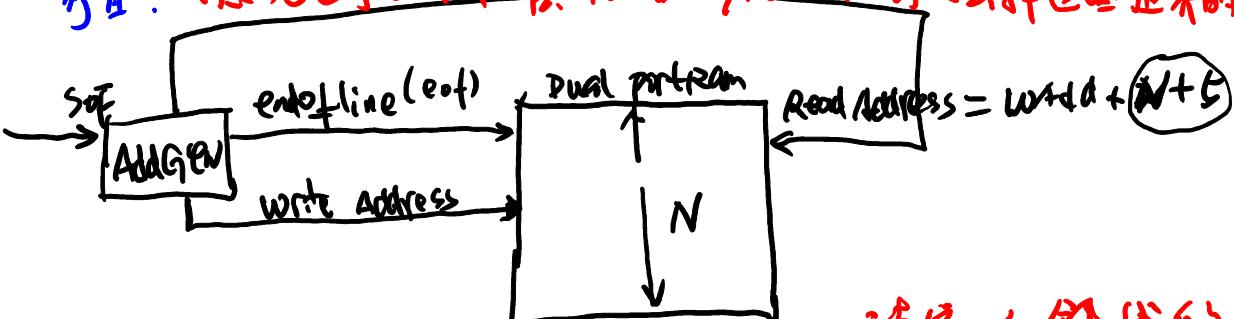
- 增加一个
AD/选择二
向路, 同时输出 $L_{r_0}(p, D_i)$.
有 D 个 output. 分别是 D_i .

Compute Right $L^R(p(x,y), d_i)$ 用 shift register array 来实现.



① 保证 Ram 里有 D 个 $L(p, d_i)$ 数据才能计算.

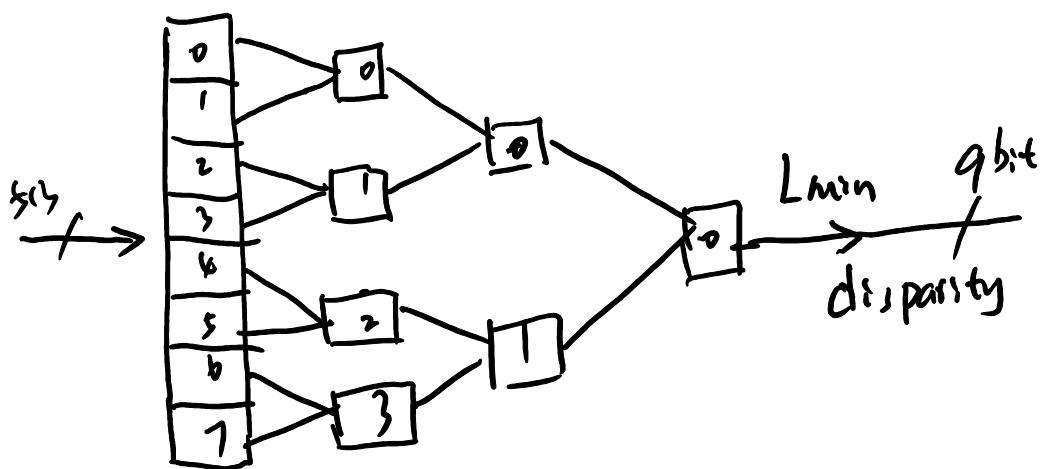
② 注意边界条件. 行头也好,主要是行尾, 计数器要控制好, 不要斜插入下一行的值. 我感觉也跟而不冗余. 后面 L/R check 可以去掉这些边界的点, 建议设三部分.



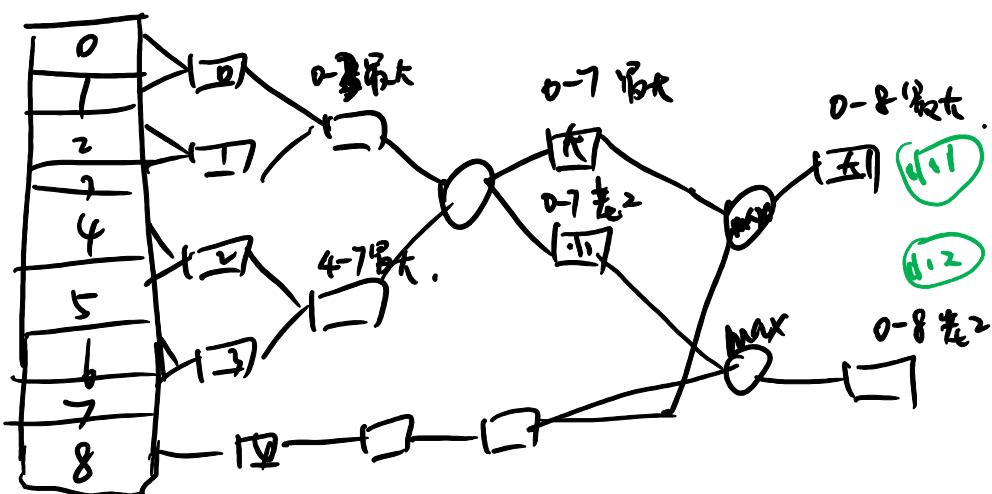
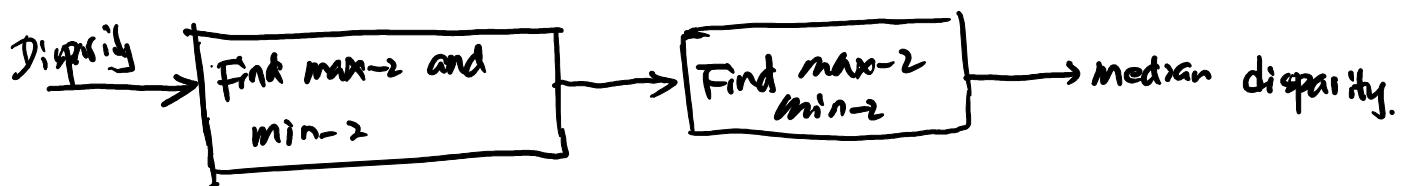
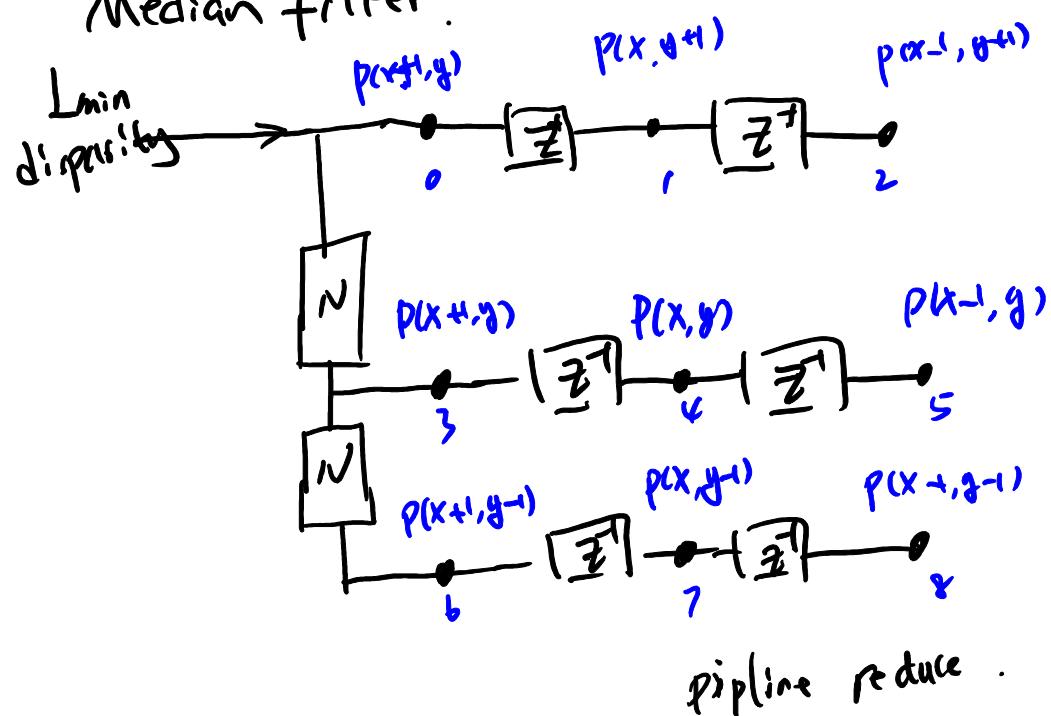
重要更正: Dual port Ram 里无法一个 clock 读完一个邻线的. 这里只和 L/R shift Register Array 来实现).

Winner takes all :

$$L(p, d_i)$$



Median filter.



或者，相同结构一个找最大，一个找最小，这样就找出前2%和倒数2%，同时去掉，剩下的再找出前2%和后2%，剩下的就是中间卯位了。

L/R check.

