FASTOR重复读出bug复现和修正

汇报人：邓思琪

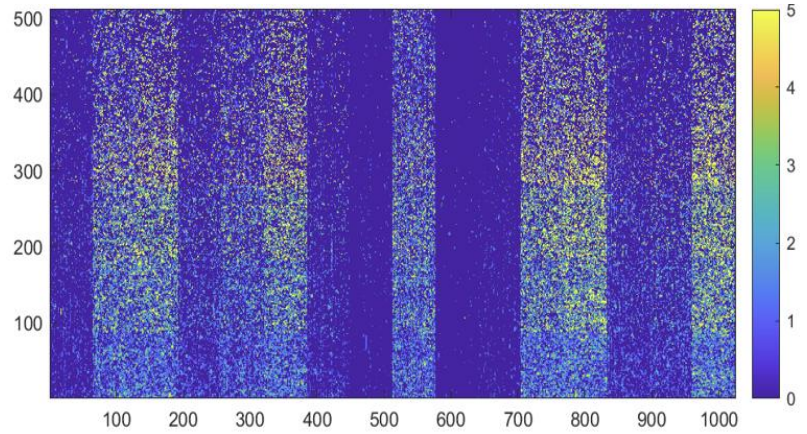
导师：魏微

时间：2025.10.23

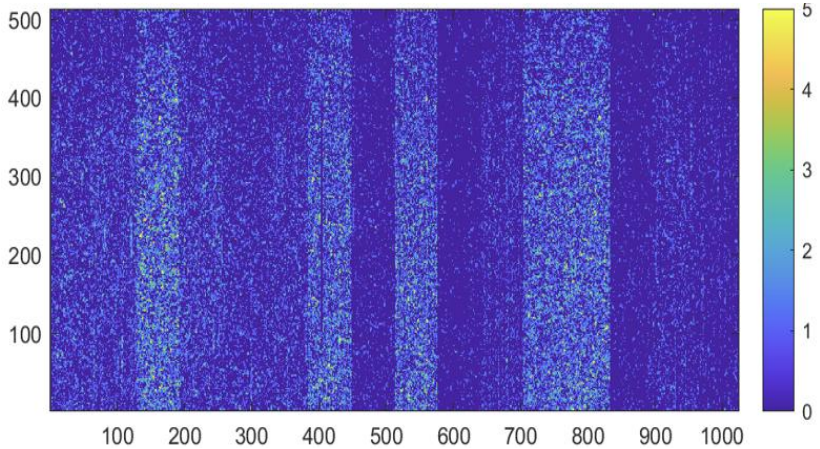
目录

- Taichu3原设计bug复现与修正
- Stitching设计参考

背景



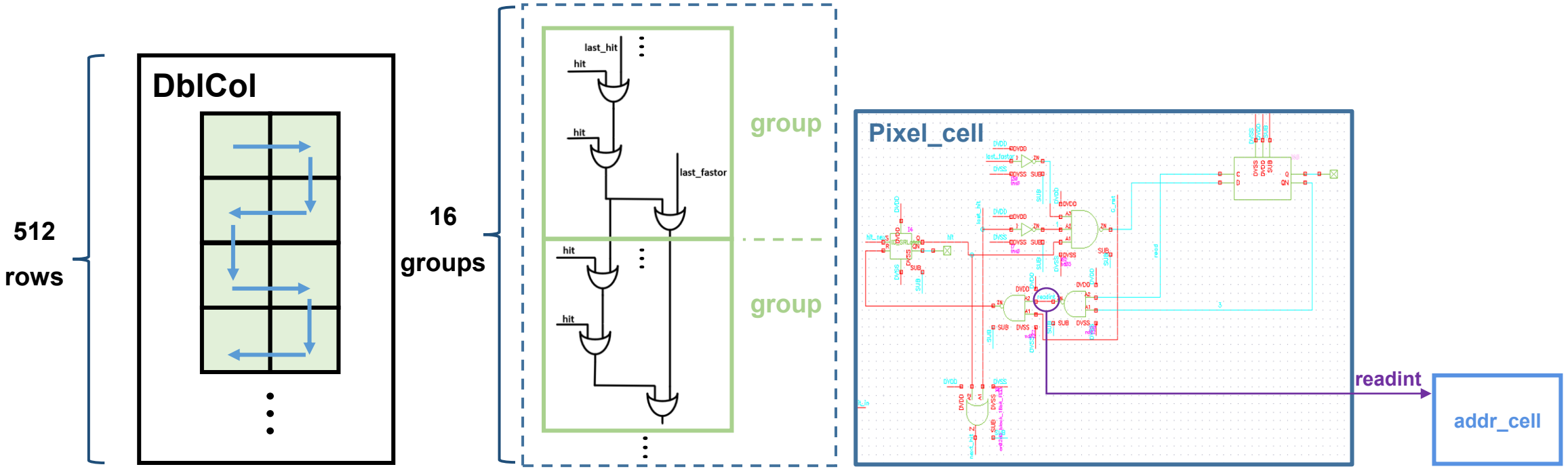
Taichu3测试成像图（原始）



Taichu3测试成像图（已清除重复数据）

- 较高行的像素在被单次击中时会有重复读出
  - 行号越高，重复数越多
- 形成原因：
  - 由于阵列和连线的延时过大，FASTOR信号无法被及时清除
- 目前解决方法
  - 在FPGA中监测重复数据并将其清除

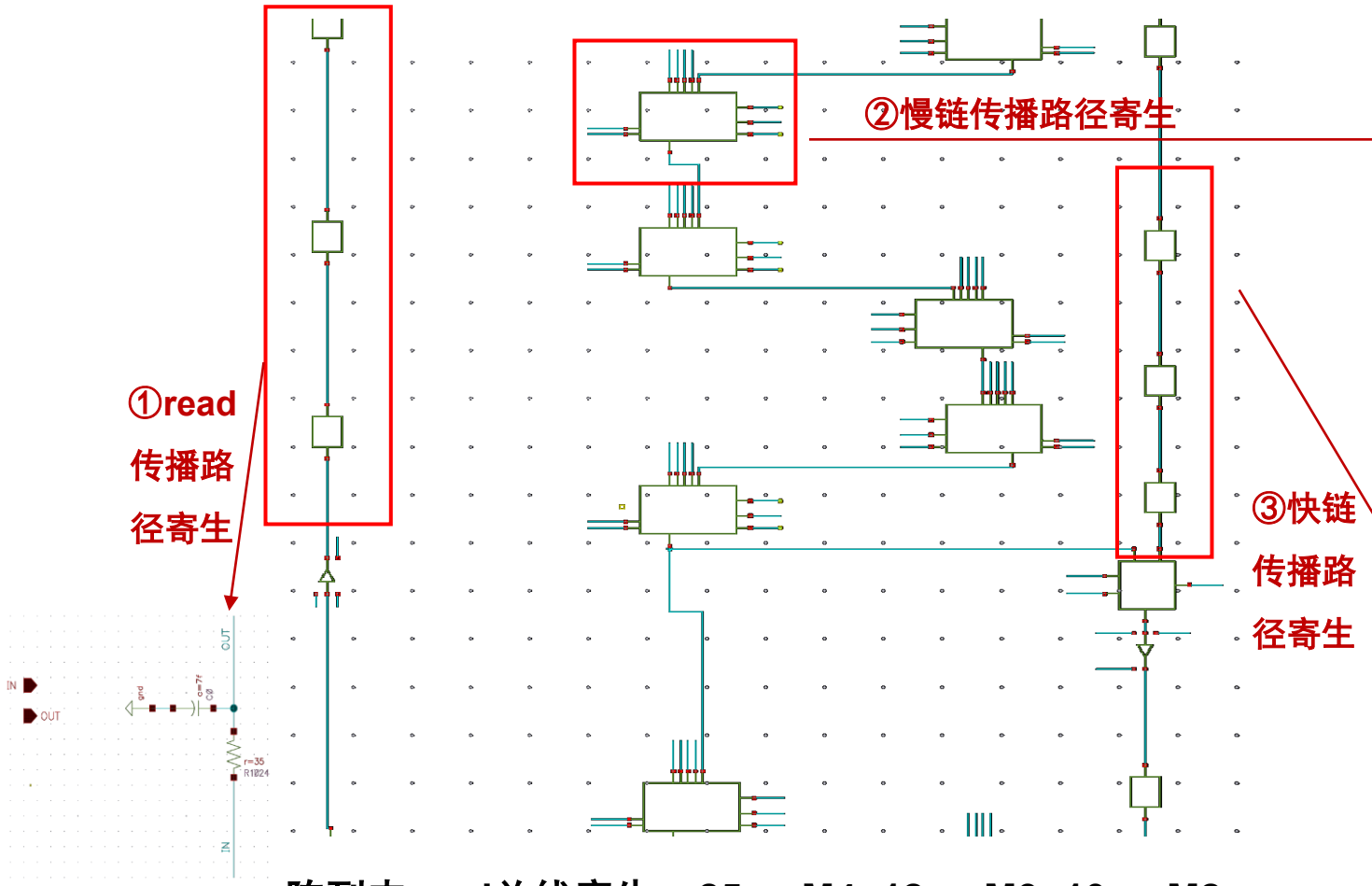
原设计基本情况及仿真模型搭建

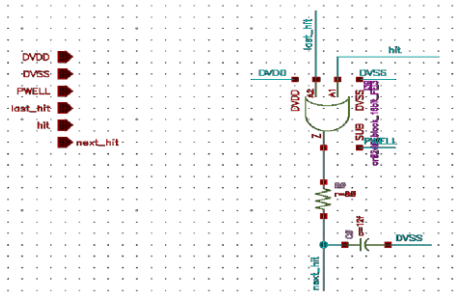


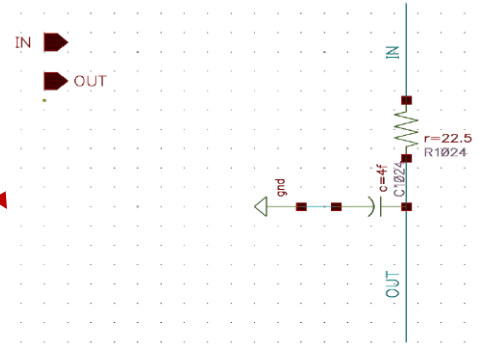
Dbcol数据流　　阵列内逻辑示意图　　仿真像素单元示意图

- 考虑到实际需求和仿真效率，选取像素中关键数字部分来代替完整像素功能
- 外围模块以列端行为级模型代替
- 考虑连线寄生

主要延时路径及寄生参数提取





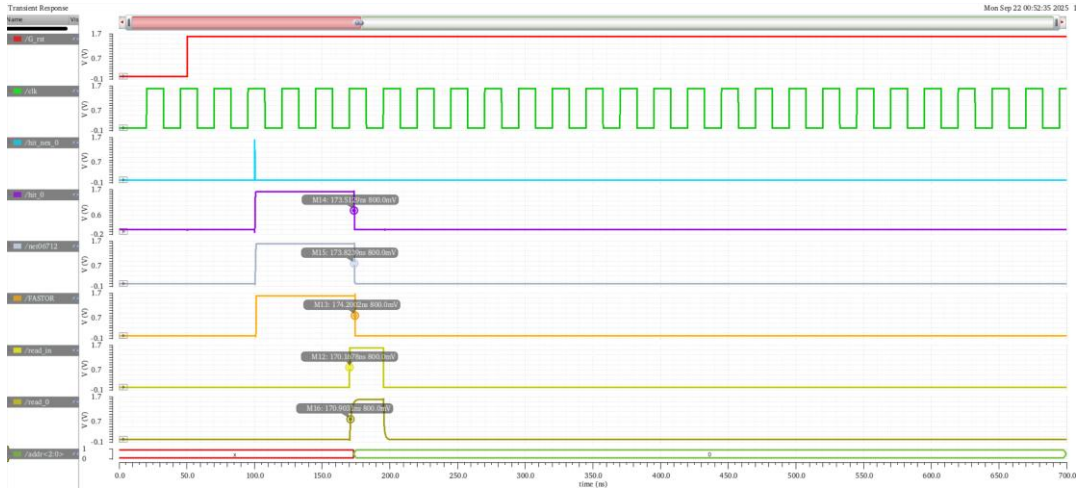


- 阵列内read总线寄生：25umM4+12umM3+10umM2+对应过孔（线宽0.28um）
- 直接提取结果：r=35ohm, c=7fF

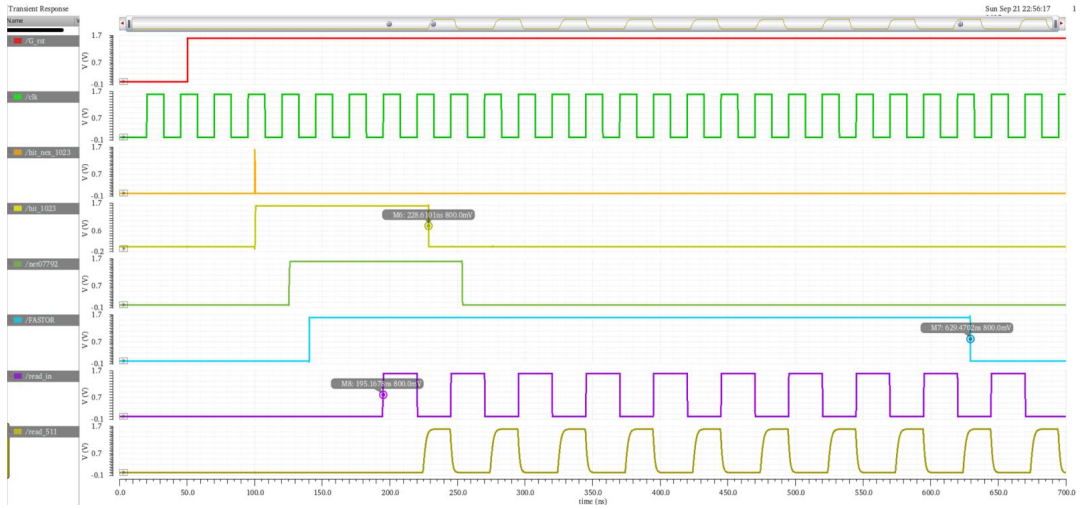
- 慢链或门间走线寄生：50umM3/M2+对应过孔（线宽0.28um）
- 直接提取结果：r=40ohm, c=6fF

- 快链或门间走线寄生：25umM4+3umM3+对应过孔（线宽0.28um）
- 直接提取结果：r=22.5ohm, c=4fF

原设计仿真

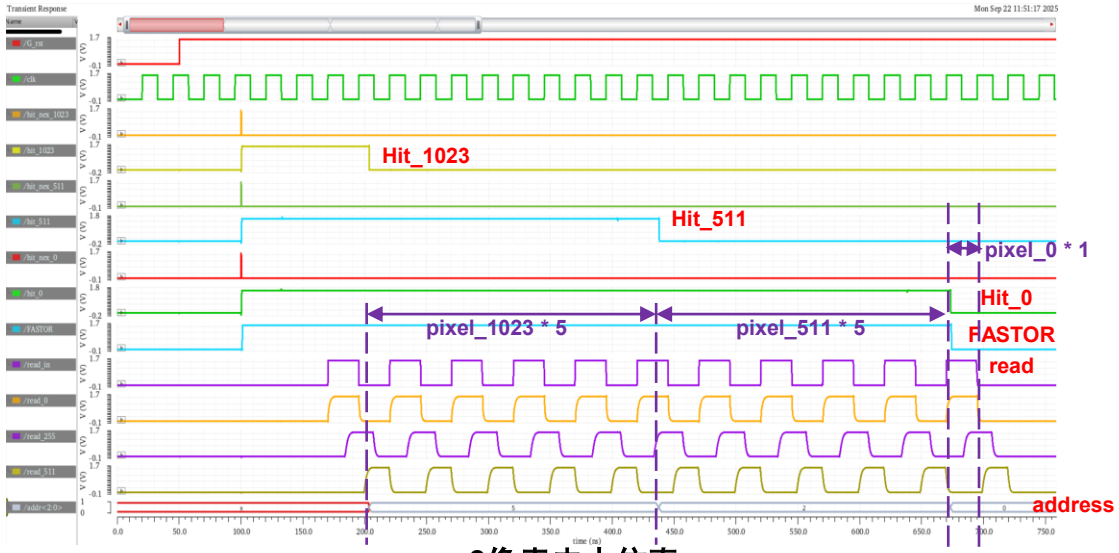


最近端1像素击中仿真



最远端1像素击中仿真

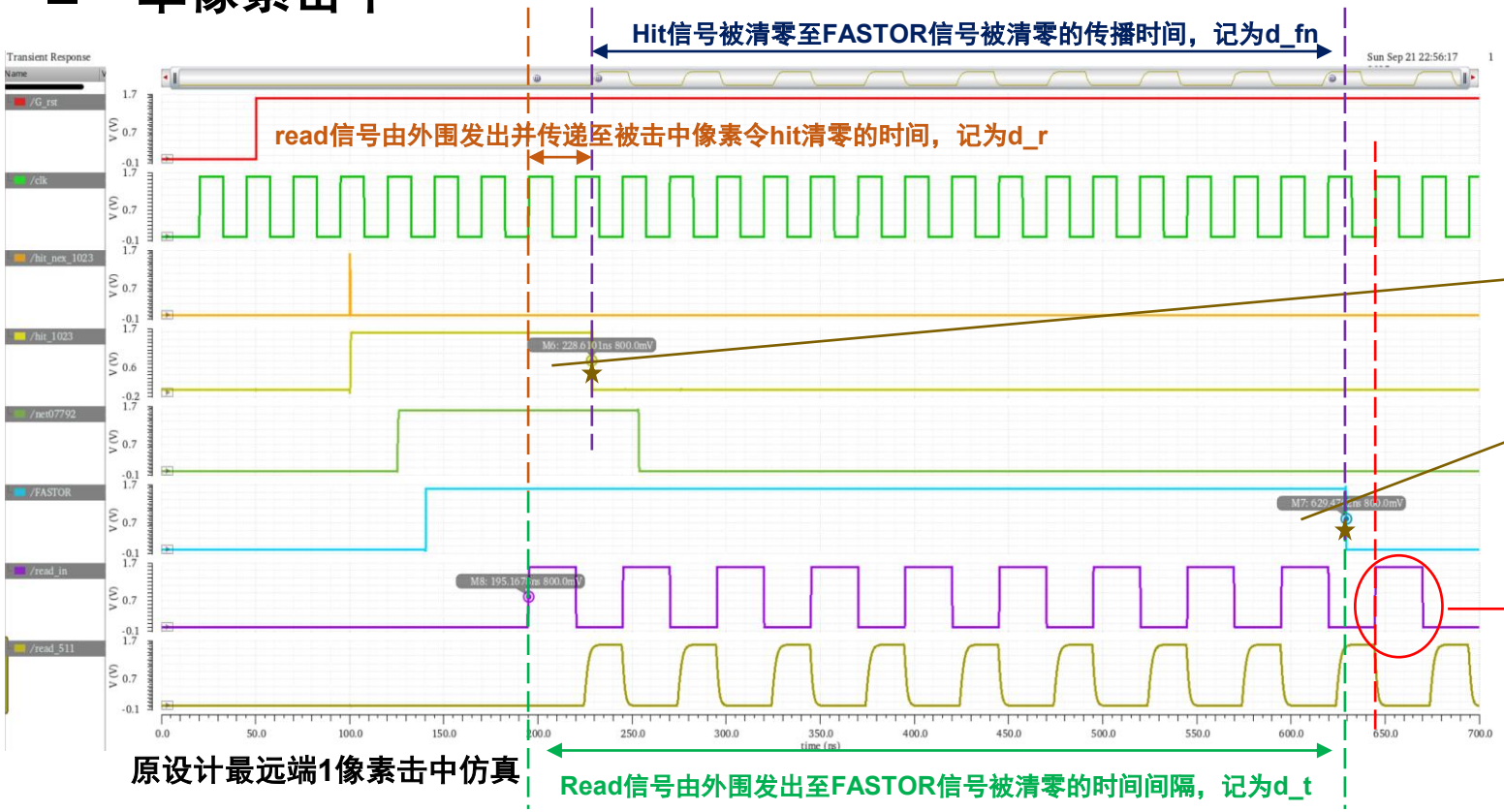
- SS + 2*RC条件下仿真（RC为所提取的寄生参数）
- 较近处像素被击中时无重复读出
  - 第1个像素（最近端）被单独击中时无重复读出
- 较远处像素被击中时重复读出
  - 第1024个像素（最远端）被单独击中时产生10个read信号，即9次重复读出
  - 第1024、512、1个像素被同时击中时，第1024个像素4次重复读出，第512个像素4次重复读出，第1个像素无重复读出
  - 重复原因：FASTOR清除信号无法在下一个read到来之前传递至下一个被击中像素



3像素击中仿真

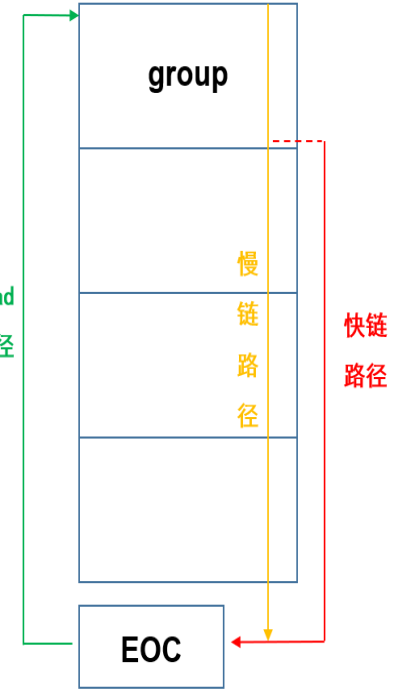
延时分析

■ 单像素击中



原设计最远端1像素击中仿真

- 被击中像素的hit被清除
- 清除信号传至末尾令“FASTOR”清零
- 由FASTOR_syn导致的1个重复read，在d_t > 25ns时产生（行为级模型中）



■ 总延时d_t = d_r + d_fn = 33.4 + 400.9 = 434.3 ns

■ 延时过大根本原因：FASTOR信号被清除需要遍历完整慢链，此段时间过长，即d_fn

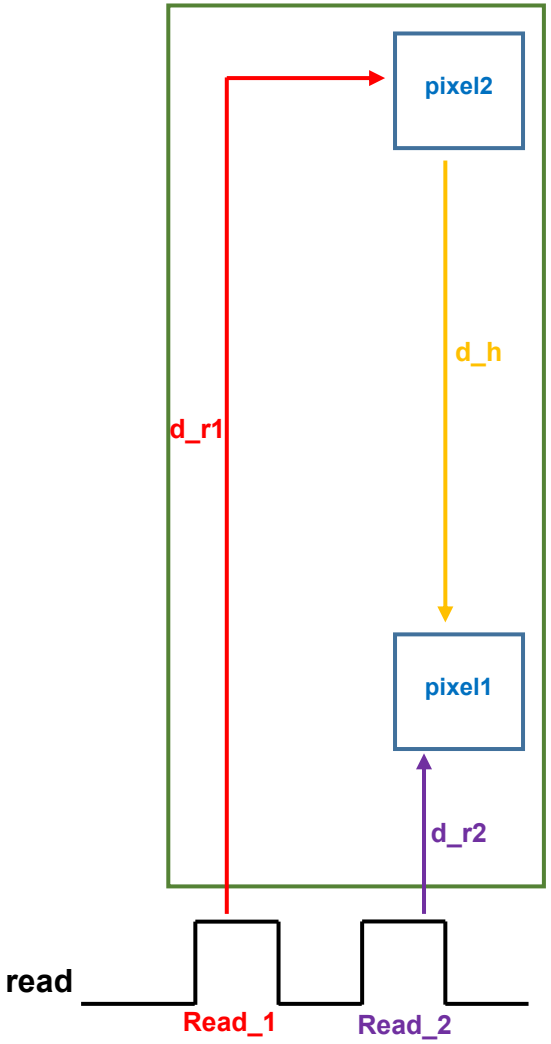
- 根本性减小延时：减小d_fn → 阵列中修改

■ 适当修改外围逻辑，清除最后一个重复read（行为级层面）

■ 不重复约束条件： 0 < d_r + d_fn < 50 ns

延时分析

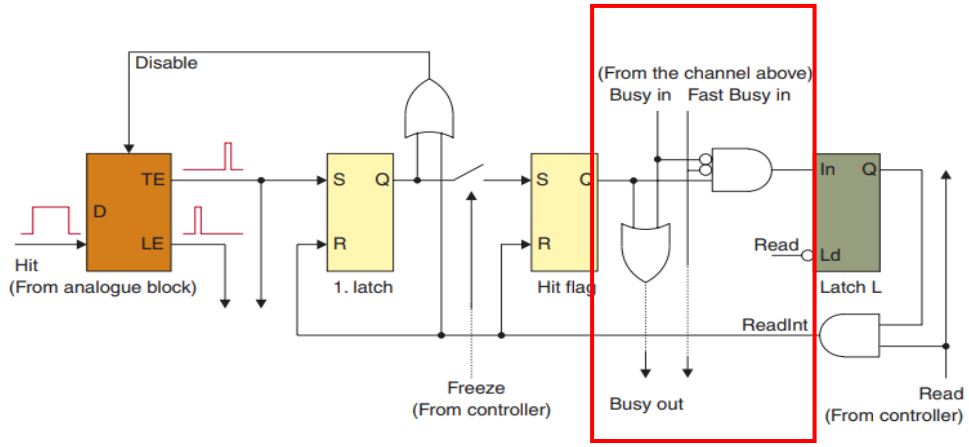
■ 多像素击中



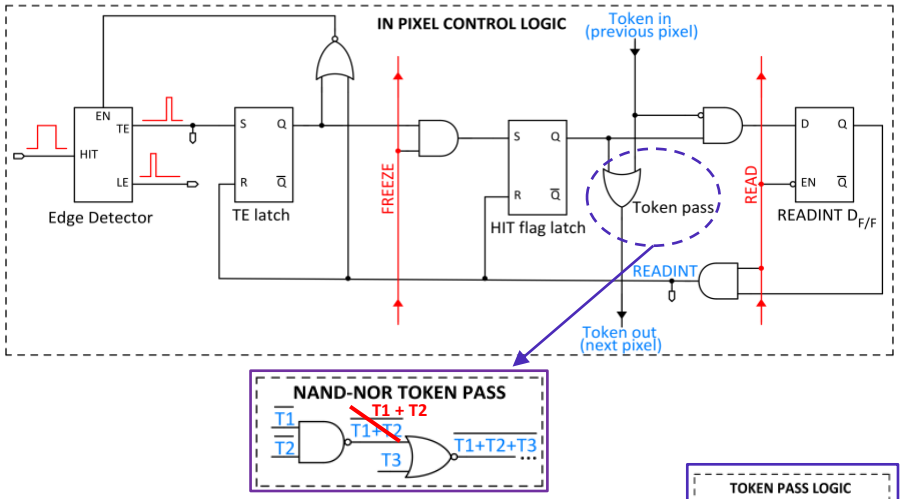
- Pixel1不重复条件：
  - 直观理解：hit清零信号需要在第2个read信号到达pixel1之前到达pixel1
  - 数学表达：d_r1 + d_h – d_r2 < 50 ns
- Pixel2不重复条件与该像素被单独击中时情况相同（参考上一页）
- 综合不同情况分析，延时最坏的情况为最远端像素被单次击中，满足上页不重复约束条件即可
- 优化：视初步修改后的结果而定

注*：d_r1为第1个read信号由外围发出传递至pixel2，并令其hit清零的时间
d_r2为第2个read信号由外围发出传递至pixel1，并令其hit清零的时间
d_h为hit清零信号由pixel2传递至pixel1的时间

相关调研



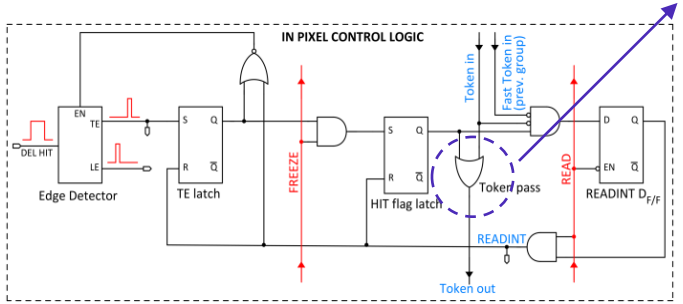
FEI3像素内数字逻辑原理图

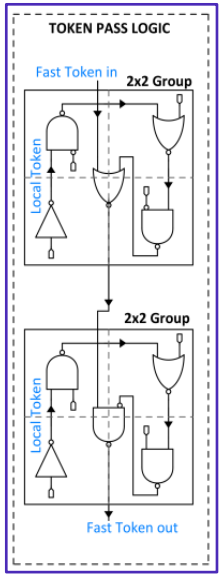


TJ-Monopix1像素内数字逻辑原理图

- FEI3：将一列分块，引入fast busy进行块级的信号传递，Taichu采用了同样的策略，只能加快busy（fastor）传“1”的速度；
- TJ-Monopix1：将“或门”替换为“与非门-或非门”交替出现；
- TJ-Monopix2：2×2像素组，引入快速令牌逻辑。

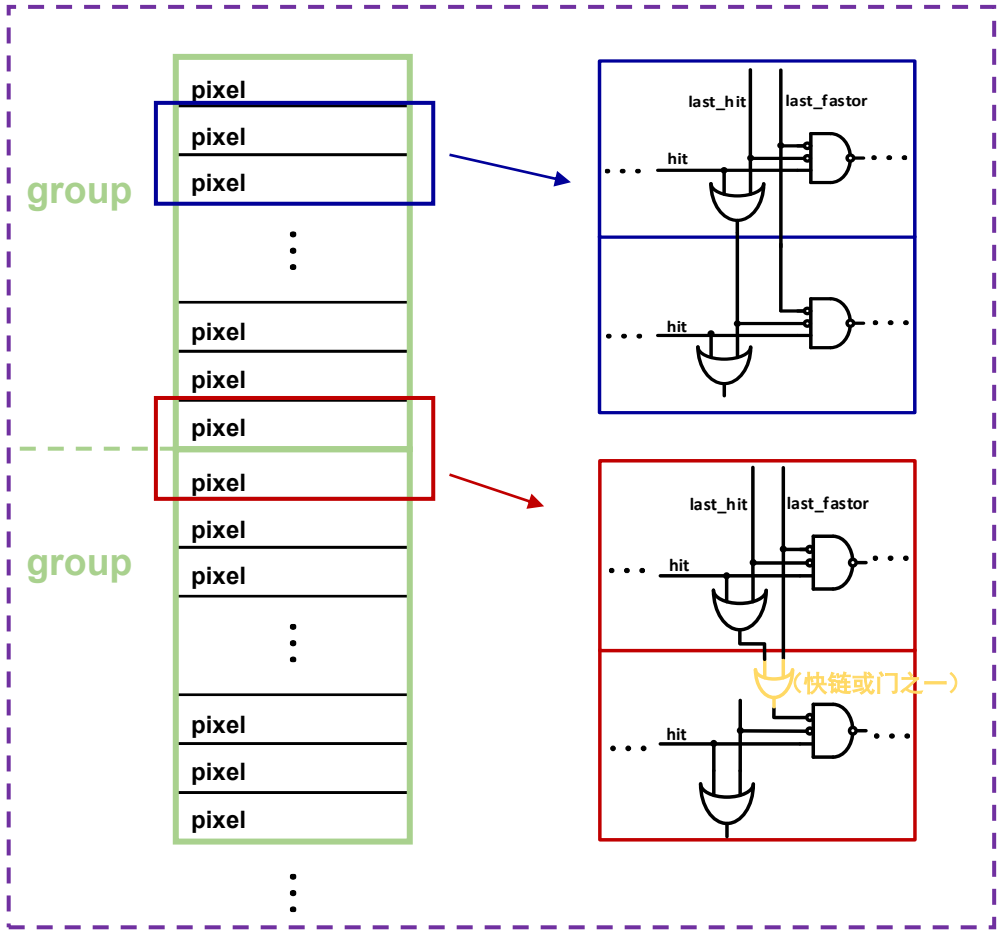
对后续修改和优化具有一定参考意义



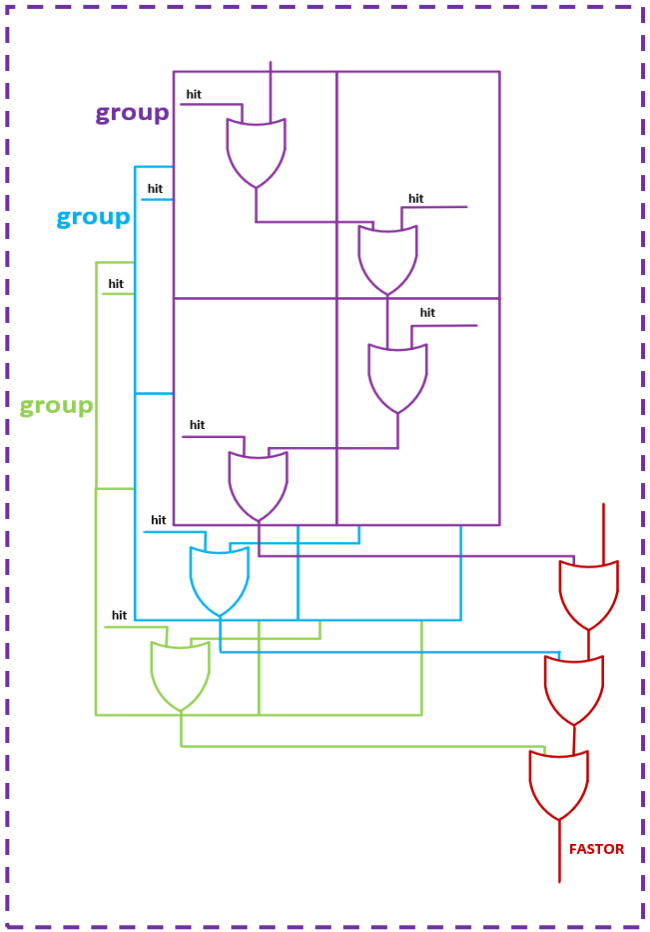


TJ-Monopix2像素内数字逻辑原理图

初步修改



阵列内逻辑修改示意图

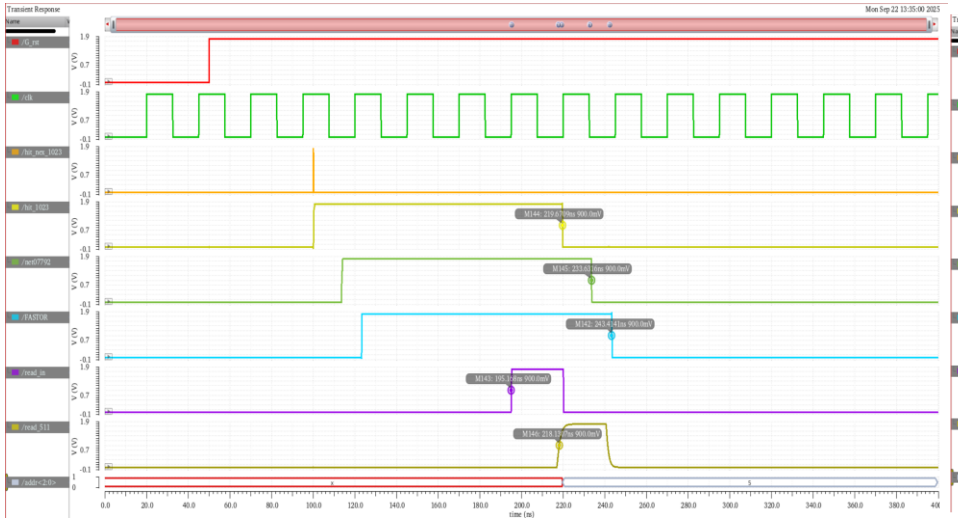


修改后逻辑示意图

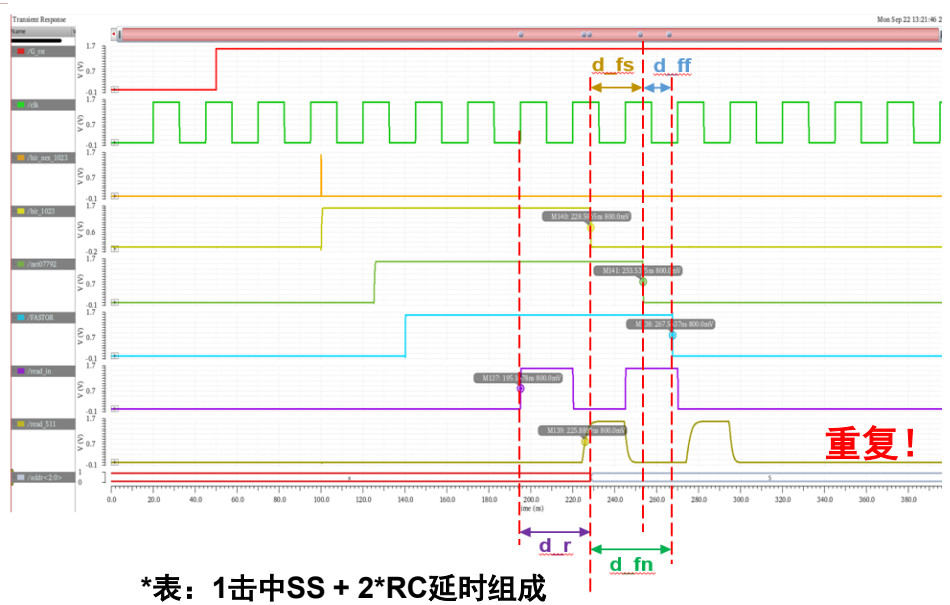
- 阵列内逻辑修改（根本）
  - 慢链（像素内）的或门在分组处断开但优先级仲裁处保持连接
  - 加快FASTOR信号的清除但不影响优先级仲裁
- 外围逻辑修改（行为级层面/辅助）
  - 修改read翻转条件
  - 原设计：FASTOR_syn为高的时钟上升沿翻转
  - 修改后：FASTOR_syn & FASTOR

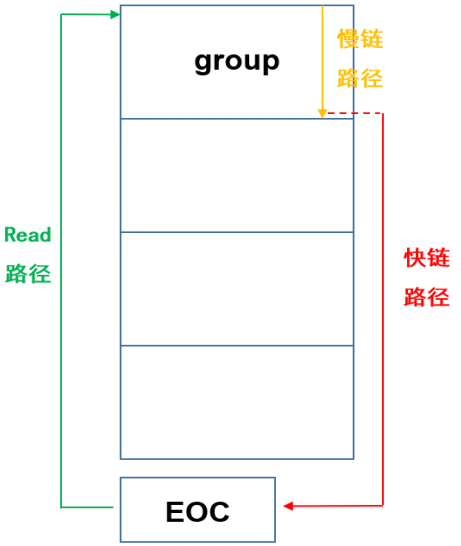
初步修改后仿真与分析

- 1击中TT（TT + 1.8V + 27℃）+ 2*RC



- 1击中SS（SS + 1.6V + 50℃） + 2*RC





注*：d_t = d_r + d_fn，为read信号由外围发出至FASTOR信号清零的时间

d_r为read信号由外围发出并传递至被击中像素令hit清零的时间

d_fn = d_fs + d_ff，为hit信号被清零至FASTOR信号被清零的传播时间

d_fs为hit信号被清零至慢链被全部清零的传播时间

d_ff为慢链被全部清零至快链被全部清零的传播时间

*表：1击中SS + 2*RC延时组成

d_t			72.4 ns		
d_r	d_fn		33.4	39	
	d_fs	d_ff		25	14

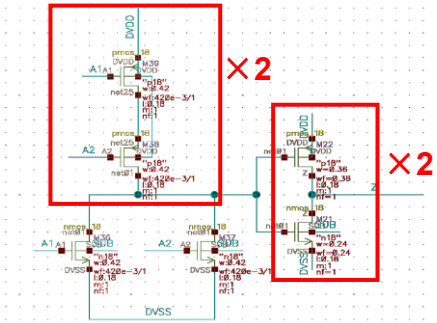
- TT仿真无重复读出，SS仿真有1次重复读出
- 需针对主要延时路径进一步优化，即read路径、慢链和快链路径

延时优化

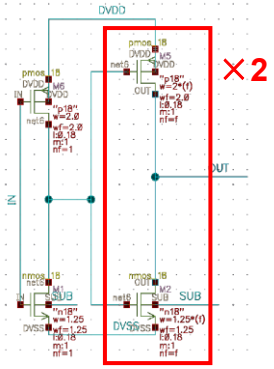
➢ 针对慢链和快链的优化：

■ 方案一：保留“纯或”逻辑，优化关键逻辑门尺寸

- 修改慢链和快链中的或门尺寸
- 修改快链路径中的buffer尺寸



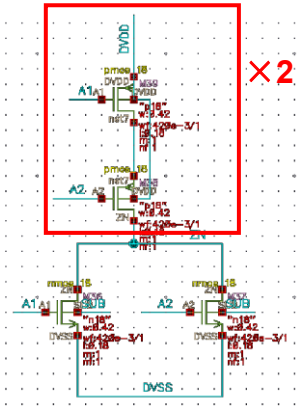
或门尺寸优化



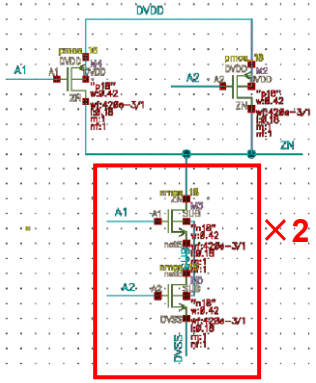
buffer尺寸优化

■ 方案二：NAND-NOR逻辑交替

- 将OR替换为NAND-NOR交替出现，并修改关键NAND/NOR尺寸
- 逻辑复杂，面积节省空间小，延时优化情况与方案一相当（见本报告P13、P14）



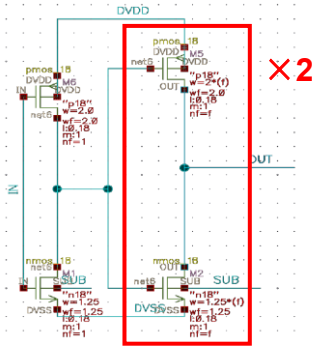
或非门尺寸优化



与非门尺寸优化

➢ 针对read路径的优化：

- 修改read路径中的buffer尺寸



buffer尺寸优化

版图代价

	慢链/快链路径						Read路径	
	像素内增加（修改OR）/像素			像素外增加（修改OR & NAND & NOR & Buffer）/组				
纯或逻辑		2*PMOS	$\left(\frac{W}{L}\right)_P=\frac{0.42u}{0.18u}$		2*PMOS	$\left(\frac{W}{L}\right)_P=\frac{0.42u}{0.18u}$	2*INV	$\left(\frac{W}{L}\right)_N=\frac{1.25u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{2.0u}{0.18u}$
		1*INV	$\left(\frac{W}{L}\right)_N=\frac{0.24u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{0.36u}{0.18u}$		1*INV	$\left(\frac{W}{L}\right)_N=\frac{0.24u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{0.36u}{0.18u}$		
					2*INV	$\left(\frac{W}{L}\right)_N=\frac{1.25u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{2.0u}{0.18u}$		
NAND-NOR交替	像素内NAND逻辑	2*NMOS	$\left(\frac{W}{L}\right)_N=\frac{0.42u}{0.18u}$	NAND逻辑组	2*NMOS	$\left(\frac{W}{L}\right)_N=\frac{0.42u}{0.18u}$	2*INV	$\left(\frac{W}{L}\right)_N=\frac{1.25u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{2.0u}{0.18u}$
		1*INV	$\left(\frac{W}{L}\right)_N=\frac{0.24u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{0.36u}{0.18u}$		（注：快链中第一个NAND需额外增加1*INV，$\left(\frac{W}{L}\right)_N=\frac{0.24u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{0.36u}{0.18u}$）			
		（注：慢链中每组第一个NAND在此基础上需额外再增加1*INV，尺寸同上）			2*INV	$\left(\frac{W}{L}\right)_N=\frac{1.25u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{2.0u}{0.18u}$		
	像素内NOR逻辑	2*PMOS	$\left(\frac{W}{L}\right)_P=\frac{0.42u}{0.18u}$	NOR逻辑组	2*PMOS	$\left(\frac{W}{L}\right)_P=\frac{0.42u}{0.18u}$		
					1*INV	$\left(\frac{W}{L}\right)_N=\frac{0.24u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{0.36u}{0.18u}$		
					2*INV	$\left(\frac{W}{L}\right)_N=\frac{1.25u}{0.18u}$, $\left(\frac{W}{L}\right)_P=\frac{2.0u}{0.18u}$		

注*：对于像素外增加的部分，尺寸为$\left(\frac{W}{L}\right)_N=\frac{0.24u}{0.18u}$，$\left(\frac{W}{L}\right)_P=\frac{0.36u}{0.18u}$的INV为逻辑门中所增加；尺寸为$\left(\frac{W}{L}\right)_N=\frac{1.25u}{0.18u}$，$\left(\frac{W}{L}\right)_P=\frac{2.0u}{0.18u}$的INV为快链中Buffer所增加

- 两种方案在**面积消耗方面相当**
- 版图评估：无法在原设计布局基础上直接修改，**需重新布局和绘制**。

不同情况延时结果1

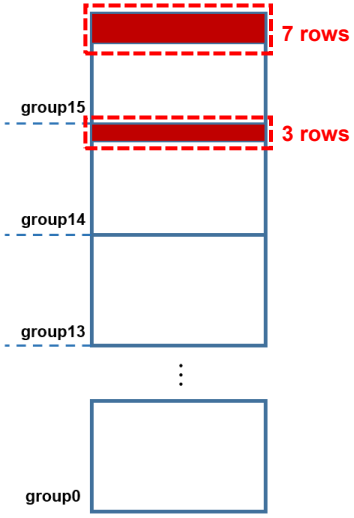
			最远端像素1击中											
			2*RC				1.5*RC				1*RC			
			TT			SS	TT			SS	TT		SS	
d_t		原设计	249.0 ns			434.3 ns					191.2 ns		339.8 ns	
			24.6	224.4		33.4	400.9				15.1	176.1	22.0	317.8
		初步修改	48.3 ns			72.4 ns								
			24.5	14.0	9.8	33.4	25.0	14.0						
d_r	d_fn	纯或优化	37.2 ns			53.4 ns			30.0 ns		44.9 ns		23.9 ns	37.3 ns
			19.8	9.6	7.8	25.7	16.7	11.0	15.0	8.8	6.2	20.2	15.4	9.3	10.9	8.0	5.0	15.5	13.9	7.9
	d_fs d_ff	Nand-nor	36.8 ns			53.7 ns								
			19.8	9.4	7.6	25.7	17.3	10.7						

未完全满足要求，具体分析见下页

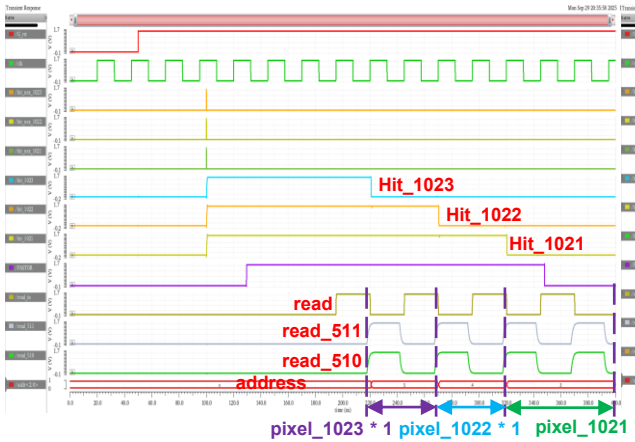
- 在优化方面，针对“纯或逻辑”和“NAND-NOR逻辑”两种方案，延时优化效果相当
- 结合两者面积消耗情况相当，且考虑到NAND-NOR方案逻辑更为复杂，选择纯或逻辑方案

纯或优化SS + 2*RC情况分析

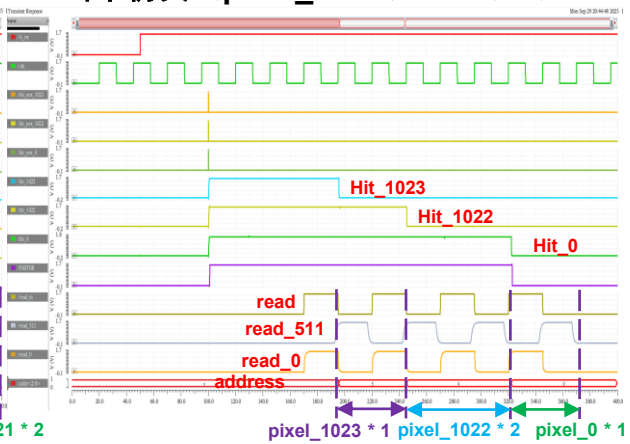
	2*RC					
	TT			SS		
纯或优化	37.2 ns			53.4 ns		
	19.8	9.6	7.8	25.7	16.7	11.0



• 3击中仿真（pixel_1023、1022、1021）



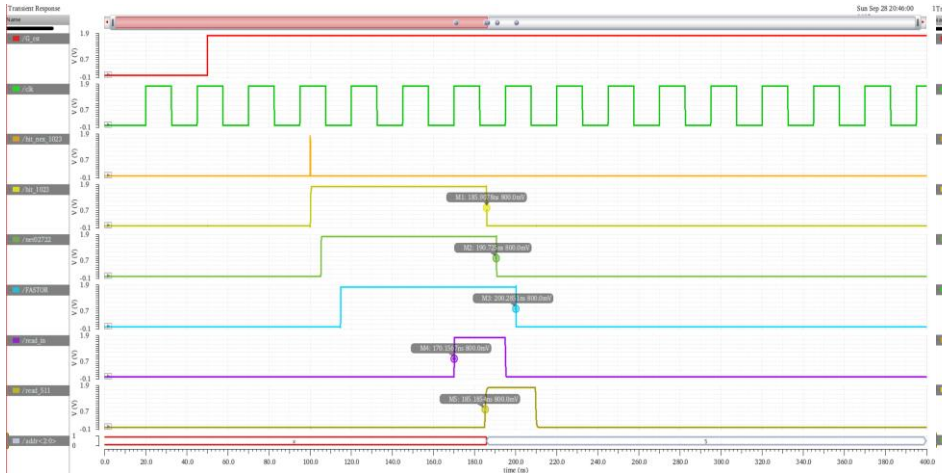
• 3击中仿真（pixel_1023、1022、0）



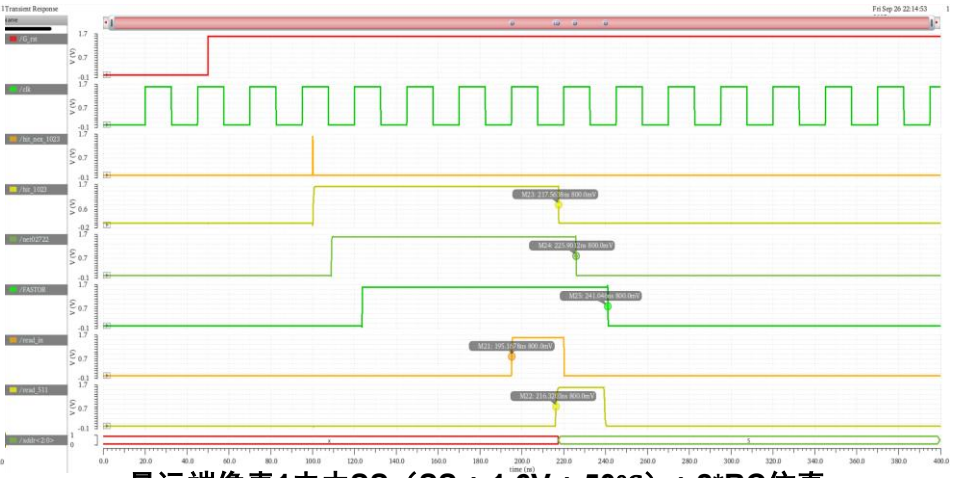
- 可能重复的像素位置：第16组（最远端的组）中远端7行像素和第15组中远端3行像素，即20个像素所在位置（针对一个Dbcol）
- 重复情况：
  - 单像素击中：以上20个位置的任意像素被单独击中时，会重复读出1次，其余1004个位置的像素不重复
  - 多像素击中：以3个像素被击中为例
    - 若3个连续像素被击中，则当3个像素中最近端的像素处在以上20个位置之中时，最近端像素重复读出1次，其余2个像素不重复。即3像素被击中，最终读出4次。
    - 若3个不连续像素被击中，则像素间隔较远时，上述20个位置中的像素有几率重复读出1次，其余像素不重复。即3像素被击中，最终读出4次。
- 综上，该情况下无论是单像素被击中还是多像素被击中，均只有1个像素有几率重复读出1次，占比小，且对死时间和数据带宽的影响不大。可以选择在FPGA中处理。

进一步优化选择

- 彻底优化方案：
  - 将Dbcol分为32组，每组16行/32像素
  - Read路径相应分为32节
  - 版图代价：相较于修正后的16组方案，慢链部分无额外开销，即像素内无需额外增加晶体管；快链部分需额外增加16个尺寸修改后的或门和buffer；read路径需额外增加16个尺寸修改后的buffer。



最远端像素1击中TT（TT + 1.8V + 27℃）+ 2*RC仿真



最远端像素1击中SS（SS + 1.6V + 50℃）+ 2*RC仿真

	TT			SS		
32组方案	30.1 ns			45.9 ns		
	15.8	4.8	9.5	22.4	8.3	15.2

✓ 该情况下彻底满足不重复要求，在条件允许范围内（如版图面积等）可优先选择该方案

✓ 原设计FASTOR重复读出bug解决

不同情况延时结果2

			最远端像素1击中																	
			2*RC						1.5*RC						1*RC					
			TT			SS			TT			SS			TT			SS		
d_t		原设计	249.0 ns			434.3 ns									191.2 ns			339.8 ns		
			24.6	224.4		33.4	400.9								15.1	176.1		22.0	317.8	
		初步修改	48.3 ns			72.4 ns														
			24.5	14.0	9.8	33.4	25.0	14.0												
d_r	d_fn	纯或优化	37.2 ns			53.4 ns			30.0 ns			44.9 ns			23.9 ns			37.3 ns		
			19.8	9.6	7.8	25.7	16.7	11.0	15.0	8.8	6.2	20.2	15.4	9.3	10.9	8.0	5.0	15.5	13.9	7.9
	d_fs d_ff	32组方案	30.1 ns			45.9 ns														
			15.8	4.8	9.5	22.4	8.3	15.2												

Stitching设计参考

	芯片尺寸 [mm2]	RSU Height [um]	RSU Width [um]	Matrix 行数 R	Matrix 列数 C	Matrix 个数 N	RSU 个数	读出方式	端口数量	读出速率 [Mbps]	Buffer功耗密度
1	24.600 x 158.400	24600	12750	440	64	14	12	独立	168	259.98	73.1 mW/cm^2
								汇总	48	1039.92	62.7 mW/cm^2

- Matrix大小：440rows × 64cols；
- 阵列布局时可采用与Taichu（修正后）相似的方案
  - 若采取Taichu-16组类似方案，则stitching-tile中一个双列可分为14组，其中有1组为24行/48像素（其余13组为32行/64像素）；
  - 若采取Taichu-32组类似方案，则stitching-tile中一个双列可分为28组，其中有1组为8行/16像素（其余27组为16行/32像素）；
- 考虑延时最小化，较短的一组放置在最远端（针对以上提到的两种方案分别记为第14组或第28组），则14组方案中延时最坏的像素为14组最远端像素或13组最远端像素，28组方案中延时最坏的像素为28组最远端像素或27组最远端像素。

均满足不重复要求！

		14组方案				28组方案			
		14组最远端1像素击中		13组最远端1像素击中		28组最远端1像素击中		27组最远端1像素击中	
d_t		44.9 ns		46.9 ns		37.2 ns		40.4 ns	
d_r	d_fn	22.7	22.2	21.3	25.6	19.7	17.5	19.1	21.3

注*：4组结果均在SS + 2*RC情况下仿真所得

总结

➢ Taichu3设计

- Bug复现：较坏情况下最远端1个像素被击中时有9次重复读出
- 问题分析：阵列内FASTOR清零延时过大，需满足$0 < d_r + d_fn < 50$ ns即可达到不重复要求
- 修改与优化：保留纯或逻辑的基础上，16组方案基本可以满足需求，32组方案可彻底解决问题，资源允许的情况下可优先选择
  - 16组方案：434.3 ns → 53.4 ns
  - 32组方案：434.4 ns → 45.9 ns

➢ Stitching设计参考

- 类似Taichu3修正后方案，可选择14组方案或28组方案，视条件选择
  - 14组方案：46.9 ns
  - 28组方案：40.4 ns

Thanks!