DV仿真结果分析方法

由于perf model现在有些行为描述与RTL不符，因此性能分析以DV仿真结果为准。当然也存在RTL实现与预期设计不符的情况，可以参考 Tcore perf tuning guide 和 TFE MAS V0.9。

环境准备及仿真

ETX权限申请请参照： IT：1-用户首次登陆ETX

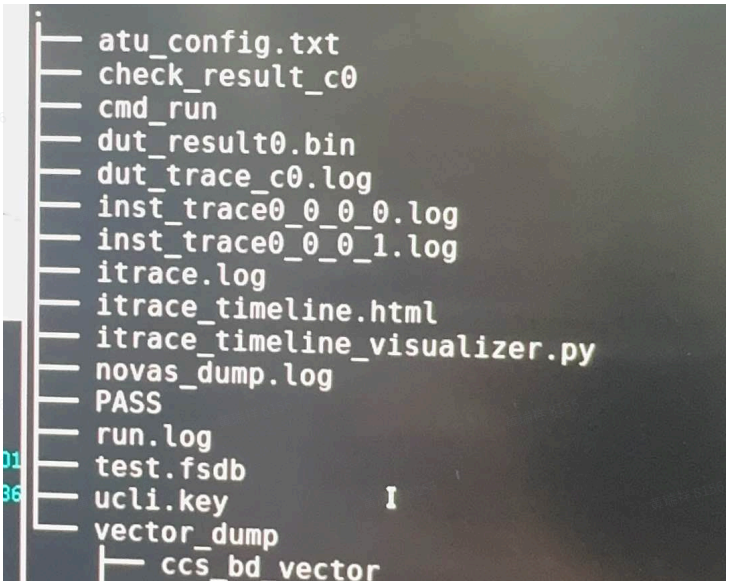
算子运行方法参照： PE DV 算子运行及分析方法

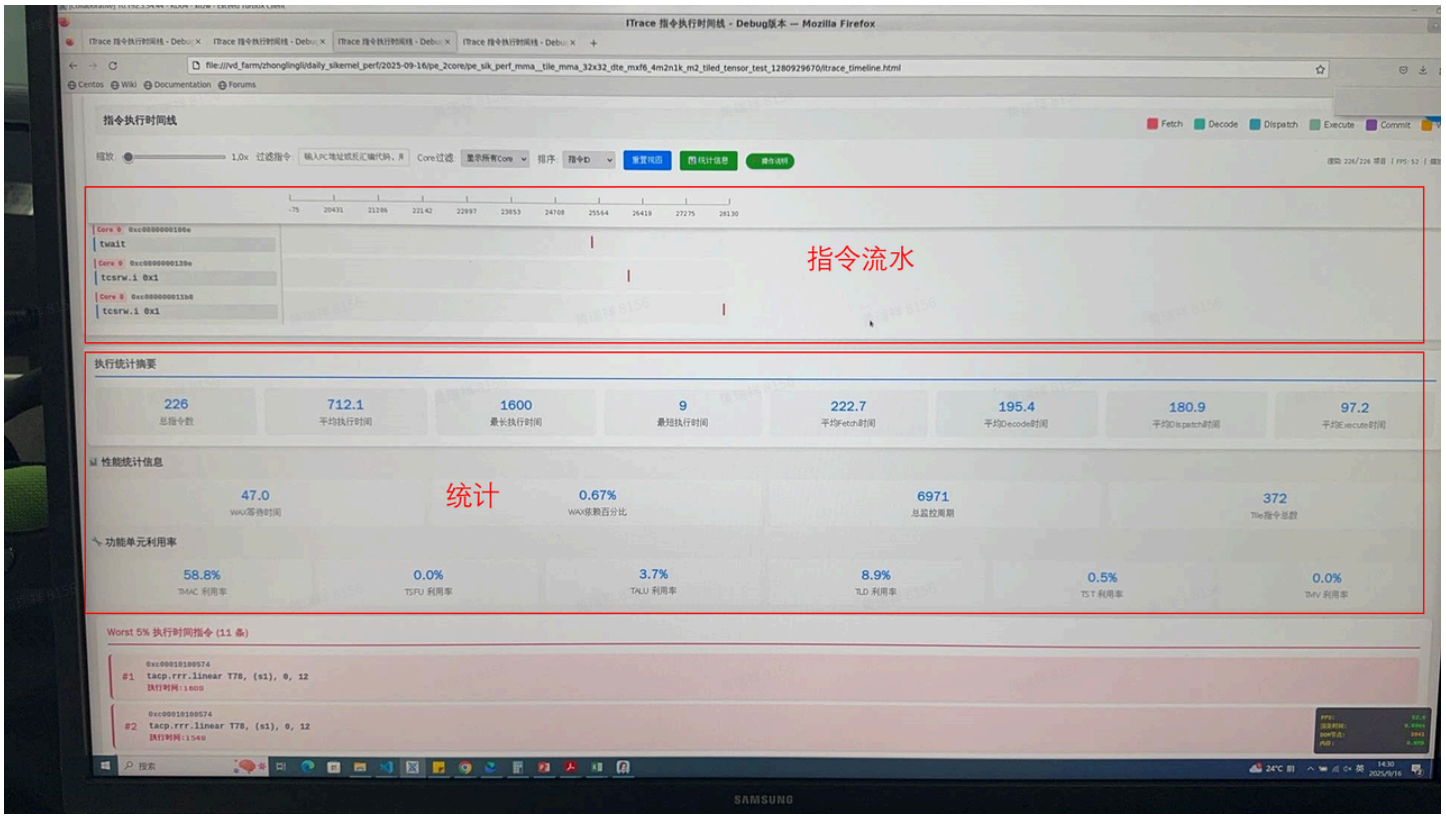
Itrace工具

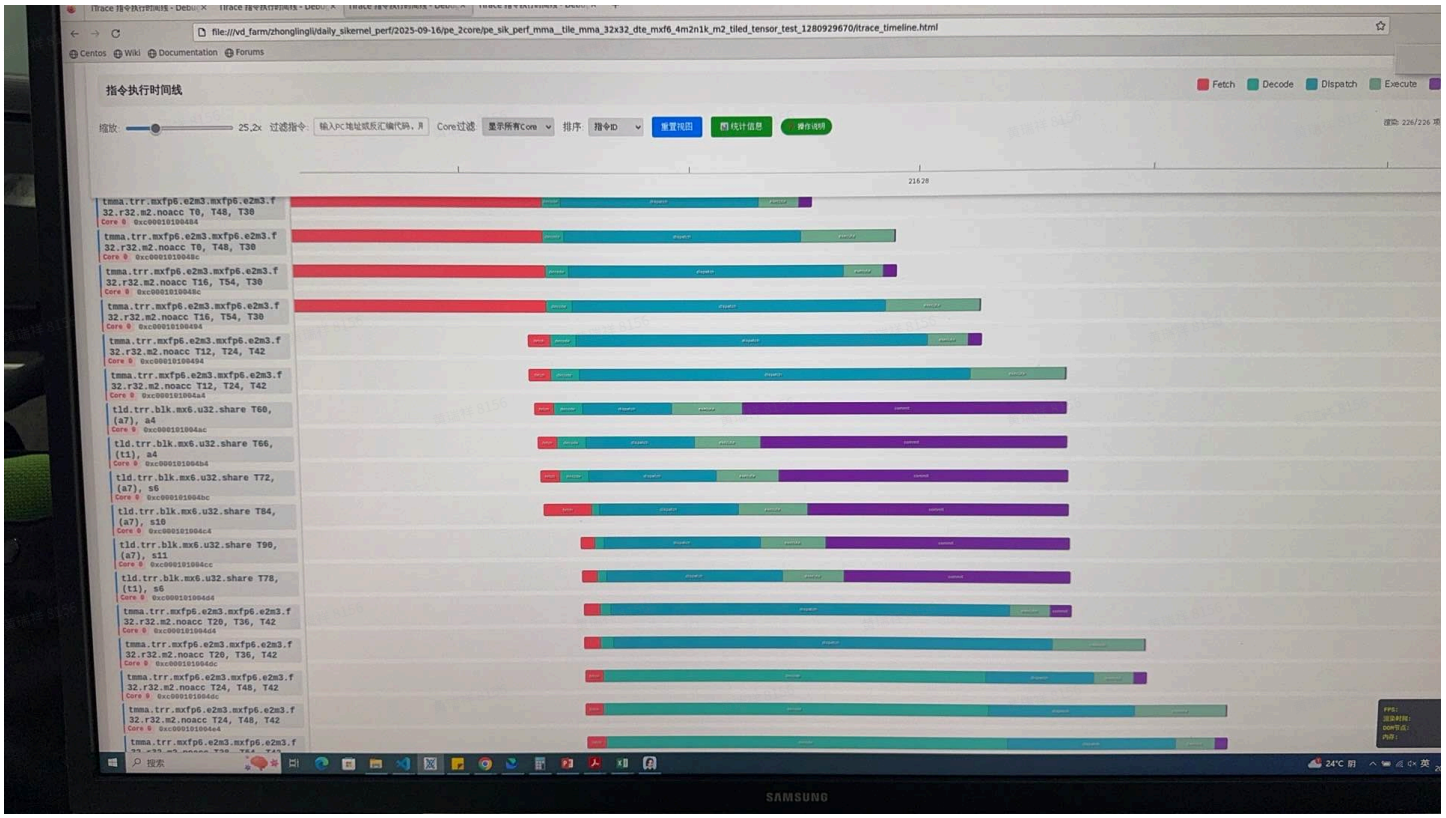
ITrace

itrace 是一个为Sipu设计的高级性能监控模块。它的核心目标是精确追踪每一条指令从“请求取指”到“执行完毕”的完整生命周期，从而为性能分析、瓶颈定位和系统优化提供关键数据。

仿真完成后，工程目录下会生成一系列文件，用浏览器打开itrace_timeline.html即可查看Itrace结果。







优点：

1. 快速查看各指令的执行起止时间
2. 分析明显的数据依赖等问题导致的执行中断

缺点：

1. 界面显示的原因，分析依赖关系需要上下核对，容易混乱；

2. 短时间的执行中断发现及分析困难，例如相邻mma之间由于itrace显示的execute有大量重叠部分，不能真实反应TMAC实际执行阶段，即使itrace上显示execute一直都有重叠，但是TMAC可能并没有连续执行。

Verdi仿真波形

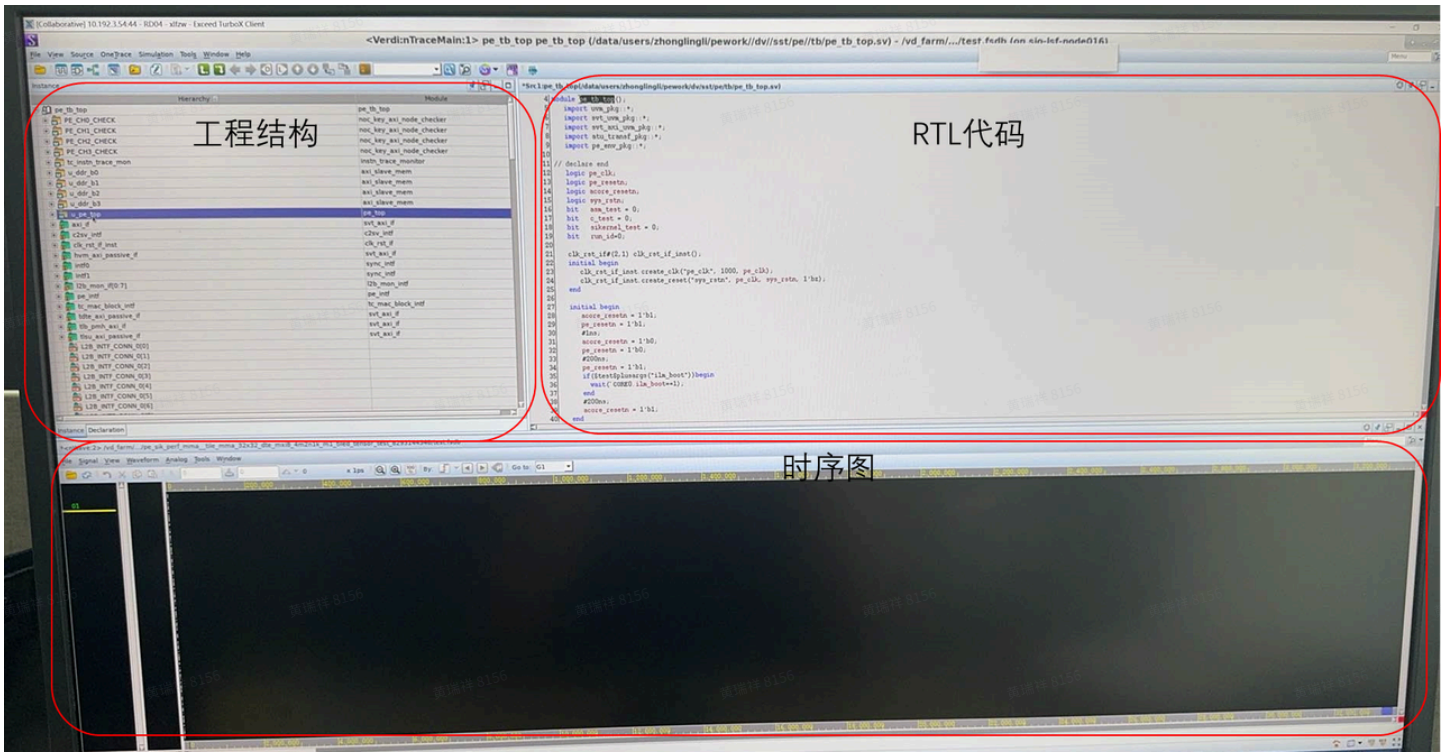
打开verdi

终端中进入到想要查看的测试案例路径下，执行以下指令，可以打开verdi debug平台。

代码块

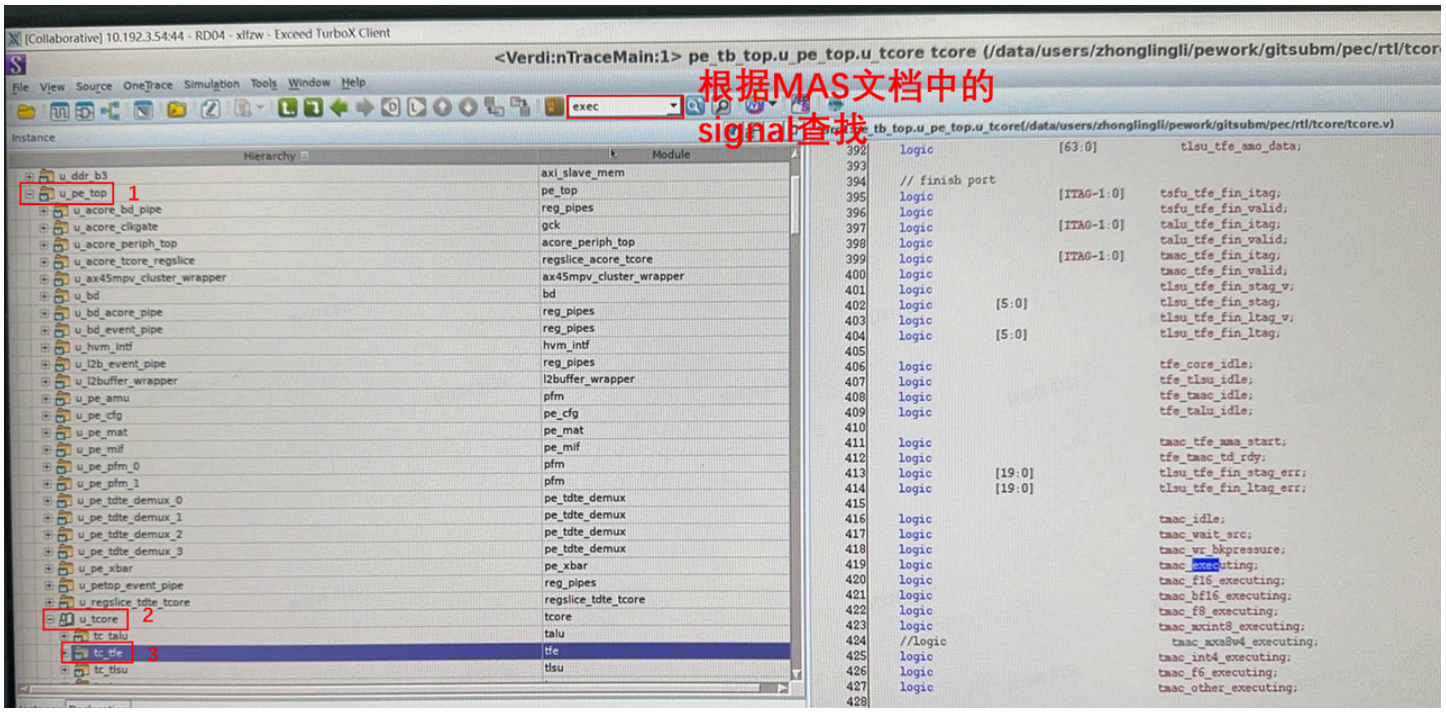
```
1   bsub -Is verdi -simflow -ssf test.fsdb &
```

左侧可以看到RTL层级结构与芯片的层级结构相同，双击选择文件能在右侧查看其RTL代码。我们主要关注u_pe_top内部的信号。



添加信号

打开想要查看的模块代码，可通过查找关键字的方式找到想要查看的信号，直接拖拽到波形窗口(快捷键ctrl+w)。与TFE模块相关的信号可以参照 TFE MAS V0.9 添加。可将常用的信号保存在rc文件中。



性能分析

下面以mxi8_mma_4m2n1k_m2说明性能分析及优化的过程。

计算总量为128M_64N_2048K, TMAC理论8192cycle。

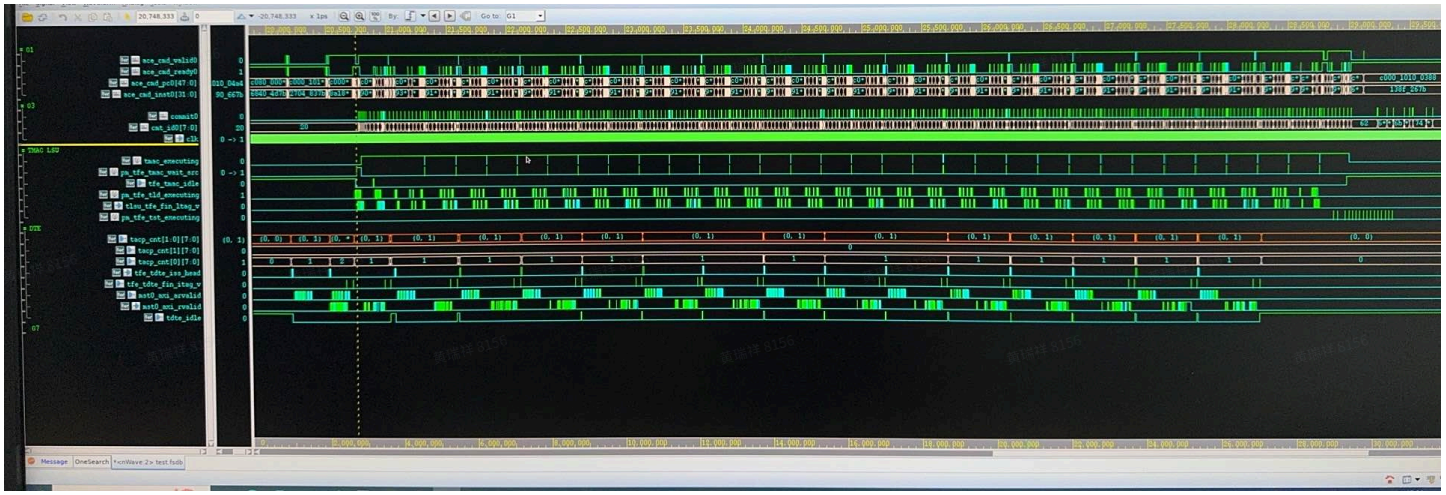
chunk size：128M_64N_128K，数据量26112B，理论dte搬运时间102cycle (256B/cycle)， TMAC理论512cycle。

信号列表

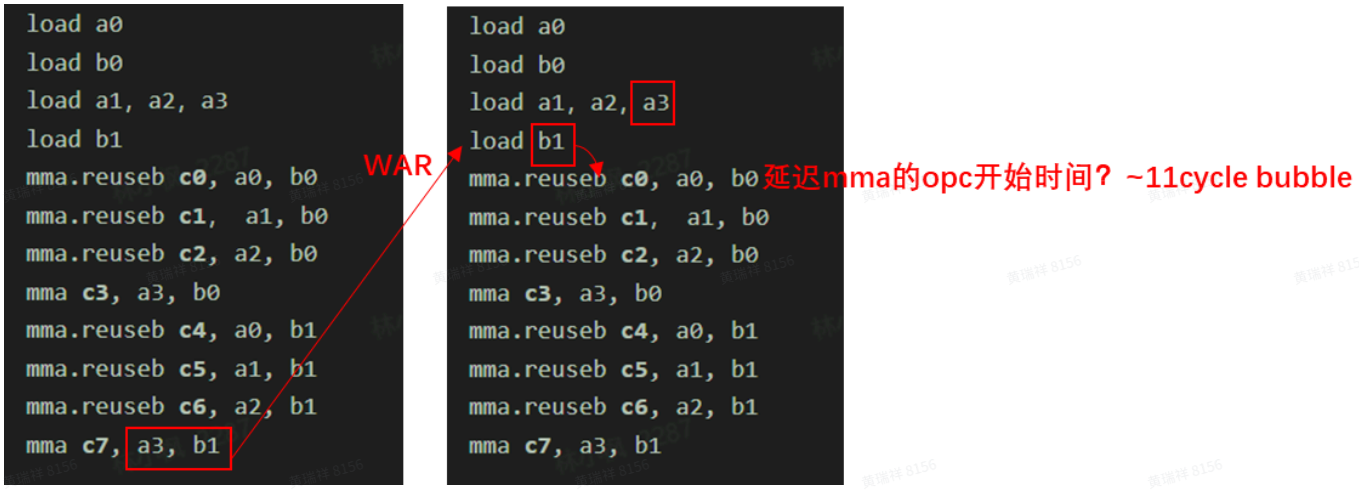
	模块-信号名	说明
	clk	时钟
TMAC	tmac_executing	TMAC在工作，目标：该信号尽量持续高电平
	pm_tfe_tmac_wait_src	TMAC在等待参与运算的数据就位
LSU	pm_tfe_tld_executing	lsu正在load
	tlsu_tfe_fin_ltag_v	lsu有load完成
	pm_tfe_tst_executing	lsu正在store
DTE	tfe_dte_iss_valid/head	发起dte搬运
	tfe_tdte_fin_itag_v	完成dte搬运
	mst0_axi_arvalid	dte内部axi发起搬运

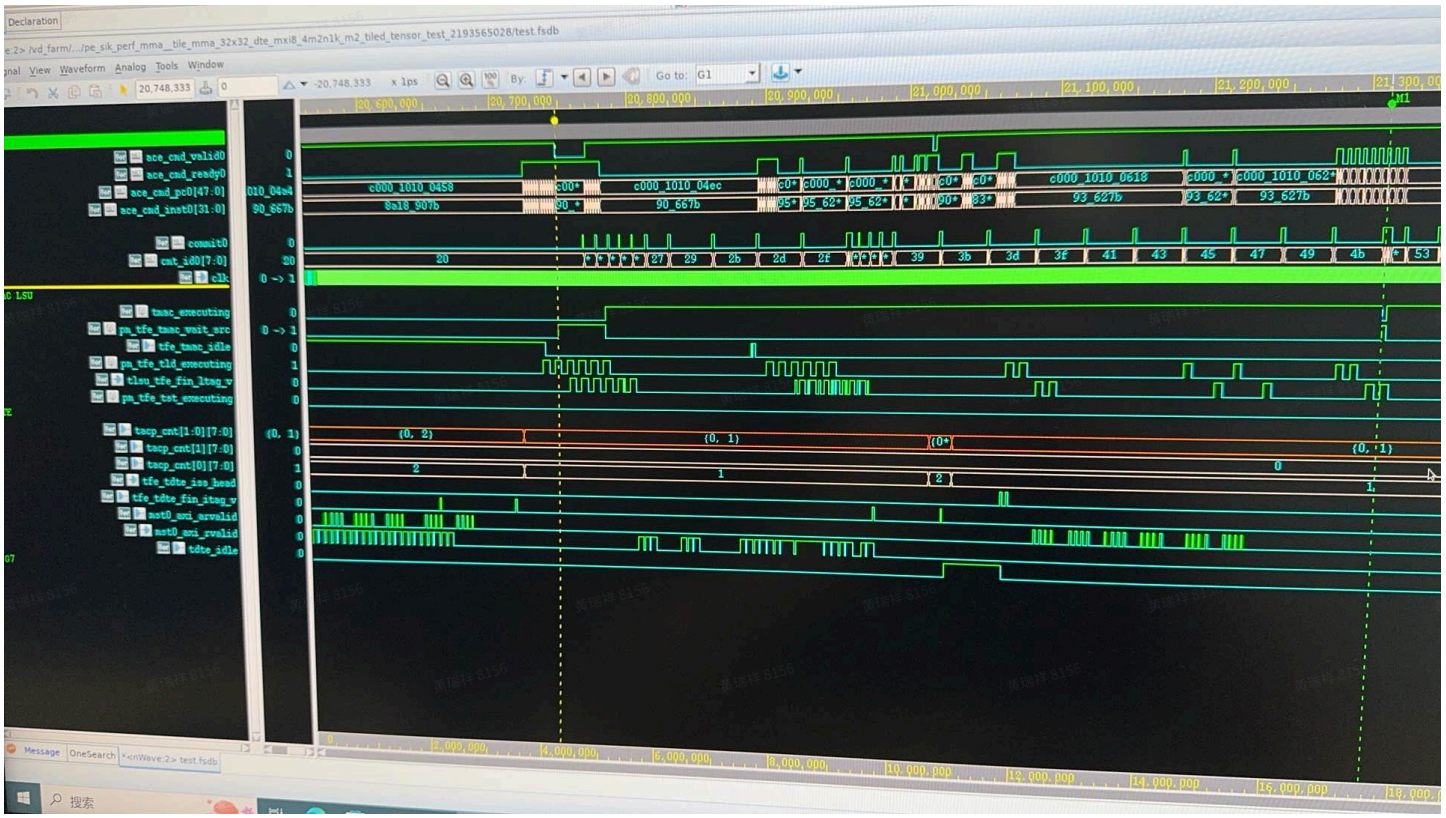
	mst0_axi_rvalid	dte内部axi完成搬运
	tdte_idle	
	tacp_cnt	tcore内pending的tacp cmt-group tile-instr的数量
Write back	wb_ctl_wb_trid[8:1]	tid
	wb_ctl_wb_trid[0]	0：低512B，1：高512B
	wb_ctl_wb_en	Write back使能

代码及波形分析



由于load a3/b1在解WAR依赖的时间比较晚，导致后面的mma decode阶段迟迟无法开始，此处引入约11cycle的bubble。



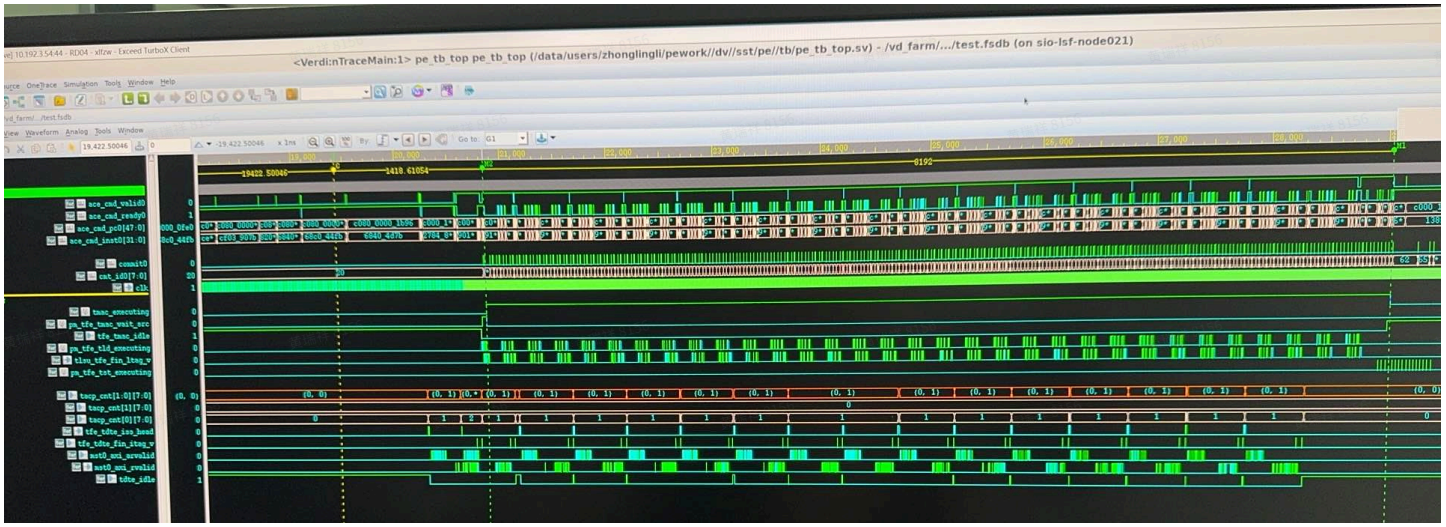


代码优化

代码块

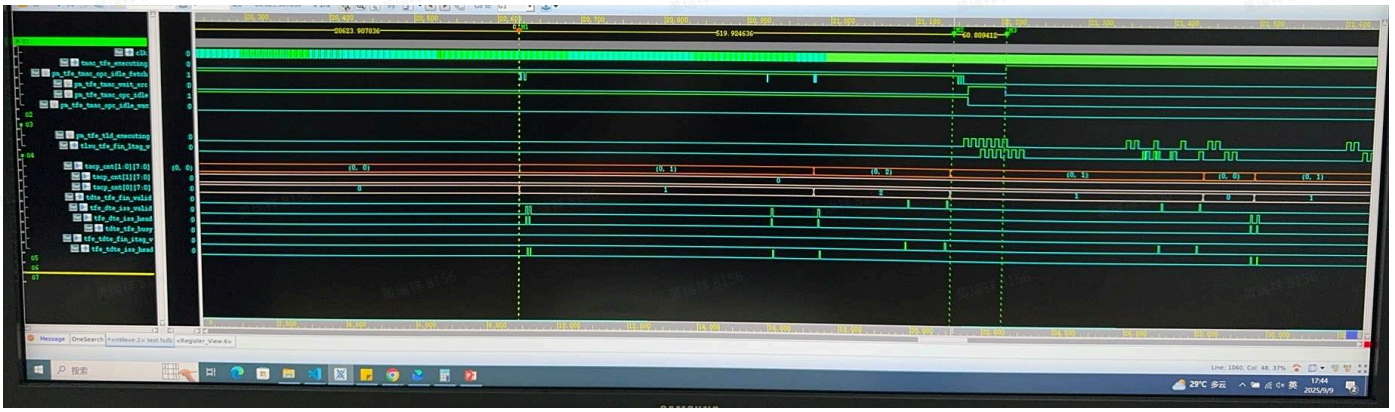
```
/*****1.原始写法******/
load a0
load b0
load a1, a2, a3
load b1
mma.m2 c0, a0, b0
mma.m2 c1,  a1, b0
mma.m2 c2, a2, b0
mma.m2 c3, a3, b0
mma.m2 c4, a0, b1
mma.m2 c5, a1, b1
mma.m2 c6, a2, b1
mma.m2 c7, a3, b1
/*****2.调整load和mma顺序后，期望减少wax依赖对后续mma启动的影响******/
load a0
load b0
mma.m2 c0, a0, b0
load a1
mma.m2 c1, a1, b0
load a2
mma.m2 c2, a2, b0
load a3
mma.m2 c3, a3, b0
load b1
```

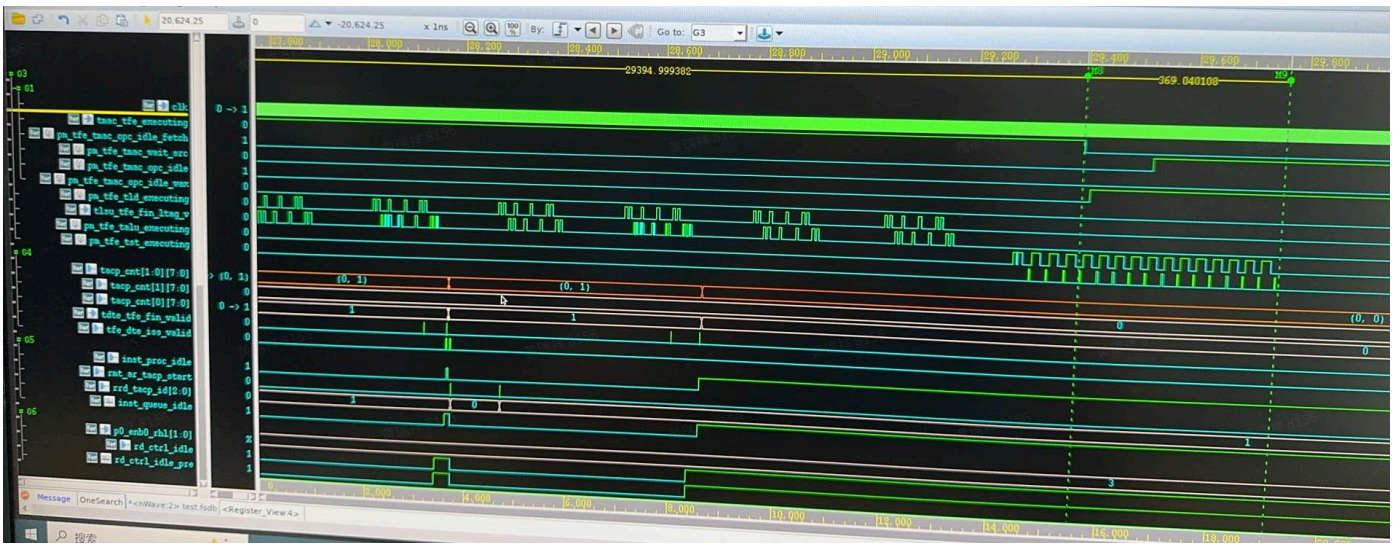
```
25   mma.m2 c4, a0, b1
26   mma.m2 c5, a1, b1
27   mma.m2 c6, a2, b1
28   mma.m2 c7, a3, b1
```

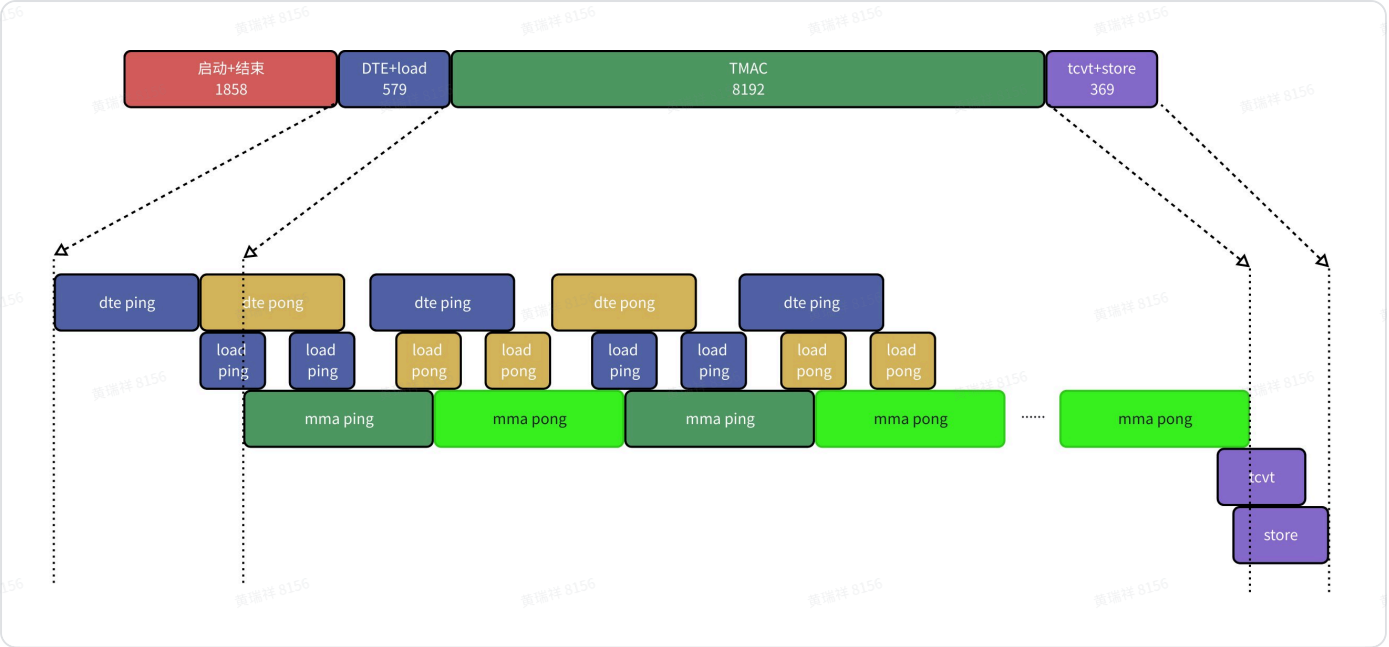


整体性能分析

1. DV计算效率为**74.5%**，周期数为10998. TMAC计算周期8192.







第一次发起dte搬运到完成，共519个时钟周期。

搬运的数据量(chunk size)为128M128K+64N128K：17408B+8704B=26112B。

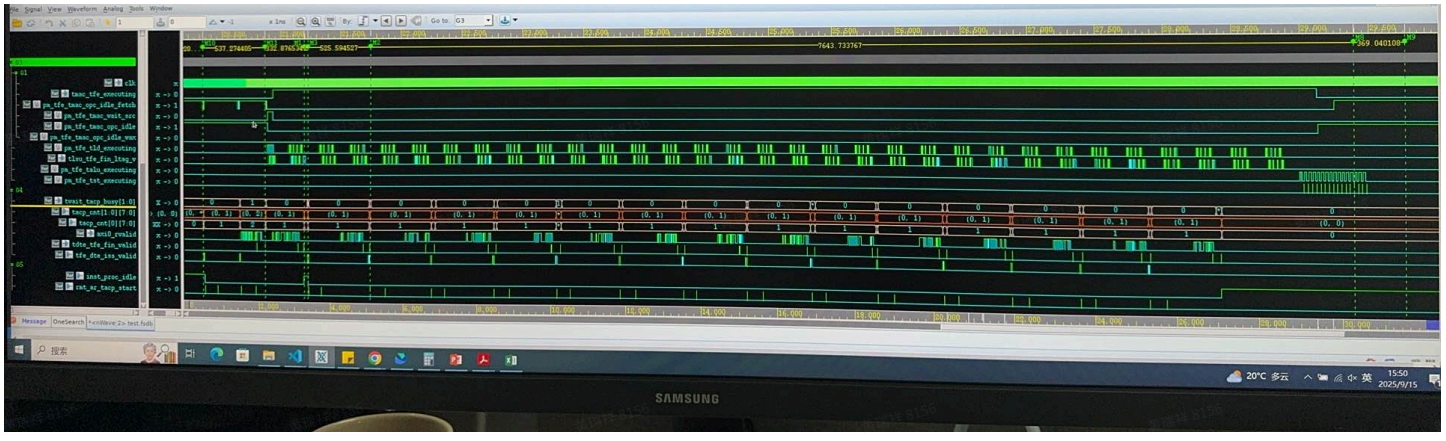
DTE搬运带宽为256B/cycle->512B/cycle, 按照256B/cycle计算，需要102cycle。

1 chunk tmac 512cycle

第一次搬运完成到TMAC开始计算，共60个时钟周期。

load的数据量为128M64K+64N64K：13056B

L2B带宽为512B/cycle，需要25cycle。



在这个case中，启动和结束的时间占比较大，导致效率低。

Mx mma性能分析结果

调用dte+TMAC有以下几个原则：

1. 结合DRAM带宽不稳定的因素，选择合适的chunk size，让chunk的搬运时间小于chunk的TMAC时长
2. mxi8尽量不使用reusea/b, 优先使用reused（减少相邻mma之间无法掩盖加载TR的时间导致的bubble）
3. m和n都展开的情况下，循环体内load使用的寄存器尽早完成mma计算（减少两次loop之间由于数据依赖产生的bubble)
4. 提高DTE的时间利用率，2thread可以提高dte工作时间(减少由于DTE搬运时间无法被TMAC掩盖导致的bubble）

gemm算子优化

r32_mxi8

不同chunk size对应的dte和tmac理论cycle。绿色表示结合DRAM带宽不稳定因素后，TMAC有比较大的概率能够覆盖住dte搬运。

row	m_tile	n_tile	k_tile	m_size	n_size	k_size	A_size	B_size	Total_size	dte cycle	dte sim
32	4	1	8	128	32	256	34816	8704	43520	170	~250双线程
32	4	2	4	128	64	128	17408	8704	26112	102	~400
32	4	2	8	128	64	256	34816	17408	52224	204	~800
32	1	1	16	32	32	512	17408	17408	34816	136	~425
32	2	2	4	64	64	128	8704	8704	17408	68	~499单线程 ~200双线程

下表统计了目前在DV测试中，每种展开方式下的TMAC执行情况。√表示TMAC从开始工作到结束没有bubble。×表示有bubble。

Unroll m	Unroll n	Unroll k	m1	m2	m4
4	4	1	原 x（相邻mma bubble)	原 × 重排未尝试，可以改用4m2n1k_m2_重排+2thread实现	• （TR超）
4	2	1	原 ×（相邻mma bubble) 原+双线程 ×	原 × 重排 √ 原+双线程 √	原 √（要注意寄存器分配） 原+双线程 √
4	1	1	原 ×	原 ×	原 x

			重排 ×	重排 x 重排+手动分配两组loop用不同TR（2k）x	重排 x 重排+手动分配两组loop用不同TR × 原+双线程 √
2	2	1	-	-	原 × 重排 x 原+双线程 √
1	1	4	-	-	原 x 重排 x 原+双线程 x

r16_mxi8

row	m_tile	n_tile	k_tile	m_size	n_size	k_size	A_size	B_size	Total_size	dte cycle	dte sim cy
16	1	4	4	16	128	256	4352	34816	39168	153	~550
16	1	2	4	16	64	256	4352	17408	21760	85	~354
16	1	1	8	16	32	512	8704	17408	26112	102	~395

Unroll m	Unroll n	Unroll k	m1	m2	m4
1	1	4	×	-	-
1	2	1	× 2thread ×	-	-
1	4	1	× 2thread ×	× 2thread ×	-

r32_mxf6

row	m_tile	n_tile	k_tile	m_size	n_size	k_size	A_size	B_size	Total_size	dte cycle	tmac cyc
32	4	2	4	128	64	512	51200	25600	76800	300	
32	4	1	4	128	32	512	51200	12800	64000	250	
32	2	2	4	64	64	512	25600	25600	51200	200	
32	4	1	4	128	32	512	51200	12800	64000	250	
32	1	1	8	32	32	1024	25600	25600	51200	200	

Unroll m	Unroll n	Unroll k	m1	m2	m4
4	4	1			
4	2	1	原？	原 √（TR90） 重排，指定寄存器 √（TR72） 重排，指定寄存器+2thread？	
4	1	1	原？		
2	2	1	原？		
1	1	4	原？		