????

????????

把 [[Pipe]] 第 6 节 §208 的算力单元抽象展开到 RTL 起手前。每个计算实例 = 一条 engine pipe 实例。

0. ? DMA ?????

算力单元与 [[DMA]] 结构同构：都是 "engine pipe 集合 + sync pool + 控制接口 + 共享数据 fabric master"。差别仅在执行通路：

	DMA	算力单元
执行通路	读 → transform → 写	读 → 计算阵列 → 写
数据端	被动单元（L1、DRAM）	被动单元（L1、DRAM）
内部状态	transform buffer	L0 寄存器组
cmd_desc 专属字段	xform_flags, xform_params	compute_op, loop, dtype, accum

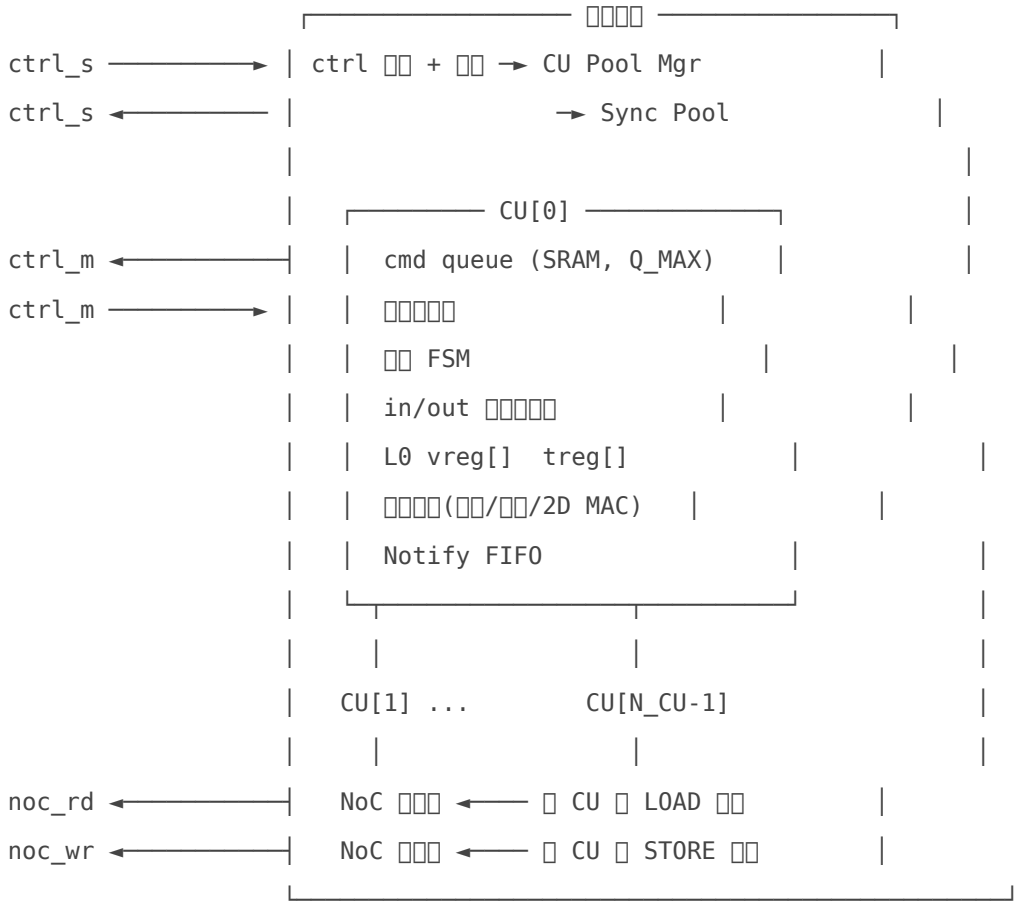
下文凡是结构与 DMA 一致的，引用 [[DMA]] 不重复展开；只描述差异。

1. ????

参数	含义	默认	备注
`N_CU`	计算实例数（每个 = 一条 engine pipe）	4	可异质（见 §13）
`N_SYNC`	sync counter 数	32	
`Q_MAX`	单实例命令队列硬件上限	16	2 的幂
`K_IN / K_OUT / K_SIG`	cmd_desc 列表上限	4/4/4	
`N_L0_VEC`	每实例 L0 向量寄存器数	8	
`N_L0_TEN`	每实例 L0 张量寄存器数	4	
`VEC_LANE`	向量阵列 lane 数	16	INT8 lane
`MAC_M / MAC_K`	2D MAC 阵列尺寸	8 / 8	systolic
`MAC_ACC_W`	MAC 累加器位宽	32	INT32
`OUT_TYPE_SET`	可选输出位宽	{8, 16, 32}	round/sat 软件控
`LOOP_W`	inner-loop 计数器位宽	16	

□□	□□	□□	□□
`STRIDE_DIM / STRIDE_W / SHAPE_W`	□ [[DMA]]	3 / 32 / 24	
□□ `COUNT_W / HANDLE_W / ADDR_W / NOC_W`	□ [[DMA]]		

2. ????



3. ????

□□□□ [[DMA]] §3 □□□□□□□

- `noc_rd_*` / `noc_wr_*` □□□□□□□□□□ L1□ DRAM□□□□ cmd_desc □□ in/out handle.unit_addr □□
- `ctrl_s_*` / `ctrl_m_*` □□□□□□

4. cmd_desc ????

□□ ≤ 1024 bit□□ [[DMA]] §4 □□□□□□ `op + □□□□□` □

□□	□□	□□	□□

0	4	`op`	0=SCALAR_ALU, 1=VEC_ALU, 2=VEC_REDUCE, 3=MAC_2D, 4=ELEMENTWISE_ACT(ReLU/GELU), 5=SOFTMAX_PARTIAL, ...
4	4	`dtype_in`	0=I8, 1=I16, 2=I32, 3=FP8(预留), 4=FP16(预留)
8	4	`dtype_out`	同上
12	2	`accum_mode`	0=覆盖，1=累加（C += A*B），2=累加+饱和
14	2	`act_kind`	ELEMENTWISE 时的激活函数选
16	16	`loop_count`	inner-loop 次数；硬件按此自动 step 地址生成器
32	48	`src_a_base`	byte_addr（落在某条被动单元 pipe 内，见 [[L1]] §4.1 / [[DRAM]] 地址映射）
80	96	`src_a_stride[3]`	
176	72	`src_a_shape[3]`	
248	48	`src_b_base`	MAC_2D / VEC_ALU 用
296	96	`src_b_stride[3]`	
368	48	`dst_base`	
416	96	`dst_stride[3]`	
512	72	`dst_shape[3]`	
584	8	`l0_a_reg`	A 操作数落到哪个 L0 寄存器
592	8	`l0_b_reg`	
600	8	`l0_c_reg`	结果累加目标
608	4+4+4+4	`n_in / n_out / n_sig / _`	
624	96	`in_list[4]`	每项 {handle:16, delta:8}
720	96	`out_list[4]`	
816	96	`sig_list[4]`	
912	112	reserved/padding	

`src_b_*` 仅对二元算子（MAC / VEC_ALU）有效；一元算子（激活、reduce）写 0。

5. ??????

完全同 [[DMA]] §5。多 beat `PSEND` 同样 8 × 128 bit。opcode 表共享。

6. CU Pool ???

结构同 [[DMA]] §6。差别:

- `meta[N_CU]` 多一个 `cu_class` 字段（标量 / 向量 / MAC，若异质例化）
- PALLOC 时 `params.cu_class` 必须与本实例 class 匹配；不匹配视为软件错（行为未定义）

7. CU ?????×N_CU?

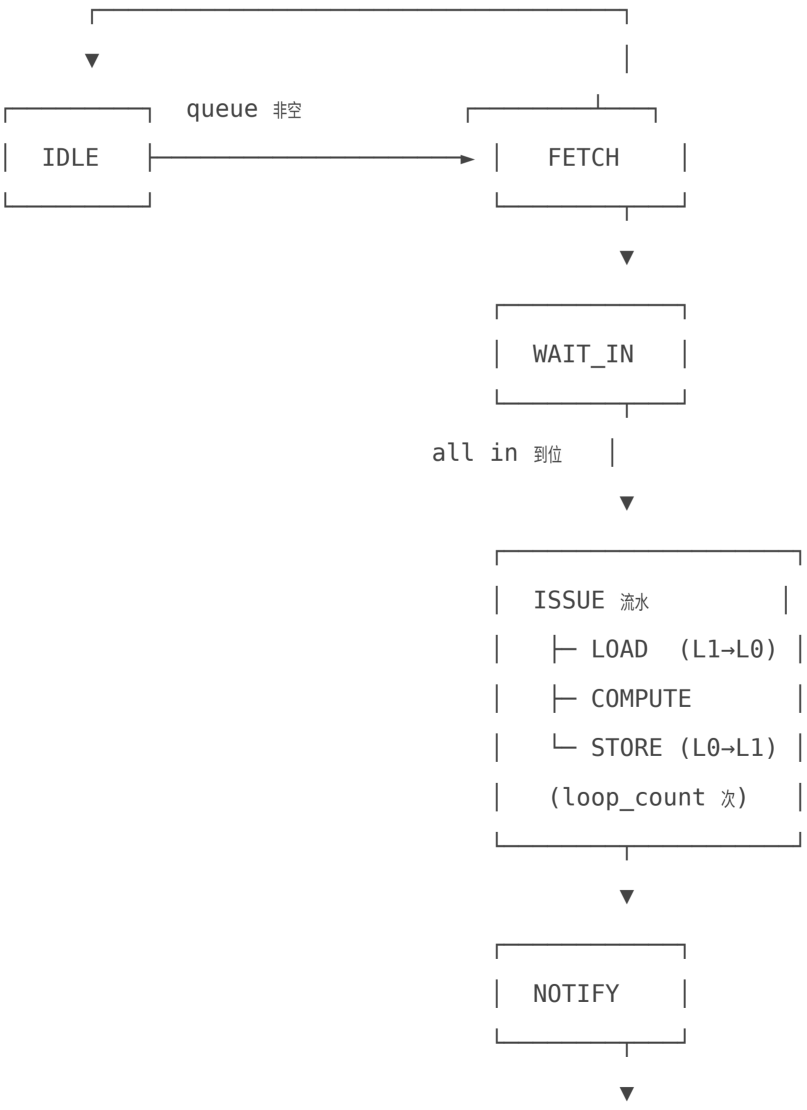
7.1 ????

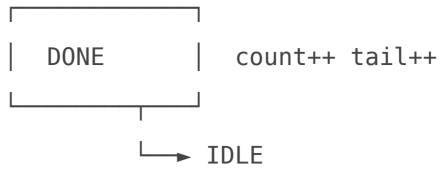
完全同 [[DMA]] §7.1。

7.2 ?????

完全同 [[DMA]] §7.2。

7.3 ?? FSM





状态	时长	行为
IDLE	1 cycle	
FETCH	1 + (STRIDE_DIM-1) cycle	读 cmd_desc + 预算 stride wrap_delta（同 DMA §7.4）
WAIT_IN	不定	顺序 COUNT_QUERY in_list
ISSUE	流水，见 §7.6	
NOTIFY	(n_in + n_out + n_sig) cycle	入 Notify FIFO
DONE	1 cycle	

7.4 L0 ????

项	规格
`vreg[N_L0_VEC]`	每个 `VEC_LANE × dtype_in_bits` 宽；存 vector 操作数 / 结果
`treg[N_L0_TEN]`	每个 `MAC_M × MAC_K × dtype_in_bits` 宽；存 tile 操作数
端口	vreg: 2R/1W； treg: 2R/1W（FMA 类够用）
复位	不复位（undefined），软件必须先 LOAD 才用
共享	实例私有，不跨 CU 共享

实现： vreg 用 register file（small）， treg 用 SRAM macro（一个 treg ≈ 8×8×8 = 512 bit，N_L0_TEN = 4 时 2 KiB）。 treg SRAM 端口配 2R/1W 满足 MAC 2 输入 1 累加。

vreg 与 treg 物理分离 —— 二者访问宽度差很大。

7.5 ????

三类阵列共存（每个实例可裁掉某类，按 cu_class 决定）：

7.5.1 ??

- 1 lane，宽度 `max(dtype_in, dtype_out)` ≤ 32 bit
- op： ADD/SUB/MUL/SHIFT/CMP/SELECT/BIT
- latency 1 cycle， throughput 1/cycle

7.5.2 ??

- `VEC_LANE` 路并行 INT8 ALU + 16-bit accumulator option
- op： lane-wise ADD/SUB/MUL/MAC/CMP，跨 lane REDUCE_SUM/MAX/MIN
- latency 1-2 cycle， throughput 1/cycle
- elementwise activation： LUT 表（ 4 KiB ROM/SRAM）实现 ReLU / GELU / sigmoid 等

7.5.3 2D MAC

- `MAC_M × MAC_K` systolic 阵列
- 单元: INT8 × INT8 → INT16 partial → 累加进 MAC_ACC_W 寄存器
- 输入: A treg (M × K), B treg (K × N, 本设计 N = MAC_M); 权重和激活方向可在 cmd_desc 内对调
- 输出: C treg (M × N), 累加进已有 treg（accum_mode）
- pipeline depth = `MAC_K + 2` cycle
- 每 cycle 注入一行 A、一列 B; MAC_M = MAC_K = 8 时 8 cycle 填满, 第 9 cycle 起每 cycle 一个稳态结果

下溢 / 上溢 / round: cmd_desc.dtype_out 与 accum_mode 决定输出阶段的 saturate + truncate / round_to_nearest。专用硬件块（输出后处理 stage）实现。

7.6 LOAD/COMPUTE/STORE inner-loop ??

把一条 cmd 拆成 `loop_count` 次 micro-op, 三级流水:

```
cycle:  N   N+1   N+2   N+3   N+4   N+5
        |    |     |     |     |     |
LOAD    op0  op1   op2   op3   op4   op5
COMPUTE      op0   op1   op2   op3   op4
STORE              op0   op1   op2   op3
```

- L0 寄存器组作为级间 buffer: LOAD 写、COMPUTE 读 + 写、STORE 读
- 软件用 `l0_a_reg / l0_b_reg / l0_c_reg` 指定不同 op 用哪些 reg, 硬件按 `op_id % N_L0_REG` 做 cyclic 调度避免读写冲突
- leading 2 cycle 只 LOAD（COMPUTE/STORE 等数据）; tailing 2 cycle 只 STORE
- inner-loop 内不可中断 / 不可被外部 stall（除 NoC backpressure）

7.7 in / out ?????

每实例 3 个独立累加器（A、B、C）。复用 [[DMA]] §7.4 的累加器实现。差别:

- 目标地址落在被动单元（L1 / DRAM）的地址空间内
- inner-loop 由 `loop_count` 驱动, 与 `shape[3]` 一起决定终止
- 地址生成单次给一个 NoC 请求; burst 长度 = 一拍 L0 寄存器宽度 / NOC_W（典型 1 beat）

7.8 Notify FIFO

完全同 [[DMA]] §7.6。

8. Sync Pipe Pool

完全同 [[DMA]] §8。

9. ??????

9.1 NoC ? / ?

结构同 [[DMA]] §9.1。NoC 主端可对接 L1 / DRAM; 对接 L1 时可按低延迟假设缩 outstanding 深度，对接 DRAM 时按 DRAM 延迟特性配置。

9.2 ctrl_m ??

完全同 [[DMA]] §9.2。

9.3 ctrl_s ????

完全同 [[DMA]] §9.3。差别: PALLOC 时 `params.cu_class` 决定走哪个 cu_class 的子池。

10. ????????

10.1 CU ? pending wait CAM

同 [[DMA]] §10.1。

10.2 CU ? ctrl_m

同 [[DMA]] §10.2。

10.3 CU ? NoC????????

- LOAD: CU 发 noc_rd_req 给被动单元（L1 / DRAM, master），地址按 §7.7 生成; resp 数据按 req_id 路由到 CU 的 L0 写口
- STORE: CU 发 noc_wr_req + wr_data 给被动单元; 不等 ack（fire-and-forget）, NOTIFY 中的 COUNT_DELTA 由 CU 内部 FSM 在 STORE 流水尾部排出（保证落到目标单元后才发，由 NoC 同 id 同地址保序保证）

10.4 ? cmd_desc ? in ? out ???? pipe

参见 [[Pipe]] §289: 合法用法。硬件执行顺序:

1. WAIT_IN 阶段查询该 pipe 的 count ≥ in 阈值
2. LOAD 阶段读被动单元（数据已在）
3. COMPUTE
4. STORE 阶段写被动单元
5. NOTIFY 阶段: 对该 pipe 先发 `COUNT_DELTA(-in_delta)`，再发 `COUNT_DELTA(+out_delta)`

被动单元端按到达顺序应用即可。In-place 累加的语义保证依赖 NoC 同 id 同地址保序。

11. ?????

信号 / 状态	复位值
同 [[DMA]] §11 全部项	
L0 vreg / treg	不复位（undefined）

信号 / 状态	复位值
计算阵列 pipeline 寄存器	复位
MAC 累加器	0

L0 不复位的影响: 未 LOAD 就 COMPUTE 是软件 bug, 硬件不阻止; 结果未定义。

12. ??????

同 [[DMA]] §12, 加:

信号	宽度	用途
`dbg_cu_state[N_CU]`	N_CU × 3	每 CU FSM 状态
`dbg_mac_active[N_CU]`	N_CU	MAC 阵列在跑
`dbg_inner_loop_idx[N_CU]`	N_CU × LOOP_W	当前 micro-op 序号
`dbg_l0_read_hazard[N_CU]`	N_CU	LOAD/COMPUTE 同 reg 冲突计数 (应为 0)
`dbg_pipeline_stall_cycles[N_CU]`	N_CU × 16	NoC backpressure 累计 stall

13. ??????

1. `N_CU` 与异质组合 (如 1 个 2D MAC 实例 + 2 个向量实例 + 1 个标量实例 vs 同质)
2. `VEC_LANE / MAC_M / MAC_K` 实际规模 —— 直接决定算力
3. `dtype` 支持集合 (INT4 / FP8 / FP16 / BF16)
4. MAC 阵列拓扑 (systolic 1D vs 2D, weight-stationary vs output-stationary)
5. L0 寄存器组的实际端口数 (FMA 用 3R+1W 还是分段 2R+1W)
6. inner-loop 流水级数 (3 级 vs 4 级, 多一级换 stall 余裕)
7. 输出 round / saturate 硬件块的位宽
8. 是否例化激活函数 LUT 共享还是 per-CU 独立

14. ??? Pipe ??

抽象层概念	本文档落点
engine pipe 契约	§7.1 队列 + §7.2 count + §7.3 FSM
sync pipe 契约	§8
落地 A	§6 + §8 各自独立 bitmap
落地 B	§3 ctrl vs noc 物理分开
落地 D	§8 sync 独立
特性 3 (单 pipe 不分叉)	§7.3 FSM 单 channel 严格串行
特性 6 (硬件零检查)	§5 / §6 / §7.4 L0 未初始化不报警
特性 7 (参数创建即不可变)	§6 alloc 后 cu_class / queue_depth 定死

抽象层概念	本文档落点
特性 8 (三类 pipe)	engine 在 §7，sync 在 §8
§289（in/out 同 pipe)	§10.4 NOTIFY 顺序

Revision #1
Created 2026-05-13 13:49:43 UTC by Colin
Updated 2026-05-13 13:49:43 UTC by Colin