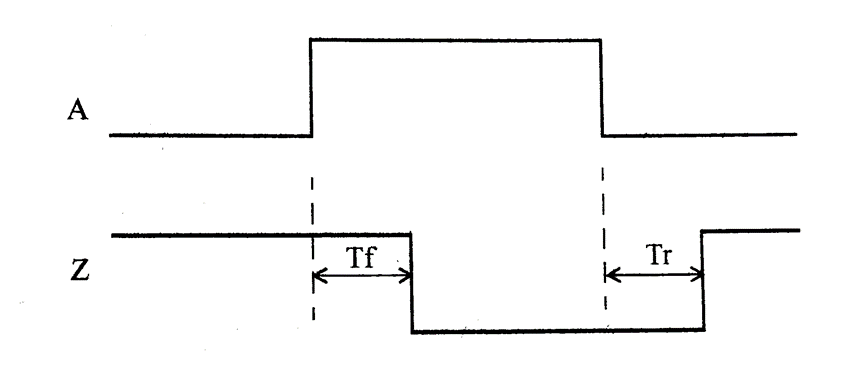
DACS NOTE FOR TIMING

1 Introduction

在电路设计流程中，EDA 工具参数的选取，对电路设计的性能、面积、功耗等指标有重要的影响。在本次实验中，我们将基于Ibex RISC-V 核心，调整和优化EDA 工具的参数，以获得电路性能指标的提升。

Function + number of inputs + driving strength

xp5：驱动强度/尺寸标识（值越小，驱动越弱，面积越小）

x1/x2/x4：驱动强度倍数（值越大，驱动越强，面积越大）

**Basic logic gate**

INVx1 INVxp67 INVx2

NAND2xp5 NAND3xp33 NAND4xp25

AND2x4 AND5x1 XOR2xp5 XOR2x1

NOR2xp33 XNOR2xp5

**Composite logic gate**

AOI21xp5 AOI21xp1 Y = !(A & (B | C)) AOI22xp5 Y = !((A & B) | (C & D))

AOI31xp33 Y = !(A & B & C | D) OAI21xp5 Y = !((A | B) & C)

AO222x2 Y = (A & B) | (C & D) | (E & F)

OAI22xp5

**Multi-level gate**

OA21x2 Y = (A | B) & C OA22x2 Y = (A | B) & (C | D)

**Pre CTS Pre Place Post Route**

一、综合前：io\_addend[29] 的负载为 10.000，扇出为 4。布局后：io\_addend[20] 的负载为 3.888，扇出为 2。

**综合阶段**：若逻辑综合工具为了满足时序要求，可能为某些路径插入\*\*缓冲器（Buffer）\*\*或调整驱动强度，导致PORT的扇出和负载电容被高估。

**布局后阶段**：通过物理优化（如单元摆放、时钟树综合），实际扇出可能减少，或驱动单元被替换为更合适的尺寸，从而降低PORT的负载电容。

e.g.

**PrePlace**：工具插入缓冲器拆分扇出，PORT扇出变为 **3**（直接驱动缓冲器）。

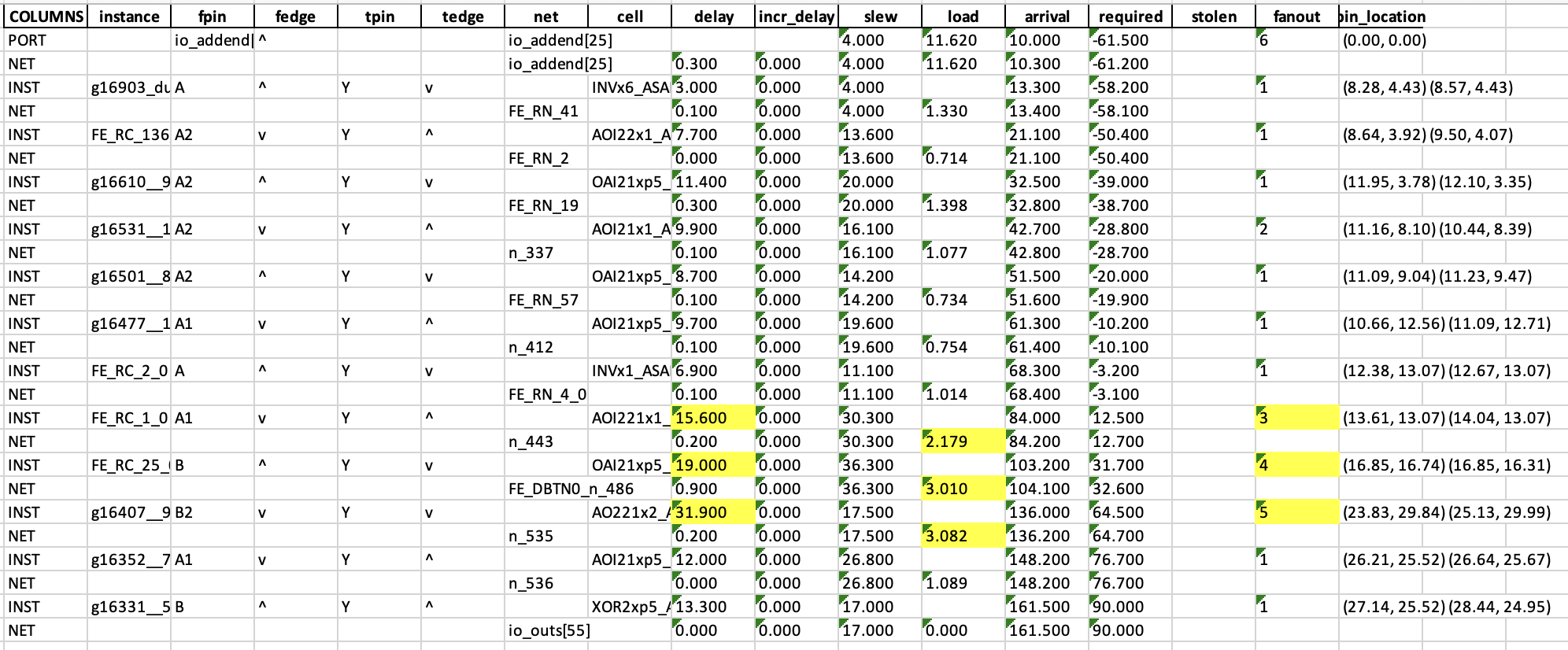
**PreCTS**：布局工具发现其中2个单元距离较远，插入本地缓冲器，PORT扇出进一步减少到 **2**。

**PostRoute**：布线后因互连电容过大，替换驱动单元为高驱动强度版本，最终PORT扇出稳定为 **1**（仅驱动一个缓冲器）。

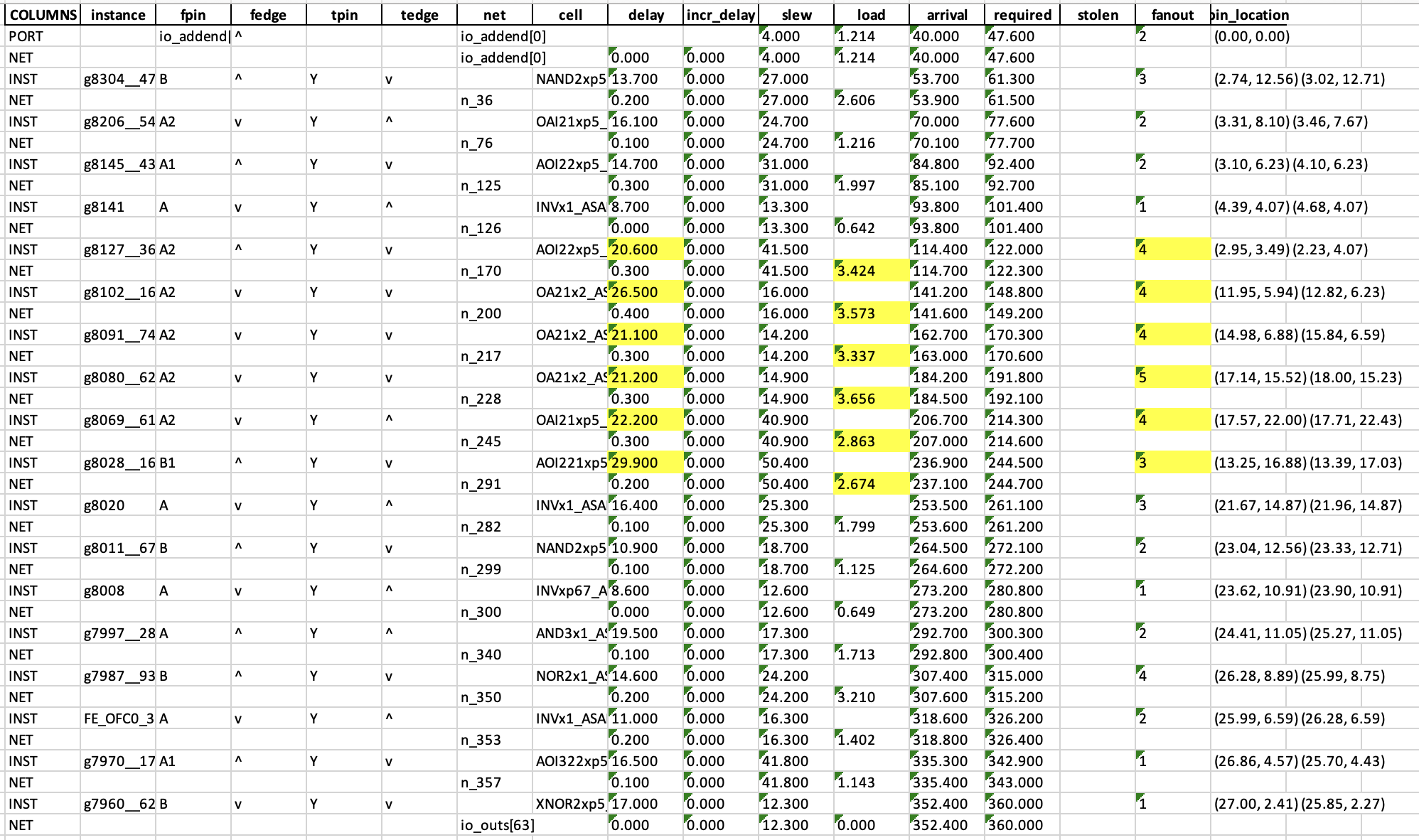
2 Prefix Adder Timing

2.1 Different Clock Period

*0.1*

**

*0.4*

**

当 clk\_period\_ns 升高（时序约束放宽）：  
综合工具不再需要为了满足严格的时序要求而采取激进优化策略，例如：

* 减少逻辑级数拆分：工具不会强制拆分关键路径的逻辑级数，允许更长的组合逻辑延迟。
* 减少流水线插入：无需插入额外的寄存器（流水线阶段）来缩短单周期路径。
* 避免使用高速单元：不再优先选择高驱动强度（低延迟但大尺寸）的标准单元。

**紧凑型结构（如Brent-Kung树）**：

* 逻辑级数少，但每级复杂度高（需更多晶体管）。
* 总延迟低，面积和功耗较高。

**简化型结构（如Sklansky树）**：

* + 逻辑级数多，但每级复杂度低（晶体管更少）。
  + 总延迟高，面积和功耗较低。

2.2 Binary Chain Connection

Reinforcement Learning

三角矩阵编码：将前缀加法器的节点矩阵（node\_mat）压缩为一维向量。每个节点(i,j)表示从第j位到第i位的进位生成逻辑

优化动作：

ADD/DELETE：动态增删节点以平衡面积与性能。

MERGE\_LEVELS：合并冗余逻辑层级，缩短关键路径。

SPLIT\_CHAIN：分割长进位链，提升并行度

启发式：随机分割进位链（30%概率）/合并同层节点

使用TD误差计算动作优势值，对非法操作（如删除不存在的节点）设置概率为0

Area: 逻辑门数量

area = np.sum(adder.node\_mat)

Timing: 关键路径长度

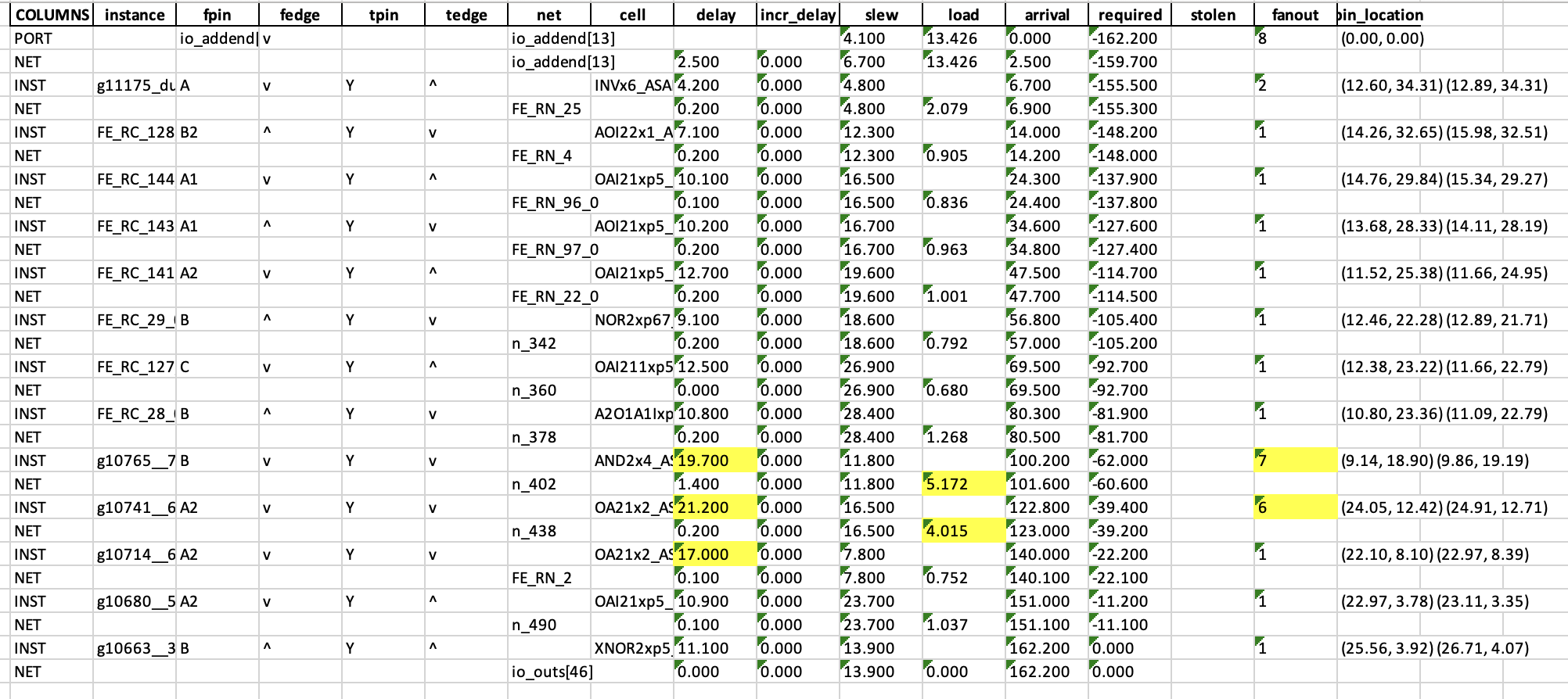
max\_level = np.max(adder.level\_mat)

Power: 估算为面积与扇出的乘积

fanout = np.sum(adder.node\_mat, axis=1)

power = np.sum(fanout \* area)

Brentkung

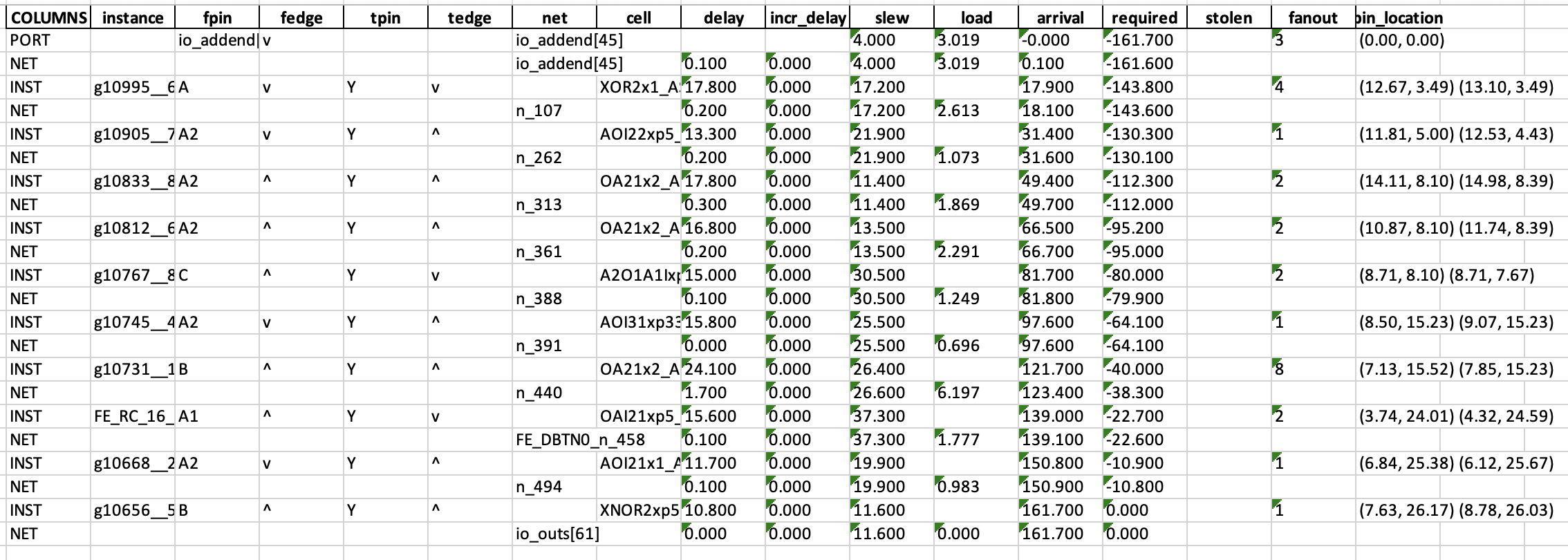


# generic logical synthesis effort: [low/medium/high]

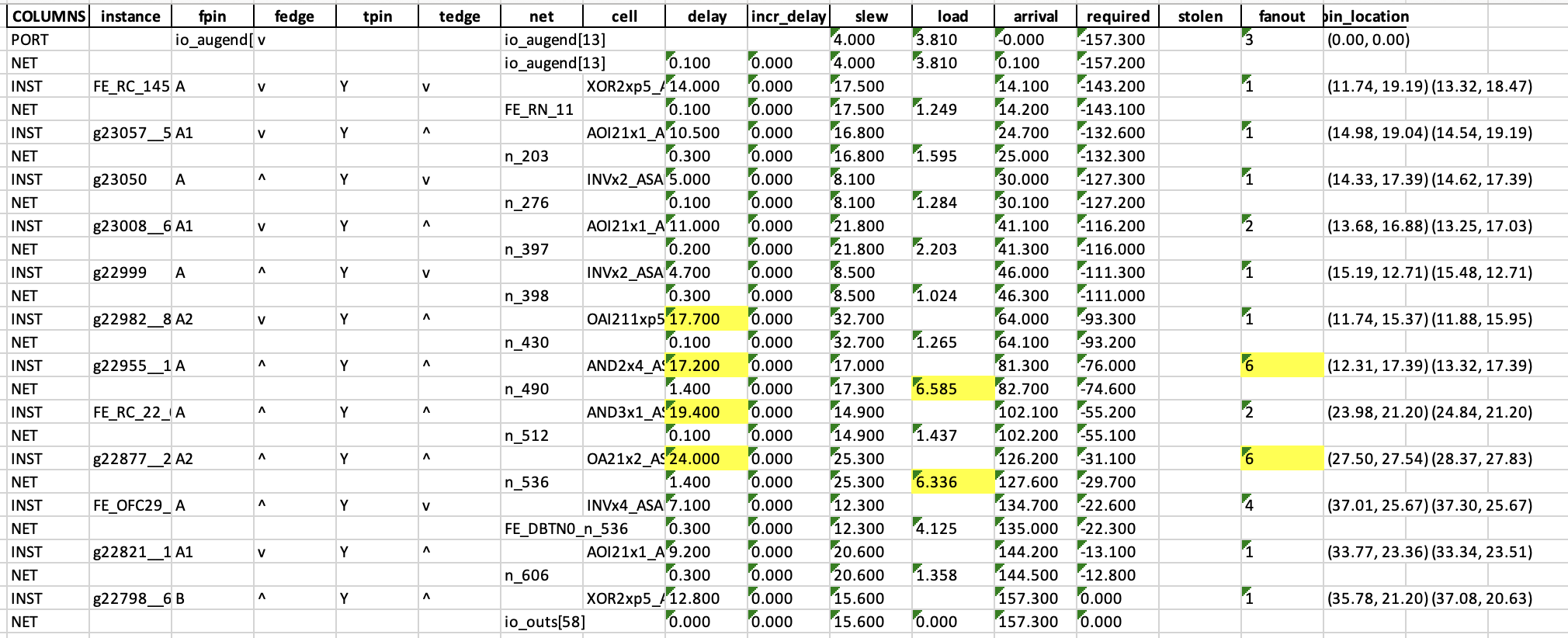
'syn\_generic\_effort': 'medium' --> ‘high’

# wire length optimization effort: [low/medium/high]

'place\_detail\_wire\_length\_opt\_effort': 'medium' --> ‘high’



RL 1000：



pseudocode

class DRLAdderOptimizer:

target\_bits = N # 目标位宽

state\_size = N\*(N+1)/2 # 三角矩阵元素数

action\_space = [ADD, DELETE, MERGE, SPLIT] # 动作集合

policy\_network # 策略网络（如PPO、DQN）

reward\_weights = [0.2, 0.7, 0.1] # 面积/延迟/功耗权重

Function optimize():

current\_adder = initialize\_adder() # 初始加法器配置

while not converged:

# 状态编码

state = encode\_adder\_config(current\_adder)

# 策略网络选择动作

action = policy\_network.predict(state)

# 执行选定的硬件操作

new\_adder = apply\_action(current\_adder, action

# 评估硬件指标

old\_ppa = evaluate(current\_adder)

new\_ppa = evaluate(new\_adder)

# 计算奖励

reward = calculate\_reward(old\_ppa, new\_ppa)

# 存储经验并更新策略

memory.store(state, action, reward, new\_state)

policy\_network.update(memory.sample())

current\_adder = new\_adder if improved else current\_adder

return best\_adder\_config

Function apply\_action(adder, action):

if action == SPLIT\_CHAIN:

# 示例：在随机位置分割进位链

split\_pos = find\_optimal\_split(adder)

for i in range(split\_pos):

adder.delete(split\_pos, i)

for i in range(split\_pos, N):

adder.add(i, split\_pos)

elif action == MERGE\_LEVELS:

# 合并同层级节点

merge\_candidates = find\_mergeable\_nodes(adder)

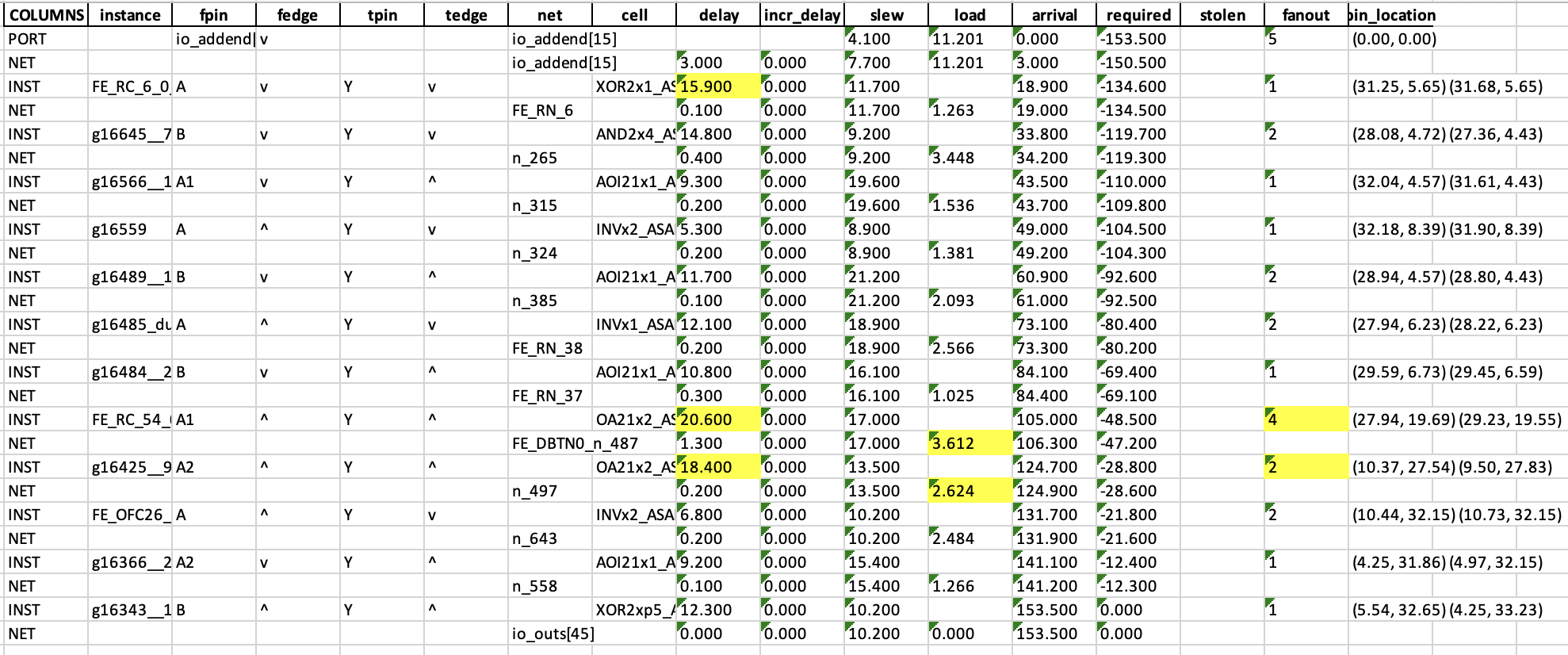
execute\_merge(merge\_candidates)

# ...其他动作实现

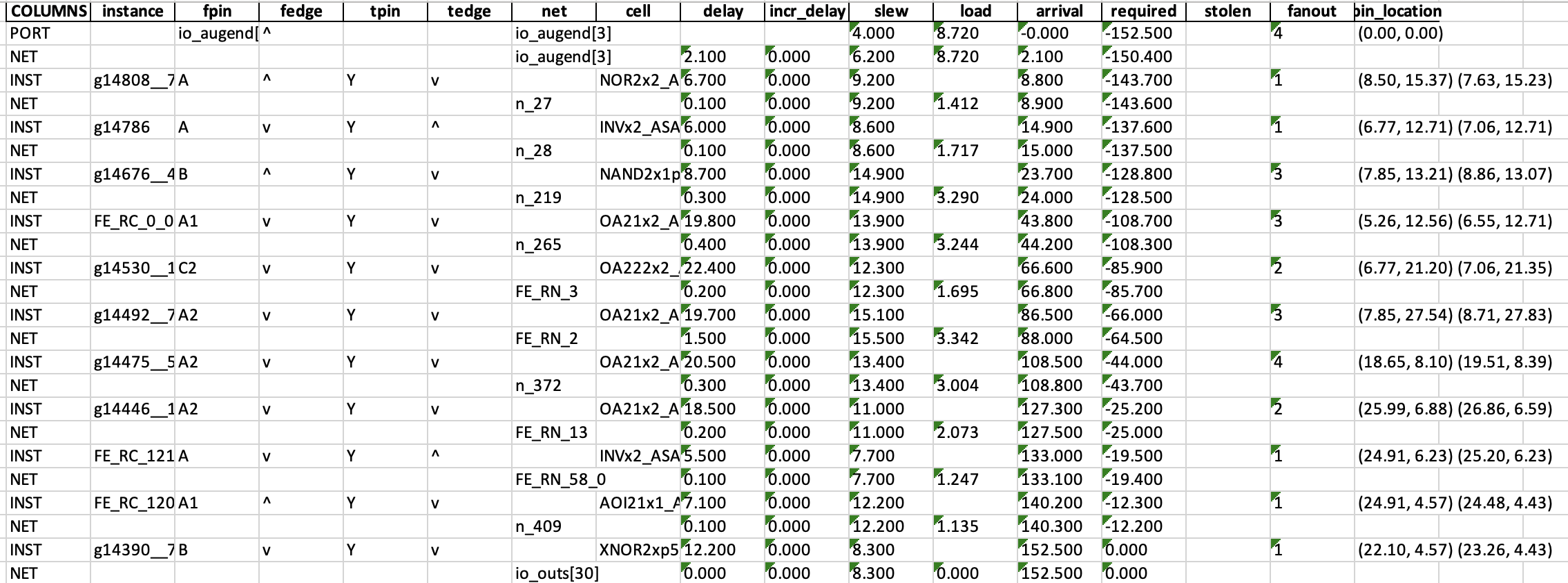
return modified\_adder

Sklansky

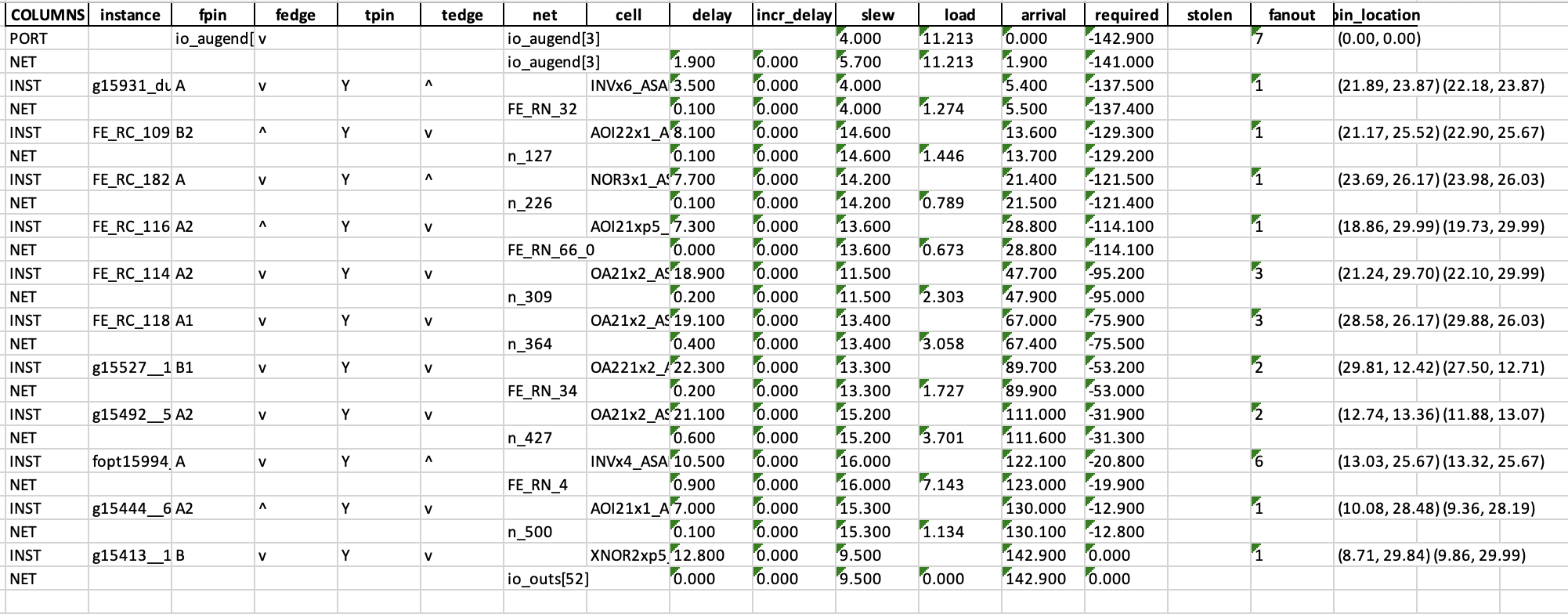
0次



100次



1000次

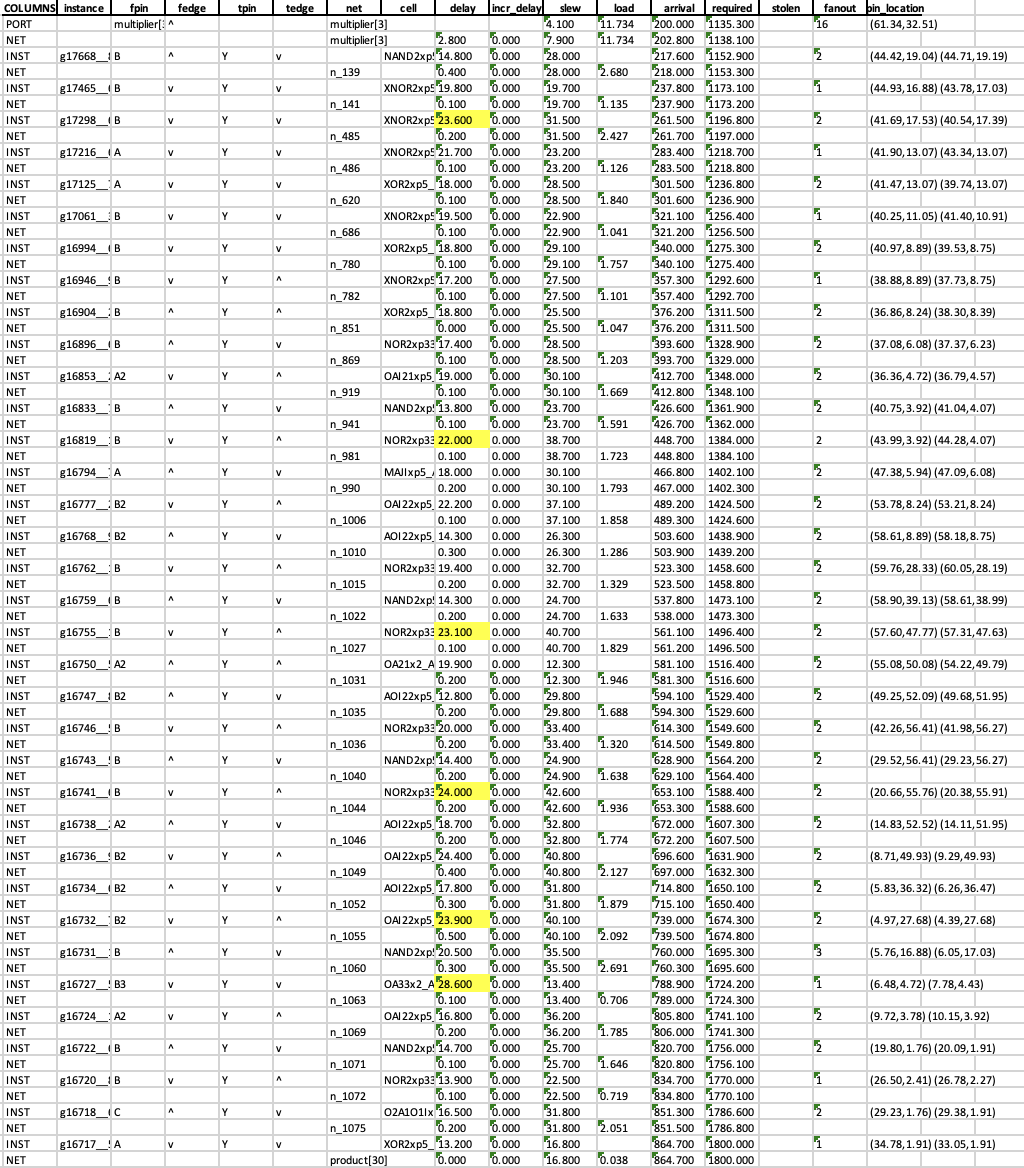


趋势：关键路径上门的数量减少，尤其是基础逻辑单元减少，倾向于选择驱动强度大的类型。

3 Multiplier

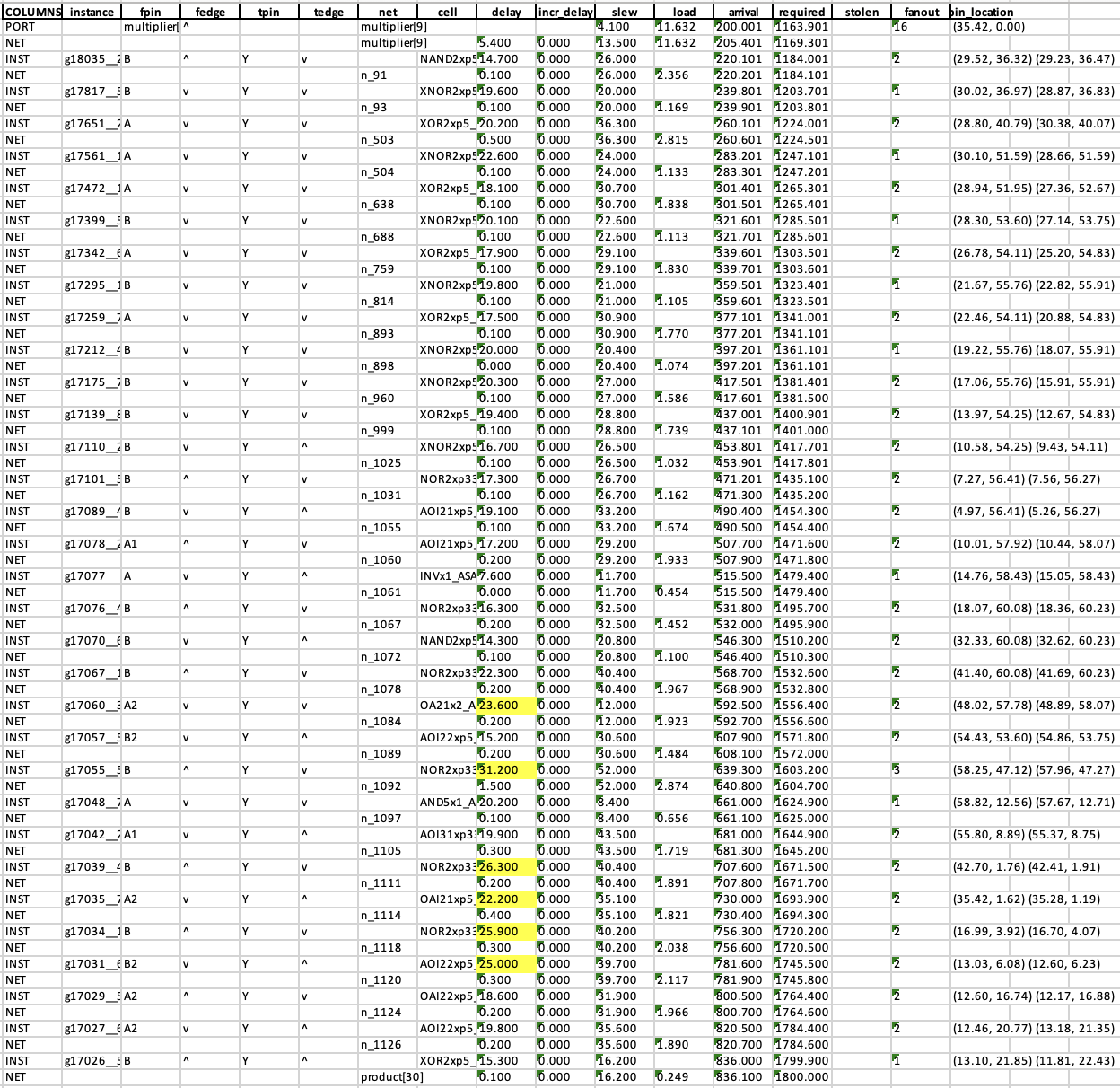
**Dadda乘法器**

* **核心逻辑单元**：
  + 更均衡的XNOR/XOR使用（XNOR 5次，XOR 4次），部分积压缩可能通过复合门优化。
  + 高频使用**NOR2**（7次）和**NAND2**（6次），但通过复合门减少级数。
* **复合逻辑门**：
  + 大量使用**AOI22**（4次）和**OAI22**（4次），以及**MAJIxp5**（多数门）、**OA33x2**（3输入OR-AND）和**O2A1O1Ixp33**（复杂组合逻辑），显著减少关键路径级数。
  + 复合门占比更高，逻辑功能集成度强。



**Wallace乘法器** 逻辑级数较少，但单级延迟和线延迟较高。

* **核心逻辑单元**：
  + 大量使用**XNOR/XOR**门（XNOR出现7次，XOR出现6次），表明部分积压缩阶段依赖加法器结构（如全加器/半加器）。
  + 较多使用**NOR2**（6次）和**NAND2**（3次）等基本门。
* **复合逻辑门**：
  + 使用少量AOI（AOI21/AOI22/AOI31）和OAI（OAI21/OAI22）门（总计约6次），如AOI22用于多输入组合逻辑。
  + 包含特殊门如**AND5x1**和**AOI31**，用于复杂逻辑优化。



**高延迟瓶颈**：

* 存在极端门延迟（如**NOR2xp33**单门延迟达31.2），可能因长路径依赖或高负载导致。
* **XNOR/XOR**门平均延迟较高（如XNOR达22.6，XOR达20.2），部分积压缩阶段延迟累积明显。
* 线延迟总和较高，可能因布局不规则导致布线较长。

以减少加法级数为目标，依赖基本门（XNOR/XOR/NOR）快速压缩部分积，但关键路径易出现高延迟门。需要解决牺牲布局规则性换取更少级数：

SA for count\_view

