北京大学
PEKING UNIVERSITY

基于AI的数字系统设计

第6讲：AI辅助组合逻辑设计

主讲：陶耀宇

2026年秋季

北京大学版权所有

北京大学
PEKING UNIVERSITY

AI辅助组合逻辑设计

北京大学版权所有

组合逻辑分类与 AI 辅助要点

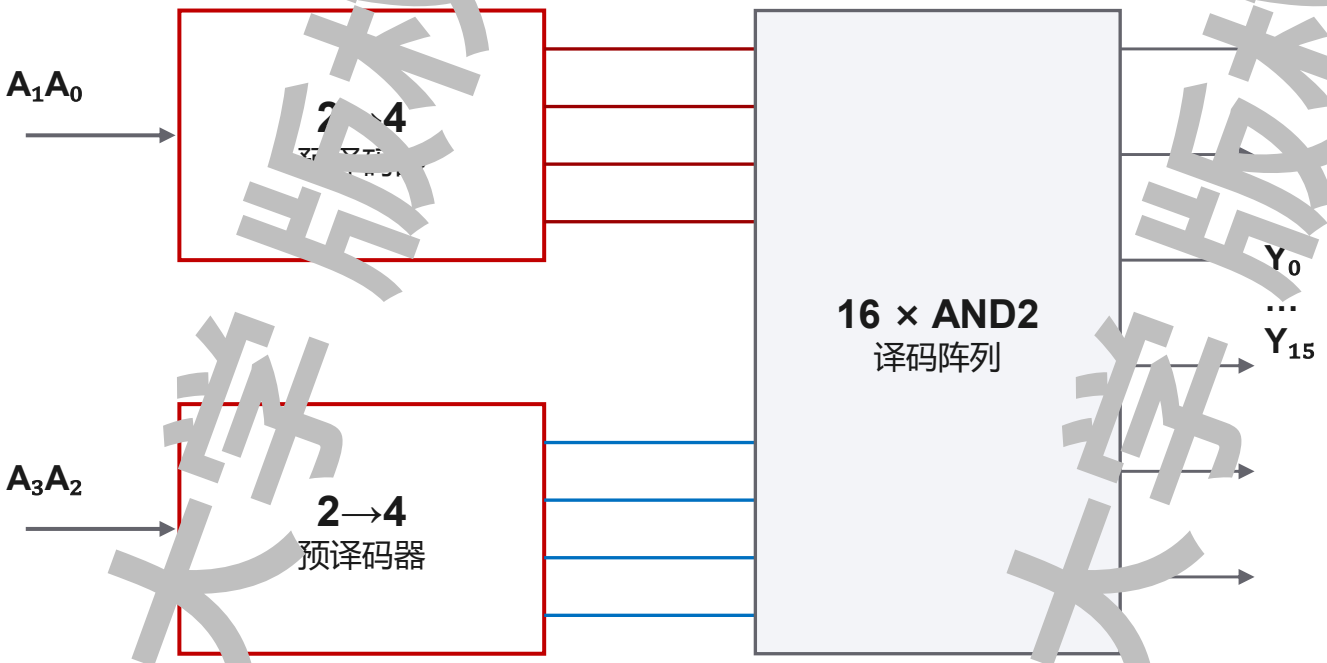
- 结构决定 PPA 上限，AI 在“生成—验证—优化”三个环节提升效率

类别	典型结构	延迟	面积	AI 辅助重点
静态 CMOS 门	NAND / NOR / AOI / OAI	逻辑努力模型	晶体管数	等价检查、尺寸建议
Ripple-Carry	全加器级联	O(N)	O(N)	参数化生成、最坏向量
Carry-Select	双路预计算 + MUX	O(√N)	≈ 2× RCA	分块方案搜索
Carry-Lookahead	分组 (G, P) + LCU	O(log N)	O(N)	扇入约束下的展开
PGK / 前缀	预处理-前缀-后处理	O(log N)	由前缀图决定	前缀图 RL 搜索
Kogge-Stone	跨度倍增，扇出 2	log₂N 级	N·log₂N 单元	布线感知、稀疏化
Sklansky	分治，高扇出	log₂N 级	(N/2)·log₂N	尺寸与扇出优化
阵列乘法器	AND 阵列 + 加法阵列	O(N)	O(N²)	穷举 / 形式验证
Booth 乘法器	Radix-4 重编码	部分积减半	编码器开销	符号与 +1 纠错
Wallace 树	3:2 / 4:2 压缩树	O(log N)	O(N²)	压缩器树 RL 优化
移位器	对数 / 漏斗	log₂N 级	N·log₂N MUX	统一漏斗模板
编码器	预译码、树形优先编码	O(log N)	O(N) / O(2ⁿ)	无效输入、树形化
比较器	相等 / 减法 / 树形	O(log N)	O(N)	符号与边界、资源共享

北京大学版权所有

6.1 译码器 – 二进制译码与预译码

- n 位输入 → 2ⁿ 路 one-hot 输出；预译码把高扇入门拆成两级小门



每条预译码线只驱动 4 个 AND2；直接实现则每个地址位（原/反）要驱动 8 个 AND4

方案	门数	最大扇入
直接实现	16 × AND4	4
两级预译码	8 × AND2 + 16 × AND2	2

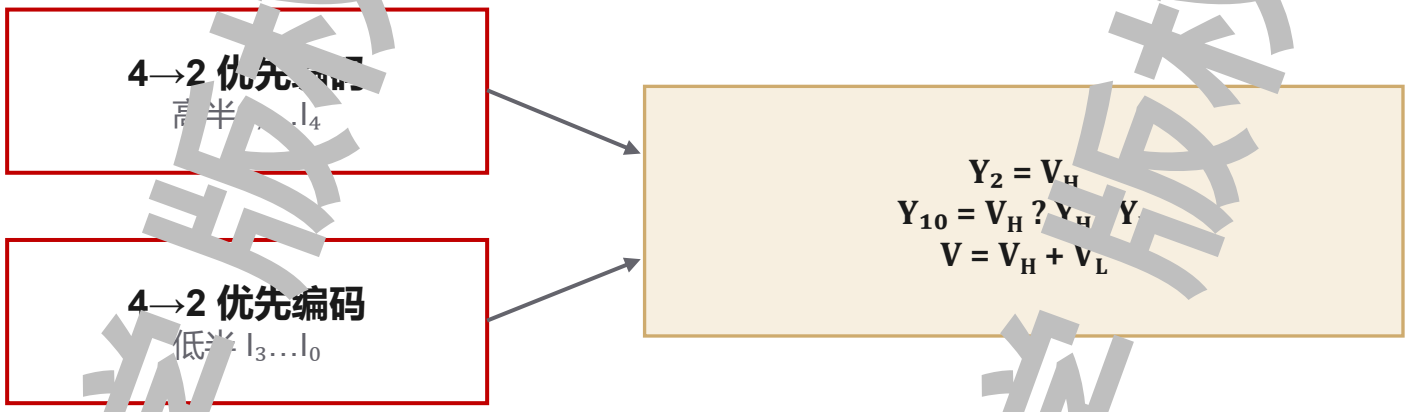
- one-hot 输出：任意时刻只有一条有效
- SRAM 行译码常用多级预译码 + 字线驱动器
- 预译码减小扇入、分摊输入负载，也便于按逻辑努力分配尺寸

6.2 编码器与优先编码器

- 优先编码器输出最高位有效请求的编号，可递归二分构成 O(log N) 树

$I_7 \dots I_0$	$Y_2Y_1Y_0$	V
1xxxxxxx	111	1
01xxxxxx	110	1
001xxxxx	101	1
0001xxxx	100	1
00001xxx	011	1
000001xx	010	1
0000001x	001	1
00000001	000	1
00000000	xxx	0

树形实现（8 → 3）



- 普通编码器（one-hot → 二进制）：Y_k 是所有第 k 位为 1 的输入之“或”
- 优先编码器：高位优先；每层一个 MUX，深度 O(log N)
- 应用：**总线仲裁、中断控制、浮点规格化中的前导零计数（LZC）**

6.3 AI辅助：编码器 / 译码器的生成与验证

- 描述简单但陷阱多：锁存器、X 传播与综合出的长链结构

提示词 Prompt

生成参数化优先[illegible]（高位优先），输出编号 idx 与有效标志 valid，**要求纯组合、无锁存器。**

LLM 生成的 Ver[illegible]

```
module pri[illegible]parameter N = 8, W = $clog2(N)) (
  input       [N-1:0] req,
  output reg [W-1:0] idx,
  output             valid);
  integer i;
  always @(*) begin
    idx = {W{1'b0}};                    // 默认赋值：避免锁存器
    for (i = 0; i < N; i = i + 1)
      if (req[i]) idx = i;              // 后赋值覆盖 → 高位优先
  end
  assign valid = |req;
endmodule
```

AI 输出检查清单

1. **默认赋值**
   always 开头先给 idx 赋默认值，否则[illegible]出锁存器
2. **casez 而非 casex**
   casex 会把输入中的 X 当无关项，掩盖仿真问题
3. **结构检查**
   循环描述可能综合为线性链；大 N 时要求 AI 生成树形
4. **穷举验证**
   N = 8 只有 256 种输入，可全部与参考模型比对

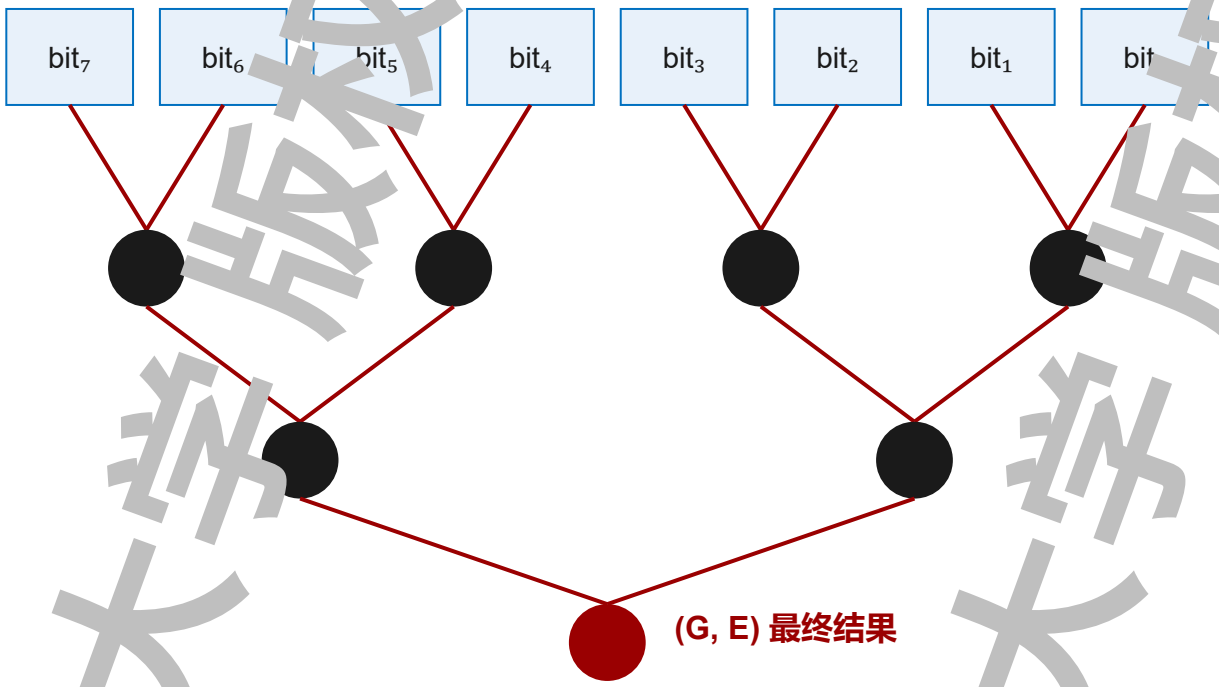
6.4 比较器 – 三种典型实现

- 相等比较用 XNOR + AND 树；大小比较可复用减法器，或用专门的树形结构

相等比较器	减法型比较器	树形比较器
$a_i \odot b_i$（XNOR） × N	A + ¬B + 1	每位 (g_i, e_i)
AND 归约树	加法器进位 cout	(G, E) 逐级合并
EQ	cout = 1 ⇒ A ≥ B	GT EQ LT
• 深度 $\log_2 N$ 级 • 只判断相等，面积最小	• 复用 ALU 加法器，几乎零额外面积 • 延迟 = 加法器进位延迟	• $g_i = a_i \cdot \neg b_i$，$e_i = a_i \odot b_i$ • 深度 O(log N)，面积 O(N)

6.5 树形比较器 - 结构与有符号处理

- (G, E) 合并运算满足结合律，与前缀加法器的 (G, P) 完全同构



叶节点：$g_i = a_i \cdot \neg b_i$，$e_i = a_i \odot b_i$；每个黑点做一次 (G, E) 合并，高位在左

$$(G, E) = (G_H + E_H \cdot G_L,\ E_H \cdot E_L)$$

$A > B \Leftrightarrow G = 1$；$A = B \Leftrightarrow E = 1$；$A <$ B $\Leftrightarrow$ [illegible] E

- **与前缀加法器 (G, P) 同构 → 可复用前缀网络与 AI 前缀图搜索成果**
- 有符号比较：两数最高位取反后按无符号比较（补码 → 偏移码）
- 深度 $\log_2 N$，面积 O(N)，适合宽位宽比较

6.6 AI辅助：比较器生成与数据通路共享

- 行为级描述交给综合器；AI 负责补齐有符号语义并发现可共享的硬件

提示词 Prompt

> 生成同时支持有符号 / 无符号的 32 位比较器，输出 lt / eq / gt，要求用最高位取反技巧处理有符号数

LLM 生成的 Ver

```
module cmp  (parameter N = 32) (
  input  [N-1:0] a, b,
  input          sgn,        // 1: 有符号比较
  output         lt, eq, gt);
  // 有符号：最高位取反，转换为无符号比较
  wire [N-1:0] ax = {a[N-1] ^ sgn, a[N-2:0]};
  wire [N-1:0] bx = {b[N-1] ^ sgn, b[N-2:0]};
  assign eq = (a == b);
  assign gt = (ax > bx);
  assign lt = ~eq & ~gt;
endmodule
```

AI 输出检查清单

1. **有符号边界**
   -2^{N-1} 与 $2^{N-1}-1$ 互比，以及相等的负数
2. **结构控制**
   行为级 > 由综合器选结构；需要 loc 深度时要求 AI 写树形
3. **资源共享**
   AI 分析数据通路，让比较与减法共用同一个加法器
4. **验证**
   随机 + 边界用例，并与 $signed 参考模型形式等价

总结：组合逻辑类与 AI 辅助要点

- 结构决定 PPA 上限，AI 在“生成—验证—优化”三个环节提升效率

类别	典型结构	延迟	面积	AI 辅助重点
静态 CMOS 门	NAND / NOR / AOI / OAI	逻辑努力模型	晶体管数	等价检查、尺寸建议
Ripple-Carry	全加器级联	O(N)	O(N)	参数化生成、最坏向量
Carry-Sele	双路预计算 + MUX	O(√N)	≈ 2× RCA	分块方案搜
Carry-Looka d	分组 (G, P) + LCU	O(log N)	O(N)	扇入约束 展开
PGK / 前缀	预处理-前缀-后处理	O(log N)	由前缀图决定	前缀图 RL 搜索
Kogge- tone	跨度倍增，扇出 2	log N 级	$N \cdot \log_2 N$ 单元	布线感知、稀疏化
Skla sky	分治，高扇出	$\log_2 N$	$(N/2) \cdot \log_2 N$	尺寸与 优化
阵列乘法器	AND 阵列 + 加法阵列	O(N)	$O(N^2)$	穷举 形式验证
Boot 乘法器	Radix-4 重编码	部分积减半	编码器开销	符号与 +1 纠错
W e 树	3:2 / 4:2 压缩树	O(log N)	$O(N^2)$	压缩树 RL 优化
移位器	对数 / 漏斗	$\log_2$ 级	$N \cdot \log_2 N$ MUX	统一漏斗模板
编码器	预译码、树形优先编码	O(log N)	O(N) / $O(2^n)$	优先 器、树形化
比较器	相等 / 减法 / 树形	O(log N)	O(N)	符号与 界、资源共享

北京大学版权所有

北京大学
PEKING UNIVERSITY

AI辅助复杂逻辑设计

北京大学版权所有

AI辅助复杂模块A...设计1 - 卷积加速模块

- 卷积 – AI计算中最常用的算子

卷积最初是一种信号处理滤波操作 –

“AI神经网络也可看作是一种滤波”

WinoGrad算法起源于1980年，是Shmuel Winograd提出用来减少FIR滤波器计算量的一个算法

输入d = [d0,d1,d2,d3]，卷积核g = [g0, g1, g2]

1维卷积可表达为

输入尺寸 = 2 + 3 -1 = 4

$$F(2,3) = \begin{bmatrix} d0 & d1 & d2 \\ d1 & d2 & d3 \end{bmatrix} \begin{bmatrix} g0 \\ g1 \\ g2 \end{bmatrix} = \begin{bmatrix} r0 \\ r1 \end{bmatrix}$$

输出尺寸

卷积核尺寸

6次乘法

AI辅助卷积加速软件设计

- 卷积 – AI计算中最常用的算子

输入d = [d0, d1, d2,d3]，卷积核g = [g0, g1, g2]

1维卷积可表达为

Winograd

$$F(2,3) = \begin{bmatrix} d0 & d1 & d2 \\ d1 & d2 & d3 \end{bmatrix} \begin{bmatrix} g0 \\ g1 \\ g2 \end{bmatrix} = \begin{bmatrix} r0 \\ r1 \end{bmatrix}$$

存在重复元素

$$F(2,3) = \begin{bmatrix} d0 & d1 & d2 \\ d1 & d2 & d3 \end{bmatrix} \begin{bmatrix} g0 \\ g1 \\ g2 \end{bmatrix} = \begin{bmatrix} m1 + m2 + m3 \\ m2 - m3 - m4 \end{bmatrix}$$

$$m1 = (d0 - d2)g0 \quad m2 = (d1 + d2)\frac{g0 + g1 + g2}{2}$$

$$m4 = (d1 - d3)g2 \quad m3 = (d2 - d1)\frac{g0 - g1 + g2}{2}$$

预算好一次g的加减后可重复复用 -> 4次乘法

AI辅助卷积加速核设计

• 卷积 – AI计算中最常用的算子

目标：$y_0 = d_0g_0 + d_1g_1 + d_2g_2$，$y_1 = d_1g_0 + d_2g_1 + d_3g_2$　乘法次数 6 → 4

1　基础技巧：相邻系数的和差法（2 次乘法）

$u = pa + qb$，$v = qa + pb$

$\Rightarrow u, v = [\,(p+q)(a+b) \pm (q-p)(b-a)\,] / 2$

2　拆分：中间块 + 边界项

$y_0 = d_0g_0 + [\,d_1g_1 + d_2g_2\,]$，$y_1 = [\,d_1g_0 + d_2g_1\,] + d_3g_2$

中间块系数矩阵 $[g_1\ g_2;\ g_0\ g_1]$ 副对角线不同，无法直接省乘法

3　对称化：借边界项修正

$y_0 = (d_0-d_2)g_0 + [\,d_1g_1 + d_2(g_0+g_2)\,]$

$y_1 = (d_1-d_3)g_2 + [\,d_1(g_0+g_2) + d_2g_1\,]$

边界项提取公因子后仍各为 1 次乘法；令 $p = g_1$，$q = g_0+g_2$

4　套用和差法：$a = d_1$，$b = d_2$

$p + q = g_0+g_1+g_2$，　$q - p = g_0-g_1+g_2$

中间块只需 2 次乘法，得到 m_2、m_3

结果

$m_1 = (d_0 - d_2)\ g_0$

$m_2 = (d_1 + d_2)(g_0+g_1+g_2)\ /\ 2$

$m_3 = (d_2 - d_1)(g_0-g_1+g_2)\ /\ 2$

$m_4 = (d_1 - d_3)\ g_2$

$y_0 = m_1 + m_2 + m_3$

$y_1 = m_2 - m_3 - m_4$

乘法次数	g 侧因子
6 → 4	离线预计算 在线仅加减+4次乘

思想：兼容并包

北京大学 版权所有

AI辅助卷积加速[illegible]设计

- 卷积 – AI计算[illegible]常用的算子

element-wise multiplication（Hadamard product）

输入d = [d0, d1, d2,d3]，卷积核g = [g0, g1, g2]

1维卷积可表达为

$$F(2,3) = \begin{bmatrix} d0 & d1 & d2 \\ d1 & d2 & d3 \end{bmatrix} \begin{bmatrix} g0 \\ g1 \\ g2 \end{bmatrix} = \begin{bmatrix} m1+m2+m3 \\ m2-m3-m4 \end{bmatrix}$$

$$m1 = (d0 - d2)g0 \quad m2 = (d1+d2)\frac{g0+g1+g2}{2}$$

$$m4 = (d1 - d3)g2 \quad m3 = (d2-d1)\frac{g0-g1+g2}{2}$$

1D Winograd

$$Y = A^T\left[(Gg) \odot (B^T d)\right]$$

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} m1+m2+m3 \\ m2-m3-m4 \end{bmatrix}$$

$$g = [g_0 \quad g_1 \quad g_2]^T$$

$$d = [d0 \quad d1 \quad d2 \quad d3]^T$$

AI辅助卷积加速算法设计

- 卷积 – AI计算中最常用的算子

将一维卷积运算定义为$F(m,r)$，m为Output Size，r为Filter Size，则输入信号的长度为$m+r-1$，卷积运算是对应位置相乘然后求和，**输入信号每个位置至少要参与一次乘法**，所以乘法数量最少与输入信号长度相同，记为

$$\mu(F(m,r)) = m+r-1$$

在行列上分别进行一维卷积运算，可得到二维卷积，记为$F(m\times n, r\times s)$，输出为$m\times n$，卷积核为$r\times s$，则输入信号为$(m+r-1)(n+s-1)$，乘法数量至少为

$$\begin{aligned}\mu(F(m\times n, r\times s)) &= \mu(F(m,r))\mu(F(n,s)) \\ &= (m+r-1)(n+s-1)\end{aligned}$$

直接按滑动窗口方式计算卷积，一维时需要$m\times r$次乘法，二维时需要$m\times n\times r\times s$次乘法，**远大于上面计算的最少乘法次数**。

使用Winograd算法计算卷积快在哪里？一言以蔽之：**快在减少了乘法的数量**，将乘法数量减少至$m+r-1$或$(m+r-1)(n+s-1)$。

AI辅助卷积加速软件设计

- 卷积 – AI计算中最常用的算子

$$Y = A^T\left[(G g) \odot (B^T d)\right]$$

g：表示卷积核

d：表示输入信号

G：卷积核变换矩阵，尺寸为$(m+r-1)\times r$

B^T：输入变换矩阵，尺寸为$(m+r-1)\times(m+r-1)$

A^T：输出变换矩阵，尺寸为$m\times(m+r-1)$

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

$$g = [g_0 \quad g_1 \quad g_2]^T$$

$$d = [d0 \quad d1 \quad d2 \quad d3]^T$$

计算过程可分为4步：

（1）输入变换

（2）卷积核变换

（3）[illegible]

（4）输出变换

1D Winograd

AI辅助卷积加速硬件设计

- 卷积 – AI计算中最常用的算子

$$Y = A^T \left[(G g) \odot (B^T d) \right]$$

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

$$g = [g_0 \quad g_1 \quad g_2]^T$$

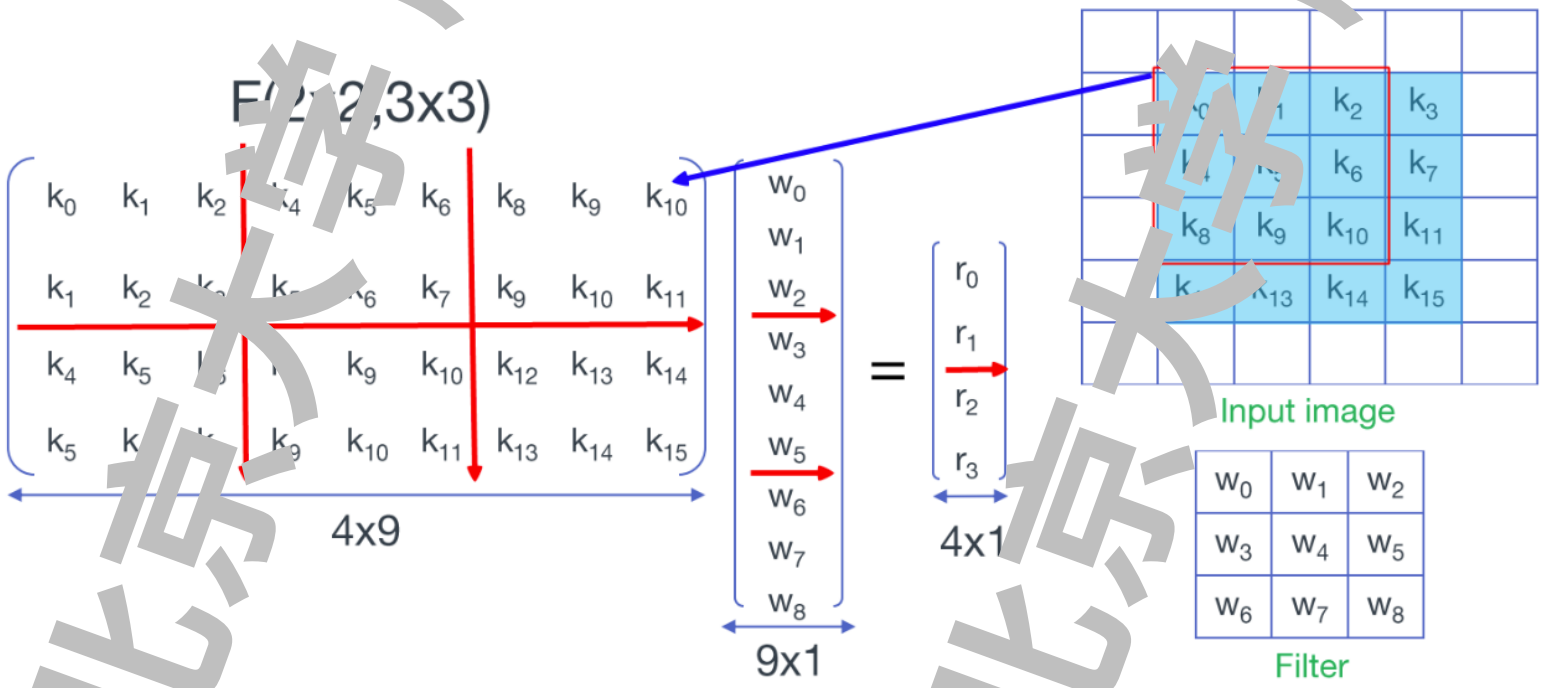
$$d = [d0 \quad d1 \quad d2 \quad d3]^T$$

2D Winograd

A minimal 1D algorithm F(m, r) is **nested with itself** to obtain a minimal 2D algorithm (m×m, r×r).

$$Y = A^T \left[\left[G g G^T \right] \odot \left[B^T d B \right] \right] A$$

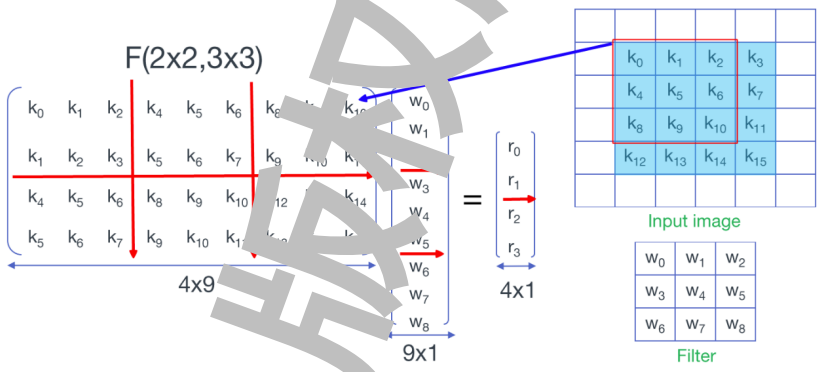
g为$r \times r$ filter，d为$(m+r-1) \times (m+r-1)$

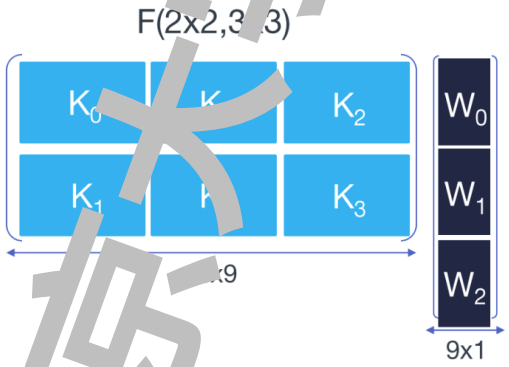


AI辅助卷积加速核设计

- 卷积 – AI计算中最常用的算子

令$D_0 = [k_0, k_1, k_2, k_3]^T$，即窗口的第0行元素，$D_1$ D_2 D_3表示第1、2、3行；$W_0 = [w_0, \quad w_2]^T$，





$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} R_0 \\ R_1 \end{bmatrix} = \begin{bmatrix} K_0W_0 + K_1W_1 + K_2W_2 \\ K_1W_0 + K_2W_1 + K_3W_2 \end{bmatrix}$$

$$= \begin{bmatrix} A^T\left[(GW_0) \odot (B^TD_0)\right] + A^T\left[(GW_1) \odot (B^TD_1)\right] + A^T\left[(GW_2) \odot (B^TD_2)\right] \\ A^T\left[(GW_0) \odot (B^TD_1)\right] + A^T\left[(GW_1) \odot (B^TD_2)\right] + A^T\left[(GW_2) \odot (B^TD_3)\right] \end{bmatrix}$$

$$= A^T\left[\left[G[W_0\ W_1\ W_2]G^T\right] \odot \left[B^T[d_0\ d_1\ d_2\ d_3]B\right]\right]A$$

$$= A^T\left[GgG^T\right] \odot \left[B^TdB\right]\Big]A$$

$$\begin{bmatrix} K_0 & K_1 & K_2 \\ K_1 & K_2 & K_3 \end{bmatrix} \begin{bmatrix} W_0 \\ W_1 \\ W_2 \end{bmatrix} = \begin{bmatrix} R_0 \\ R_1 \end{bmatrix} = \begin{bmatrix} M_0 + M_1 + M_2 \\ M_1 - M_2 - M_3 \end{bmatrix}$$

Matrix multiply F(2,3)! **4** multiplications

$$M_0 = (K_0 - K_2) \cdot W_0 \qquad M_1 = (K_1 + K_2) \cdot \frac{W_0 + W_1 + W_2}{2}$$

$$M_3 = (K_1 - K_3) \cdot W_2 \qquad M_2 = (K_2 - K_3) \cdot \frac{W_0 - W_1 + W_2}{2}$$

AI辅助卷积加速器设计

- NVDLA中的Winograd卷积核设计

在MAC并行计算单元中计算

$$A^T\left[\left[GgG^T\right]\odot\left[B^TdB\right]\right]A$$

预先算好　　在输入datapath计算

For m = 3, r = 2

$$B^T=\begin{bmatrix}1&0&-1&0\\0&1&1&0\\0&-1&1&0\\0&-1&0&1\end{bmatrix},G=\begin{bmatrix}1&0\\\frac{1}{2}&\frac{1}{2}\\\frac{1}{2}&-\frac{1}{2}\\0&1\end{bmatrix}$$

$$A^T=\begin{bmatrix}1&1&1&0\\0&1&-1&0\\0&1&1&1\end{bmatrix}$$

For m = 4, r = 3

$$B^T=\begin{bmatrix}4&0&-5&0&1&0\\0&-4&-4&1&1&0\\0&4&-4&-1&1&0\\0&-2&-1&2&1&0\\0&-2&-1&-2&1&0\\0&4&0&-5&0&1\end{bmatrix}$$

$$G=\begin{bmatrix}\frac{1}{4}&0&0\\-\frac{1}{6}&-\frac{1}{6}&-\frac{1}{6}\\-\frac{1}{6}&\frac{1}{6}&-\frac{1}{6}\\\frac{1}{24}&\frac{1}{12}&\frac{1}{6}\\\frac{1}{24}&-\frac{1}{12}&\frac{1}{6}\\0&0&1\end{bmatrix}$$

$$A^T=\begin{bmatrix}1&1&1&1&1&0\\0&1&-1&2&-2&0\\0&1&1&4&4&0\\0&1&-1&8&-8&1\end{bmatrix}$$

北京大学版权所有

Winograd卷积硬件实现 – 新方法：AI生成Verilog

- 用加减法换乘法 -> AI按变换矩阵自动生成Winograd卷积RTL

Winograd 最小滤波算法 F(2,3)

$$Y = A^T [(G\,g) \odot (B^T\,d)]$$

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \quad G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

输入变换 / 输出变换只含 0、±1 → 纯加减法；G 含 ½ → 权重离线预变换

6 → 4
1D F(2,3)
乘法次数

36 → 16
2D F(2×2,3×3)
乘法次数

2.25×
2D 乘法
节省倍数

AI生成Winograd RTL的流程

① 数学规格输入
给出 B^T、G、A^T 矩阵与 m、r 取值

↓

② 定点化约束
位宽增长、有符号运算、½ 系数处理

↓

③ 分层生成
先生成 1D 变换单元，再组合为 2D

↓

④ 自动验证
NumPy 参考模型逐位比对，[illegible]

关键：把“矩阵变换”翻译成“加减网络”，由AI完成而非手工推导

AI生成1D Winograd F(2,3) Verilog代码

- 权重离线预变换 U = 2·G·g -> 片上仅 4 次乘法 + 加减网络

Prompt 关键约束

1. **算法锁定**
   实现 F(2,3)，给出 B^T、G、A^T
2. **系数处理**
   权重用 2·G·g 整数化，输出 >>>1
3. **位宽管理**
   输入 DW 位，逐级推导位宽防溢出
4. **时序结构**
   乘法、输出变换各打一拍流水
5. **验证交付**
   附 testbench 与 NumPy 参考模型

AI生成的核心代码

```
module wino_1d_f23 #(parameter DW=8) (
  input  clk,
  input  signed [DW-1:0] d0, d1, d2, d3,   // 输入tile
  input  signed [DW+1:0] u0, u1, u2, u3,   // 2·G·g 预变换权重
  output reg signed [2*DW+4:0] y0, y1);

  // 输入变换 Bᵀd: 仅加减法
  wire signed [DW:0] v0 = d0 - d2,  v1 = d1 + d2,
                     v2 = d2 - d1,  v3 = d1 - d3;

  // 4 个元素乘: 4 次乘法 (直接卷积需 6 次)
  reg signed [2*DW+2:0] m0, m1, m2, m3;
  always @(posedge clk) begin
    m0 <= v0 * u0;  m1 <= v1 * u1;
    m2 <= v2 * u2;  m3 <= v3 * u3;
  end

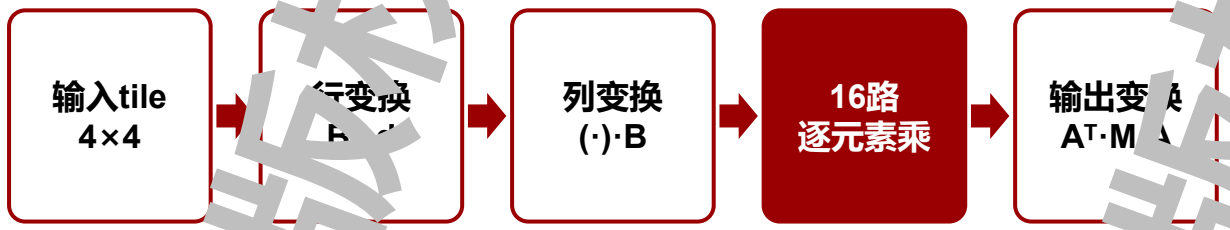
  // 输出变换 AᵀM, 再 >>>1 抵消权重的 ×2
  always @(posedge clk) begin
    y0 <= (m0 + m1 + m2) >>> 1;
    y1 <= (m1 - m2 - m3) >>> 1;
  end
endmodule
```

整数化方案已用 NumPy 验证：1000 组随机 int8 输入与直接卷积结果逐位一致

AI生成2D Winograd F(2×2,3×3) Verilog代码

- 复用1D单元分[illegible]生成 -> 16次乘法替代直接卷积的[illegible]次

$$Y = A^T [(G g G^T) \odot (B^T d B)] A$$

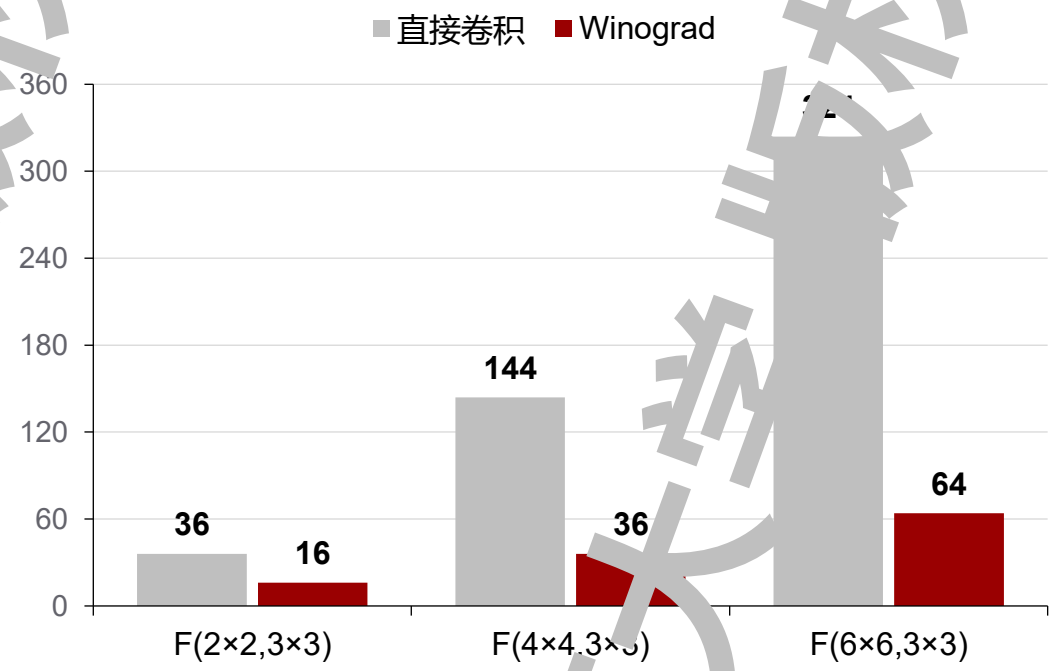


行/列变换由同一1D单元例化；权重离线计算 4·G·g·Gᵀ，输出 >>>2

AI生成[illegible]的常见问题与对策

常见问题	对策（写入Prompt）
行/列[illegible]转置混淆	明确 Bᵀ·d·B，按行后列两级[illegible]
中间位宽不[illegible]导致溢出	要求逐级标注位宽：DW→DW[illegible]2→.
G[illegible]系数引入误差	整数化为 4·G·g·Gᵀ，输出[illegible]
[illegible]长，难以维护	先生成1D单元，再用 g[illegible]e 例化

2D卷积乘法次数对比（3×3卷积核）

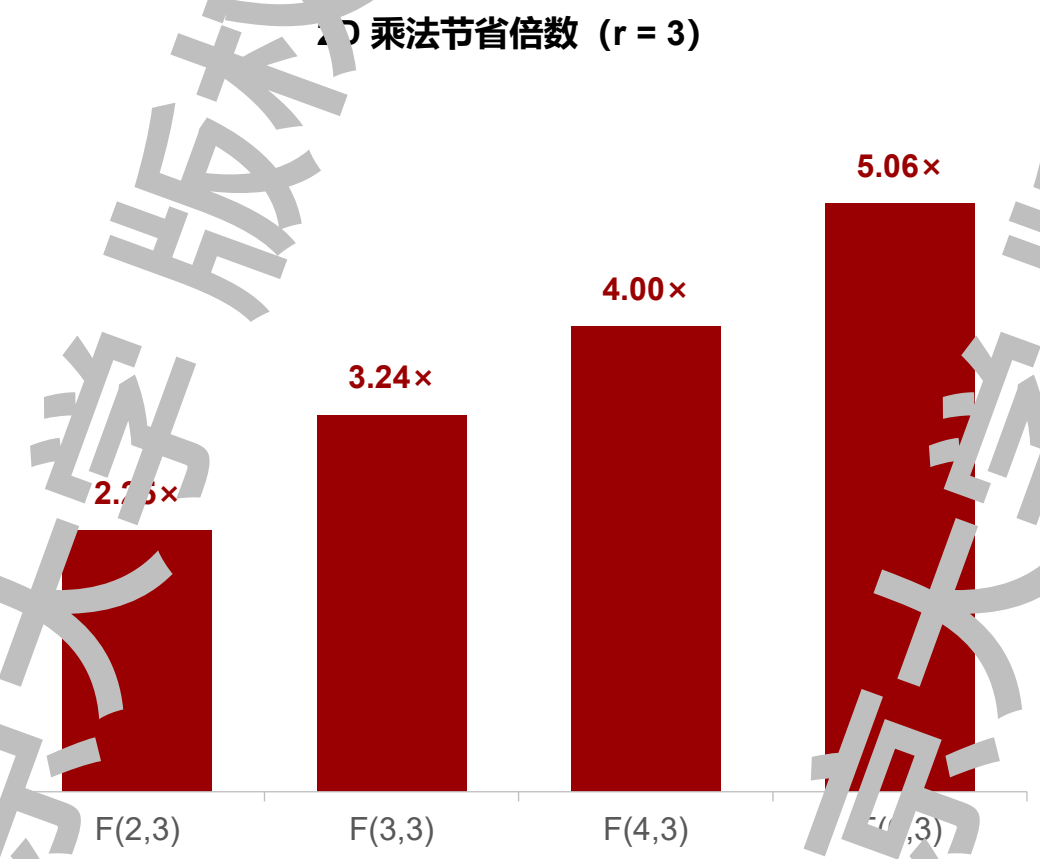


节省 2.25× / 4× / 5.06×

tile 越大乘法越省，但变换系数更复杂、数值误差增大

Winograd卷积 - 变数空间与折中

- tile 越大乘法越省，但变换中的加法数量、系数动态范围与数值误差同步上升



1 **tile 尺寸 m**

决定乘法数 n^2 与变换矩阵规模

2 **插值点选择**

0、±1、±½、±2、∞ 等；决定系数大小与数[illegible]注

3 **定点位宽**

变换后动态范围扩大，需要额外保护位

4 **变换电路结构**

B^T/A^T 为常系数矩阵，可做公共子表[illegible]消除

5 **并行度**

PE 阵列规模需与存储带宽匹配

Winograd卷积 - 联合搜索

- 硬件参数与数值参数联合优化，把精度作为约束放进搜索，而不是事后检查

搜索变量

x = (m, 插值点集合, 位宽 b, 变换加法树, PE 数)

优化目标

最小化 每次推理能耗 与 芯片面积

约束条件

量化后网络精度损失 ≤ ε（如 0.5%）

搜索方法

多保真贝叶斯优化：低保真快速筛选，高保真精确确认

解析模型 + 误差估计

毫秒级 · 数万候选

↓

快速综合 + 定点误差仿真

分钟级 · 数百候选

↓

完整综合 + 端到端精度

小时级 · 数个候选

常见结论：低位宽时，中等 tile + 优化插值点 主 过盲目增大 tile

Winograd卷积 - AI联合搜索

- AI 让组合逻辑设计从“挑选经典结构”走向“搜索最优结构”

	Kogge-Stone 加法器	Winograd 卷积
关键参数	稀疏度、基数、扇出/布线、节点结构	tile 尺寸、插值点、位宽、变换加法树
空间形态	图结构、离散、规模极大	离散/连续混合、规模中等
适合方法	RL 图编辑、进化算法	多保真贝叶斯优化
优化目标	延迟–面积 Pareto 前沿	能耗/面积 + 精度约束

1 参数化生成器是前提

必须能由参数自动生成 RTL，搜索才能闭环

2 评估保真度要分层

便宜模型负责广度，精确综合负责深度

3 正确性由构造保证

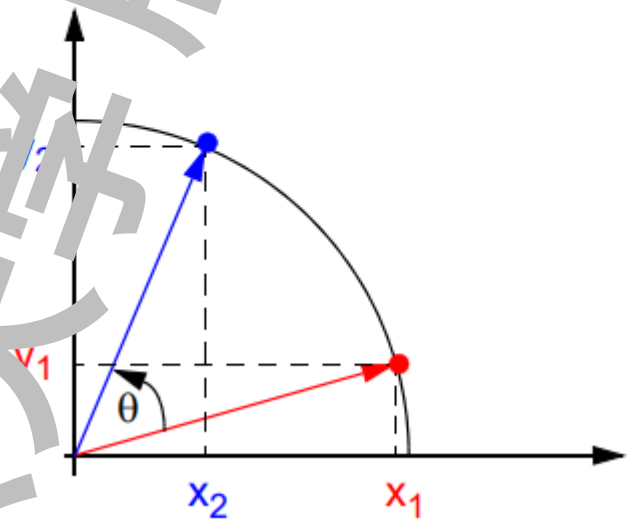
合法化步骤确保候选在功能上都正确

AI辅助CORDIC核设计

- CORDIC可以用来实现多种复杂非线性函数

$$x_2 = x_1\cos\theta - y_1\sin\theta$$
$$y_2 = x_1\sin\theta + y_1\cos\theta$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$



$$x_2 = x_1\cos\theta - y_1\sin\theta = \cos\theta(x_1 - y_1\tan\theta)$$
$$y_2 = x_1\sin\theta + y_1\cos\theta = \cos\theta(y_1 + x_1\tan\theta)$$

$$\hat{x}_2 = \cos\theta(x_1 - y_1\tan\theta) = x_1 - y_1\tan\theta$$
$$\hat{y}_2 = \cos\theta(y_1 + x_1\tan\theta) = y_1 + x_1\tan\theta$$

当θ足够小接近于0时，先忽略cosθ（后面会scale回来）

伪旋转（pseudo rotations）

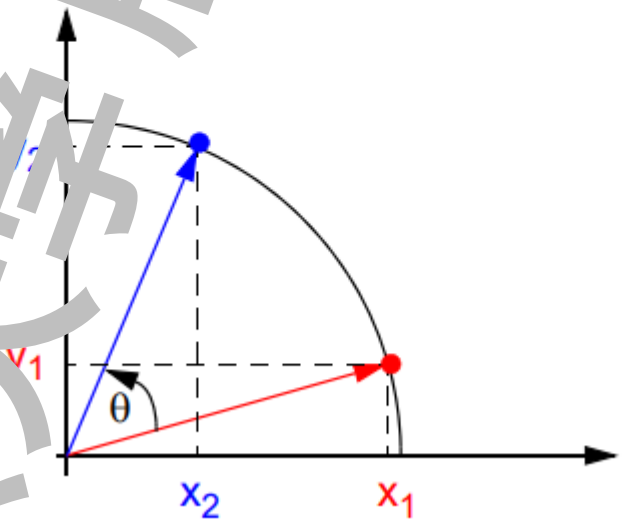
AI辅助CORDIC硬件设计

- CORDIC可以用来实现多种复杂非线性函数

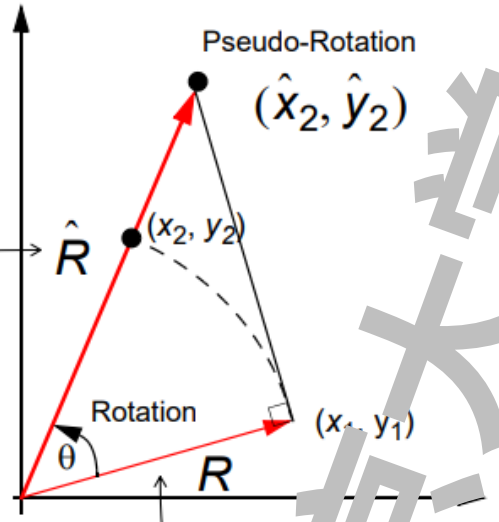
$$x_2 = x_1\cos\theta - y_1\sin\theta$$
$$y_2 = x_1\sin\theta + y_1\cos\theta$$

伪旋转（pseudo rotations）

$$\hat{x}_2 = \cos\theta(x_1 - y_1\tan\theta) = x_1 - y_1\tan\theta$$
$$\hat{y}_2 = \cos\theta(y_1 + x_1\tan\theta) = y_1 + x_1\tan\theta$$



$$\hat{R} = \frac{R}{\cos\theta} = \frac{\sqrt{x_1^2 + y_1^2}}{\cos\theta}$$



$$R = \sqrt{x_1^2 + y_1^2} = \sqrt{x_2^2 + y_2^2}$$

思想包袱 兼容并包

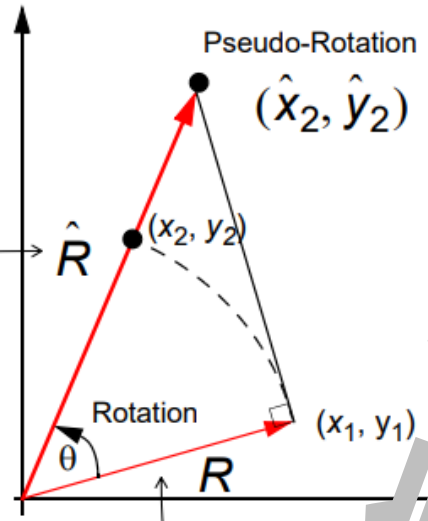
AI辅助CORDIC[illegible]设计

- CORDIC可以用来实现多种复杂非线性函数

伪旋转 (pseudo rotations)

$$\hat{x}_2 = \cos\theta(x_1 - y_1\tan\theta) = x_1 - y_1\tan\theta$$

$$\hat{y}_2 = \cos\theta(y_1 + x_1\tan\theta) = y_1 + x_1\tan\theta$$



$$\hat{R} = \frac{R}{\cos\theta} = \frac{\sqrt{x_1^2 + y_1^2}}{\cos\theta}$$

$$R = \sqrt{x_1^2 + y_1^2} = \sqrt{x_2^2 + y_2^2}$$

选择旋转角度 $\tan\theta^i = 2^{-i}$

$$\hat{x}_2 = x_1 - y_1\tan\theta = x_1 - y_1 2^{-i}$$

$$\hat{y}_2 = y_1 + x_1\tan\theta = y_1 + x_1 2^{-i}$$

i	θ^i (Degrees)	$\tan\theta^i = 2^{-i}$
0	45.0	1
1	26.555051177...	0.5
2	14.036243467...	0.25
3	7.125016348...	0.125
4	3.576334374...	0.0625

北京大学版权所有

AI辅助CORDIC模块设计

- CORDIC可以用来实现多种复杂非线性函数

1st iteration: rotate by 45°; 2nd iteration: rotate by 26.6°; 3rd iteration: rotate by 14°

i	tan	Angle, θ	cosθ
1	1	45.0000000000	0.7071067812
2	0.5	26.5650511771	0.8944271910
3	0.25	14.0362434679	0.9701425001
4	0.125	7.1250163489	0.9922778767
5	0.0625	3.5763343750	0.9980525784
6	0.03125	1.7899106082	0.9995117076
7	0.015625	0.8951737102	0.9998779192
8	0.0078125	0.4476141709	0.9999694842
9	0.00390625	0.2238105004	0.9999923712
10	0.001953125	0.1119056771	0.9999980928
11	0.000976563	0.0559528919	0.9999995232
12	0.000488281	0.0279764526	0.9999998808
13	0.000244141	0.0139882271	0.9999999702

cos 45 x cos 26.5 x cos 14.03 x cos 7.12 ... x cos 0.0139 = 0.607252941

13次伪旋转后，需要乘以

1/0.607252941 =

1.6467602

思想自由 兼容并包

北京大学版权所有

AI辅助CORDIC硬件设计

- CORDIC可以用来实现多种复杂非线性函数

$$\hat{x}_2 = x_1 - y_1 \tan\theta = x_1 - y_1 2^{-i}$$

$$\hat{y}_2 = y_1 + x_1 \tan\theta = y_1 + x_1 2^{-i}$$

$$x^{(i+1)} = x^{(i)} - d_i(2^{-i}y^{(i)})$$

$$y^{(i+1)} = y^{(i)} + d_i(2^{-i}x^{(i)})$$

$$z^{(i+1)} = z^{(i)} - d_i\theta^{(i)} \quad \text{(Angle Accumulator)}$$

where d_i = +/- 1

The symbol d_i is a decision operator and is used to decide which direction to rotate.

2 shifts

1 table lookup ($\theta^{(i)}$ values)

3 additions

AI辅助CORDIC[illegible]设计

- CORDIC可以用来实现多种复杂非线性函数

$$x^{(i+1)} = x^{(i)} - d_i(2^{-i}y^{(i)})$$

$$y^{(i+1)} = y^{(i)} + d_i(2^{-i}x^{(i)})$$

$$z^{(i+1)} = z^{(i)} - d_i\theta^{(i)}$$

where d_i = +/- 1

2 shifts

1 table lookup ($\theta^{(i)}$ values)

2 additions

Scaling Factor

$$K_n = \prod_n 1/(\cos\theta^{(i)}) = \prod_n (\sqrt{1+2^{(-2i)}})$$

$$K_n = \prod_n 1/(\cos\theta^{(i)}) = \prod_n \left(\sqrt{1+\tan^2\theta^{(i)}}\right) = \prod_n \left(\sqrt{1+2^{(-2i)}}\right)$$

$$K_n \to 1.6476 \text{ as } n \to$$

$$1/K_n \to 0.6073 \text{ as } n \to \infty$$

n = number of iterations

北京大学版权所有

AI辅助CORDIC模块设计

- CORDIC可以用来实现多种复杂非线性函数

$$x^{(i+1)} = x^{(i)} - d_i(2^{-i}y^{(i)})$$

$$y^{(i+1)} = y^{(i)} + d_i(2^{-i}x^{(i)})$$

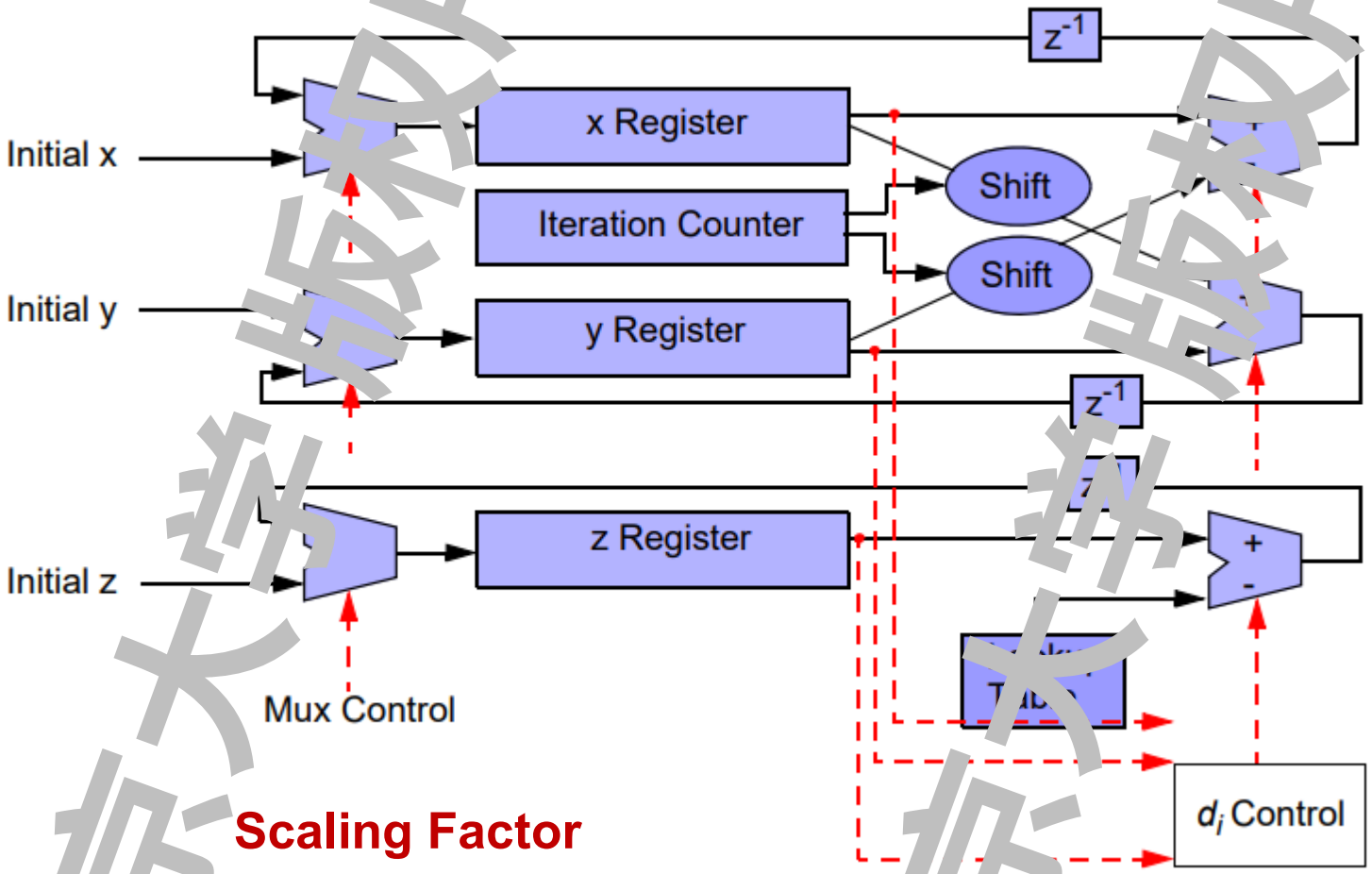
$$z^{(i+1)} = z^{(i)} - d_i\theta^{(i)}$$

where d_i = +/- 1

2 shifts

1 table lookup ($\theta^{(i)}$ values)

2 additions



Scaling Factor

$$A_n = \prod_n 1/(\cos\theta^{(i)}) = \prod_n (\sqrt{1+2^{(-2i)}})$$

AI辅助CORDIC模块设计

- 手工编写RTL + 经验调参 -> AI生成Verilog + AI驱动设计空间探索

CORDIC：仅用移位与加减实现三角/双曲函数

迭代公式（旋转模式）

$$x_{i+1} = x_i - d_i \cdot y_i \cdot 2^{-i}$$

$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i}$$

$$z_{i+1} = z_i - d_i \cdot \arctan(2^{-i})$$

无需乘法器，增益 K ≈ 0.6073 需在初值中补偿

三类典型硬件架构

迭代型
1套加法器
面积最小

部分展开
U级复用
面积/吞吐折中

全流水
N级流水
每周期1个结果

AI辅助的两阶段设计流程

阶段一 AI生成CORDIC Verilog

结构化Prompt → 大模型生成参数化RTL

仿真比对 → 错误反馈 → 自动修正

阶段二 AI优化参数设计空间探索

位宽 / 迭代次数 / 展开因子 / 补偿方式

精度–面积–频率多目标 → Pareto最优配置

RTL模板复用：阶段一产出即阶段二的参数化生成器

AI辅助CORDIC模块设计

- 结构化Prompt > 参数化、可综合的全流水CORDIC RTL

Prompt 设计要点

1. **功能规格**
   旋转模式，输入角θ，输出 cosθ / sinθ
2. **数据格式**
   有符号定点 Q2.14，W=16，N=16
3. **结构约束**
   全流水，generate 生成各级，参数化
4. **实现细节**
   使用 >>> 算术右移；x0 预置增益 K
5. **交付物**
   RTL + testbench + Python 参考模型

AI生成的核心代码（节选）

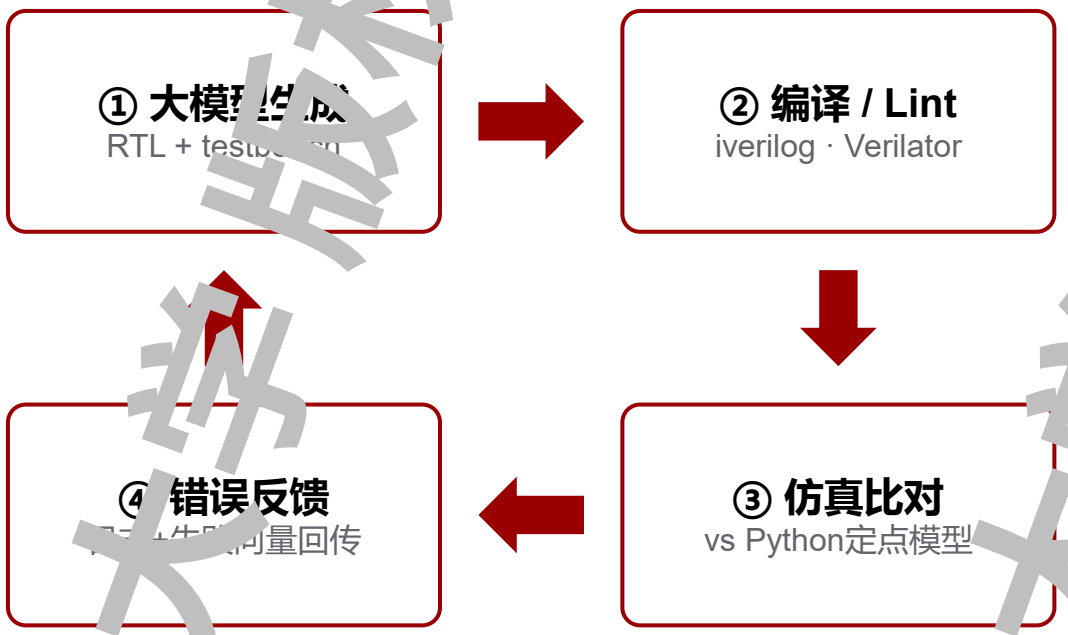
```
module cordic #(parameter W=16, N=16) (
  input  clk
  input  signed [W-1:0] theta,          // Q2.(W-2)
  output signed [W-1:0] cos_o, sin_o);
  reg  signed [W-1:0] x[0:N], y[0:N], z[0:N];
  wire signed [W-1:0] atan[0:N-1];      // 常数表由脚本生成
  always @(posedge clk) begin           // 预置增益 K
    x[0] <= K_FIT; y[0] <= 0; z[0] <= theta;
  end
  genvar i;
  generate for (i=0; i<N; i=i+1) begin: stg
    always @(posedge clk) begin
      if (z[i][W-1]) begin                // z<0
        x[i+1] <= x[i] + (y[i] >>> i);
        y[i+1] <= y[i] - (x[i] >>> i);
        z[i+1] <= z[i] + atan[i];
      end else begin
        x[i+1] <= x[i] - (y[i] >>> i);
        y[i+1] <= y[i] + (x[i] >>> i);
        z[i+1] <= z[i] - atan[i];
      end
    end
  end endgenerate
  assign cos_o = x[N];  assign sin_o = y[N];
endmodule
```

参数 W、N 暴露为 parameter，可直接作为阶段二DSE的搜索变量

AI辅助CORDIC模块设计

- AI生成的RTL不能直接信任 -> “仿真–比对–反馈”闭环自动修正

自动化验证闭环



通过标准：随机+边界角度向量全部通过，误差 ≤ 定点量化界

通常需多轮迭代；每轮将失败用例精确回传给模型

AI生成CORDIC代码的常见错误

常见错误	现象	修正提示
用 >> 代替 >>>	负数结果错误	有符号数声明 signed，用算术右移
atan 表定标错误	角度不收敛	给定 Q 格式，由脚本生成常数
迭代增益补偿	幅度放大 1.647 倍	x0 预置 K ≈ 0.6073
未处理象限	|θ|>π/2 时出错	增加 CORDIC 预处理级
流水级未对齐	输出错拍	同步输出 valid 延迟链

AI辅助CORDIC参数设计

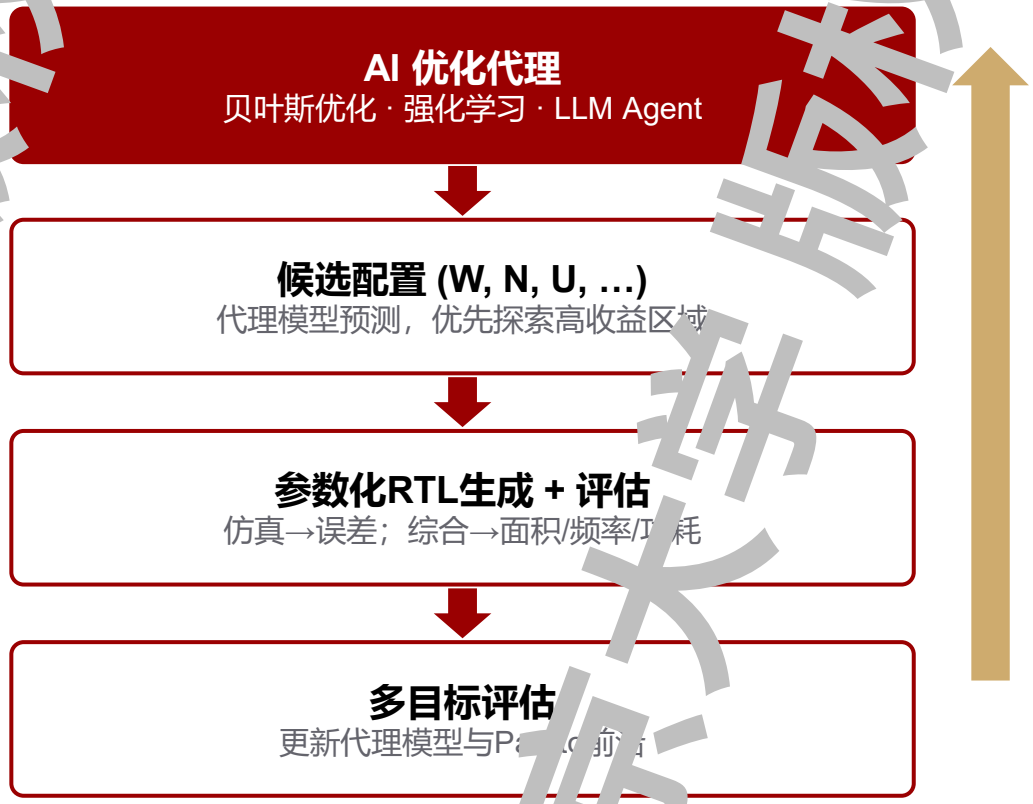
- 参数组合爆炸、目标相互冲突 -> AI代理自动搜索Pareto最优配置

CORDIC 参数化设计空间

设计参数	取值范围	主要影响
数据位宽 W	10 – 32 bit	精度、面积
迭代次数 N	6 – 24	精度、延迟
架构 / 展开 U	迭代 · 部分展开 · 全流水	吞吐、面积
增益补偿方式	初值预置 · 后乘法 · 无	精度、DSP数
舍入方式	截断 · 就近舍入	误差累积

优化目标： min 面积 / 功耗
约束： 最大误差 ≤ 2^{-12}，fmax ≥ 目标频率

AI驱动的DSE闭环

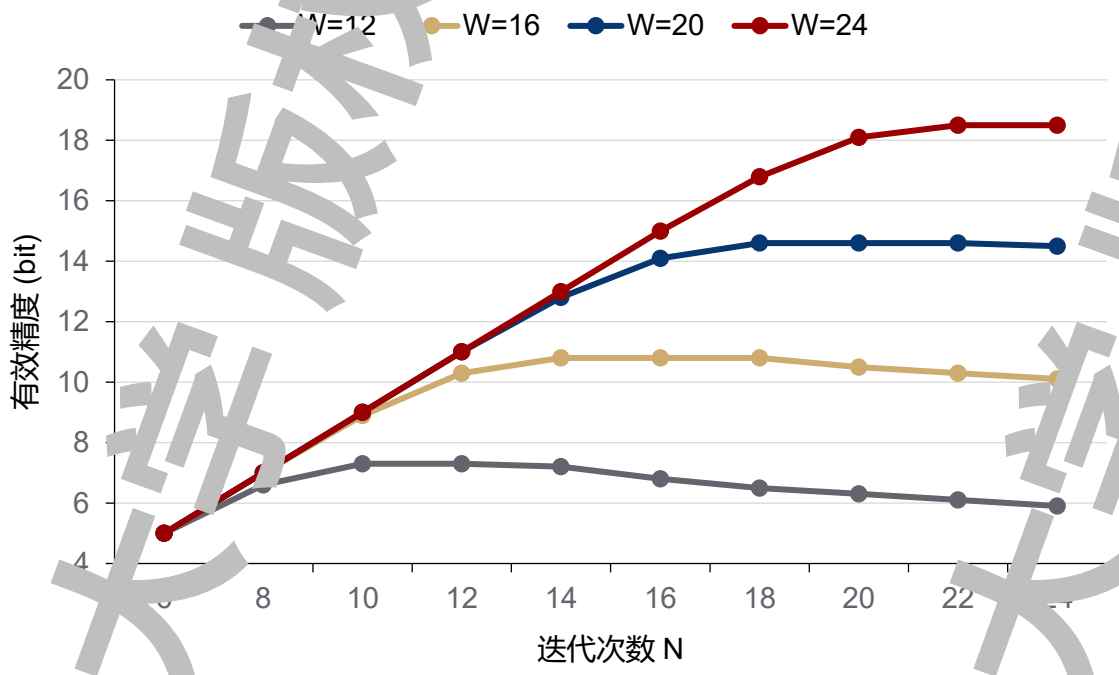


用少量昂贵的综合运行逼近全空间最优解

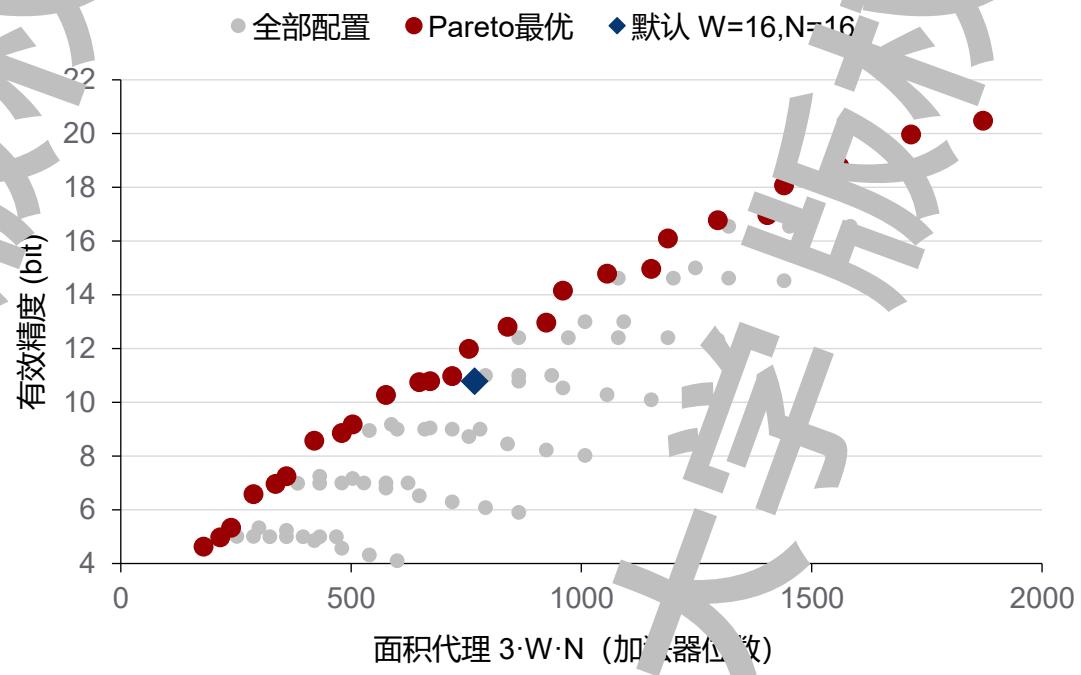
AI辅助CORDIC[illegible]设计

- 精度随迭代次数[illegible]和 -> 默认配置并非最优，AI可[illegible]Pareto前沿

有效精度 vs 迭代次数



精度–面积 Pareto 前沿



N[illegible]-4 后精度饱和甚至回落：多余迭代只增加面积

W=18, N=14：面积 756 < 768，精度 1[illegible] bit，优于默认配置

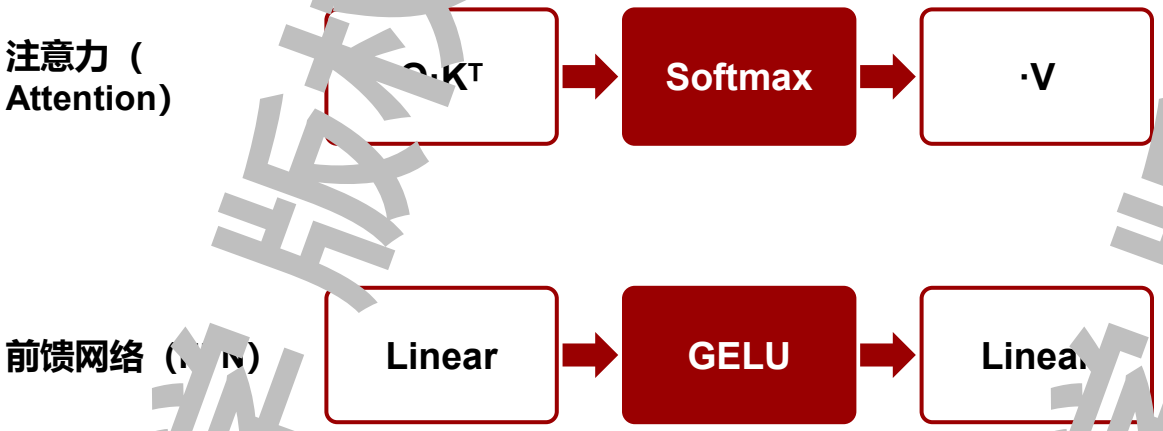
定点CORDIC位精确模型（2000[illegible]随机角度）；有效精度 = −log2(sin/cos最大绝对误差);

CORDIC[illegible]实现精准的RMSNorm等AI芯片算子

背景：Transformer中的非线性硬件瓶颈

- 算力集中于MAC阵列 -> 非线性单元成为新的延迟瓶颈

Transformer 中的两个关键非线性算子



$$\mathrm{Softmax}(x_i) = e^{x_i - \max} / \Sigma_j e^{x_j - \max}$$

$$\mathrm{GELU}(x) = x \cdot \Phi(x) = \tfrac{1}{2}\, x\, [1 + \mathrm{erf}(x/\sqrt{2})]$$

硬件实现的三大难点

1. **超越函数**
   exp、erf 无法用加法/乘法直接精确计算，需要近似
2. **全局数据依赖**
   Softmax 的 max 与 Σ 需遍历整行，难以流水
3. **除法与动态范围**
   归一化需要除法；指数运算输出范围极大

长序列下 Softmax 计算量随序列长度平方增长

Softmax硬件单元：当前主流设计思路

- 减最大值保证数值稳定 -> 以2为底的指数 + 移位查表 + 在线归一化

Softmax 硬件数据通路



Base-2 变换

$x' = \log_2 e$

指数拆成整数 I 与小数 f：2^I 是移位，2^f 在 [1,2) 用小表+线性插值，完全避免乘方运算

在线归一化（Online Softmax）

$m_n = \max(m_{n-1}, x_n)$

$d_n = d_{n-1} \cdot 2^{(m_{n-1}-m_n)} + 2^{(x_n-m_n)}$：一次扫描即得到最大值和分母，Softmax、FlashAttention 采用此思路

除法消除

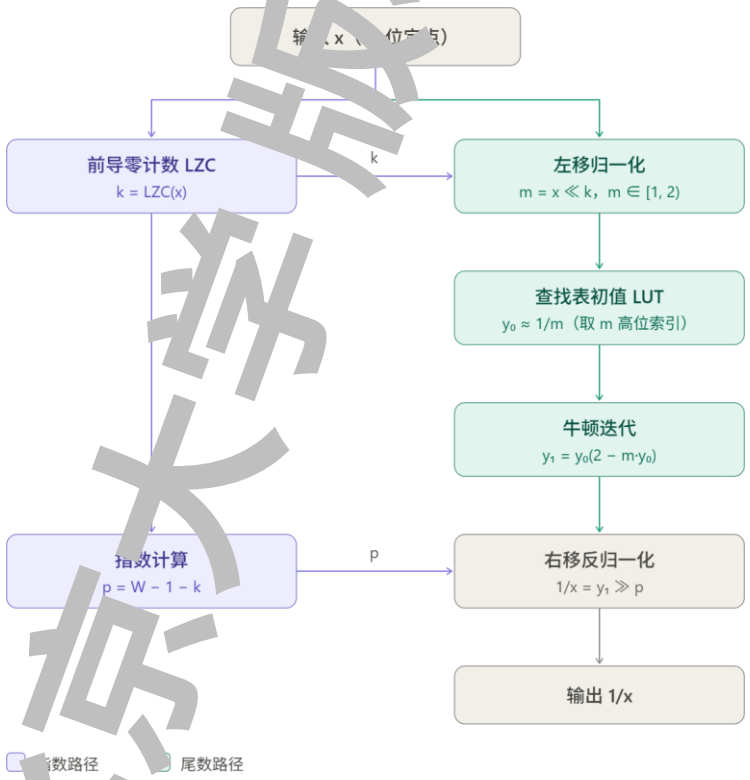
$y_i = 2^{z_i - \log_2 \Sigma}$

倒数单元（LZC + 查表 + 牛顿迭代）替代除法器；或在对数域把除法变为减法，输出仍是一次 $2^{\cdot}$

Softmax硬件单元：当前主流设计思路

- 减最大值保证数值稳定 -> 以2为底的指数 + 移位查表 + 在线归一化

先用前导零计数器（Leading Zero Counter, LZC）把输入归一化，再对归一化后的尾数求倒数，最后移位还原。它不需要除法器，在 Softmax 分母求倒、LayerNorm 等场景中很常见。



参数	取值
输入位宽 W	8 位无符号定点数
输入 x	12（二进制 0000 1100）
查找表 (LUT)	以 m 小数部分高 3 位为索引，共 8 项；每项存区间中点的倒数
牛顿迭代	$y_{i+1} = y_i(2 - m \cdot y_i)$，演示 1～2 次
输出格式	Q0.16 定点小数（16 位小数位）

将正数 x 写成“尾数 × 2 的幂”的形式，求倒数即可拆成尾数部分与移位两部分：

$$x = m \cdot 2^p, m \in [1, 2) \Rightarrow \frac{1}{x} = \frac{1}{m} \cdot 2^{-p}$$

其中 p 由前导零计数器（LZC）得到，2^{-p} 仅需移位。**m 的取值范围被压缩到 (0.5, 1)，可用小查找表加牛顿迭代高精度逼近，无需除法器。**

步骤	运算	数值 / 二进制	说明
①	前导零计数	0000 1100 → k = 4	最高位 1 之前有 4 个 0
②	指数计算	p = 8 − 1 − 4 = 3	1 的最高有效位位于第 3 位
③	左移归一化	x ≪ 4 = 1100 0000	视为 1.1000000_2，即 m = 1.5
④	查找表初值	索引 100_2 = 4 → y_0 = 0.64	区间 [1.5, 1.625) 中点 1.5625 的倒数
⑤	第 1 次牛顿迭代	$y_1 = 0.64 \times (2 - 1.5 \times 0.64) = 0.665$	误差由 2.7×10^{-2} 降至 1.1×10^{-3}
⑥	第 2 次牛顿迭代	$y_2 = 0.6656 \times (2 - 1.5 \times 0.6656) \approx 0.666665$	误差降至 1.7×10^{-6}
⑦	右移反归一化	$y_2 \gg 3 = 0.0833331$	除以 $2^3 = 8$，得到 1/x
⑧	定点输出	$0.0833331 \times 2^{16} \approx 5461 =$ 0x1555	Q0.16 格式，与精确值一致

Softmax硬件单元：典型实现方案对比

- 精度、面积、[illegible]中 -> 低位宽推理多用Base-2 + 查表插值

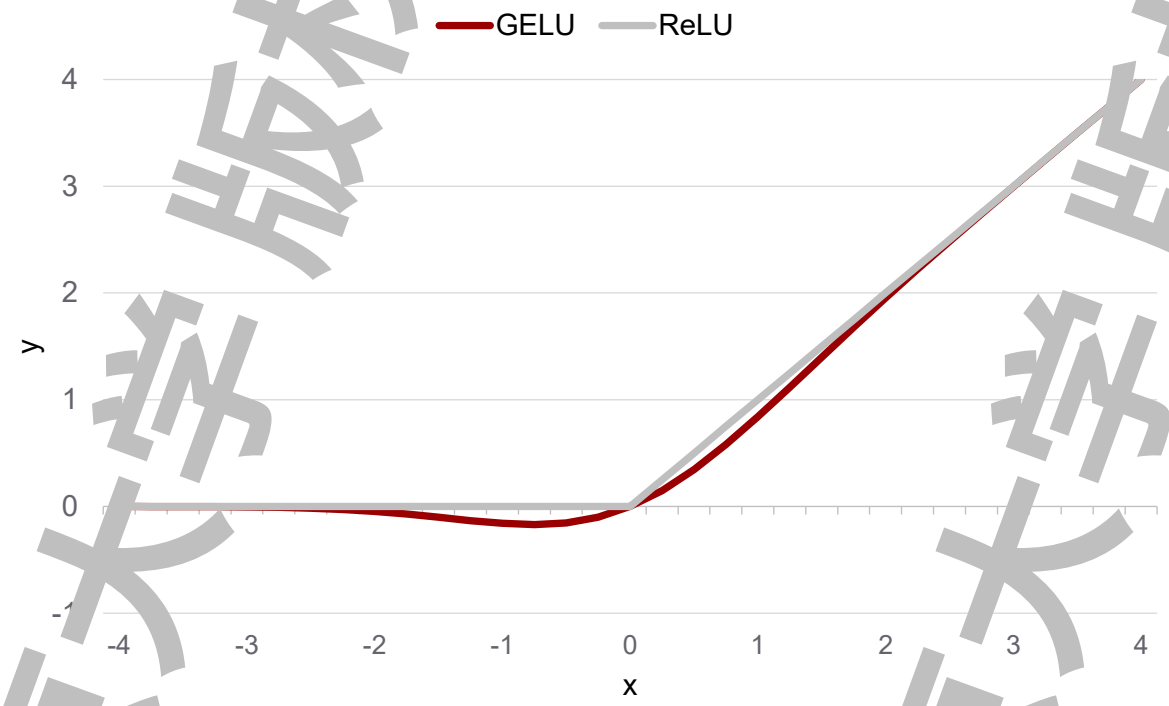
方案	指数实现	归一化	优点	代价
浮点精确实现	FP32 exp 单元	浮点除法器	精度最高	面积、功耗最大
直接查表	LUT 存 e^x	除法器	结构最简单	高精度需大表
Base-2 + 查表插值	移位 + 小表 + 插值	倒数 × 乘法	面积小、精度可调	需插值乘法器
整数多项式 [illegible]T)	二次多项式 + 移位	整数除法	纯整数运算	多项式系数需[illegible]化
在线归一化（Softermax）	Base-2 + 移位	在线累加分母	单次扫描、易流水	需要重缩放逻辑
Log 域[illegible]化	2^x 单元复用	$\log_2 x$ [illegible]	消除除法器	需额外 log[illegible]化

[illegible]键：指数小表的大小、插值与否、归一化方式共[illegible]定精度与面积，天然适合做设计空间探索

GELU硬件单元：当前主流设计思路

- 利用对称性与饱和区缩小近似区间 -> 分段线性 / 多项式近似

GELU 与 ReLU 曲线



x < −4 时 GELU ≈ 0，x > 4 时 GELU ≈ x：只需在 [−4, 4] 内近似

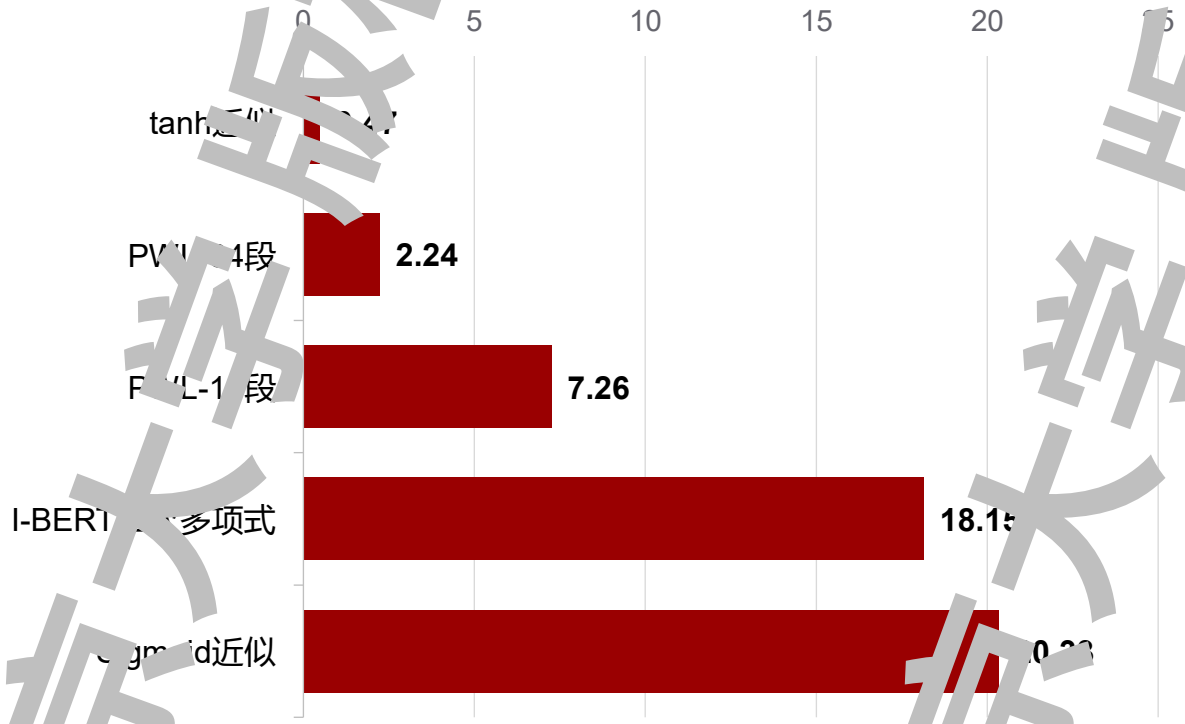
关键硬件技巧

1. **对称性**
   GELU(x) = x + GELU(−x)，只存负半轴，查找表减半
2. **饱和区直通**
   |x| > 4 用比较器直接输出 0 或 x，不进近似单元
3. **分段线性（PWL）**
   每段 y = s·x + c，1 次乘法 + 1 次加法，系数存 ROM
4. **非均匀分段**
   曲率大的区域（x ≈ −1）分段更细，同精度下段数更少
5. **多项式 / 函数替换**
   erf 用二次多项式（I-BERT），或 x·σ(1.702x)

GELU硬件单元：近似方案精度与代价对比

- 没有“万能”方案 -> 需按精度要求与硬件预算联合选择

最大绝对误差（[-8, 8] 区间，×10⁻³）



硬件代价对比

方案	主要运算	硬件代价
tanh 近似	3次乘法 + tanh	高：需 tanh 单元
Sigmoid 近似	1次乘法 + exp + 除法	中高：复用 Softmax 的 exp
I-BERT 多项式	2次乘法 + 移位	中：纯整数
PWL 分段线性	1次乘法 + 查表	低：段数可调

结论： PWL 精度随段数连续可调，仅需1次乘法，是 INT8/FP8 加速器最常用的 GELU 方案

基于AI的Verilog生成：方法与流程

- 先生成位精确参考模型 -> 多智能体“生成–验证–修复”闭环

多智能体生成流程

① 规格智能体
数学公式 → 定点格式（Q格式、位宽）→ 设计规格文档

② 参考模型智能体
生成 Python 位精确模型 + 常数表生成脚本

③ RTL 生成智能体
按规格生成参数化 Verilog，常数表由脚本导出

④ 验证与修复智能体
仿真 + 边界激励逐位比对，失败用例回传修复

非线性单元特有的AI生成难点

难点	应对策略
常数表数值易算错	禁止模型“心算”，一律由脚本生成
Q 格式在各级不一致	规格中逐级标注位宽与小数位
边界条件遗漏	显式列出 x=0、饱和点、负输入
移位量越界	要求对移位量做饱和处理
有符号 / 无符号混用	统一声明 signed，使用 >>>

核心原则：让 AI 负责“结构与连线”，让脚本负责“数值”

AI生成Softmax核心单元：2^x指数单元

- 整数部分移位 + 小数部分查表插值 -> 无乘方运算的指数单元

Prompt 关键约束

1. **输入格式**
   z = (x − max)·$\log_2 e$ ≤ 0，Q8.8
2. **分解方式**
   z = I + f，I 用右移，2^f 查表
3. **查表插值**
   [illegible] 2^{K+1} 项表，低位线性插值
4. **边界处理**
   移位量 ≥ 输出位宽时直接输出 0
5. **参数化**
   IW、FB、K、OW 均为 parameter

AI 生成的核心代码（节选）

```
module exp2_[illegible] #(parameter IW=16, FB=8, K=4, OW=16) (
  input [illegible]lk,
  input [illegible] signed [IW-1:0] z,          // z <= 0, Q(IW-FB).FB
  output [illegible]g [OW:0] e);                // 2^z, Q1.(OW-1)
  // z = I + f, I 为负整数, f ∈ [0,1)
  wire [illegible] [IW-FB-1:0] I = z[IW-1:FB];
  wire [K-1:0]     idx = z[FB-1 -: K];    // 高 K 位查表
  wire [FB-K-1:0] r   = z[FB-K-1:0];      // 低位插值
  // 2^(i/2^K) 常数表: 由 Python 脚本生成的 case 函数
  [illegible] [OW:0] y0 = exp2_lut(idx);
  [illegible] [OW:0] y1 = exp2_lut(idx + 1);
  // 线性插值: y = y0 + (y1 - y0)·r
  wire [OW:0] y = y0 + (((y1 - y0) * r) >> (FB-K));
  // 2^I: 按 -I 右移, 移位量越界则输出 0
  wire [IW-FB-1:0] sh = -I;
  always @(posedge clk)
    e <= (sh >= OW) ? 0 : (y >> sh);
endmodule
```

同一 2^x 单元可复用于 Softmax、Sigmoid 近似的 GELU 以及 log 域归一化

AI生成GELU单元：对称分段线性实现

- 对称性 + 饱和区直通 + 均匀PWL -> 1次乘法完成GELU

AI 生成的核心代码（经验证闭环修正后）

```
module gelu_pwl #(parameter W=16, FB=8, NL=4, QB=12) (
  input  clk,
  input  signed [W-1:0] x,              // Q(W-FB).FB
  output reg signed [W-1:0] y);
  // 对称性：GELU(x) = x + GELU(-x)，只近似负半轴
  wire signed [W-1:0] t = x[W-1] ? x : -x;    // t = -|x|
  wire sat = (t < -(4 <<< FB));               // 饱和区 → 0
  // [-4, 0] 均匀分 2^NL 段
  wire [NL:0] kr = (t + (4 <<< FB)) >> (FB + 2 - NL);
  wire [NL-1:0] k = kr[NL] ? {NL{1'b1}} : kr[NL-1:0];  // 修正
  wire signed [QB+1:0] s, c;                  // 斜率/截距 ROM
  pwl_lut #(.NL(NL), .QB(QB)) u_rom (.k(k), .s(s), .c(c));
  wire signed [W+QB+1:0] gf = s * t + (c <<< FB);
  wire signed [W-1:0] g = sat ? 0 : (gf >>> QB);
  always @(posedge clk)
    y <= x[W-1] ? g : x + g;
endmodule
```

系数 LUT 由脚本按分段端点插值生成；W、NL、QB 作为下一步设计空间探索的搜索变量

验证闭环发现的典型问题

x = 0 时段索引溢出

t + 4 恰好等于 2^(FB+2)，右移后为 2^NL，截断成 0，错误地落入第 0 段

修正：多保留 1 位并饱和到最后一段

启示

这类边界错误在随机激励下很难触发，必须在 Prompt 中要求 AI 生成边界测试向量

基于AI的设计空间探索：搜索空间与方法

- 精度与面积相互冲突 -> AI代理自动搜索Pareto最优配置

参数化设计空间

单元	设计参数	取值范围
Softmax	输入小数位宽 FB	4 – 12 bit
	查表地址位 K	2 – 10（4 – 1024 项）
	插值方式	无 / 线性插值
	归一化方式	倒数×乘法 / log域
GELU	分段数 2^NL	4 – 128 段
	系数小数位宽 QB	6 – 12 bit
	分段方式 / 饱和点	均匀 · 非均匀 / … ±6

约束： 输出精度损失 ≤ 阈值；**目标：** 最小化面积 / 功耗

AI 驱动的探索闭环

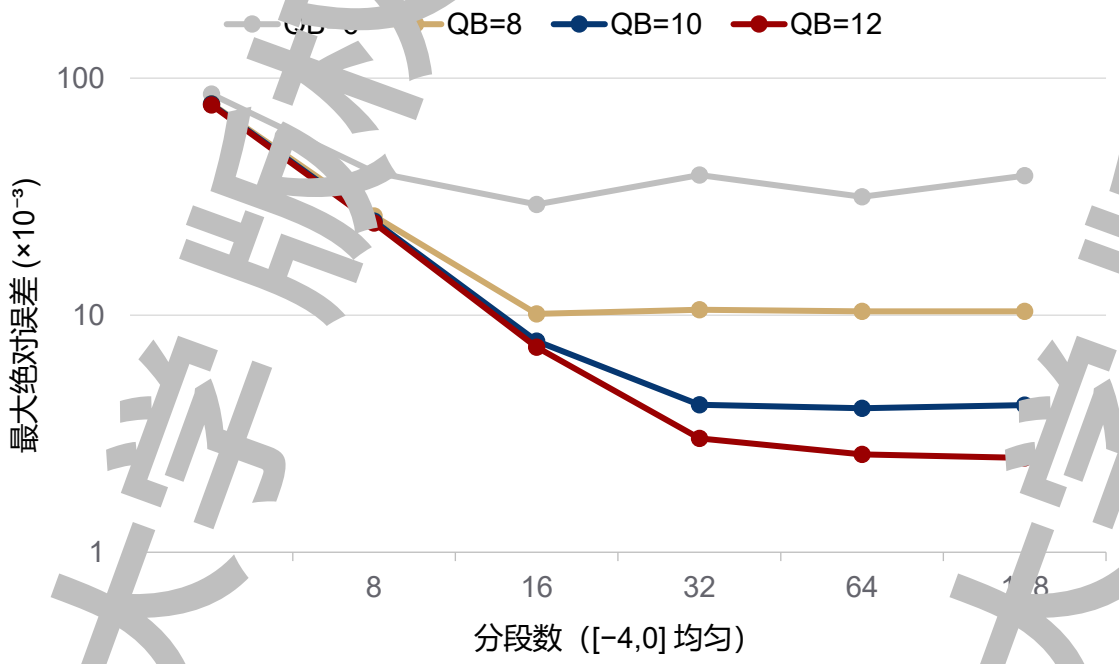


两级评估：便宜的误差模型筛选，昂贵的综合只跑少数点

设计空间探索结果：精度–资源权衡规律

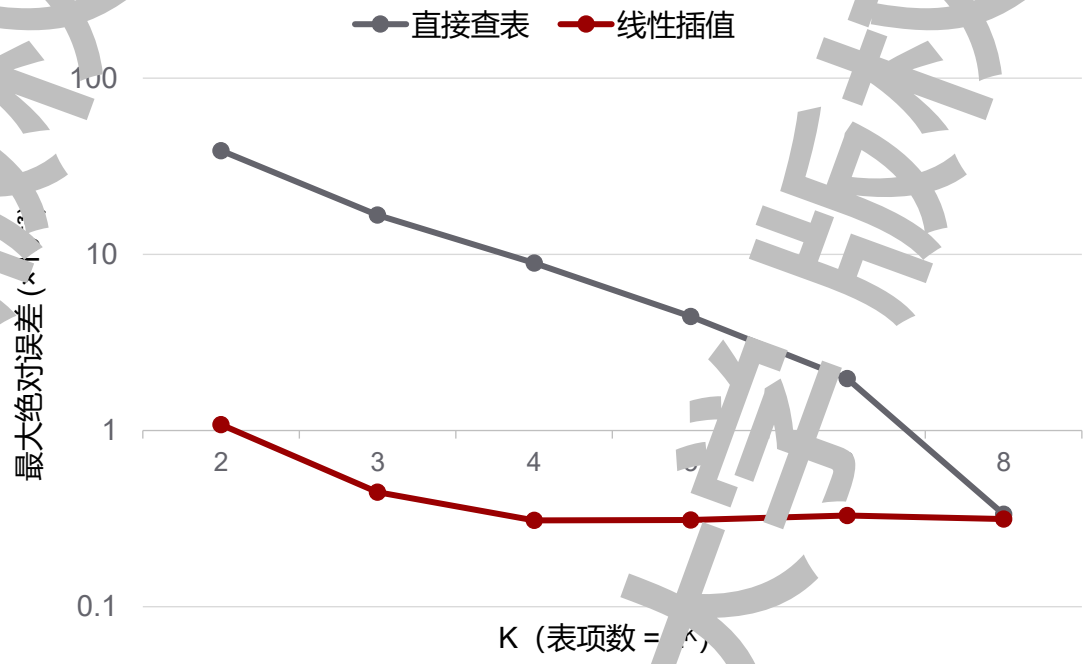
- 存在明显的“收益拐点” -> AI可据此剪枝，避免无效的大表与宽系数

GELU：PWL 误差 vs 分段数（×10⁻³）



系数位宽不足时，增加段数无效：QB=8 在 16 段后误差停在约 1.[illegible]×10⁻²

Softmax：端到端误差 vs 指数表地址位 K



16 项表 + 线性插值 ≈ 256 项直接查表的精度（误差均约 3×10⁻⁴）

GELU 输入 Q.8，[−8,8] 区间；Softmax 为 300 组 64 维 N(0,3²) 随机向量，输入 Q[illegible]，除法精确

总结与展望

- AI 正在从“写代码”走向“做设计”：生成、验证、优化一体化

硬件设计思路

- Softmax：Base-2 + 移位查表插值 + 在线归一化
- GELU：对称性 + 饱和直通 + 分段线性
- 核心：用加法、移位、小表替代超越函数与除法

AI 生成 Verilog

- 多智能体：规格 → 参考模型 → RTL → 验证修复
- 数值交给脚本，结构交给 AI
- 边界测试向量是发现 AI 错误的关键

AI 设计空间探索

- 参数化 RTL 天然构成设计空间
- 两级评估：误差模型筛选 + 小量综合
- 利用收益拐点剪枝，逼近 Pareto 前沿

展望

- 从单元级走向算子融合级：Attention 整体（QK^T + Softmax + ·V）的 AI 协同设计
- 以模型精度（而非函数误差）直接作为 DSE 目标，实现算法-硬件联合优化

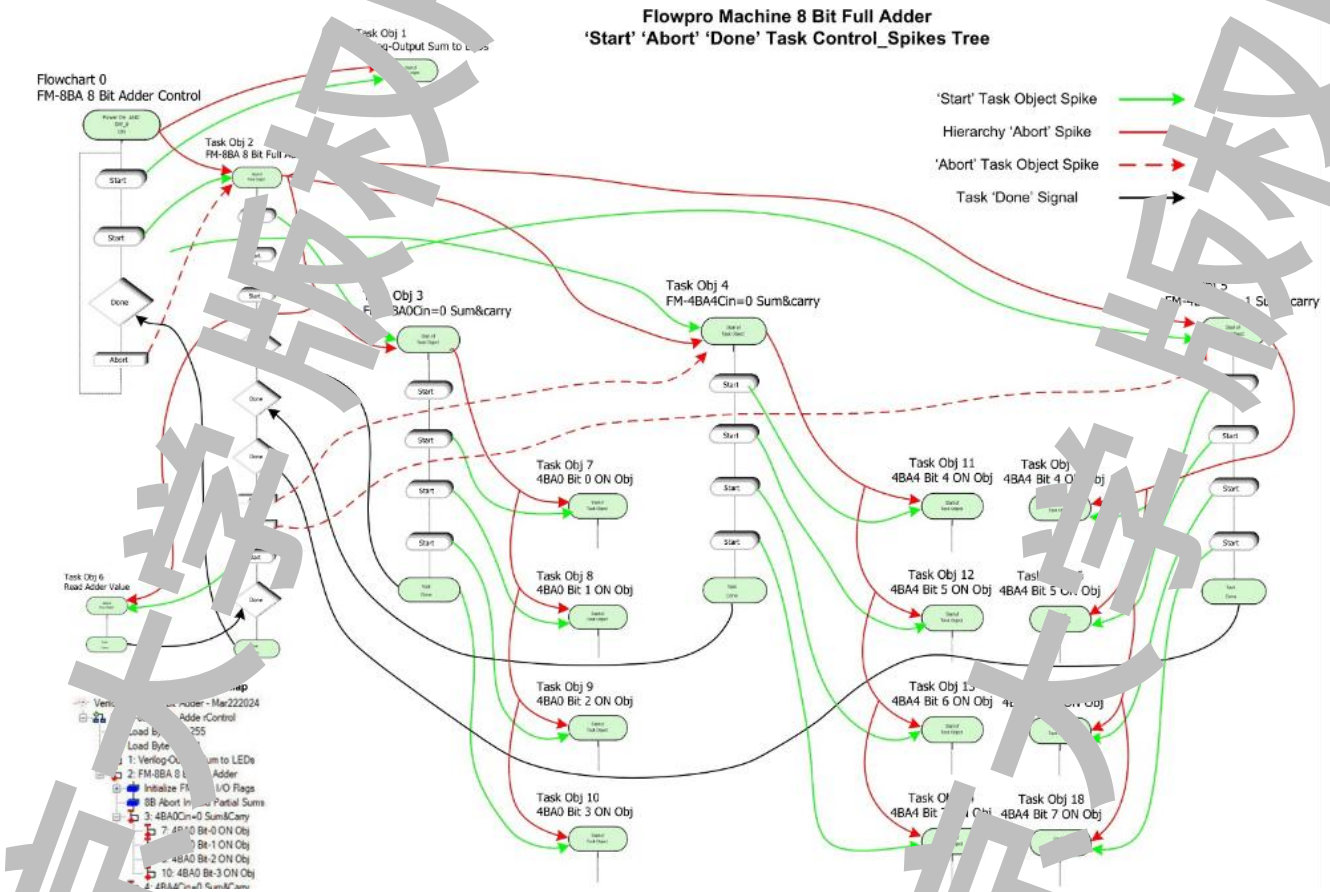
北京大学
PEKING UNIVERSITY

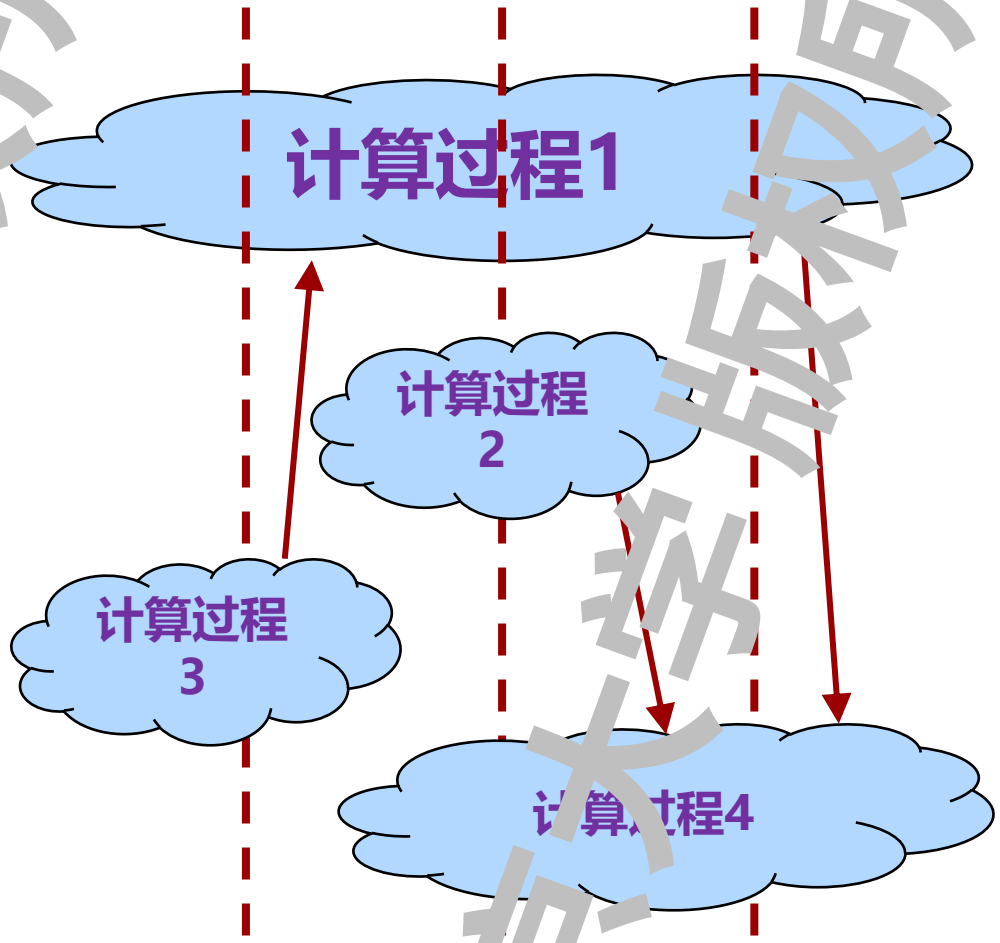
AI辅助时序逻辑设计

北京大学版权所有

电路时序的基本概念

- 电路为什么需要一个时钟?



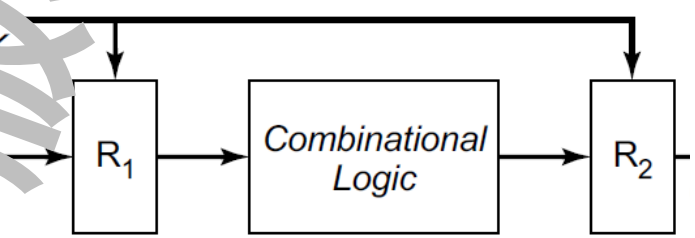


无时钟：非常难以控制每一个信号的有效时间

引入时钟：每隔一段计算将结果同步一次

电路时序的基本概念

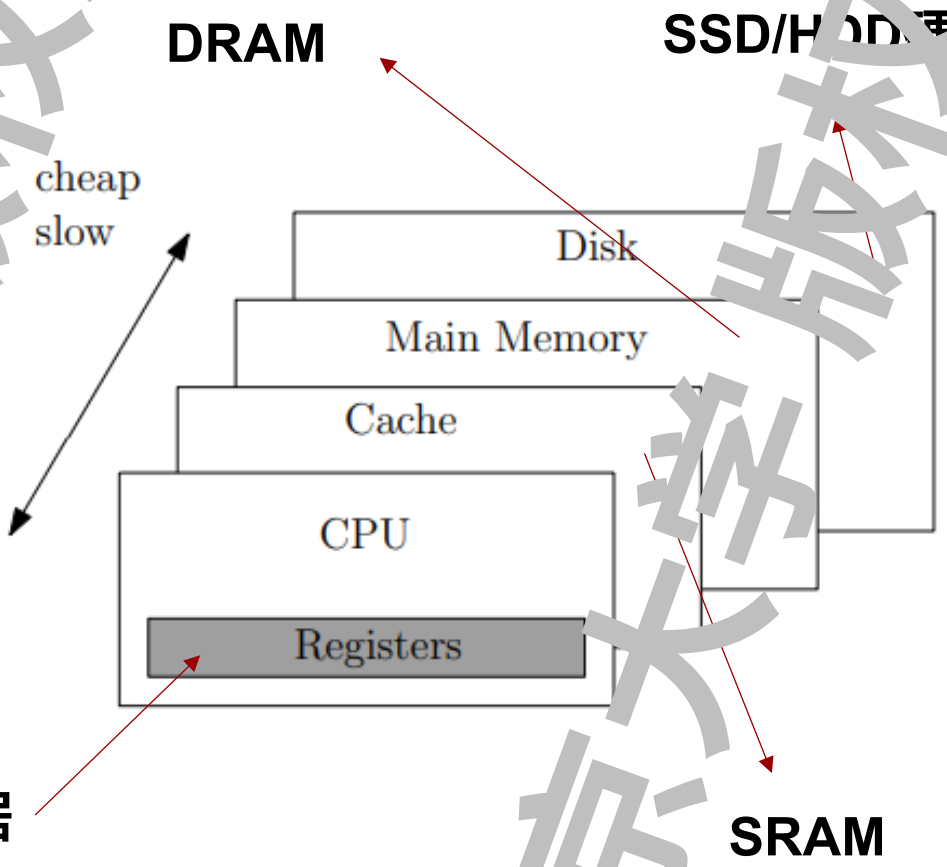
- 同步时序（Synchronous Timing）



寄存器　组合逻辑（各种逻辑门电路）

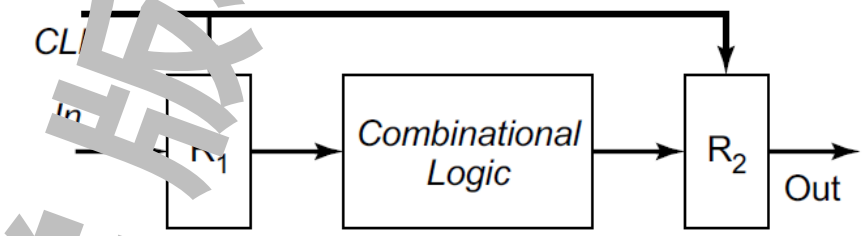
（register）

在芯片内用于暂存随时可能需要改变的数据



电路时序的基本概念

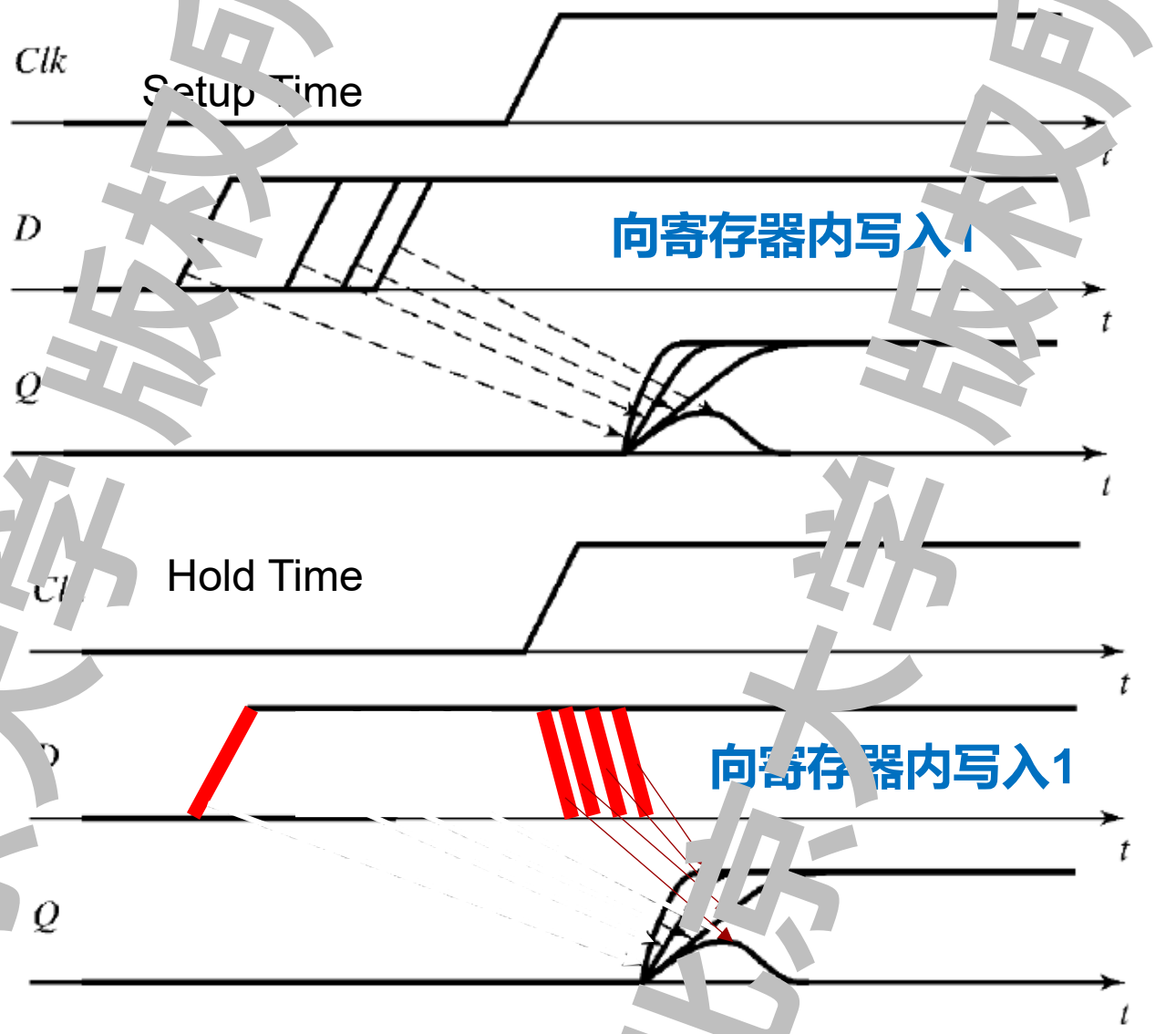
- 同步时序（Synchronous Timing）



寄存器 组合逻辑（各种逻辑门电路）

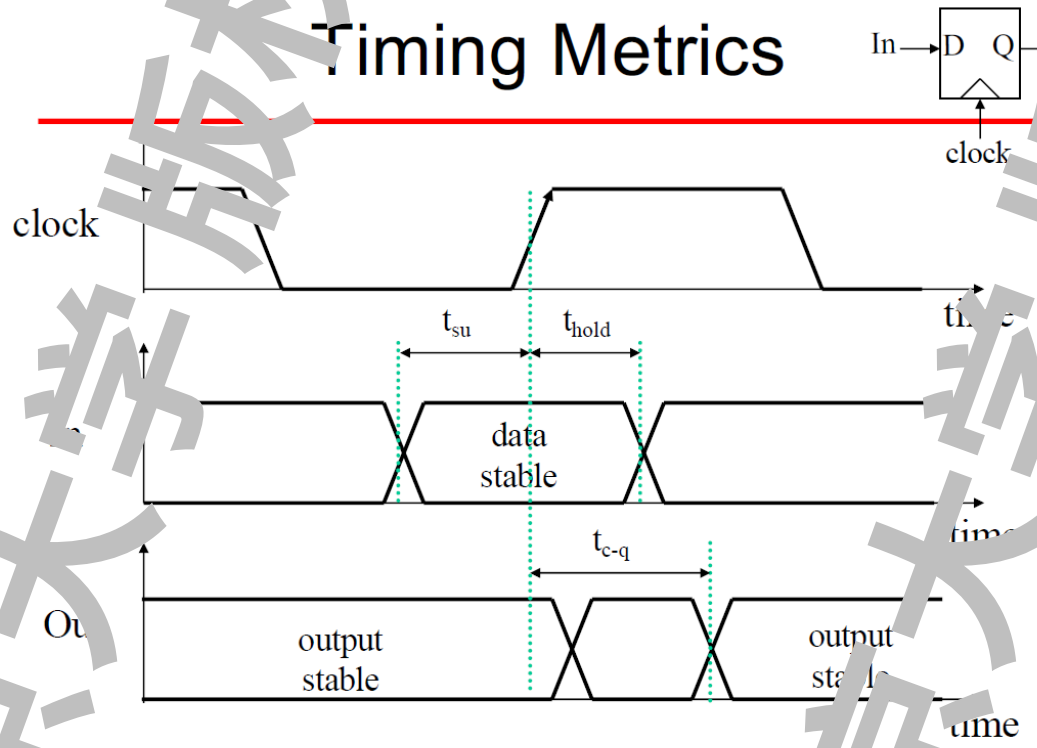
(Register)

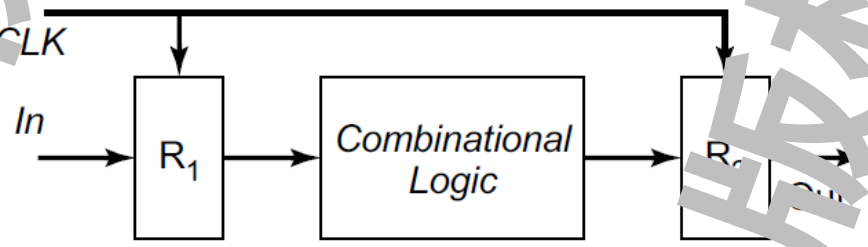
用于暂存随时可能需要改变的数据



电路时序的基本概念

- 同步时序（Synchronous Timing）





$$T_{c\text{-}q} + t_{plogic,min} \geq t_{hold}$$

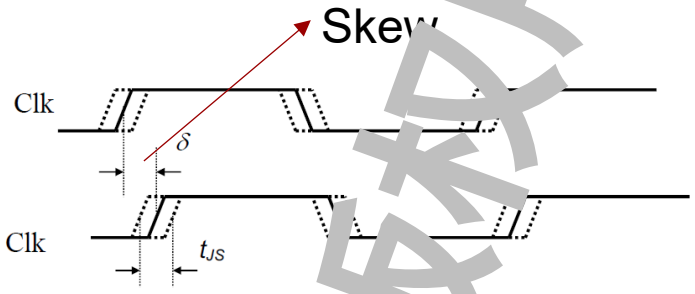
min

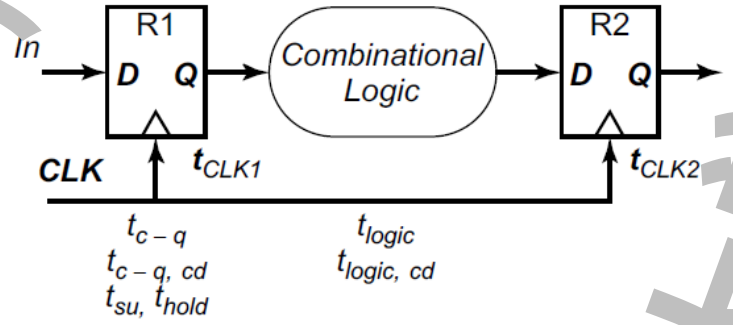
$$T \geq t_{c\text{-}q} + t_{plogic,max} + t_{su}$$

max

电路时序的基本概念

- 时钟的不稳定性

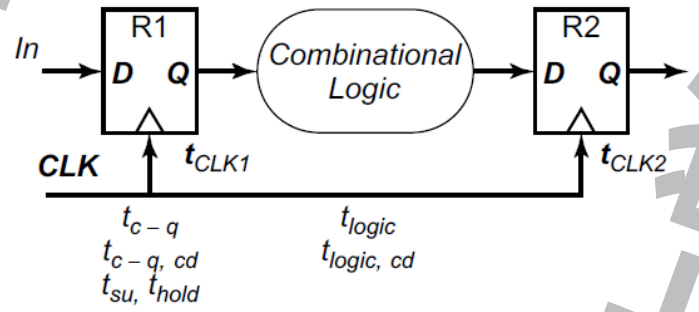




Minimum cycle time:

$T \geq t_{c\text{-}q} + t_{su} + t_{logic} - \delta$

最坏情况为接收边沿过早到达（negative δ）



Hold time constraint:

$t_{(c\text{-}q,\ cd)} + t_{(logic,\ cd)} > t_{hold} + \delta$

最坏情况为接收边沿过晚到达（正向偏差）

数据和时钟之间的竞争

Cd：contamination delay（最快可能延迟）

为什么需要状态机

- 控制电路的基本

Step 1 – 定义状态并画出状态转换图

Step 2 – 给每一个状态赋值并更新状态转换图

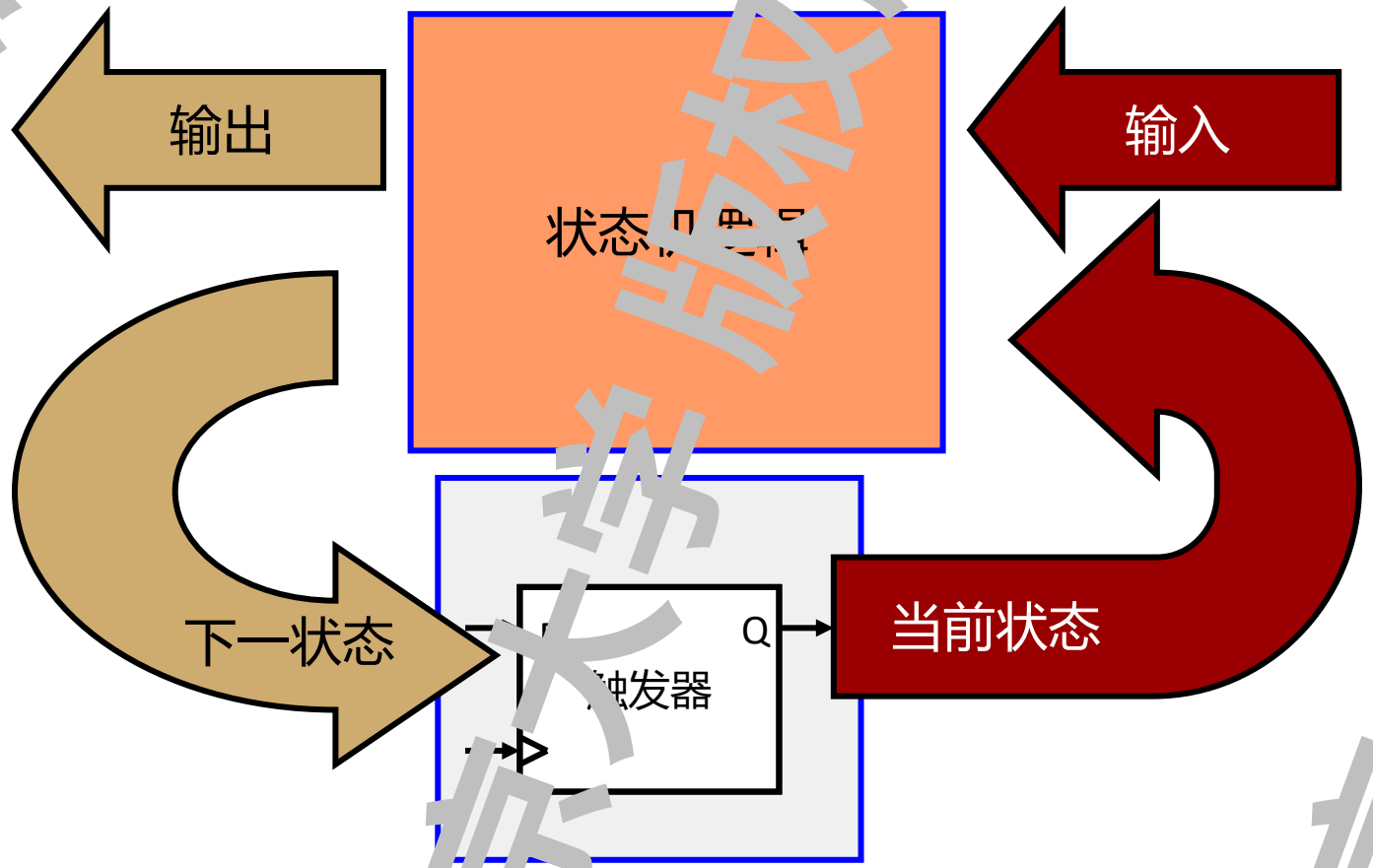
Step 3 – 根据状态转换图写出下一状态和输出的逻辑表达式

Step 4 – 画出实际电路图

状态将在每一个时钟上升沿更新

为什么需要状态机

- 控制电路的基本



为什么需要状态机

- 控制电路的基本

状态机实例1 – 控制一个红绿灯



- 仅考虑红灯和绿灯，灯转换的速度不快于每次30s（0.033 Hz）
- 2个输出
  - NSlight：1=南北向为绿灯；0=南北向红灯
  - EWlight：1=东西向为绿灯；0=东西向红灯
- 2个输入
  - Nscar：1=南北向有车等；0=南北向无车等
  - Ewcar：1=东西向有车等；0=南北向无车等
- 规则
  - 交通灯切换到另一个方向当且仅当另一方向有车等
  - 否则，保持当前交通灯不变

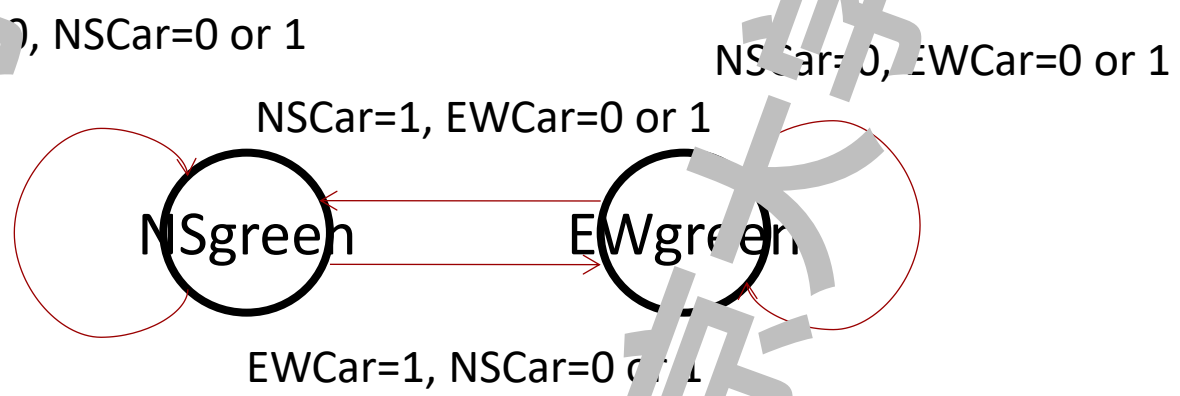
为什么需要状态机

- 控制电路的基本

状态机实例1 – 控制一个红绿灯

- 2个输出
  - NSlight：1=南北向为绿灯；0=南北向红灯
  - EWlight：1=东西向为绿灯；0=东西向为红灯
- 2个输入
  - Nscar：1=南北向有车等；0=南北向无车等
  - Ewcar：1=东西向有车等；0=南北向无车等
- 规则
  - 交通灯切换到另一个方向当且仅当另一方向有车等
  - 否则，保持当前交通灯不变

- 需要2个状态
  - Nsgreen、EWgreen



为什么需要状态机

- 控制电路的基本

状态机实例1 – 控制一个红绿灯

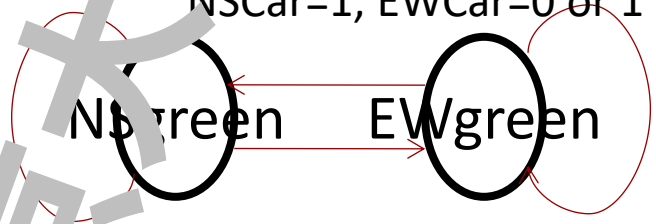
- 需要2个状态
  - NSgreen、EWgreen

Current state	Inputs		Next state
NSgreen	0	0	NSgreen
NSgreen	0	1	NSgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

EWCar=0, NSCar=0 or 1

NSCar=0, EWCar=0 or 1

NSCar=1, EWCar=0 or 1



EWCar=1, NSCar=0 or 1

Current state	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

$$\text{NextState} = (\text{CurrentState} \cdot \text{EWcar}) + (\text{CurrentState} \cdot \overline{\text{NScar}})$$

$$\text{NSlite} = \overline{\text{CurrentState}}$$

$$\text{EWlite} = \text{CurrentState}$$

为什么需要状态机

- 控制电路的基本

状态机实例1 – 控制一个红绿灯

Current state	Inputs		Next state
	NScar	EWcar	
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

Current state	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

$$NextState = (\overline{CurrentState} \cdot EWcar) + (CurrentState \cdot \overline{NScar})$$

$$NSlite = \overline{CurrentState}$$

$$EWlite = CurrentState$$

