

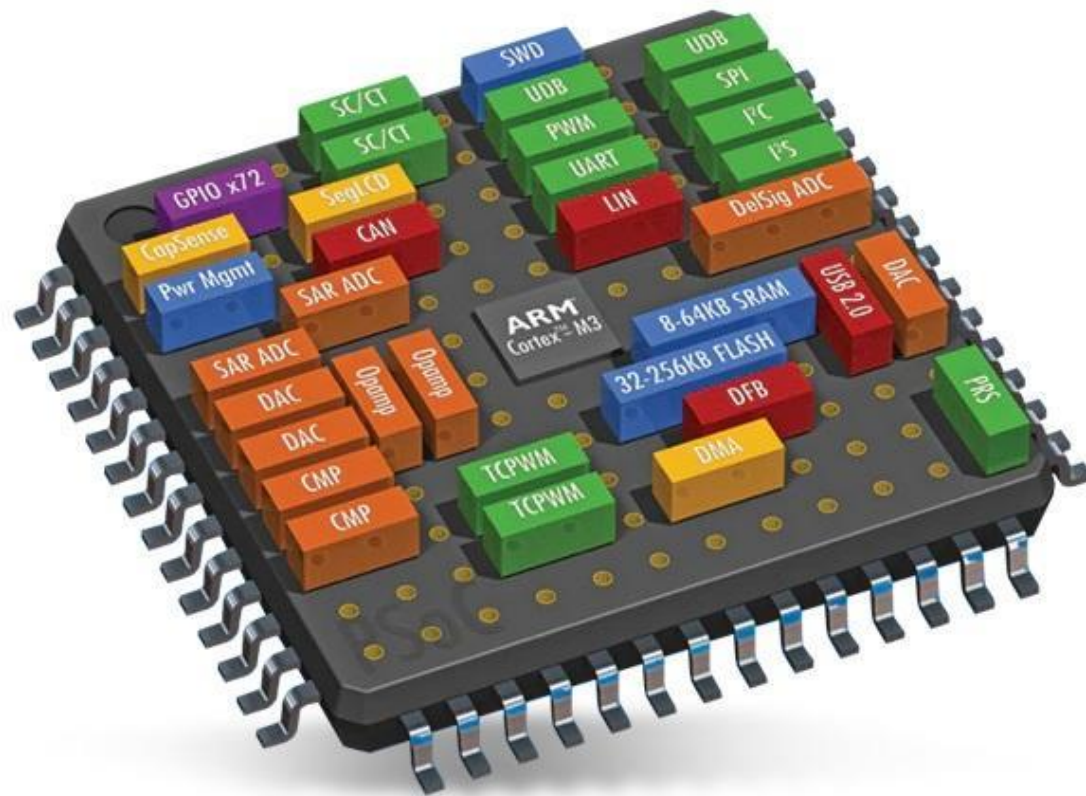
Lecture 02 SoC Design Flow and OpenEDA

SoC设计流程与OpenEDA

2025-10

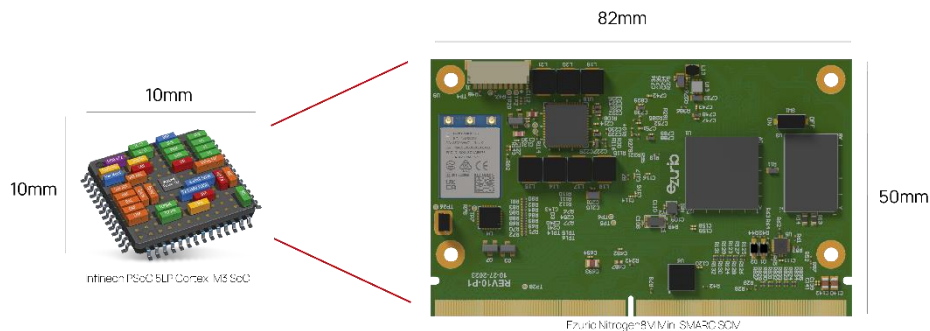
SoC (System on Chip)

- 片上系统 (SoC) 是一种集成电路, 将系统的所有必需组件合并到一块芯片上
- SoC 有助于简化PCB电路板设计, 从而在不影响系统功能的情况下提高性能

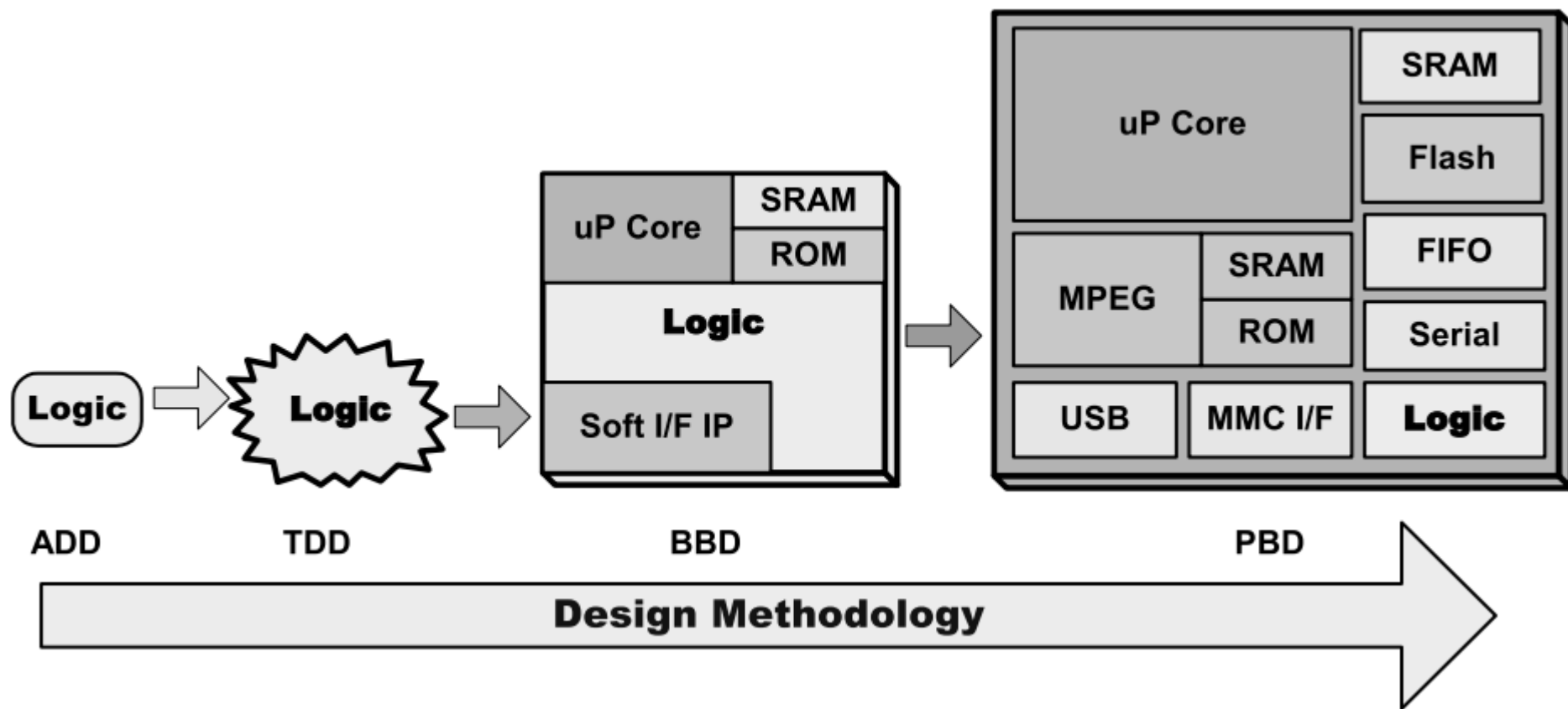


System-on-Chip (SoC)

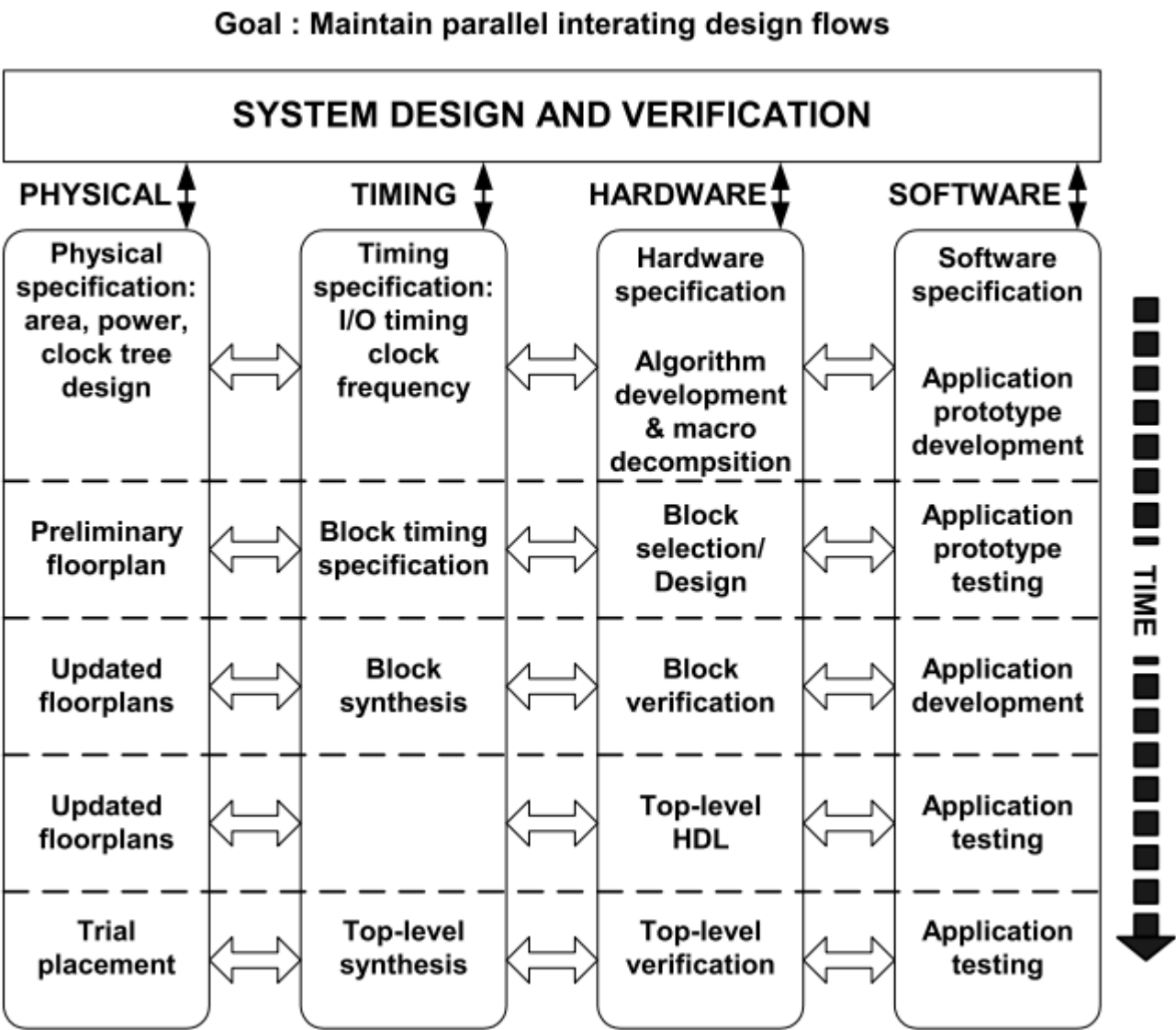
System-on-Module (SOM)



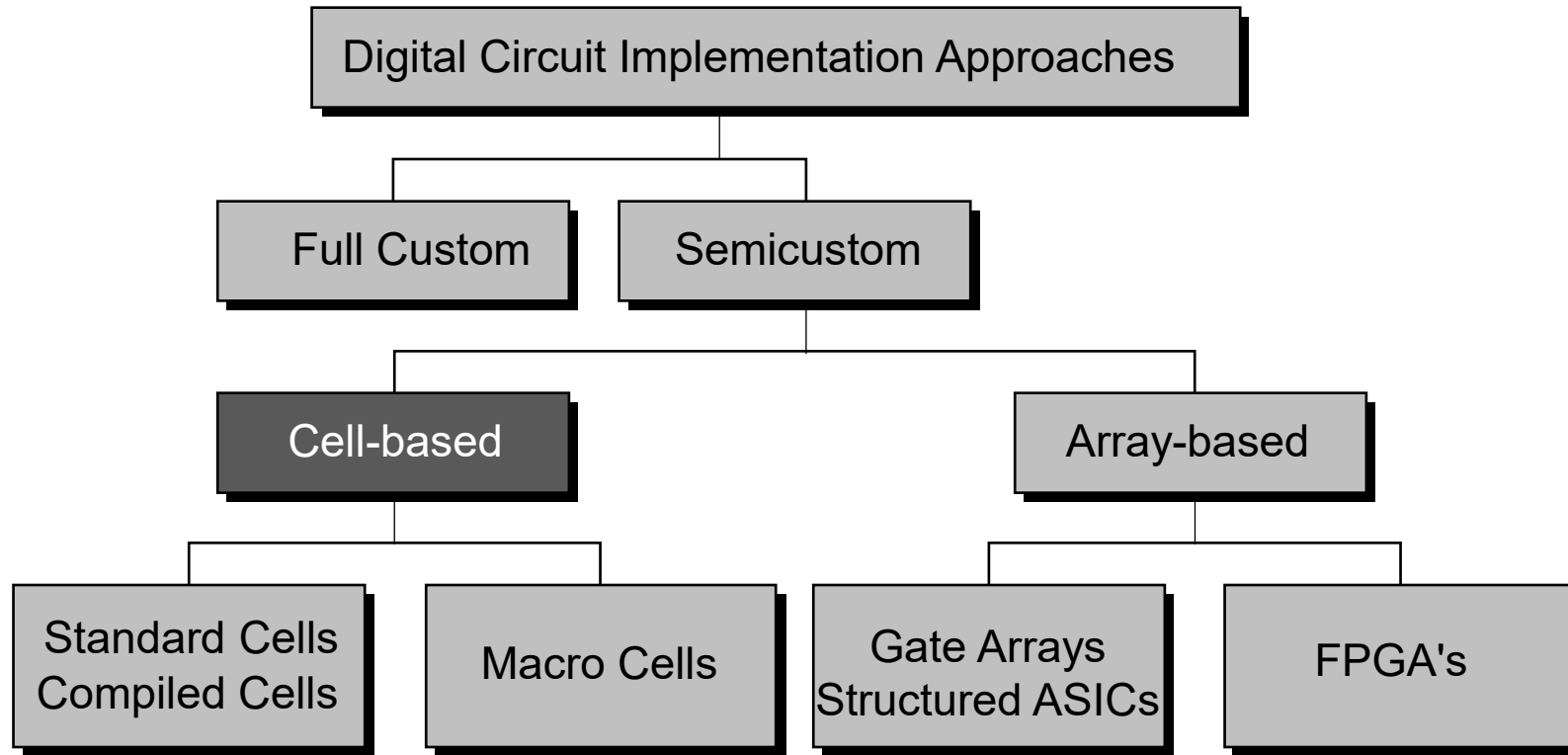
设计范式的转变



Spiral Model

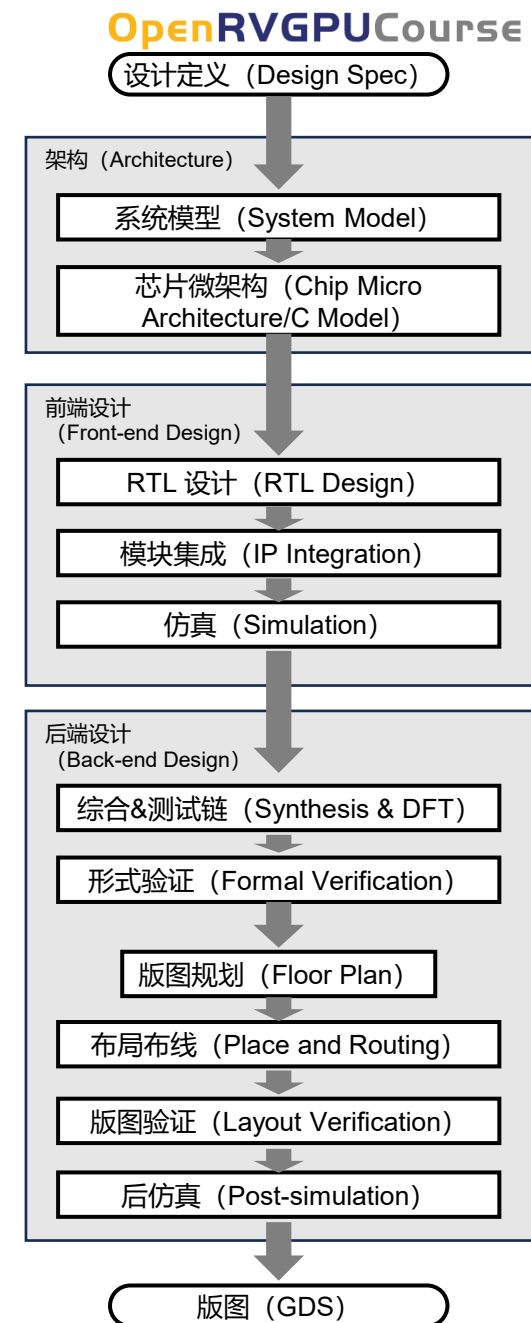
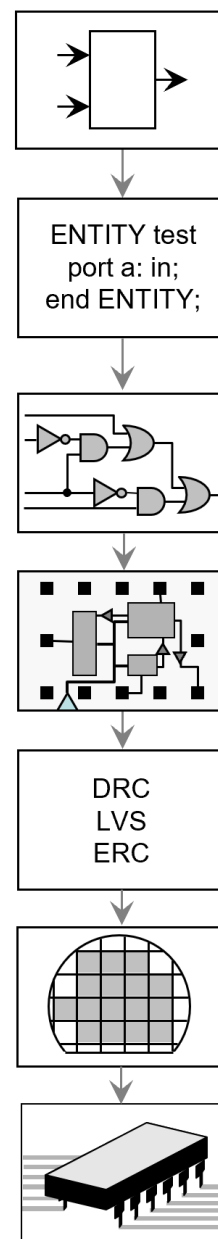


数字集成电路实现方式



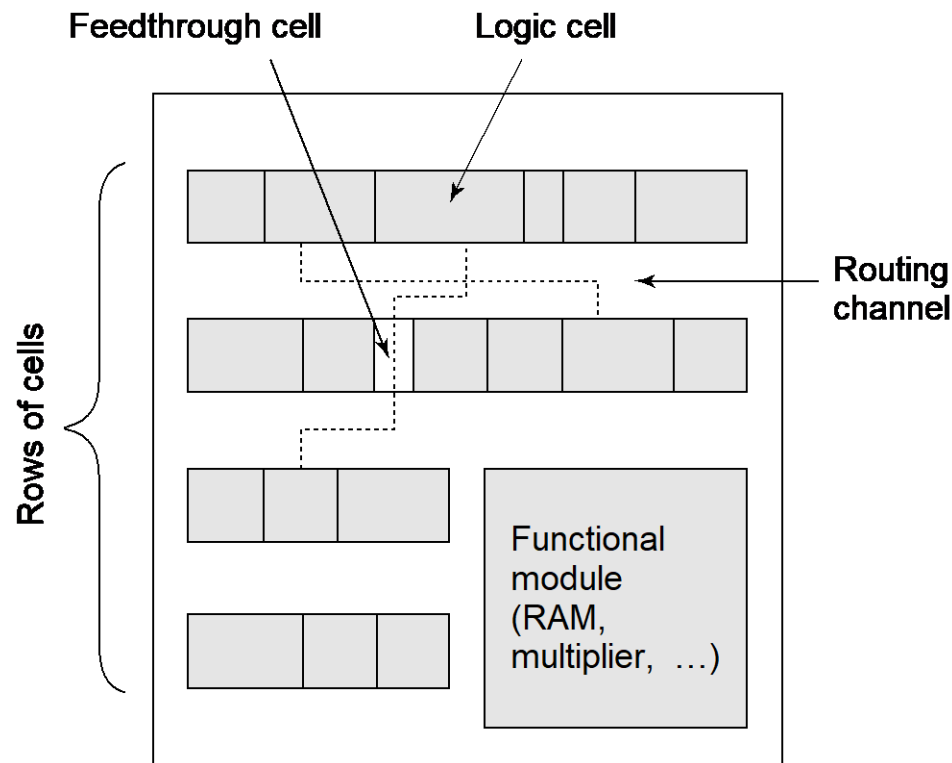
SoC芯片设计流程

- SoC设计流程涵盖软硬件协同开发，核心模块由可重复使用的IP核构成，采用深亚微米以上工艺技术
- 本节课主要聚焦SoC芯片设计流程
- 大致可分为三个阶段
 - 架构设计
 - 前端设计
 - 后端设计










基于标准单元库的设计方法

- 标准单元法 (Standard Cell Method) 是一种基于预设计单元库的半定制集成电路设计方法，通过调用标准化单元完成芯片的逻辑和物理设计
- 标准单元设计具有如下优点：
 - 标准单元设计法的布图方式更加灵活，使得标准单元设计可具有100%的连线布通率
 - 芯片中没有无用的单元和晶体管，所以面积利用率更高
 - 可以与全定制设计方法相结合，在芯片中加入全定制设计功能块，提高电路的性能



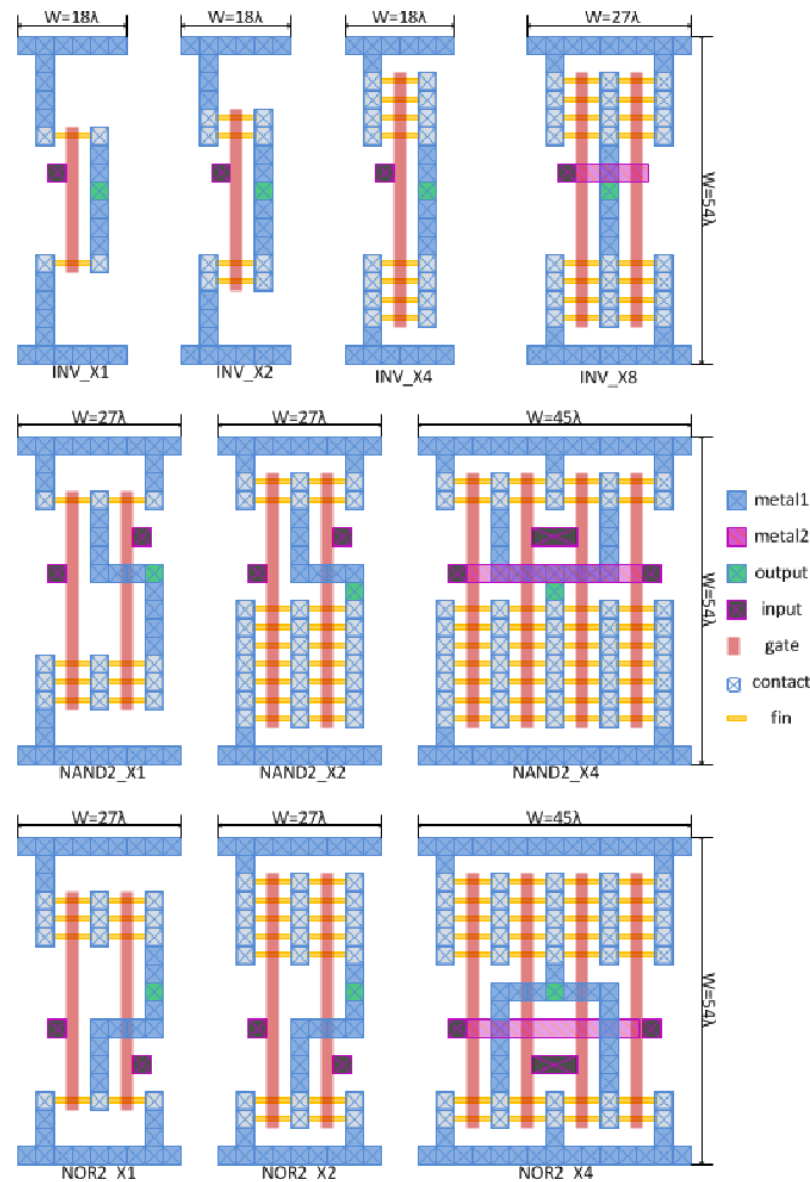
门级 (Gate-level) 电路

- 门电路是实现基本逻辑运算和复合逻辑运算的单元电路，基于二进制原理通过高低电平表示 “0” 和 “1” 两种状态
- 所谓 “门” 就是只能实现基本逻辑关系的电路
- 用逻辑门来实现逻辑运算
- 进而组成复杂逻辑和运算
- 简化数字电路设计与实现

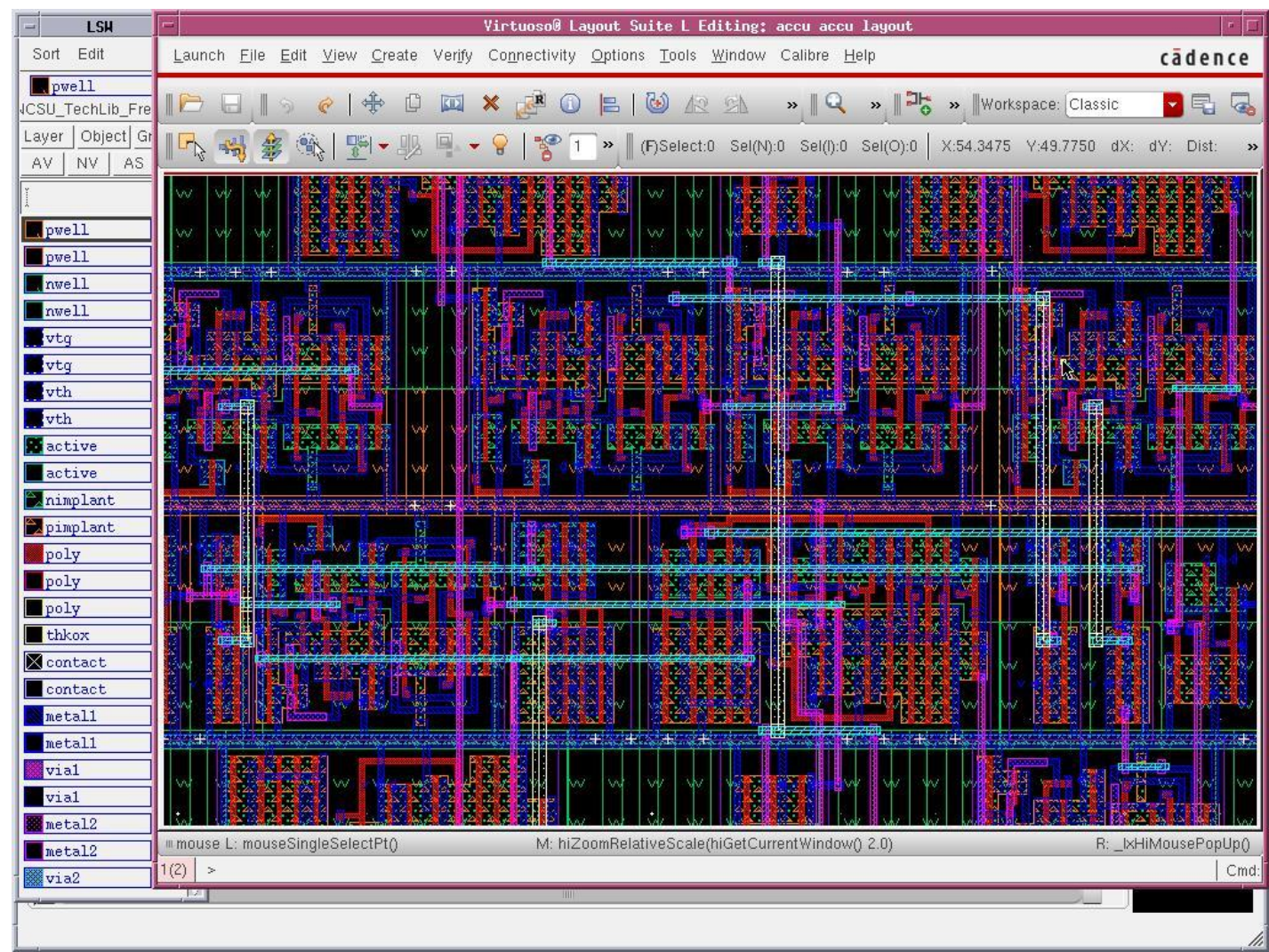
LOGIC FUNCTION	LOGIC SYMBOL	BOOLEAN EXPRESSION	TRUTH TABLE		
			INPUTS		OUTPUTS
			B	A	Y
AND		$A+B=Y$	0	0	0
			0	1	0
			1	0	0
			1	1	1
			0	0	0
OR		$A+B=Y$	0	1	1
			1	0	1
			1	1	1
			0	0	0
inverter		$A=\bar{A}$	0	1	1
			1	0	0
NAND		$\overline{A+B}=Y$	0	1	1
			0	0	1
			1	1	0
			1	0	1
NOR		$\overline{A+B}=Y$	0	1	0
			0	0	0
			1	1	0
			1	0	0
XOR		$A\oplus B=Y$	0	1	0
			0	0	1
			1	1	1
			1	0	0
XNOR		$\overline{A\oplus B}=Y$	0	0	1
			0	1	0
			1	0	0
			1	1	1
			1	0	0

标准单元 (Standard Cell)

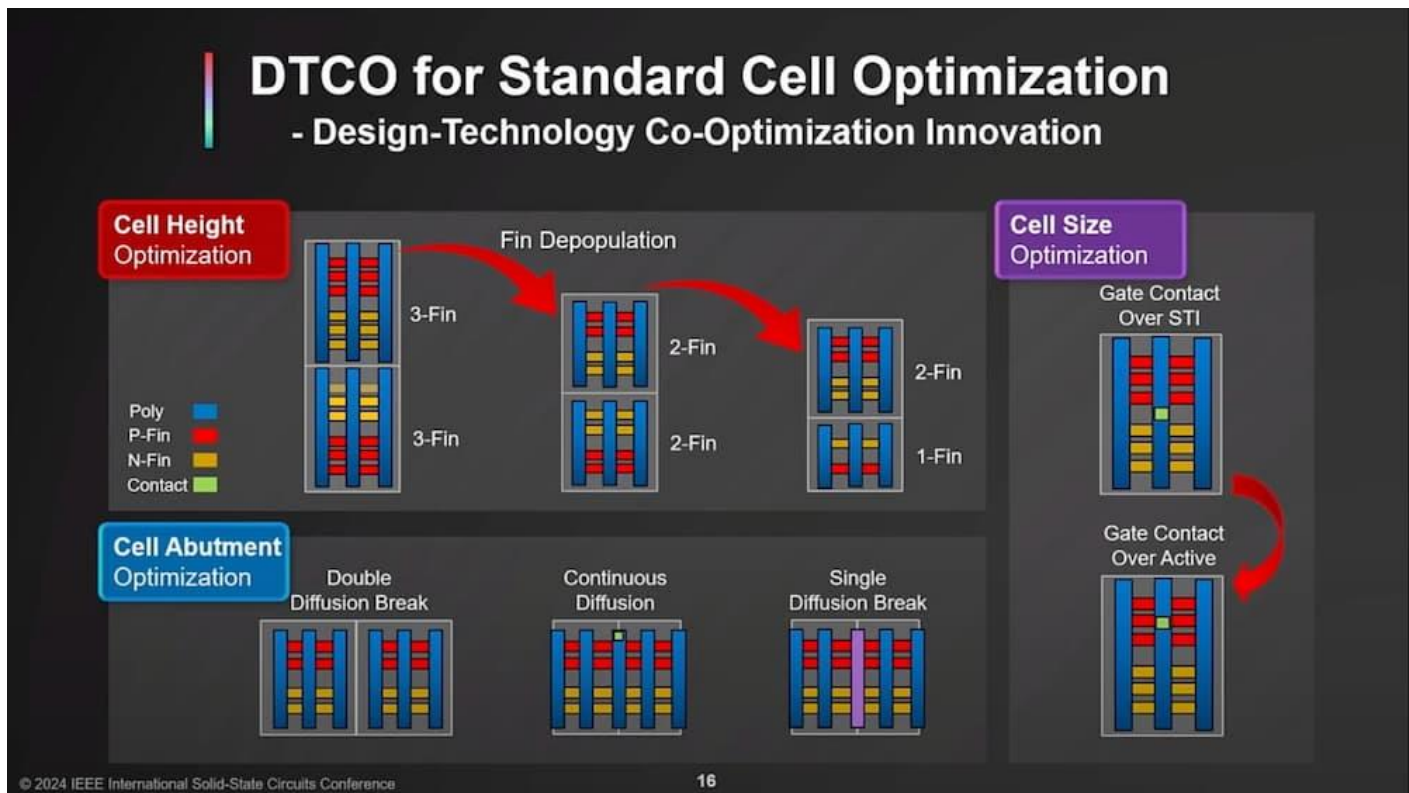
- 标准单元是预先设计和预先验证的功能块，这些功能块封装了特定的逻辑函数
- 这些单元布局布线在预定义的高度，无缝互连，允许创建复杂的数字电路
- 标准单元是芯片中的基本概念，彻底改变了集成电路（IC）的设计和制造。这些模块化基本模块构成了高效和可扩展的芯片设计的基础，在灵活性和性能之间提供了理想的平衡



基于标准单元法的IC版图

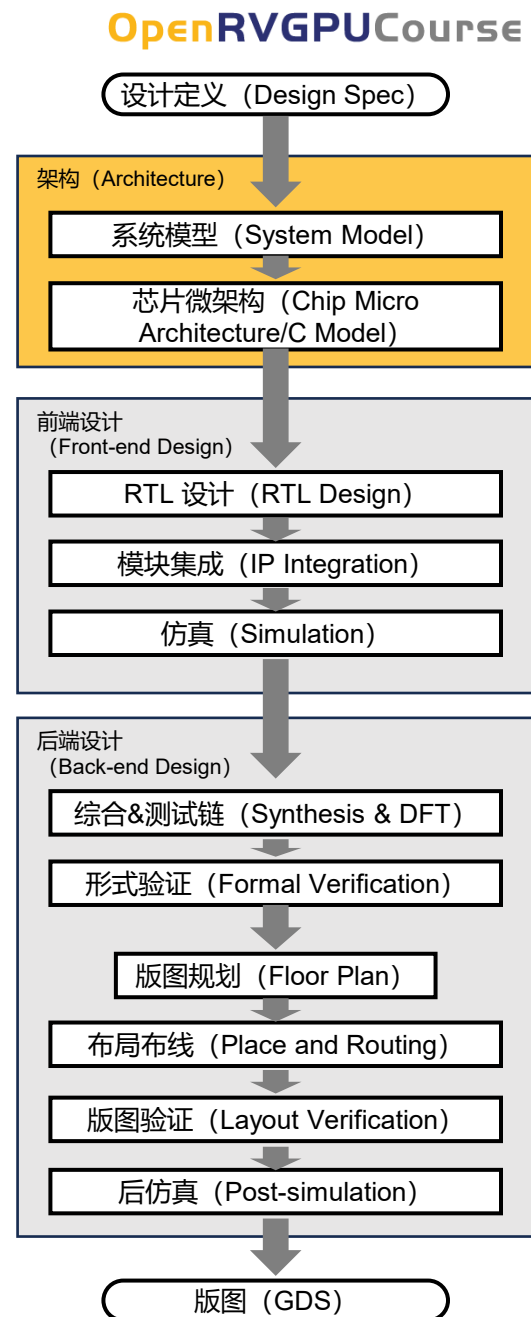


- 设计工艺协同优化（DTCO）是计算光刻与版图优化领域的技术术语，指通过工艺开发与芯片设计的协作，在器件性能与制造可行性间寻求平衡
- 未来半导体技术将主要通过三个方向来实现性能和能效的提升：
 - 微缩技术，提高晶体管密度；
 - DTCO/STCO，推进设计与工艺的协同优化；
 - 2.5D/3D先进封装与硅堆叠，实现系统集成



架构设计

- 芯片架构设计是芯片开发的核心环节，架构设计定义了芯片的功能、性能及硬件组件之间的协同工作方式
- 芯片架构需实现功能、性能、功耗、面积（FPPA）的平衡，例如在移动设备中兼顾低功耗与高性能，在高性能计算中优化并行处理能力
- 系统模型
 - 系统模型是对一个系统进行简化和抽象的表示，通过描述系统的各个组成部分以及它们之间的相互作用和关系，来帮助我们理解和研究该系统
- 芯片微架构
 - 在芯片的架构设计阶段需要对芯片进行建模（modeling），建模的主要目的是在大规模投入人力之前先用低成本的计算机软件实现目标芯片的主要功能，评估芯片的工作效果，验证设计思想，并作为后续开发活动的标准和依据



微架构设计-C Model

- C Model是使用C语言建立模型，对CPU/GPU等复杂芯片硬件的功能进行仿真
- C Model常用于在进行RTL级建模与测试之前对功能进行验证与分析
- C Model的优势在于仿真速度快（比Verilog快很多），效率高，便于进行系统混合仿真。当然C Model并不能代替Verilog仿真验证
- C Model的分类
 - 常见的电路C Model可以分为两类：function model 与 cycle model
 - function model是对硬件的功能进行仿真的模块。如在vortex的C Model中，execute.cpp文件负责根据opcode的值对不同的指令进行运算
 - cycle model是对硬件功能和时钟级的时序进行仿真的模块，负责进行模块间的连接并对时延进行模拟
- 一般有两种C Model的搭建方式。
- 第一种（以vortex为例）将C Model的cycle model分为function model与 timing model两个部分。时序模块之间通过接口直接连接。function model接收来自对应的cycle model的时序模块发送的数据及控制信号，并执行指令。这种方式的好处是避免了同时分立维护function model和 timing model
- 第二种（以gpgpusim为例）不完全区分function model与 timing model，同时时序模块之间并不连接。该种方式将所有需要的数据与控制信号打包在warp_t中。在执行流水线时每个模块按顺序改变寄存器与warp_t的值来模拟流水线的进行
- 为了避免维护两套model的复杂度，一般直接将cycle model分为function model和timing model 两部分，即第一种方式

C Model功能模型与时序模型设计思路

• 功能模型设计思路

使用case模拟指令译码：

使用case根据opcode、funct7、func3等用于判断指令类型和数据类型的值来判断指令的种类。一般在decode阶段将指令数据解码为抽象的指令类型。在exec阶段通过抽象的指令类型来判断执行哪个函数

使用trace模拟数据的集合：

在模块外部定义公共的类trace，在trace中定义包括指令、数据、控制信号在内的变量。在运行时通过更改trace中的变量的值来模拟运行

for循环来模拟执行的并行运行：

但在一些情况下for循环也能表示不同周期。比如for($i < \text{thread}$) 是模拟并行的thread,而for ($i < t$) 可能是模拟串行数据在经过t个周期后执行的结果。这个无明显的规律，需要根据实际情况来判断。此外，向量也可以用来表示串行数据。如在tensorcore中，使用v[0:7]来表示一个thread在8个周期接收到的数据。同样需要根据实际情况来判断

使用函数来执行具体的功能：

应该在功能模块的内部仍然根据不同功能将其分为如处理trace的函数、执行运算的函数、修改寄存器的函数等模块，再在函数内实现功能。通过模块的主函数来按顺序调用相关的函数，达到模拟执行功能的目的

• 时序模型设计思路

时序模块包含处理时序和功能仿真两个部分。功能仿真的部分主要处理与时序相关的功能，如dispatch，writeback等

时序模块的设计思路大部分和功能模块相似

用trace发送latency来模拟时延：

trace中包含用于传递时延的寄存器，在时序模块中在执行指令时将对应的时延发送到该寄存器中，随后该时延会被传递到sim部分进行处理函数用于模拟时序：

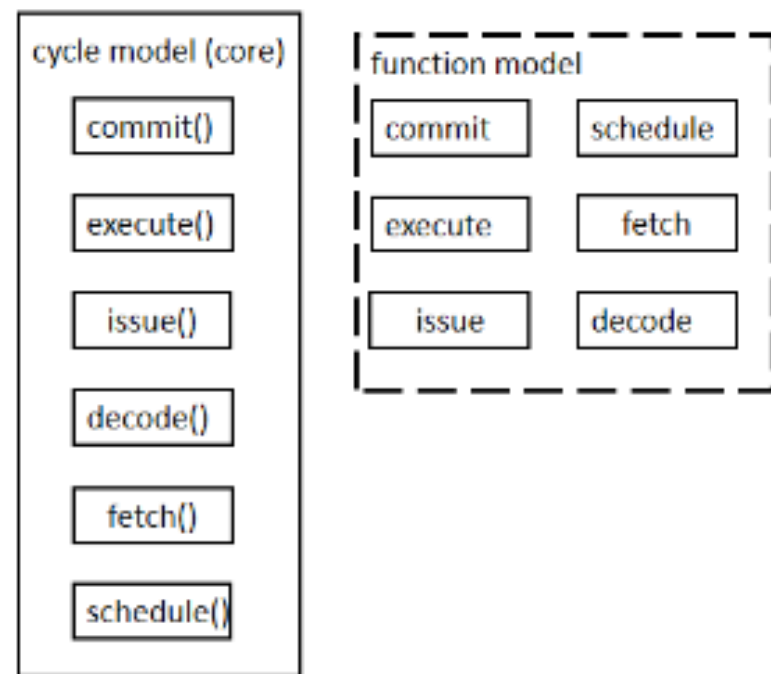
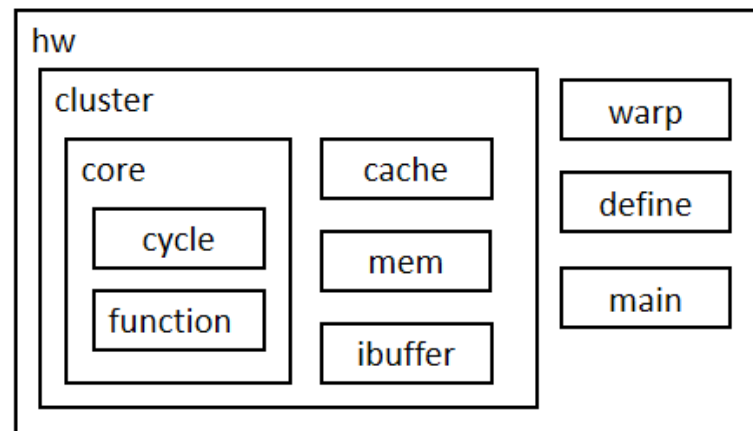
时序模型的主函数一般是cycle()函数，在该函数中按顺序调用writeback()、execute()等模拟流水线单元的同名函数。该同名函数负责连接与发送trace

与时序相关的功能模块：

部分模块的功能与时序相关，如writeback需要向流水线执行写回。因此在该模块的同名函数中执行功能

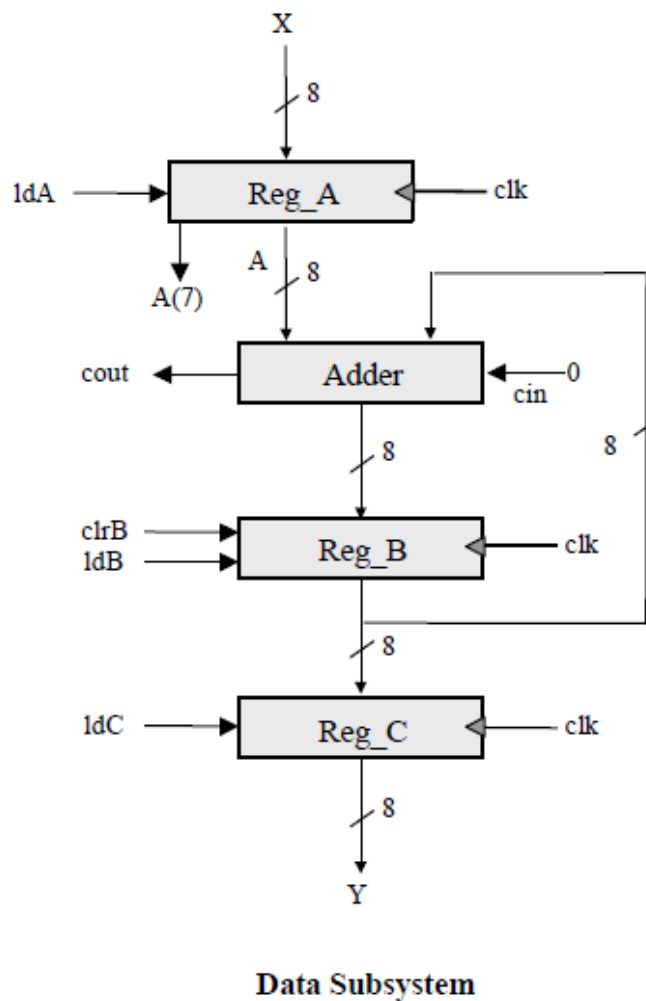
C Model设计的一般规范

- 变量名应使用小写，在不同的单词之间用_隔开 (num_operand) 或是从第二个单词开始将首字母大写 (numOperand, 这一方法目前不推荐)
- 指令类型、指令名称应全部大写 (FL、T_TYPE)
- 向量类型的变量、宏定义的命名应在前面加上v (vMul、vMop)
- 表示数量的常量或变量应在前面加上num_ (num_operand)
- 执行单元的命名应当使用xxUnit, 首字母大写 (FpuUnit)
- 名称使用缩写时应全部大写

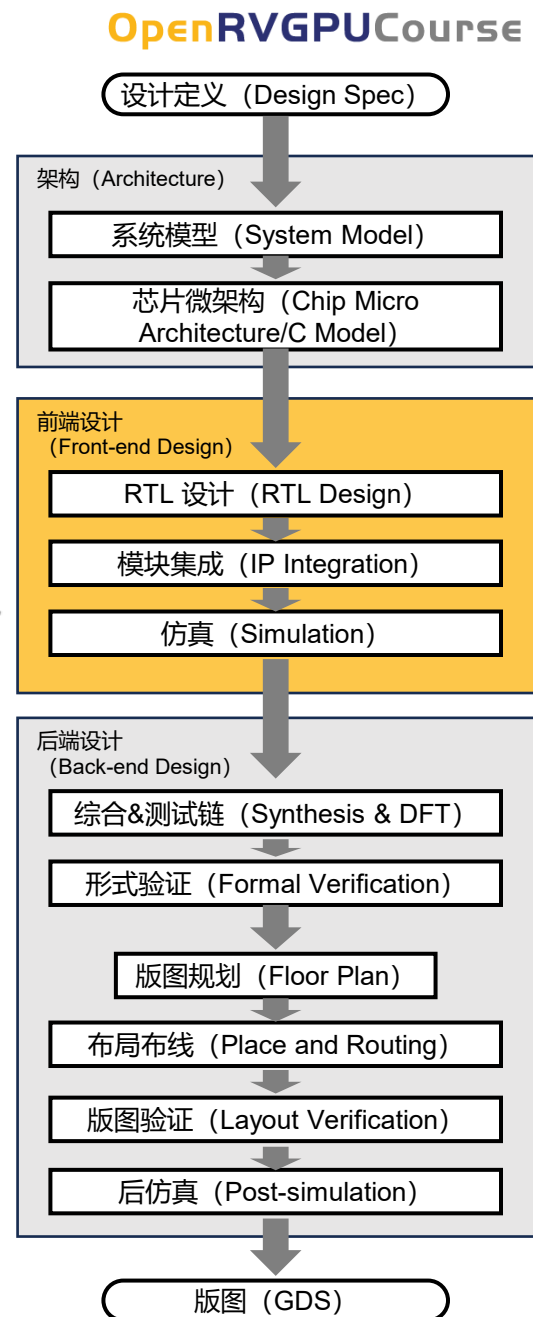


前端设计

- 前端设计，也被称为逻辑设计，主要关注于电路的功能性和逻辑性
- RTL设计 = Register Transfer Level Design
 - RTL 编码是数字电路设计中使用的—种抽象技术
 - 使用硬件描述语言（HDL）描述数字电路的行为和功能
 - RTL 充当设计规范和电路物理实现之间的中间表示。
 - 典型的 HDL 是 Verilog 或 VHDL



Data Subsystem



Verilog代码规范

编程风格 (Coding Style) 要求

• 文件

- 每个模块 (module) 一般应存在于单独的源文件中, 通常源文件名与所包含模块名相同。
- 每个设计文件开头应包含如下注释内容:
 - 年份及公司名称
 - 作者
 - 文件名
 - 所属项目
 - 顶层模块
 - 模块名称及其描述
 - 修改纪录

• 大小写

- 如无特别需要, 模块名和信号名一律采用小写字母。
- 为醒目起见, 常数 (`define定义) /参数 (parameter定义) 采用大写字母。

• 标识符

- 标识符采用传统C语言的命名方法, 即在单词之间以"_"分开, 如: max_delay、data_size、clk_cpu等等。
- 采用有意义的、能反映对象特征、作用和性质的单词命名标识符, 以增强程序的可读性。
- 为避免标识符过于冗长, 对较长单词的应当采用适当的缩写形式, 如用'buf'代替'buffer', 'en'代替'enable', 'addr'代替'address'等, 低电平采用_n后缀, 例如en_n。

• 参数化设计

- 为了源代码的可读性和可移植性起见, 不要在程序中直接写特定数值, 尽可能采用`define语句或parameter语句定义常数或参数。

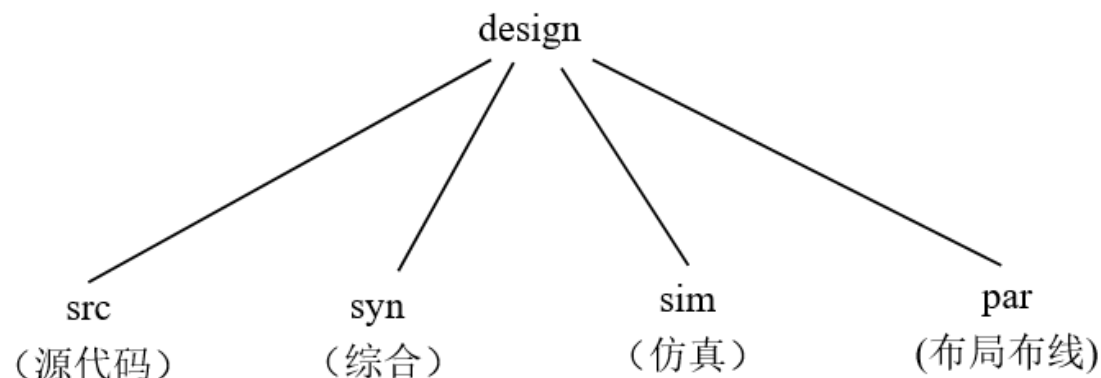
• 空行和空格

- 适当地在代码的不同部分中插入空行, 避免因程序拥挤不利阅读。
 - 在表达式中插入空格, 避免代码拥挤, 包括:
 - 赋值符号两边要有空格;
 - 双目运算符两边要有空格;
 - 单目运算符和操作数之间可没有空格,
- 示例如下:
- ```
a <= b;
c <= a + b;
if (a == b) then ...
a <= ~a & c;
```

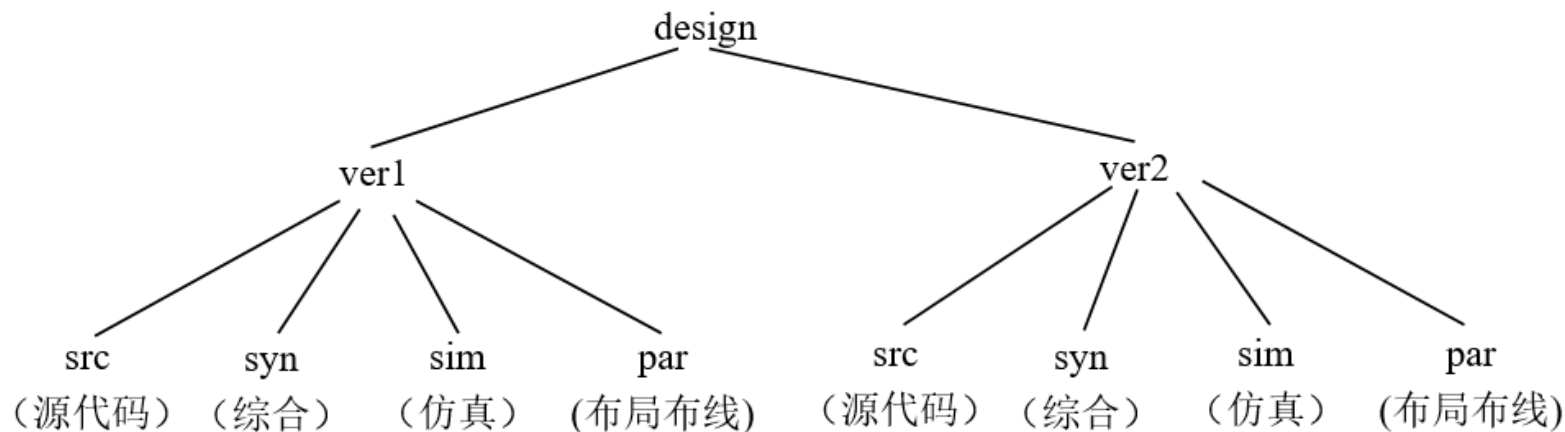
# RTL设计目录

- 采用合理、条理清晰的设计目录结构有助于提高设计的效率、可维护性

(1)

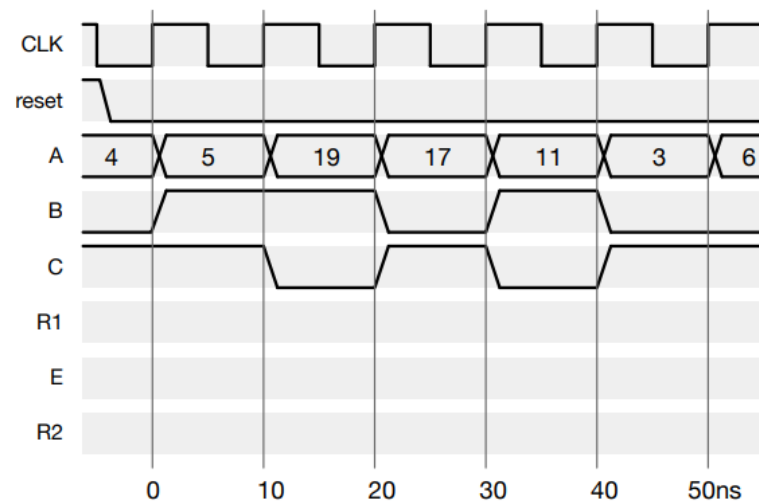
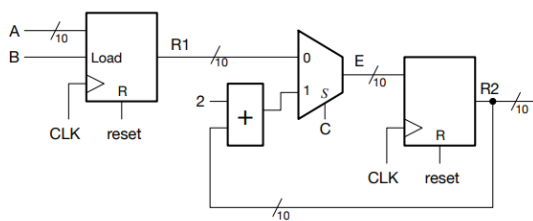
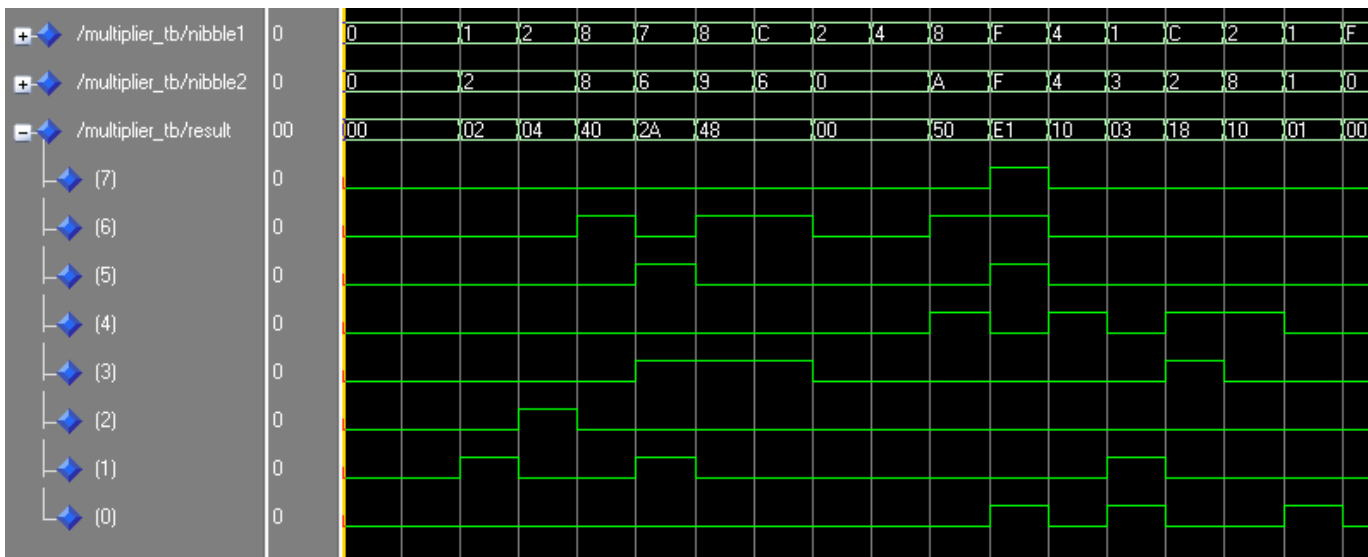


(2)



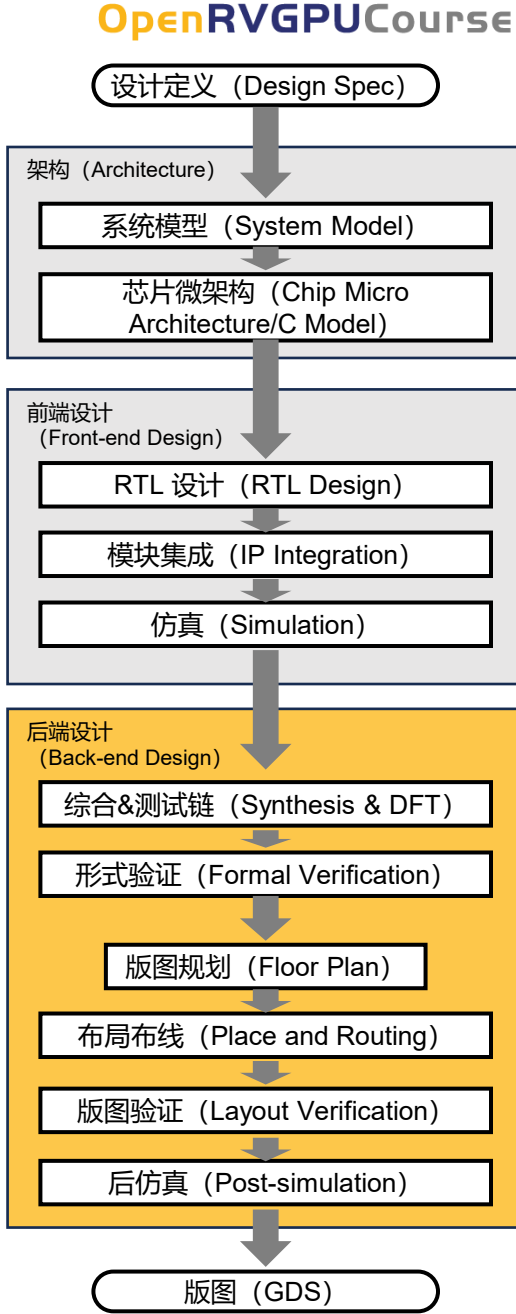
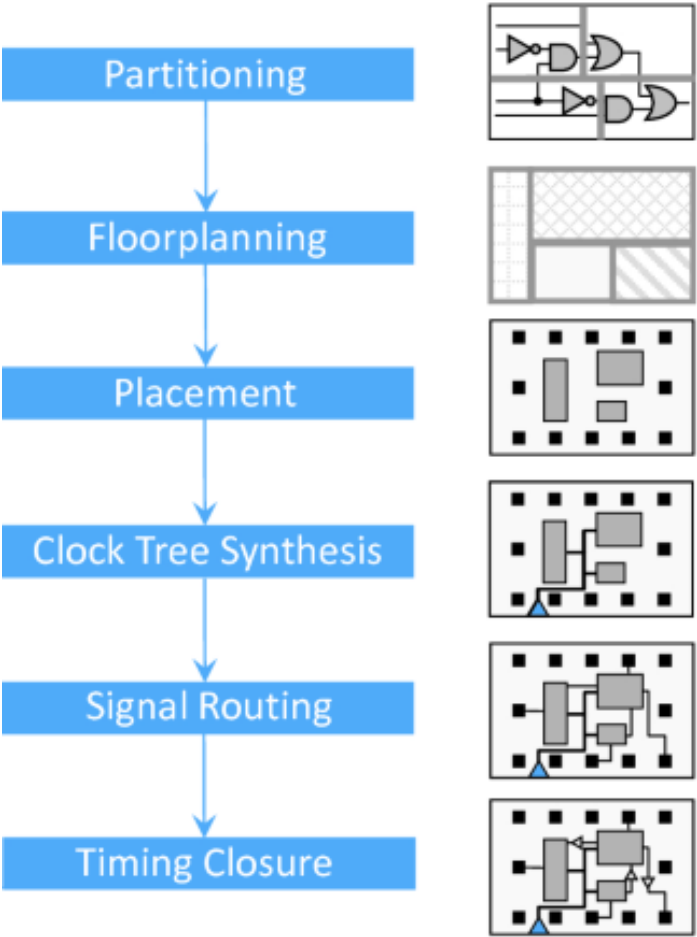
# 仿真

- 仿真是指通过计算机模拟和建模技术验证集成电路设计的正确性, 涵盖功能验证、性能评估等多个方面
- 功能仿真: 即RTL仿真, 这是仿真验证的第一步, 也叫前仿真, 目标是在理想的情况下, 确认设计的功能是否符合预期。在这个阶段, 我们通过模拟设计在特定输入下的输出来验证其行为



# 后端设计

- 芯片后端设计是芯片制造前的物理实现阶段，将前端设计的电路逻辑转化为可制造的芯片版图。其核心任务是将抽象的电路模型转换为符合工艺规则的物理电路结构，确保芯片功能、性能和可靠性达标



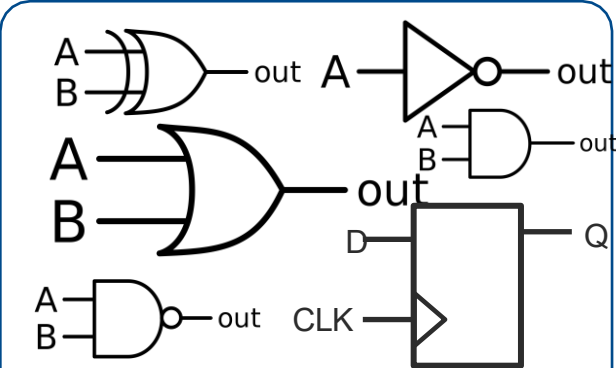
# RTL门级综合

```
always @(posedge clk or negedge reset_q_i)
begin
 if (reset_q_i == 1'b0) begin
 r_result <=1'b0;
 end
 else begin
 r_result<=input0_i && input1_i;
 end
end
```

RTL

- Clock Frequency
- Input/Output Delays
- False paths
- Case Settings
- ...

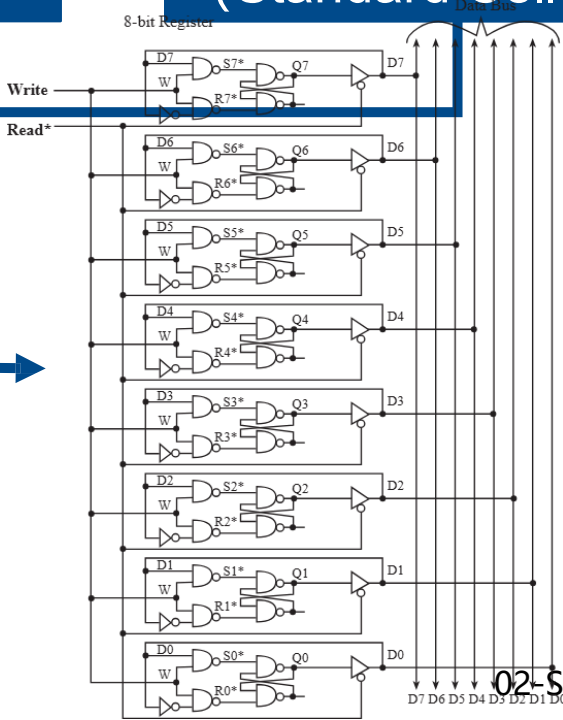
约束条件  
(Constraints)



标准单元库  
(Standard Cell Lib)

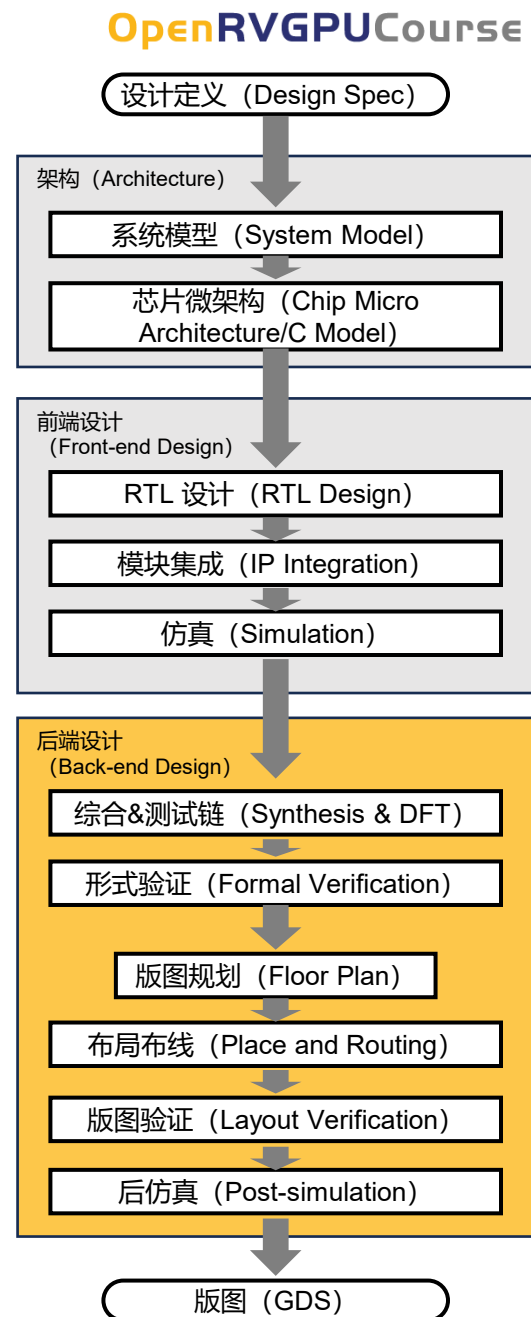
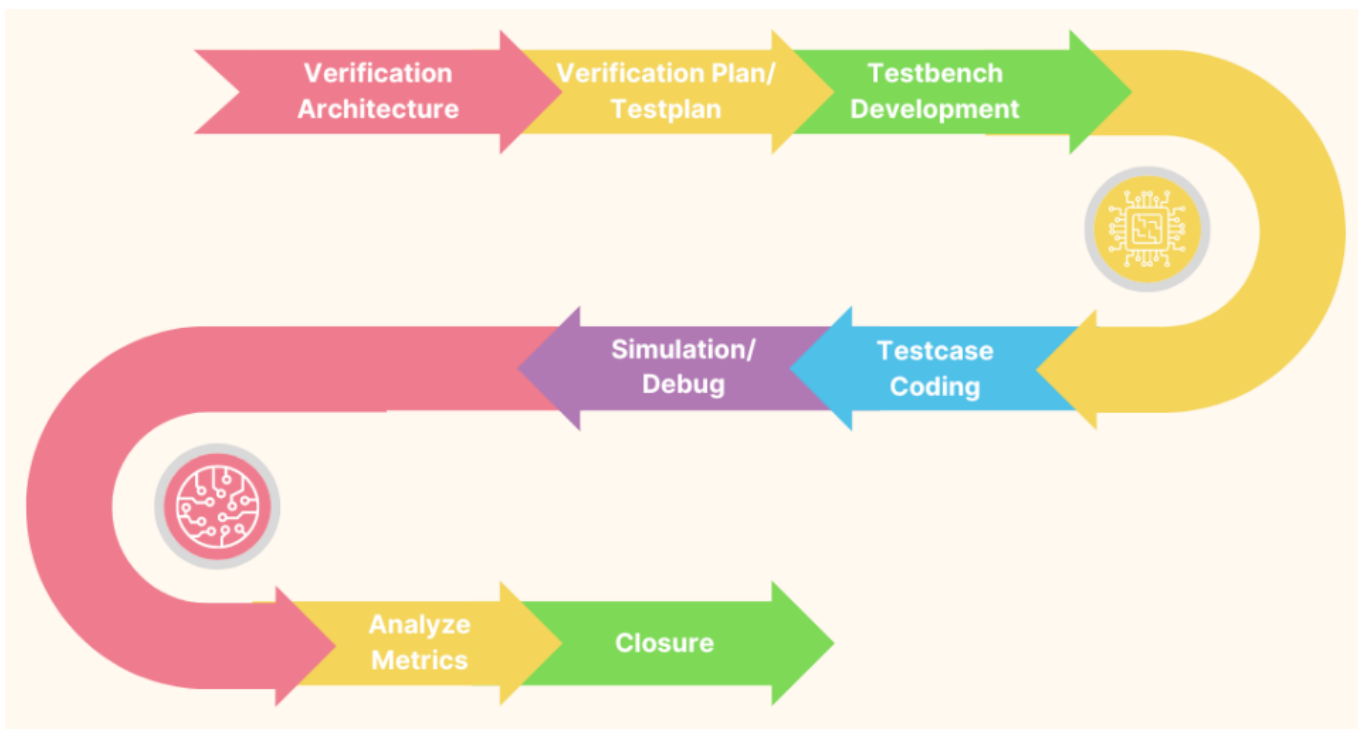
Synthesis Tool

Netlist



# 形式验证

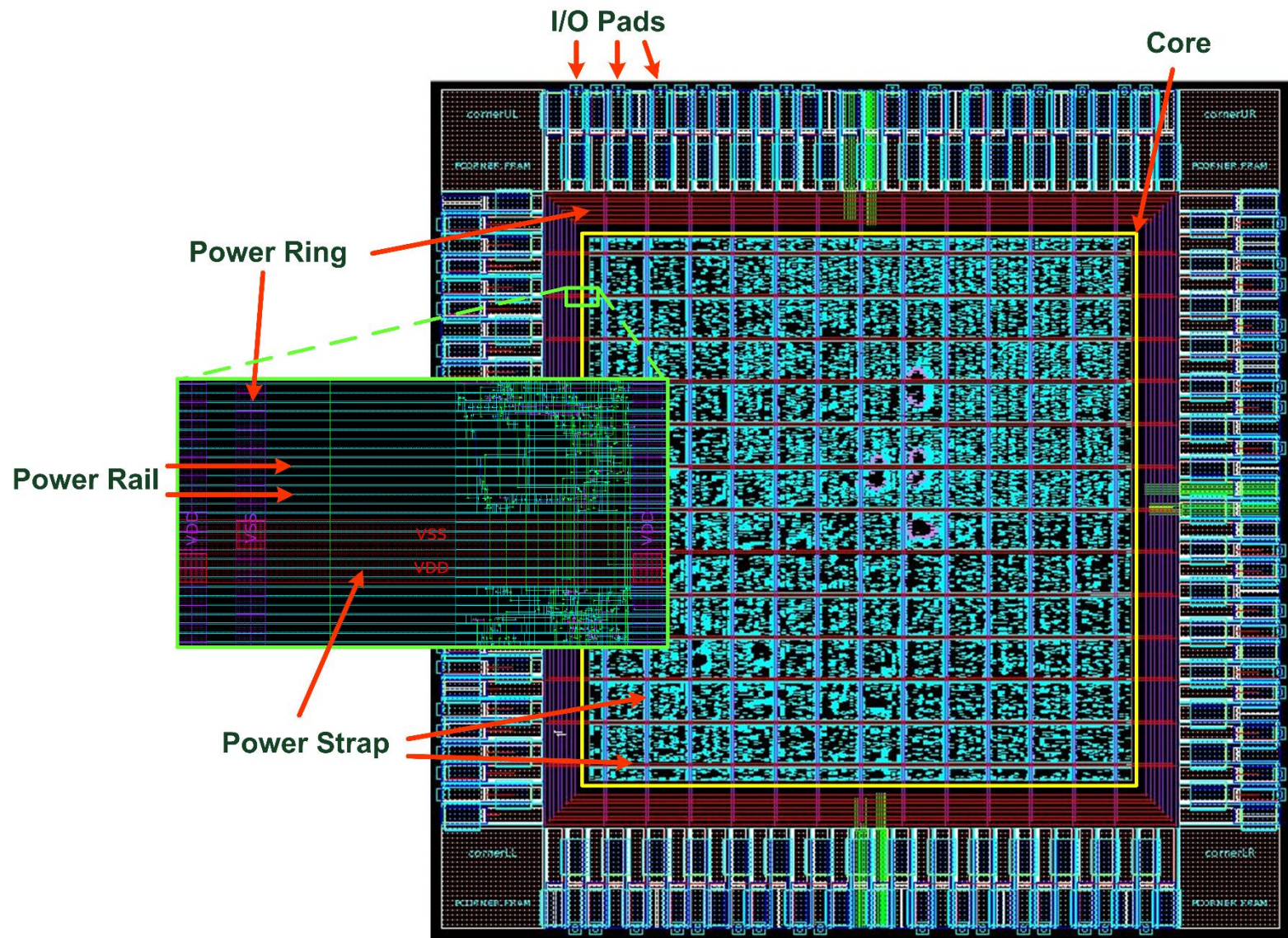
- 芯片中的形式验证是一种利用数学方法和模型检查技术来验证设计是否符合预期功能的验证方式
- 通过穷举所有可能输入组合并证明其满足特定属性，弥补传统仿真的覆盖不足和调试困难等问题





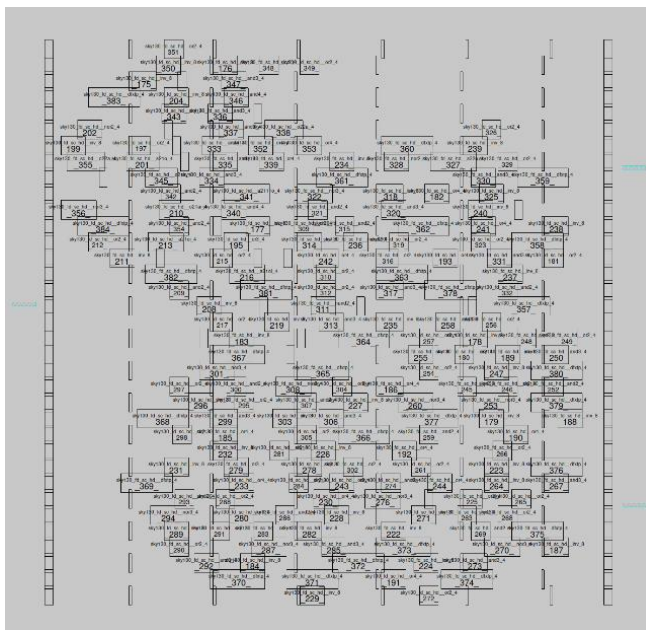
# Floorplan

- Floorplan用于确定芯片内部主要功能模块的布局框架, 为后续精确布线提供基础
- 其核心内容包括芯片尺寸规划、I/O单元布局、硬核模块分布以及特殊布线通道设定
- Floorplan的目标是优化芯片的布局, 以最小化信号延迟、提高性能并降低功耗

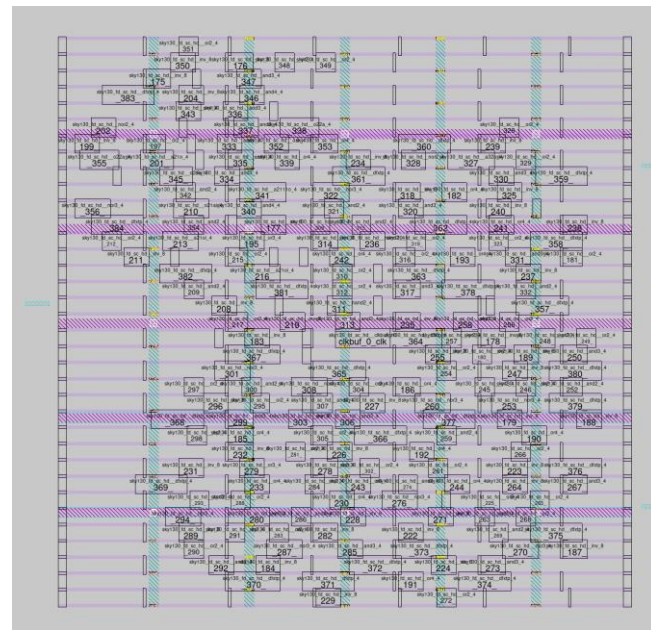


# Place and Route

- Place and Route主要涉及如何在版图图上合理地安排电路元器件的位置,并通过布线将这些元器件连接起来,以确保芯片能够正确工作



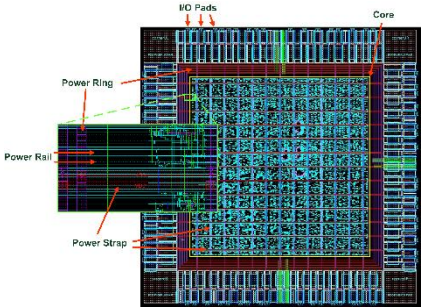
Place



Route



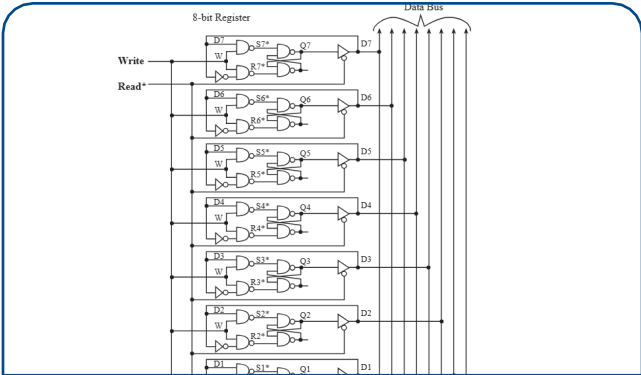
# Place



Floorplan

- No overlapping
- Correct orientation
- Timing constraints
- ...

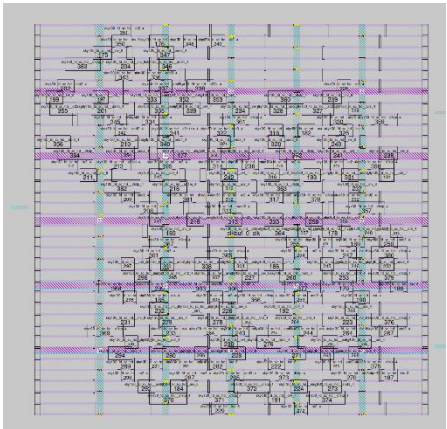
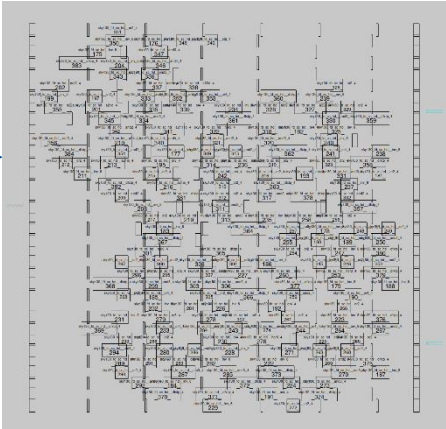
约束条件  
(Constraints)



电路网表

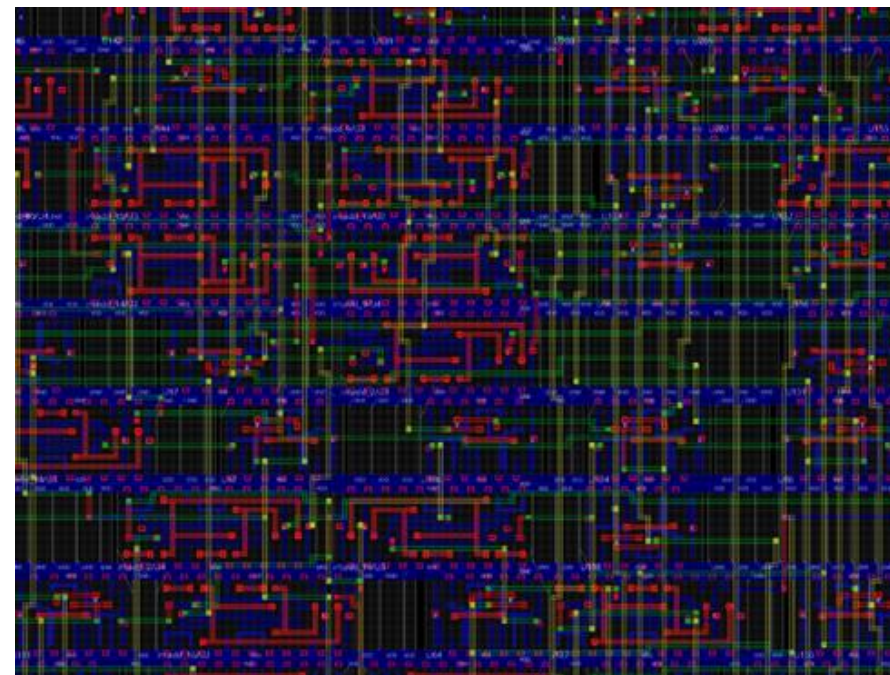
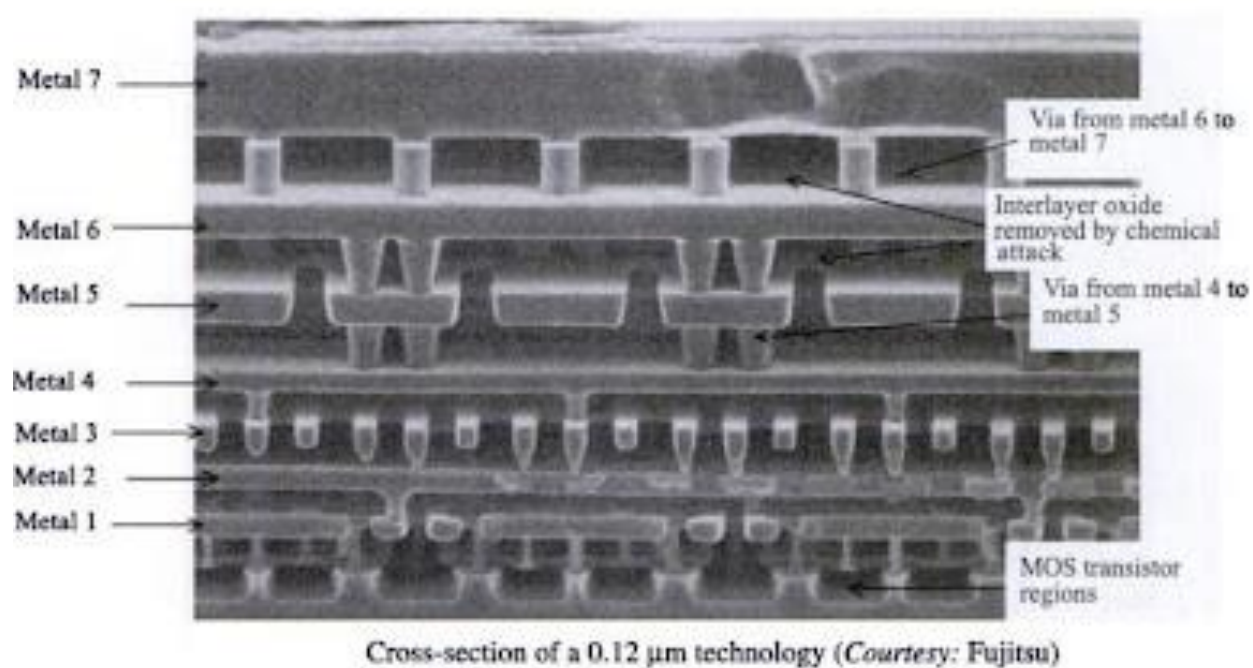
P&R Tool

Floorplan +  
Placed Std.-  
Cells



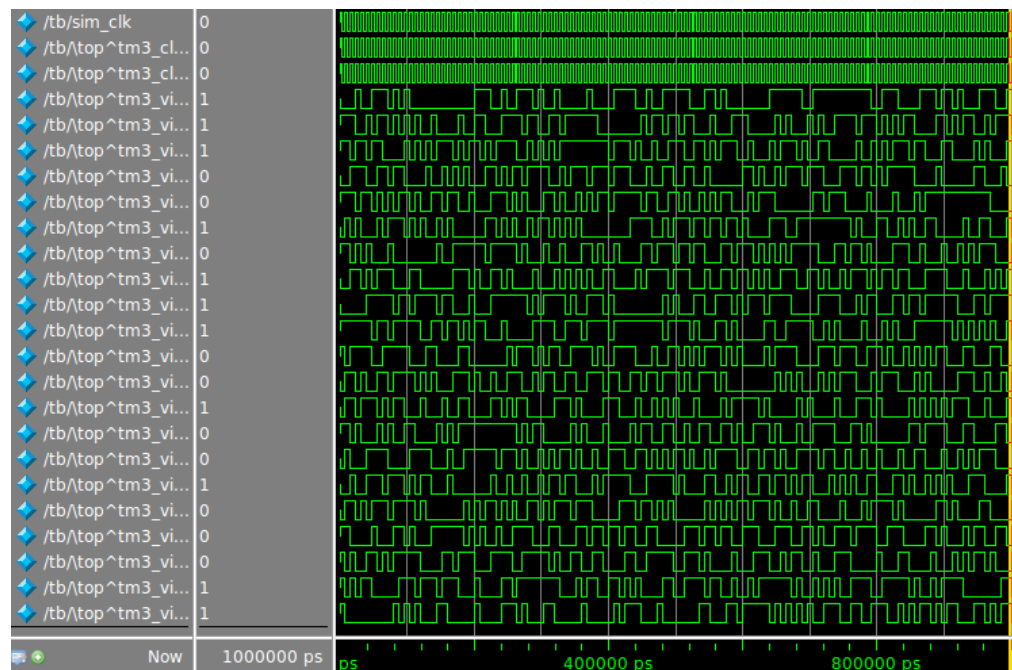
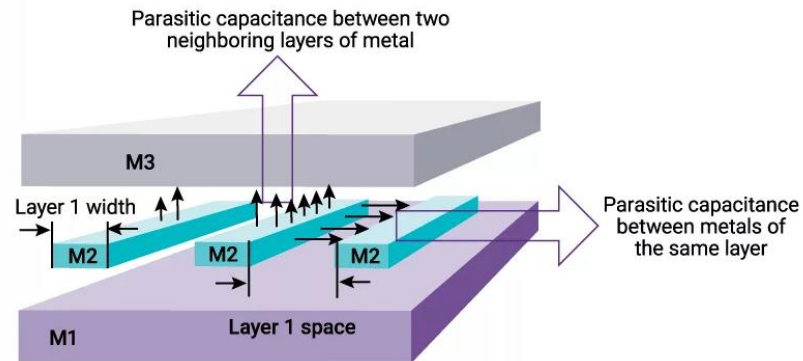
# Routing

- Routing (布线) 是将逻辑电路网表转化为物理金属连线的核心阶段
- 主要分为两大阶段：全局布线(Global Routing)和详细布线(Detailed Routing)



# 后仿真 (Post-Simulation)

- 芯片后仿真 (Post-Layout Simulation) 是集成电路设计流程中的关键验证环节，用于评估芯片在完成布局布线后的实际性能，确保时序约束和信号完整性符合设计预期
- 相较于前仿真，后仿真能更真实反映芯片在实际工作条件下的行为特性，包括时序延迟、功耗分布和温度效应等
  - 时序验证：通过提取寄生参数（如金属线延迟、门延迟、耦合电容等），验证电路在真实物理环境下的时序表现，确保信号传输满足时钟域约束
  - 功能验证：确认电路逻辑与RTL代码一致，排除因布局布线导致的功能错误
  - 信号完整性分析：检查串扰、反射、IR压降等问题，避免信号失真



# Sign-Off

逻辑和功能检查：

逻辑等效性校验 (LEC)

功能仿真

时序检查：静态时序分析 (STA)

物理检查：

设计规则检查 (DRC)

天线规则检查 (ANT)

电气规则检查 (ERC)

布局与原理图 (LVS)

ESD检查

功耗评估

电源网格检查

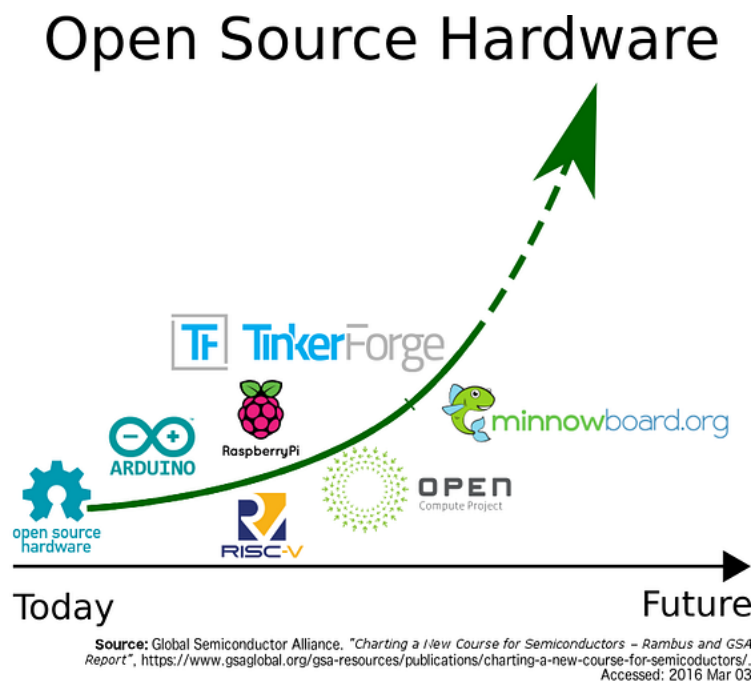
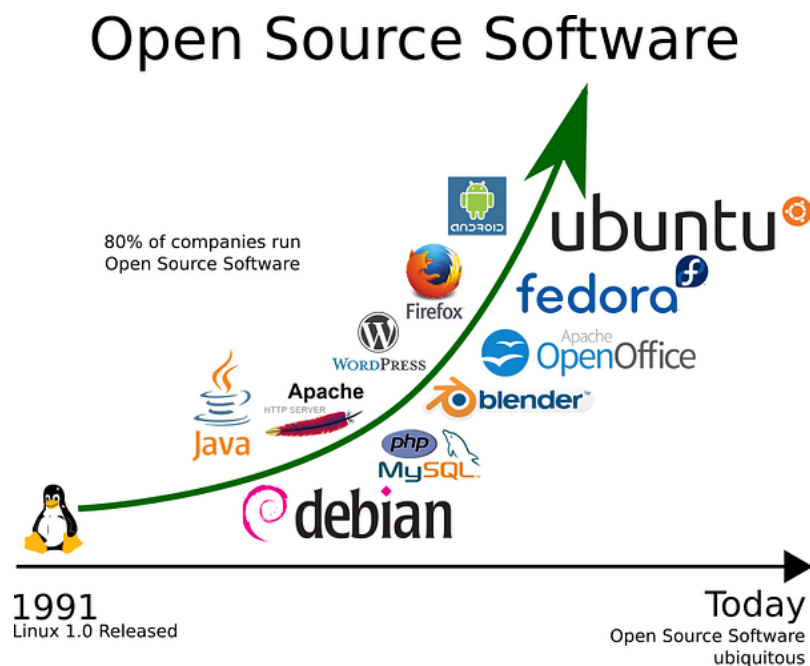
压降分析和电迁移检查 (EMIR)

| Items                                                                                               | Status | Comment |
|-----------------------------------------------------------------------------------------------------|--------|---------|
| Confirm enough isolation region (>30um) around analog IP                                            |        |         |
| Confirm IP placement and connection                                                                 |        |         |
| Confirm chip ESD capability meet target                                                             |        |         |
| Confirm Full Chip ESD Discharge Path                                                                |        |         |
| Confirm IO driver strength and SSO if needed                                                        |        |         |
| Confirm all power supply groups and voltage level                                                   |        |         |
| Confirm IP's BUS use MSB                                                                            |        |         |
| Confirm pads order is consistent with pin assignment                                                |        |         |
| Confirm sufficient connection between PG IO and core mesh for better quality on IR-drop             |        |         |
| Confirm compatible power mesh structure between top and blocks for better quality on IR-drop and EM |        |         |
| Confirm memory compiler setting on frequency, ring layer/width, mux and write mask                  |        |         |
| Confirm memory cells orientation in R0/R180 for better power mesh connection                        |        |         |
| Confirm STD cell rail connection which located between memory cells                                 |        |         |
| Confirm Verify PG Report Clean                                                                      |        |         |
| Confirm FILLTIE cell usage for latch-up concern if have                                             |        |         |
| Confirm ENDCAP cell usage for triple-well concern if have                                           |        |         |
| Confirm and report spare cells distribution                                                         |        |         |
| Confirm CLKBUF* and CLKINV* cells used in CTS                                                       |        |         |
| Confirm CLKBUF* and CLKINV* driver strength                                                         |        |         |
| Confirm no data cell in clock path                                                                  |        |         |



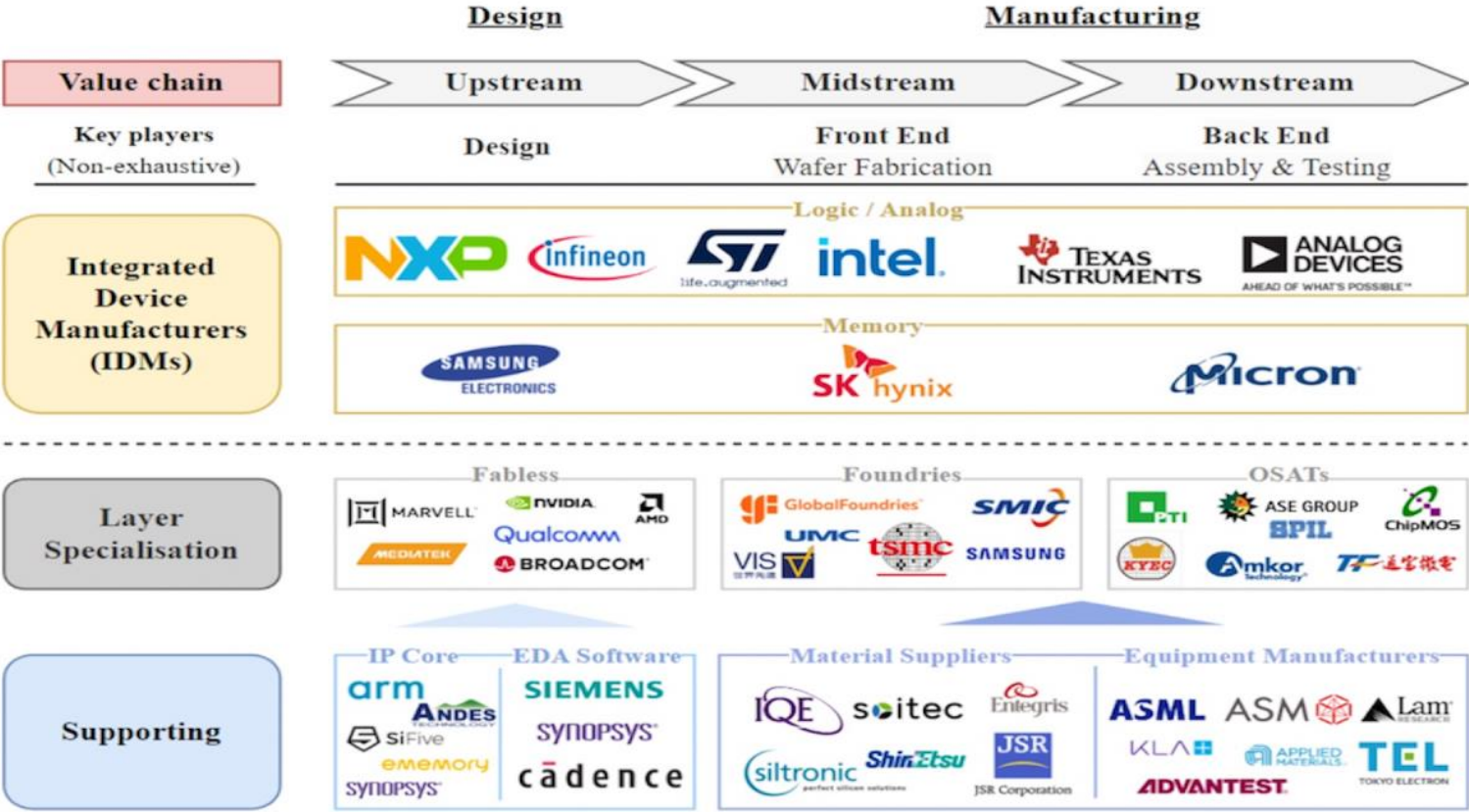
# 开源 (Open Source)

- 开源 (Open Source) 指的是事物规划为可以公开访问的, 人们可以修改并分享
- 最初是起源于软件开发中, 指的是一种开发软件的特殊形式。但到了今天, “开源” 已经泛指一组概念——就是我们称之为的 “开源的方式”。这些概念包括开源项目、硬件、产品, 或是自发倡导并欢迎开放变化、协作参与、快速原型、公开透明、精英体制以及面向社区开发的原则

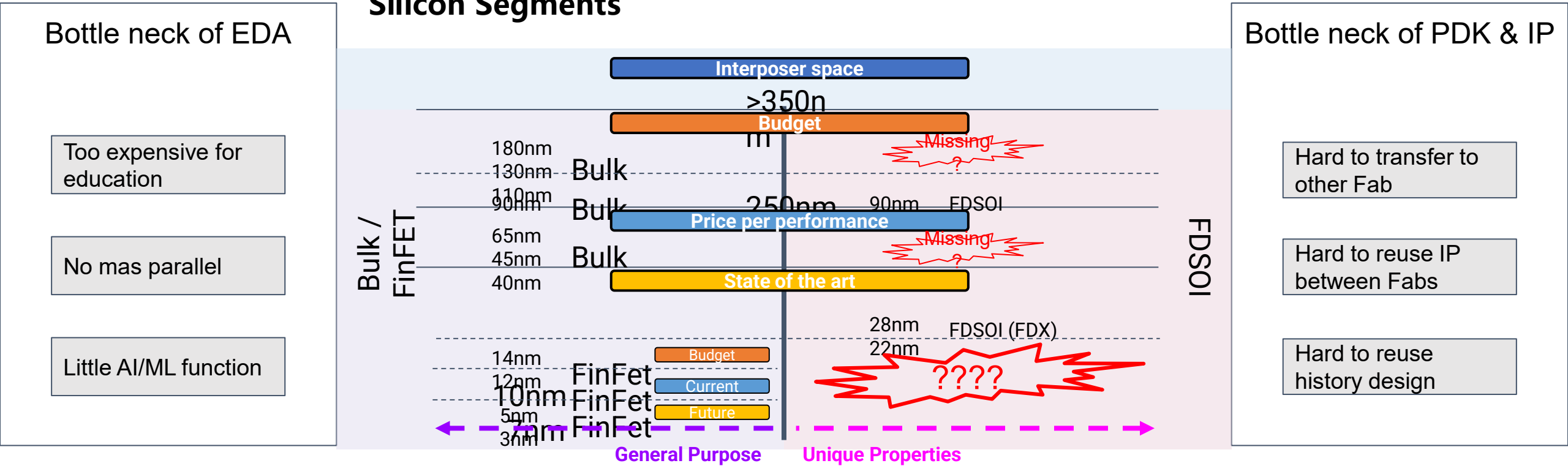


# EDA

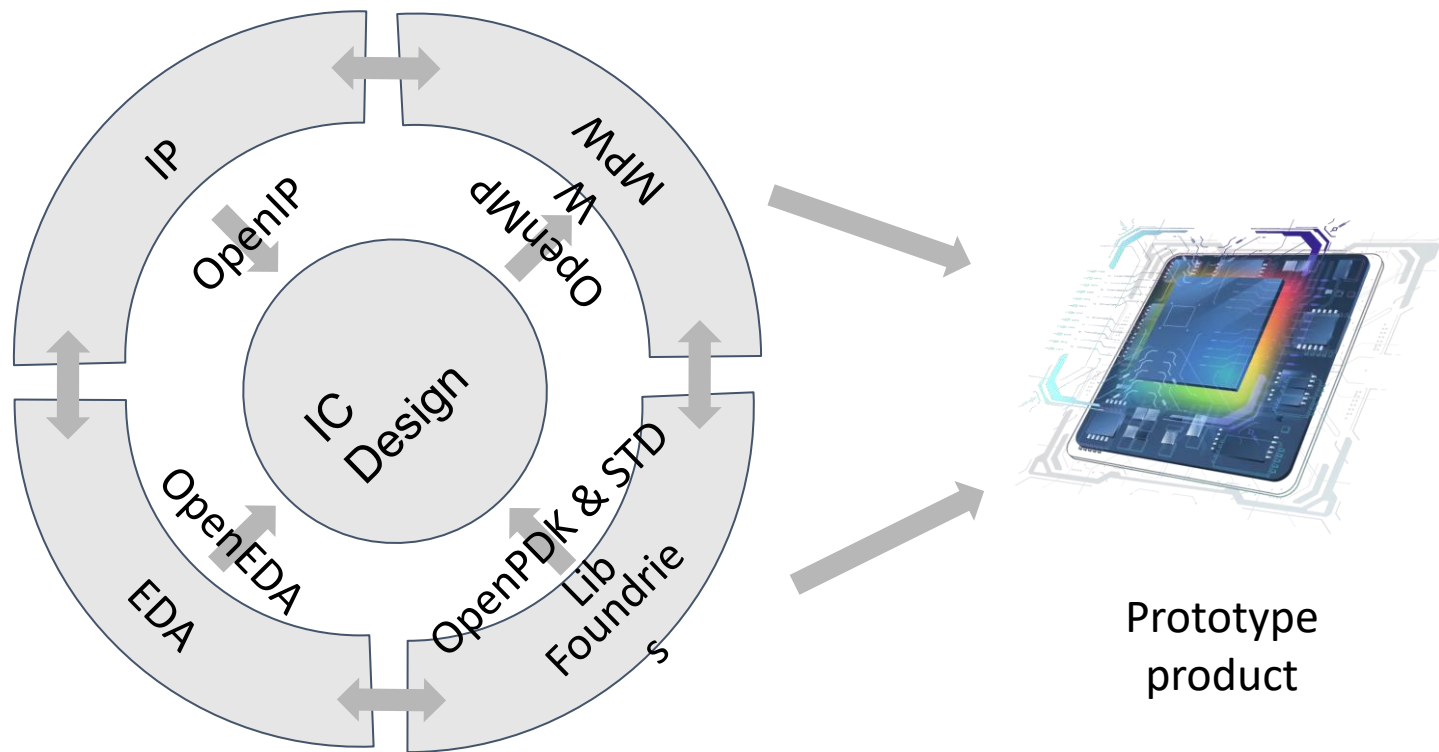
- 电子设计自动化 (Electronic Design Automation, EDA) 是指利用计算机辅助设计软件，来完成超大规模集成电路 (VLSI) 芯片的功能设计、综合、验证、物理设计等流程的设计方式
- EDA工具可以极大的提升芯片设计效率



# Bottle Neck of Nowadays IC Design



# Vision: Open Source Design Ecosystem Give More Growth Space to IC Design and Foundries



OpenPDK & STD Libs

Bridge between IC design house and foundries

OpenMPW

Attracting traffic both for OpenEDA and foundries

OpenEDA

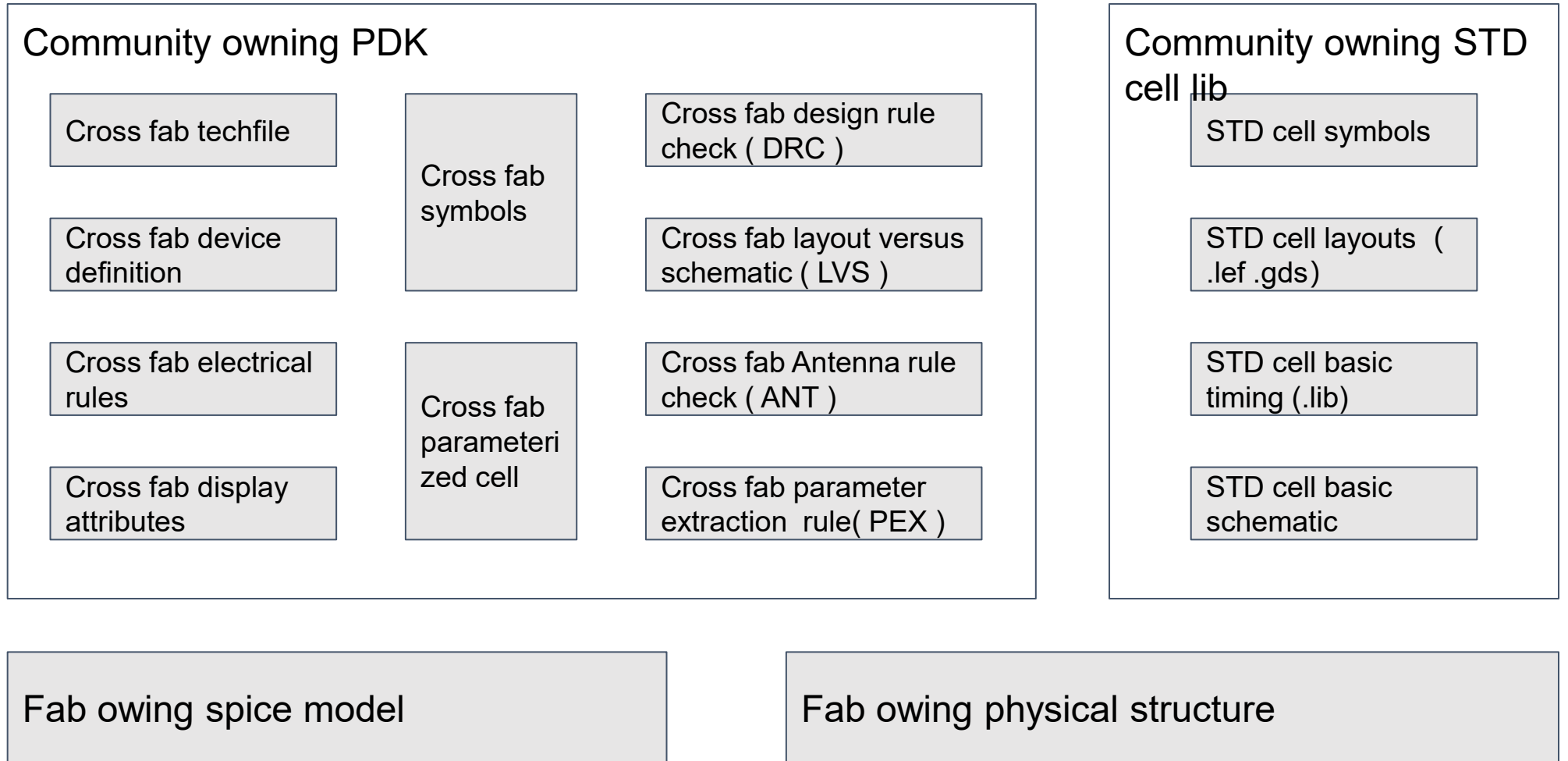
Provide Open EDA design tools to design houses and IP vendor

Prototype product



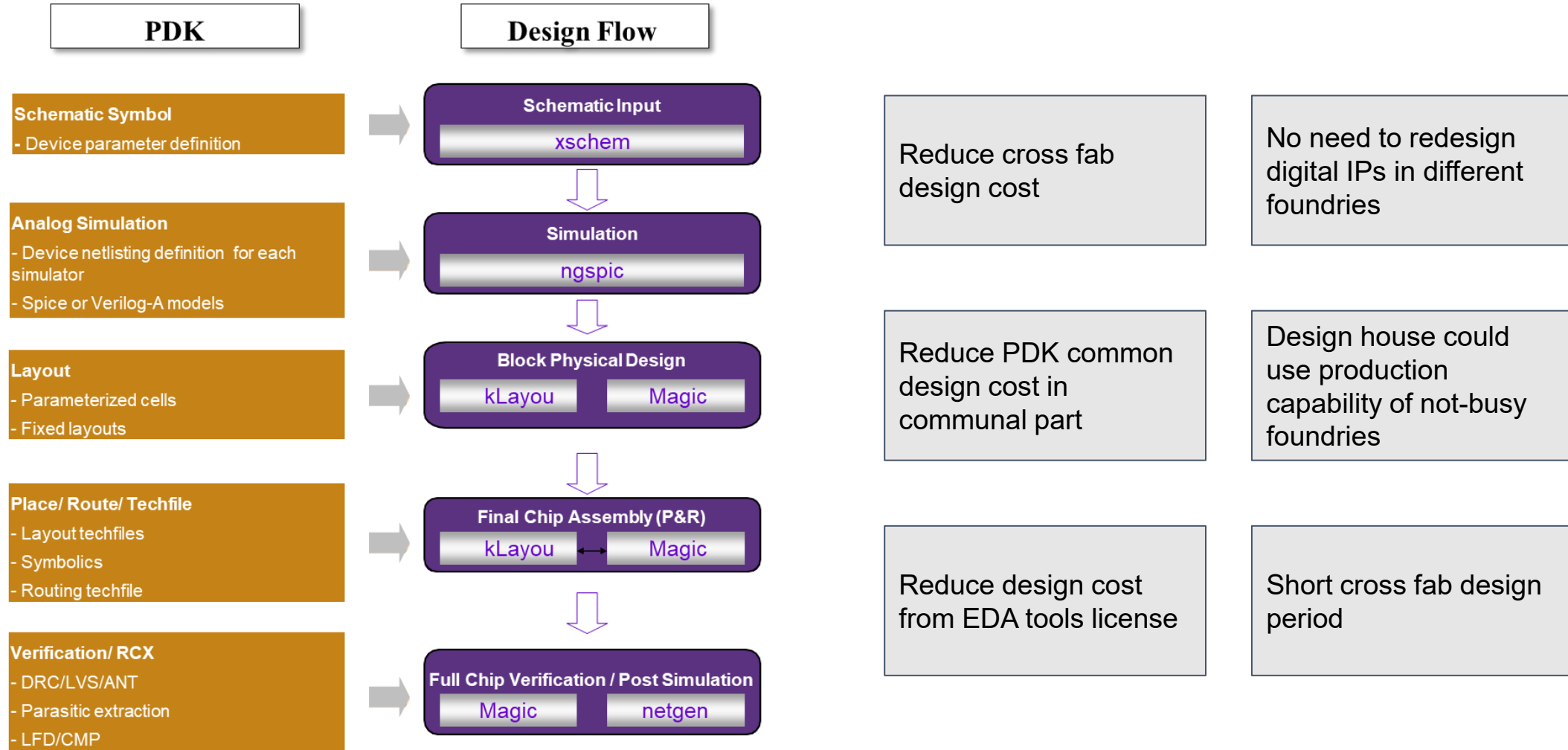
# Community Owning PDK & STD Cell Lib Will Reduce Overall Cost of Design House and Foundries

Communal part  
( Community  
owning PDK &  
STD cell lib )

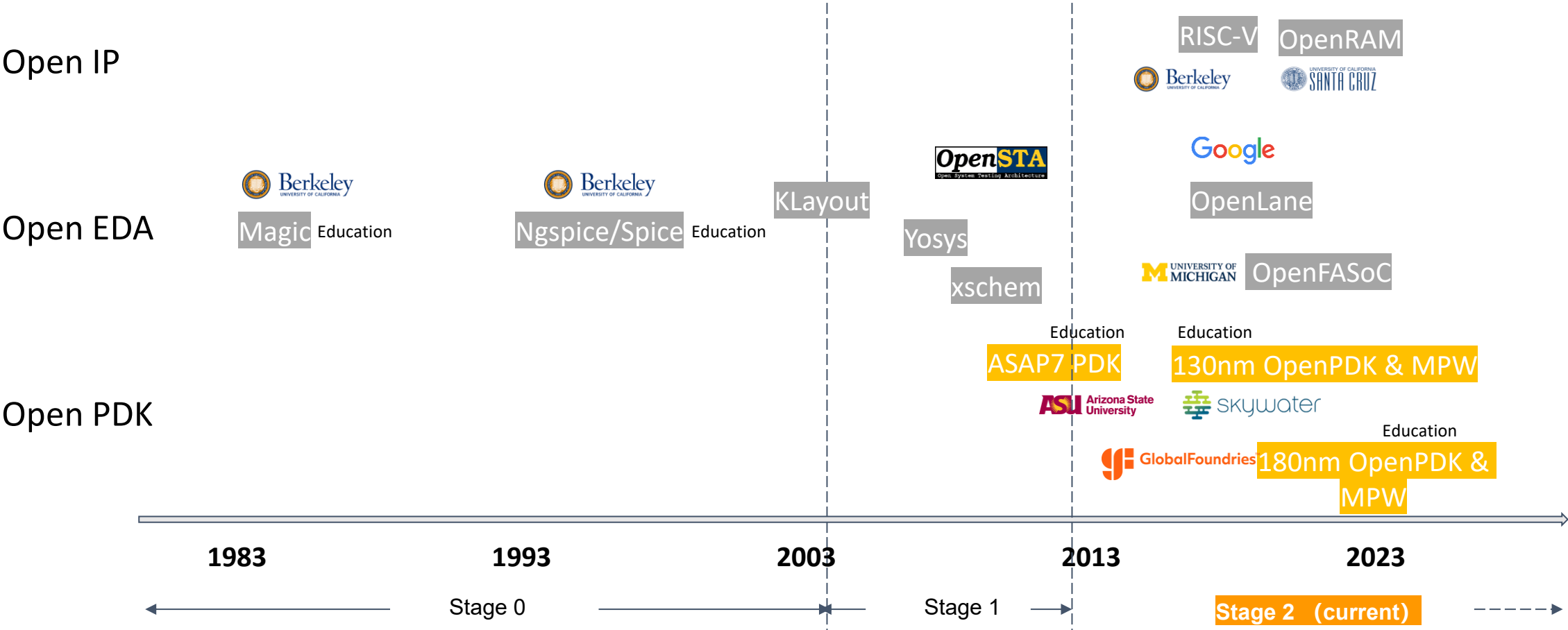


Foundry private  
part

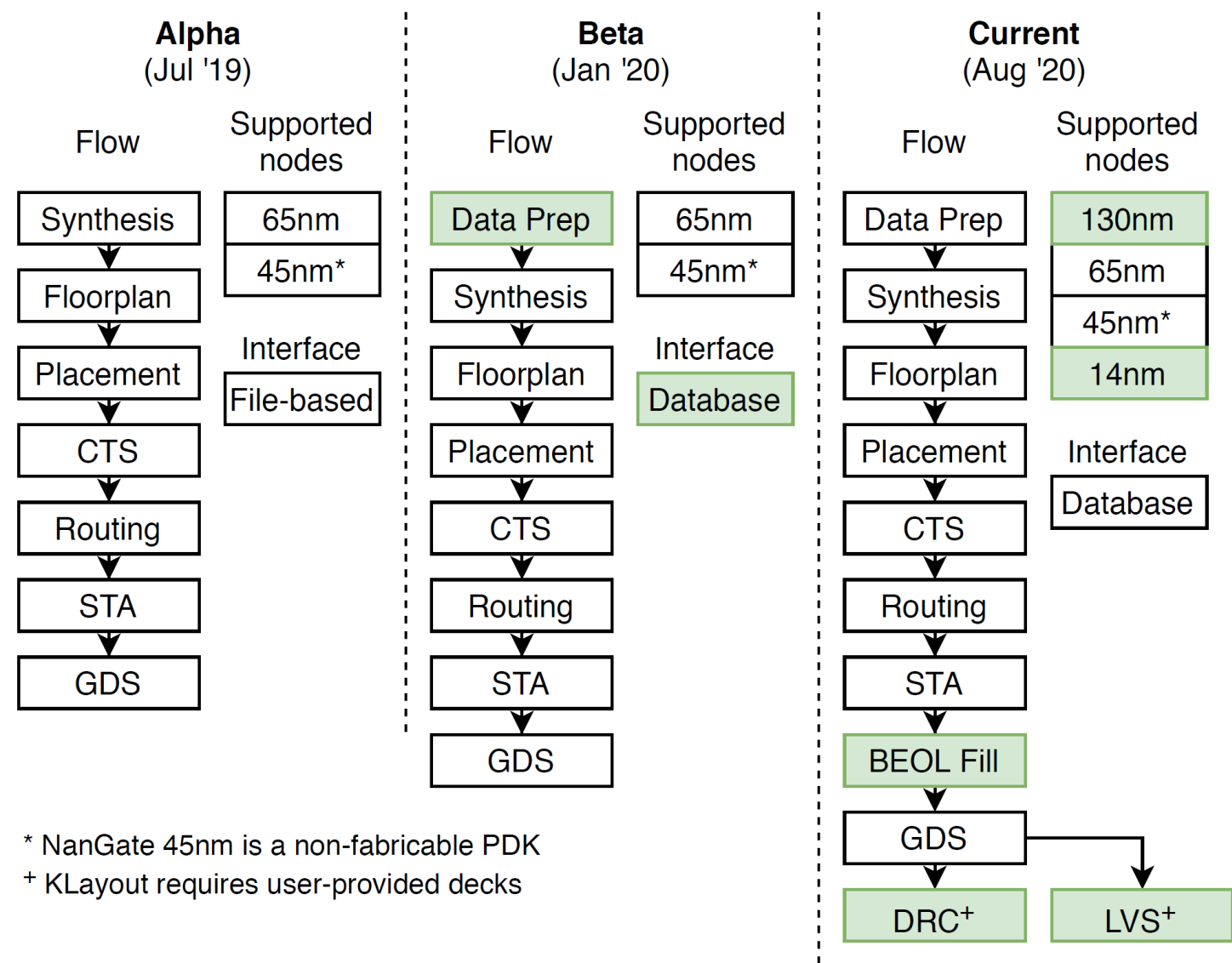
# Benefit from OpenPDK & STD Cell Lib ( OS in IC Design)



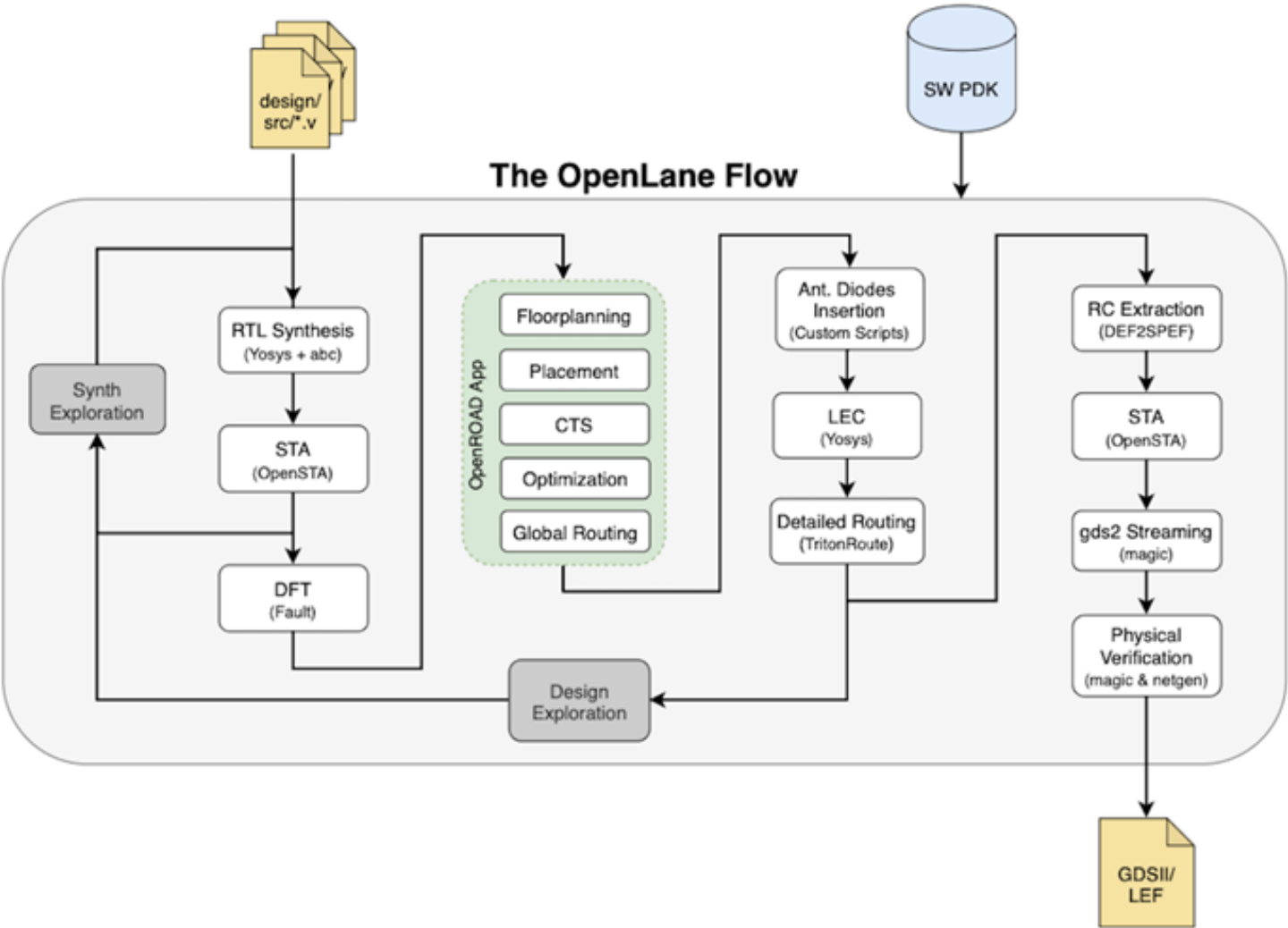
# OpenEDA Ecosystem History



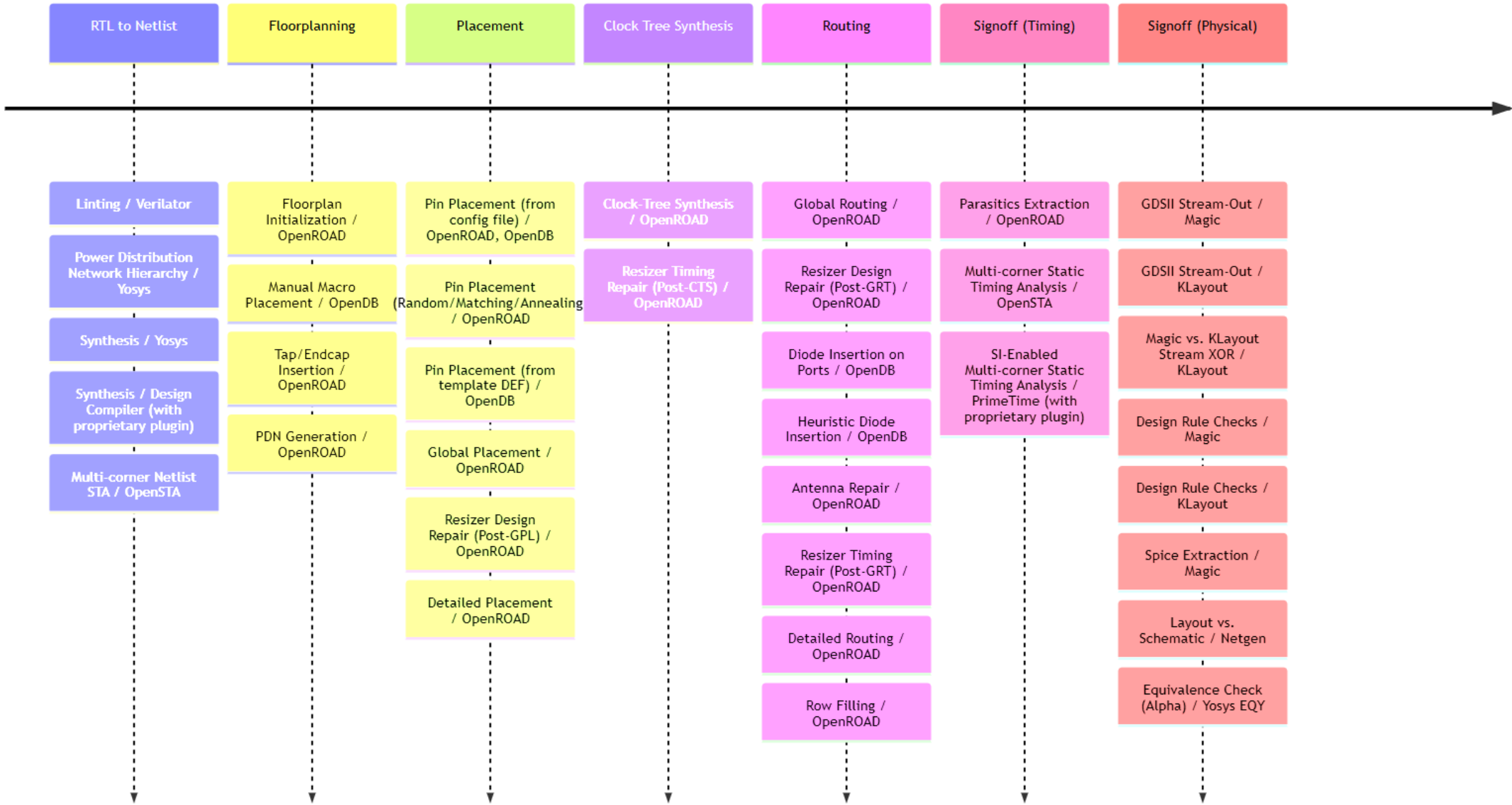
# OpenEDA的演进



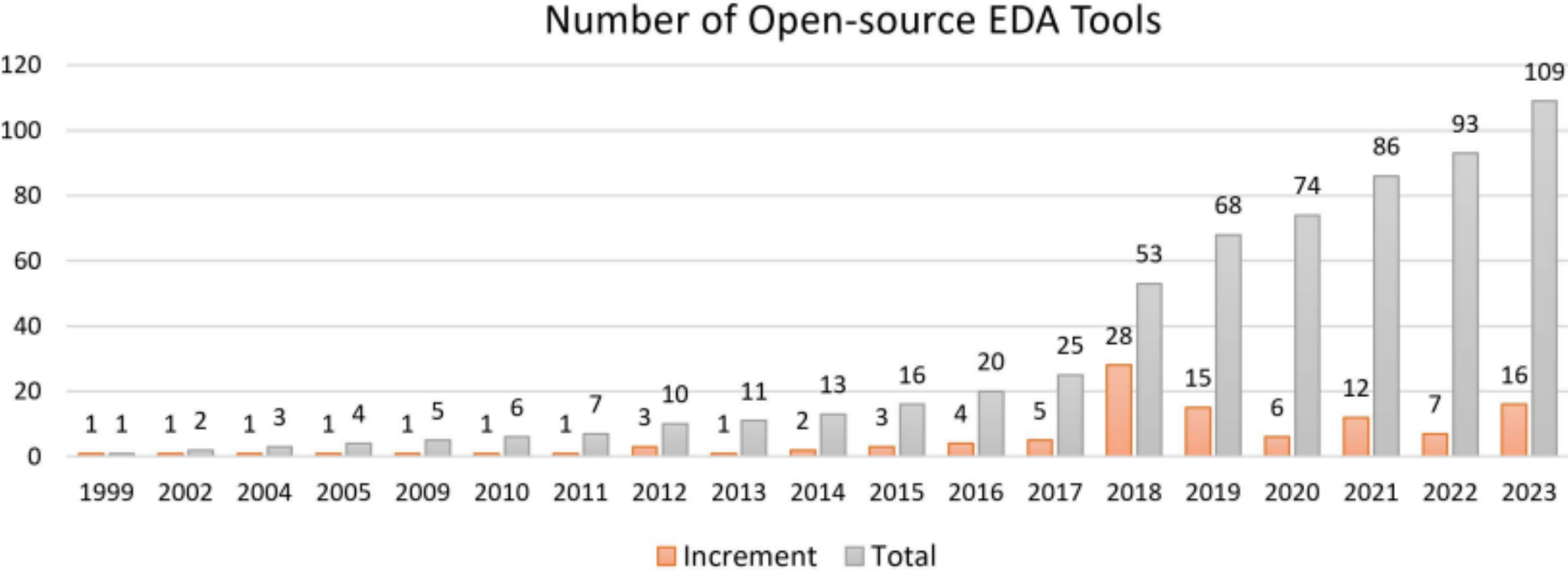
# OpenLane Flow



# OpenLane2



# 开源EDA工具的茁壮成长



# 课程基本信息

- 课程名称：基于RISC-V的开源GPU架构与设计探索
- 授课地点：深圳市学苑大道南山智园C3栋20层2005（清华大学深圳国际研究生院）
- 授课时间：十周（每周六，9:00-12:00 14:00-18:00）
- 本课程聚焦RISC-V开源架构与GPU设计的**热门交叉前沿**，构建从指令集到众核计算架构的**探索性知识体系**，以**最短路径**实现RISC-V GPU体系架构入门，并探讨3D/3.5D Chiplet、开源EDA等技术在开源GPU中的应用潜力
- 课程核心内容涵盖：RISC-V指令集架构（ISA）与GPU并行计算架构的融合设计，开源GPU核心模块（如流处理器、存储架构、TensorCore），核心编译器的原理与探索实现。通过C-Model仿真与FPGA实现，学生将掌握从架构建模、功能验证到部署实现的精简芯片设计流程，并探索开源RISC-V GPU的大模型（Transformer）应用，为进入AI芯片/GPU编程等前沿领域的研究生学习或职业发展奠定**核心竞争力与先发优势**
- <https://github.com/chenweiphd/OpenRVGPUCourse>





清華大學 深圳国际研究生院  
Tsinghua Shenzhen International Graduate School



为培养下一代图灵奖获得者传递火种