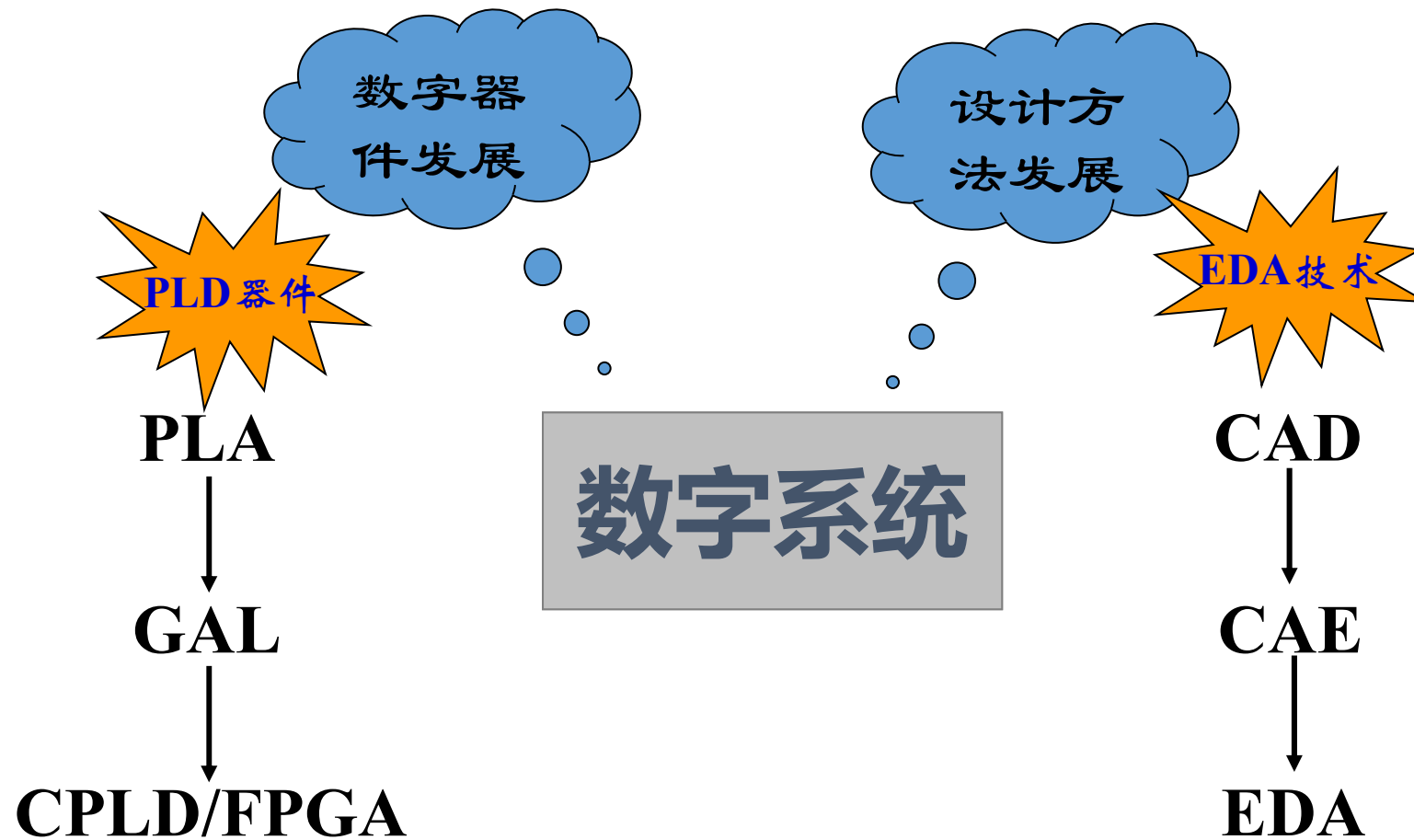
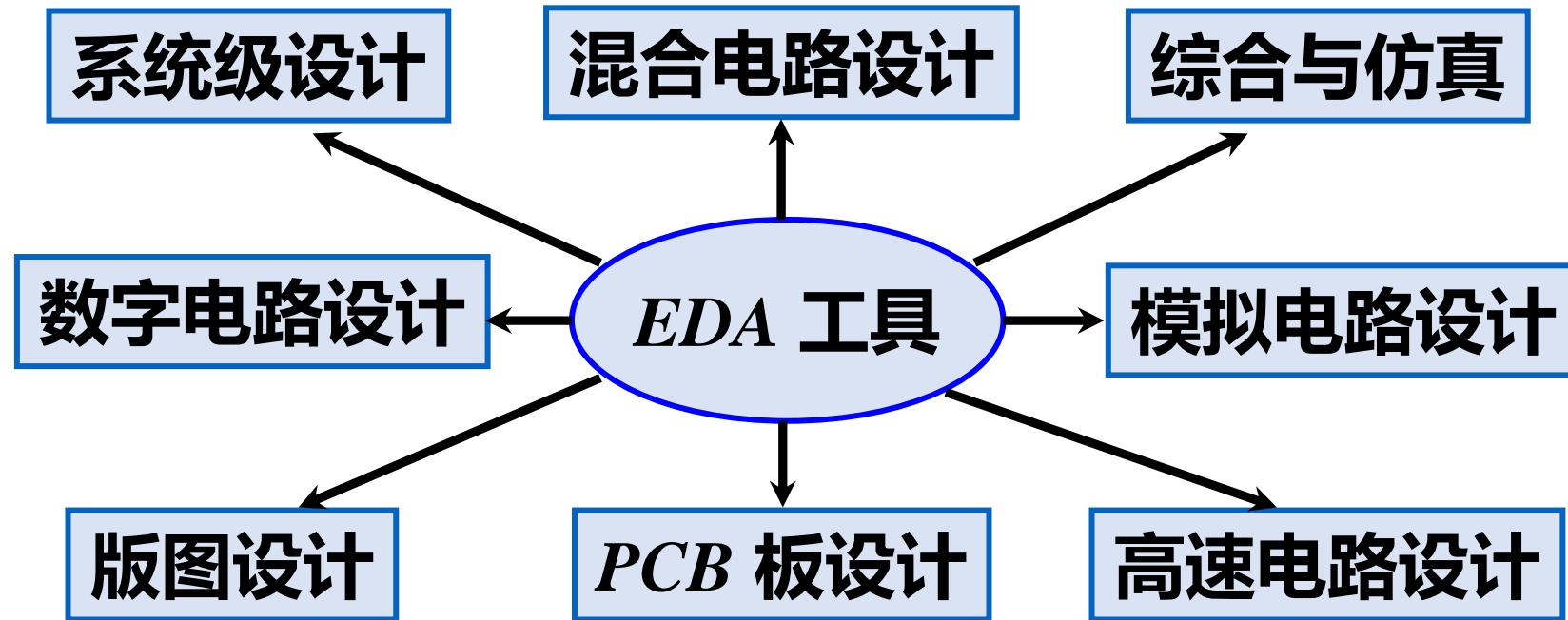


数字系统设计与 Verilog HDL

数字电路 CAD 技术的发展历史



EDA技术的功能和范畴



EDA 技术

➤ **EDA 技术：**即电子设计自动化，其依赖功能强大的计算机，在 *EDA* 工具软件平台上，对以硬件描述语言(*HDL*)为系统逻辑描述手段完成的设计文件，自动地完成逻辑编译、逻辑化简、逻辑分割、逻辑综合、布局布线，以及逻辑优化和仿真测试，直至物理实现既定的电子设计系统功能。可以说：**EDA 技术 等于 *HDL*+EDA 工具。**

**那么我们为什么要使用 *EDA* 技术，
利用其设计电子电路时有什么优点呢？我
们首先讨论一下数字集成电路的描述方法
和设计方法？**

数字集成电路的描述与设计

数字集成电路的可以从不同的**层次**（系统级、算法级、寄存器传输级、门级、开关级）以及不同的**领域**（行为领域、结构领域、物理领域）进行描述。

从设计方法学的角度来看，基于 *EDA* 技术设计中，有两种基本的设计思路（自顶向下和自底向上）。

数字集成电路的描述

三个描述领域的主要含义：

- 1. 行为领域：**描述一个设计的基本功能，或者说所设计的电路应该做什么。从概念上讲，是纯行为的描述，是输入和输出映射关系的描述。
- 2. 结构领域：**描述一个设计的逻辑结构，或者说一个设计的抽象实现。典型的表达方式是一些抽象功能模块之间相互连接的网表。
- 3. 物理领域：**描述一个设计的物理实现，或者说把结构描述中的抽象元件代之以真正的物理元件。物理域的设计常常与具体的电路工艺条件相关联。

数字集成电路的描述

五个描述层次的主要含义：

- 1. 系统级：**是针对整个数字系统性能的描述，是系统最高层次的抽象描述。系统级只描述电路的行为而不涉及电路的结构。
- 2. 算法级：**是在系统级性能分析和结构划分之后，对每个模块功能行为的描述。这一层次又称为行为层或功能层。
- 3. 寄存器传输级：**是从信号存储、传输的角度去描述整个系统的。
- 4. 门级：**是从各种逻辑门的组合、连接的角度去描述整个系统的。
- 5. 开关级：**门级中的逻辑门是由晶体管电路组成的。因此开关级描述是从晶体管的组合、连接的角度来描述整个系统的。

数字集成电路的描述

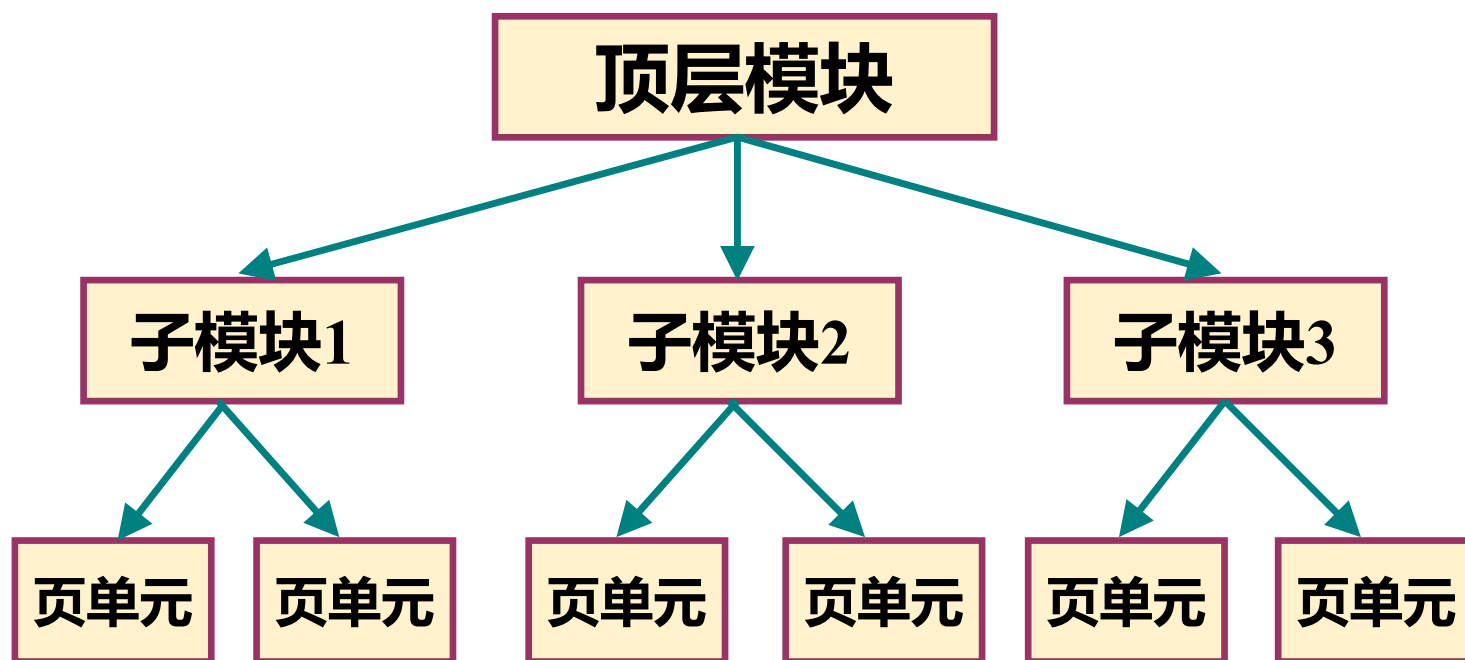
设计层次	描述领域		
	行为领域描述	结构领域描述	物理领域描述
系统级	行为、性能、输入输出映射的关系(自然语言描述)	CPU、存储器、控制器及总线之间的联系(系统逻辑方框图)	芯片、模块、电路板及子系统的物理划分
算法级	实现行为的算法	硬件模块、数据结构	部件之间的物理连接、电路板
寄存器传输级	寄存器传输、状态表、状态图等	ALU、多路选择器、寄存器、存储器、控制器等模块互连	芯片、宏单元等
门级	布尔方程、卡诺图	逻辑门、触发器、锁存器构成的逻辑图	门级单元布图
开关级	电路微分方程	晶体管、电阻、电容	版图

数字集成电路的设计方法

自顶向下的设计 (*TOP-DOWN*)：首先从整体上规划整个系统的功能和性质，然后对系统进行分割，将系统划分为各个功能模块，每个模块也可进一步细化，并借助于 EDA 技术完成到工艺的映射直至物理实现。

特点是：借助 EDA 工具，自动地实现从高层次到低层次的转换，从而使得自顶向下的过程得以实现。

自顶向下的设计 (TOP-DOWN)

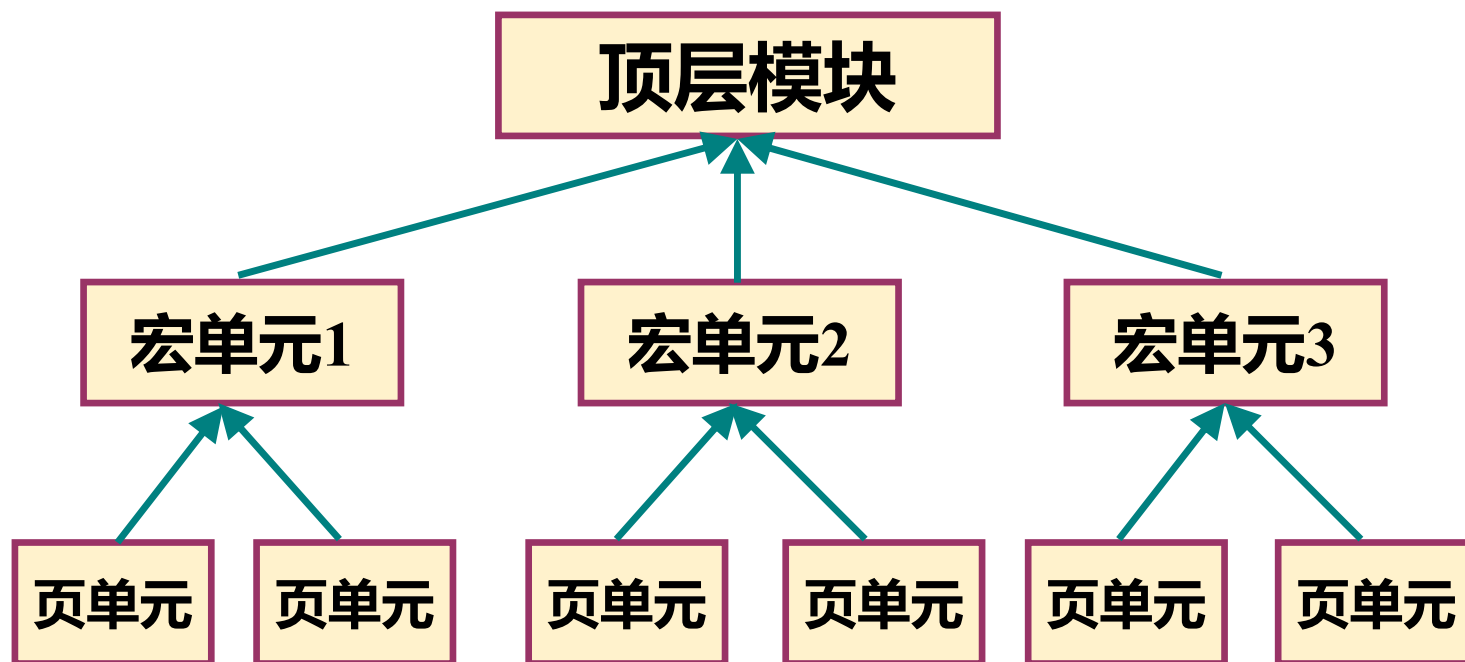


数字集成电路的设计方法

自底向上的设计 (DOWN-TOP): 这是一种传统的设计方法。即首先确定可用的元器件，然后根据器件进行逻辑设计，完成各模块后进行连接，最后形成一个完整的系统。

由于它首先进行的是低层设计，因此，缺乏对整个系统总体性能的把握。系统规模越大，复杂度越高，其缺点越突出。

自底向上的设计 (DOWN-TOP)



EDA 技术的特点

高层综合和优化：更好地支持自顶向下的设计方法，能够进行系统级的综合与优化，缩短了设计周期提高了效率。

采用硬件描述语言进行设计：更适合描述规模大的数字系统，够使设计者在比较抽象的层次上对所设计的系统结构和逻辑功能进行设计。语言的设计与工艺的无关性，便于复用、交流、保存和修改等。

开放性和标准化：采用了标准化和开放性框架结构，可实现各种 EDA 工具间的优化组合，并集成在一个易于管理的统一环境之下，实现资源共享。

传统与现代数字系统设计的比较

**传统
设计**



指集成电路制造工艺处于
微米、亚微米级时期
($0.35\mu\text{m}$) 的
设计流程。

**现代
设计**

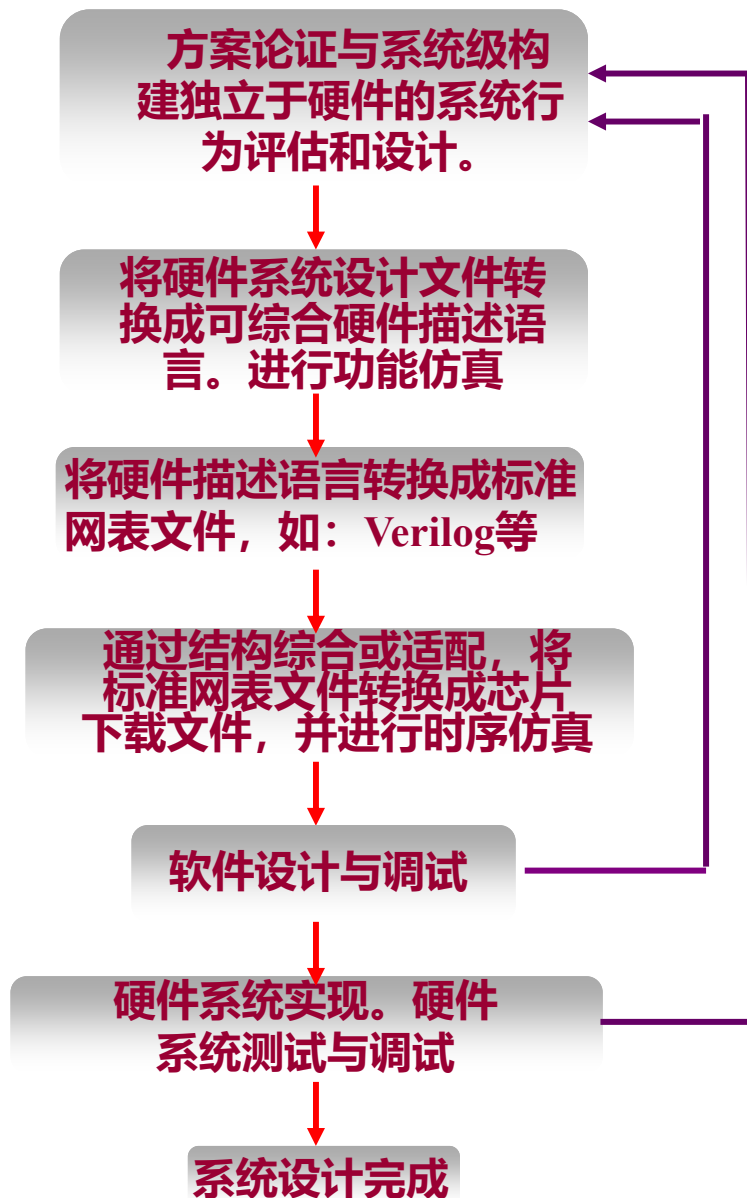


指当前集成电路制造工艺
已经达到深亚微米水平
($0.35\mu\text{m}$ 以下) 后与之相
适应的设计流程。

- 手工设计：设计者 + 纸 + 笔
- 自动设计：设计者 + EDA技术

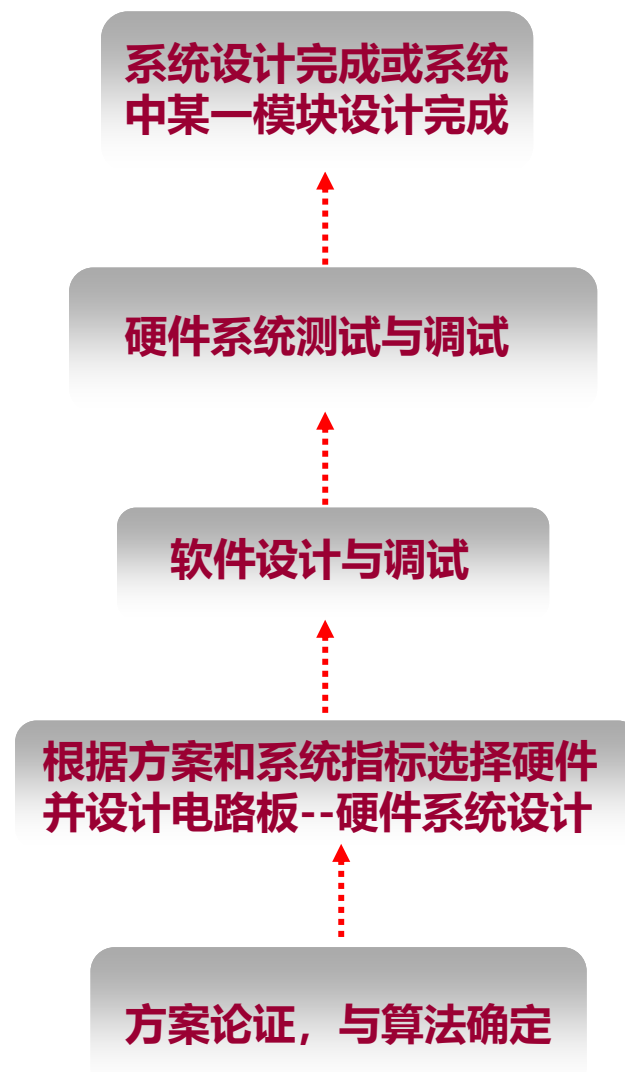
现代电子系统设计流程

自顶向下的设计流程



传统电子系统设计流程

自底向上的设计流程



传统与现代数字系统设计流程的比较

	传统集成电路设计	现代集成电路设计
设计方法	自底向上	自顶向下
设计手段	电路原理图	<i>HDL</i> 语言
系统构成	通用元器件	可编程逻辑器件
仿真调试	在设计的后期进行	在设计的早期进行
实现手段	手工实现	自动实现

硬件描述语言 HDL

□ **硬件描述语言：** 硬件描述语言是一种用形式化方法来描述数字电路和设计数字逻辑系统的语言，即用文本形式来描述电路的语言。

□ ***HDL* 的来历：** 1980年开始在美国国防部的 *VHSIC* 计划的指导下开发，完成于1983年。最初的目的是为了在各个承担国防部订货的集成电路厂商之间建立一个统一的设计数据和文档交换格式而产生的。

为什么使用硬件描述语言（HDL）？

• 使用HDL描述设计具有下列优点：

- 设计在高层次进行，与具体实现无关，设计开发更加容易
- 早在设计期间就能发现问题
- 能够自动的将高级描述映射到具体工艺实现
- *HDL* 具有更大的灵活性：可重用、可以选择工具及生产厂
- *HDL* 能够利用先进的软件，能够更快的输入、且易于管理

HDL (Hardware Description Language) 是一种用形式化方法来描述数字电路和数字逻辑系统的硬件描述语言。有两种类型：VHDL 和 Verilog HDL。

Verilog HDL语言具有这样的描述能力：设计的行为特性、设计的数据流特性、设计的结构组成，以及包含响应监控与设计验证方面的时延和波形产生机制。

Verilog HDL语言不仅定义了语法，而且对每个语法结构都定义了清晰的模拟和仿真语义。因此，用这种语言编写的模型能够使用Verilog仿真器进行验证。

VHDL侧重于系统级描述，通常被系统级设计人员所采用，Verilog侧重于电路级描述，从而更多地为电路级设计人员所采用。Verilog非常容易掌握，只需要有C语言编程基础就可很快上手。

Verilog HDL 与 VHDL的共同点

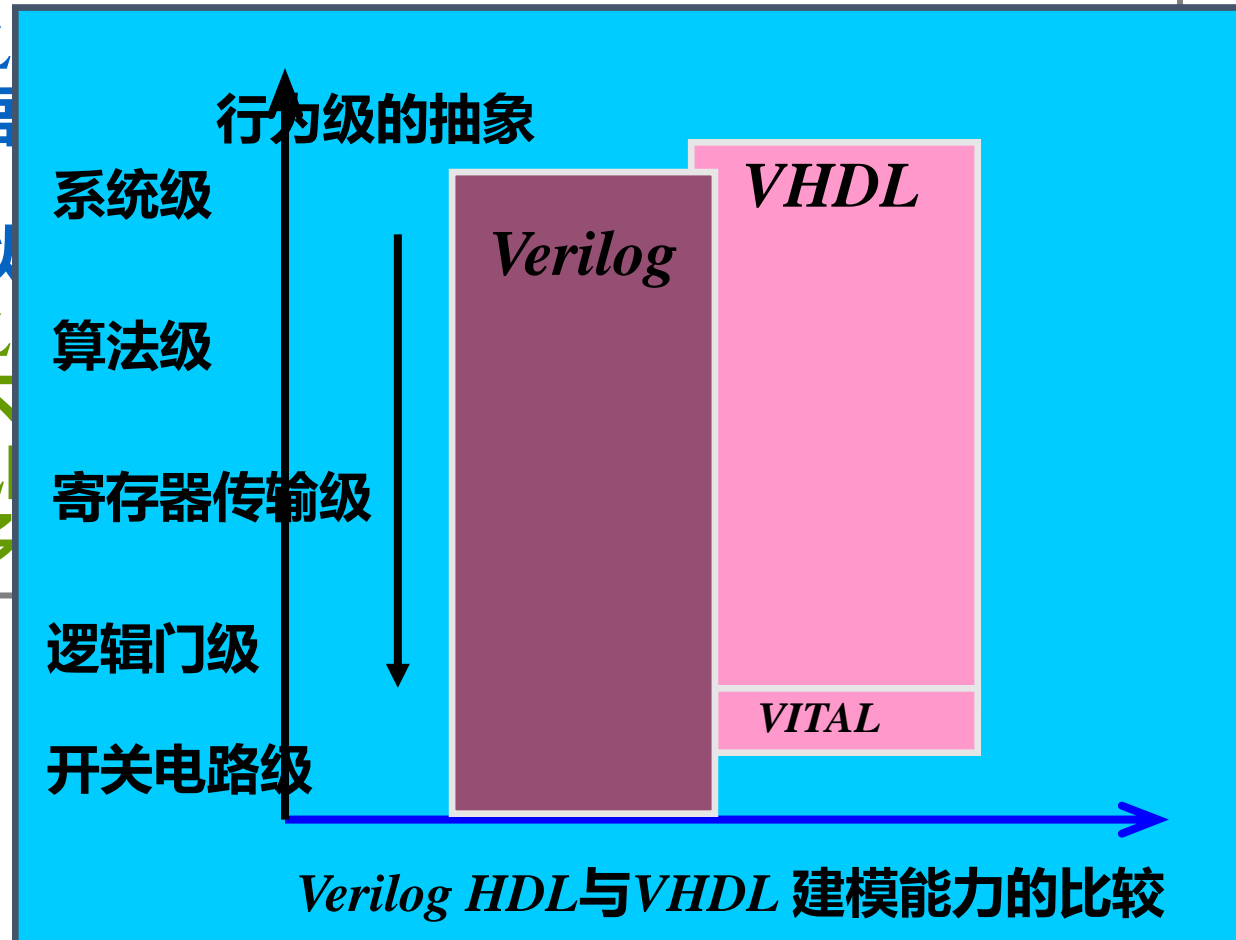
- 能形式化地抽象表示电路的结构和行为;
- 支持逻辑设计中层次描述;
- 具有电路仿真与验证机制;
- 支持电路描述由高层到低层的综合转换;
- 硬件描述与实现工艺无关;
- 便于文档管理、易于理解和设计重用。

Verilog HDL 与 VHDL的不同点

□ *Verilog HDL* 拥有更广泛的设计群体，成熟的资源也远比*VHDL*丰富。

□ *Verilog HDL* 的硬件描述语言发展而来的。语言有很多相似

□ *Verilog HDL* 方面也有所不同，*Verilog HDL* 方面比*VHDL*强得多



什么是verilog HDL语言

Verilog HDL 是硬件描述语言的一种，用于数字电子系统设计。设计者可用它进行各种级别的逻辑设计，可用它进行数字逻辑系统的仿真验证、时序分析、逻辑综合。它是目前应用最广泛的一种硬件描述语言。

Verilog与C语言的比较

(1) Verilog是一种硬件语言，最终是为了产生实际的硬件电路或对硬件电路进行仿真；而C语言是一种软件语言，是控制硬件来实现某些功能的语言。

(2) C语言只要是语法正确，都可以编译执行；而Verilog语言有可综合的限制，即在所有的Verilog语句中，只有一部分可以被综合，而另外的部分则不能被综合，只能用来仿真。

(3) C语言是一种软件编程语言，其基本思想是语句的循序执行；而Verilog语言的基本思想是模块的并行执行。

(4) 利用Verilog编程时，要时刻记得Verilog是硬件语言，要时刻将Verilog与硬件电路对应起来。

常用的C语言与Verilog语言相对应的运算符

C语言	Verilog语言
sub-function	module, function, task
if-then-else	if-then-else
Case	Case
{,}	begin, end
For	For
While	While
Break	Disable
Define	Define
Int	Int
Printf	monitor, display, strobe

C语言	Verilog语言	功 能
*	*	乘
/	/	除
+	+	加
-	-	减
%	%	取模
!	!	反逻辑
&&	&&	逻辑且
		逻辑或
>	>	大于
<	<	小于
>=	>=	大于等于
<=	<=	小于等于
==	==	等于
!=	!=	不等于
~	~	位反相
&	&	按位逻辑与
		按位逻辑或
^	^	按位逻辑异或
~^	~^	按位逻辑同或
>>	>>	右移
<<	<<	左移
?:	?:	同等于if-else叙述

Verilog HDL 的起源与发展

- 1、**Verilog HDL 是在1983年由 GDA (GateWay Design Automation) 公司的 Phil Moorby 所创。Phi Moorby后来成为 Verilog-XL 的主要设计者和 Cadence 公司的第一个合伙人。**
- 2、**在1984~1985年间, Moorby 设计出了第一个 Verilog-XL 的仿真器。**
- 3、**1986年, Moorby 提出了用于快速门级仿真的XL算法。**
- 4、**1990年, Cadence 公司收购了 GDA 公司。**
- 5、**1991年, Cadence 公司公开发表Verilog HDL语言, 成立了OVI (Open Verilog International) 组织来负责 Verilog HDL 语言的发展。**
- 6、**1995年制定了Verilog HDL 的IEEE标准, 即 IEEE1364。**

Verilog HDL设计复杂数字电路的优点

• *Verilog HDL* 设计法

- 采用*Verilog HDL*设计电路的逻辑功能容易理解;
- 把逻辑设计与具体电路的实现分成两个独立的阶段来操作;
- 逻辑设计与实现的工艺无关;
- 逻辑设计的资源积累可以重复利用;
- 可以由多人共同更好更快地设计非常复杂的逻辑电路（几十万门以上的逻辑系统）。

Verilog HDL设计复杂数字电路的优点

- **硬件描述语自身就是设计规格书。**
- **可以在设计初期发现错误。**
- **可以进行仿真。而且，仿真可以在电路系统不同的层次进行。**
- **设计的文档化。**

可以大大提高硬件的生产效率

HDL设计复杂数字电路的优点

- **HDL 软核、固核和硬核的重用：**
 - **软核：**把功能经过验证的、可综合的、实现后电路结构总门数在5000门以上的**Verilog HDL模型**称之为“软核”(Soft Core)。把由软核构成的器件称为虚拟器件。软核和虚拟器件的重用性就可大大缩短设计周期，加快复杂电路的设计。
 - **固核：**把在某一种**FPGA器件上实现的**，经验证是正确的，总门数在5000门以上**电路结构编码文件**，称为“固核”。

HDL软核、固核和硬核的重用

• **硬核：**把在某一种ASIC器件上实现的，经验证是正确的，总门数在5000门以上的电路结构掩膜，称为“硬核”。

在逻辑设计阶段，软核具有最大的灵活性，它可以很容易地借助EDA综合工具与其它外部逻辑结合为一体。相比之下固核和硬核与其它外部逻辑结合为一体的灵活性要差得多，特别是电路实现工艺技术改变时更是如此。

HDL 的应用方面

- *ASIC* 和 *FPGA* 设计师可用它来编写可综合的代码。
- 描述系统的结构，做高层次的仿真。
- 验证工程师编写各种层次的测试模块对具体电路设计工程师所设计的模块进行全面细致的验证。
- 库模型的设计：可以用于描述 *ASIC* 和 *FPGA* 的基本单元 (*Cell*) 部件，也可以描述复杂的宏单元 (*Macro Cell*) 。

数字系统设计实现

数字系统的实现一般可选择两种器件：

- **基于可编程逻辑 (PLD) 器件**

- 以 *CPLD* 和 *FPGA* 为代表
- 用户可以对它进行编程来实现所需功能，并可以反复修改、反复编程。
- 其它方法无法比拟的方便、灵活

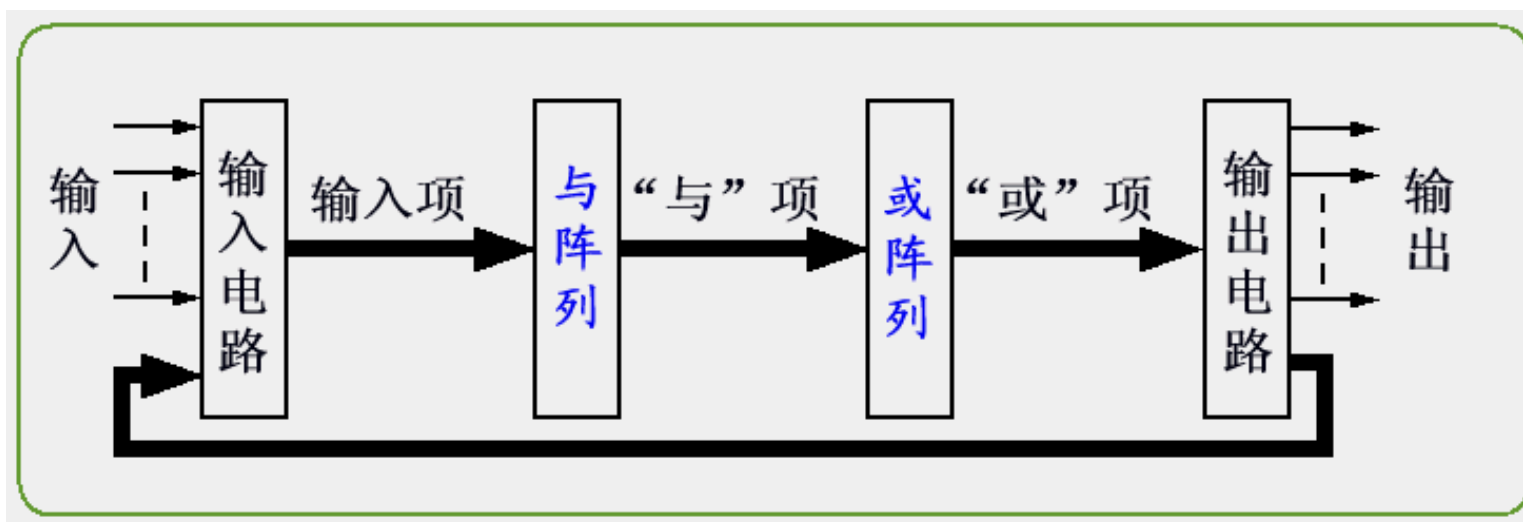
- **专用集成电路 (ASIC)**

- 用全定制方法来实现，在最底层，即物理版图级实现设计。
- 高速度、低功耗、省面积设计周期长、成本高适用于要求高或批量很大的芯片。

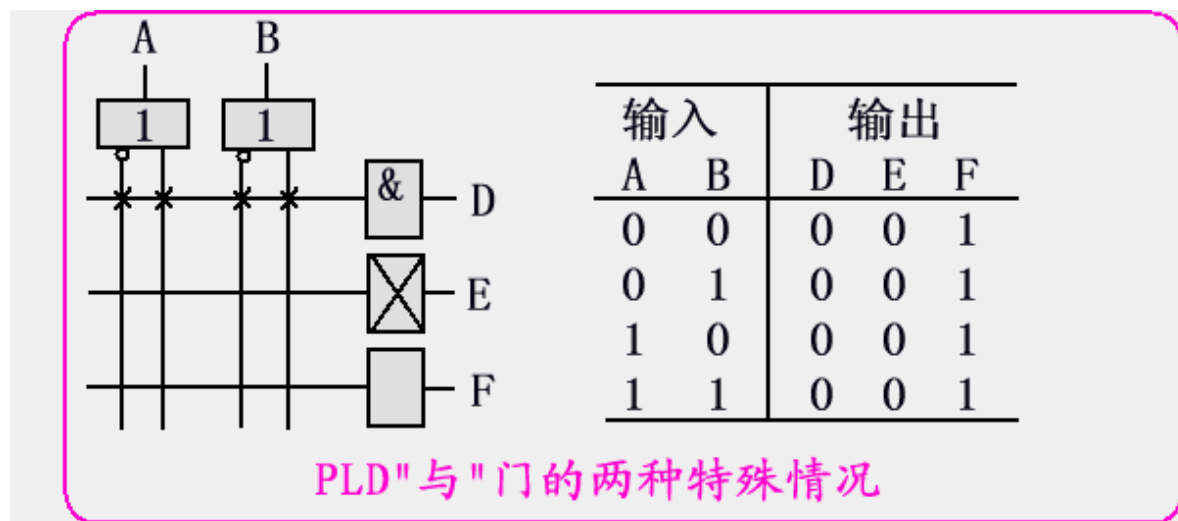
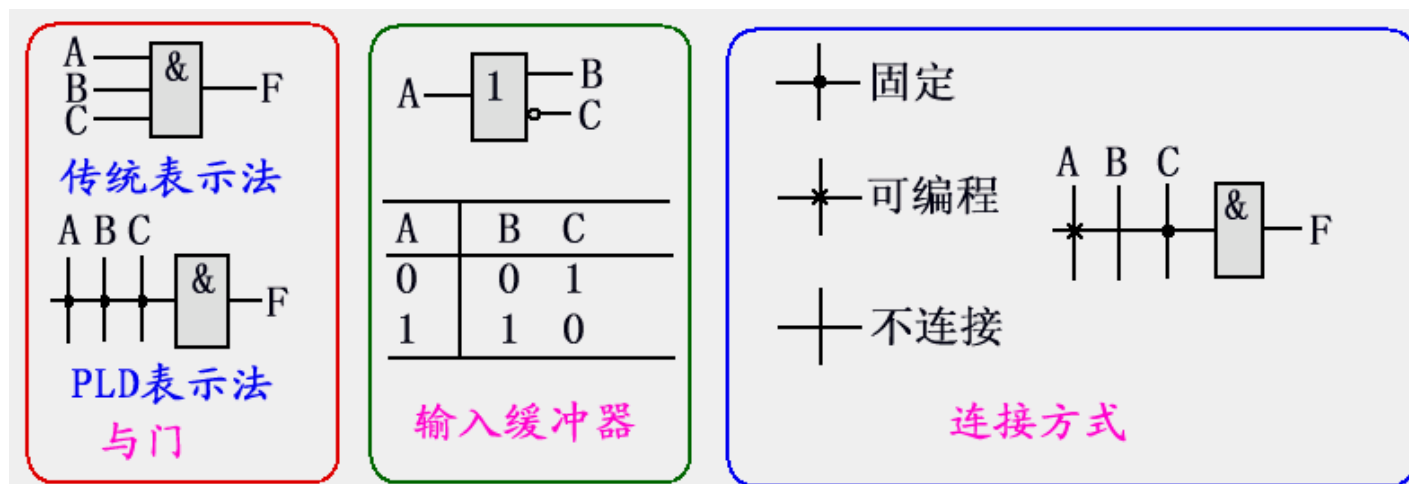
可编程逻辑器件
---为什么硬件可以软件化？
为什么可以“编程”

PLD的基本结构

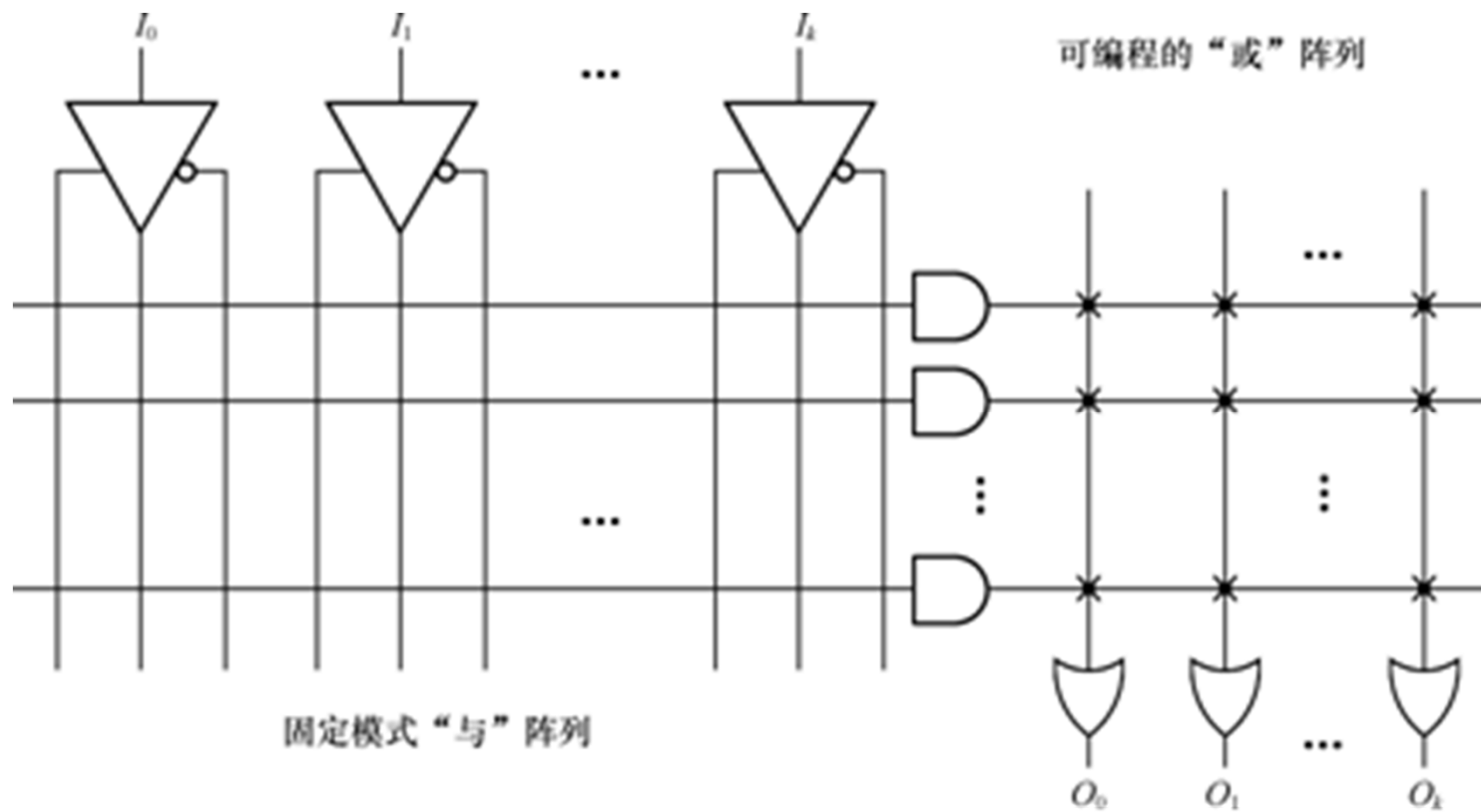
- 任何组合逻辑函数均可以转换成“与-或”式，任何时序逻辑电路都是由组合电路加上存储元件(触发器)构成的，这就是PLD最基本的依据。下图是PLD的基本结构框图。



PLD逻辑表示方法



PROM



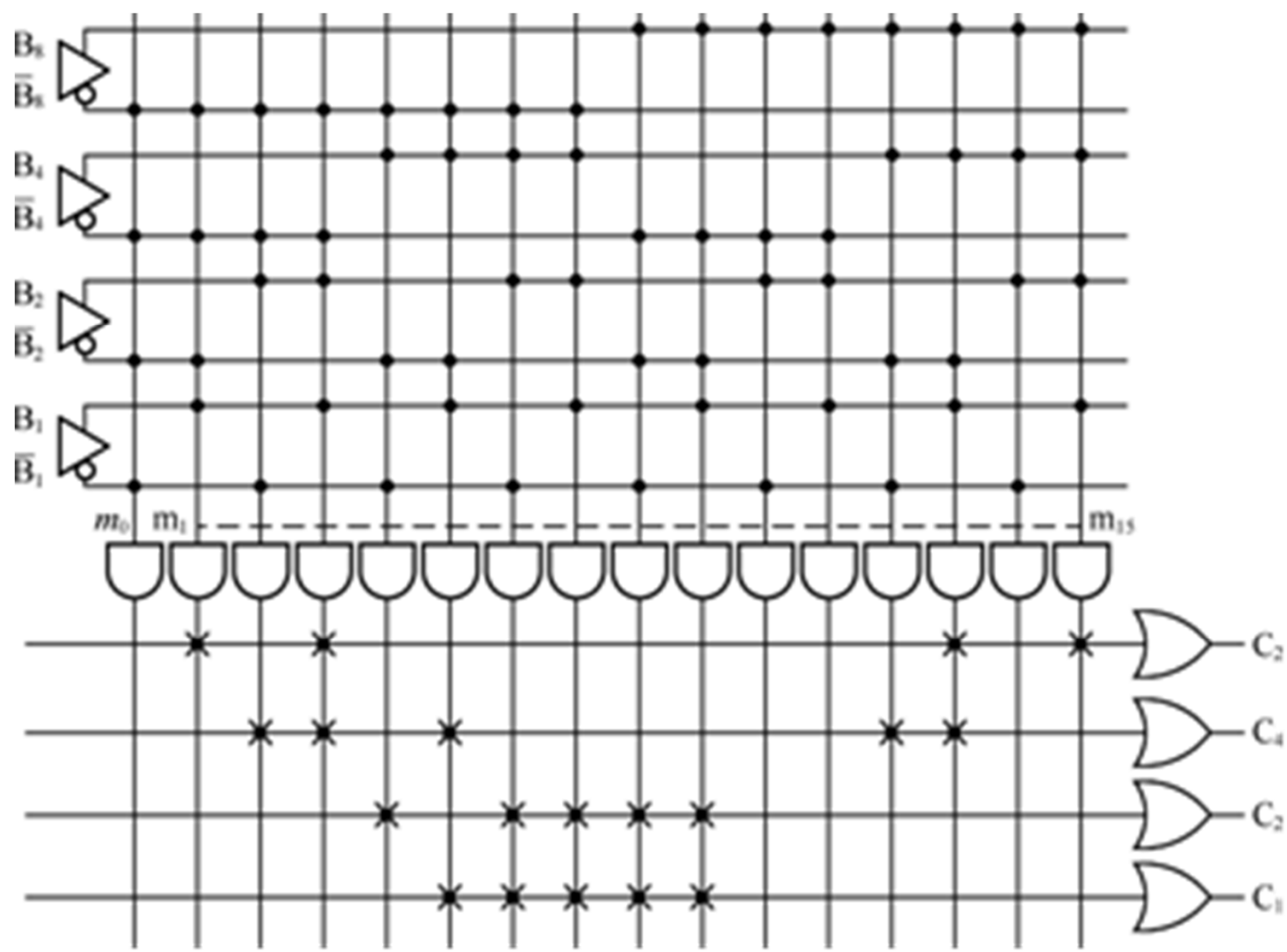
产生所有最小项，再来组合（根据输出函数个数确定或门个数）

例4.2 用PROM器件设计一个由8421码到2421码的代码转换电路。

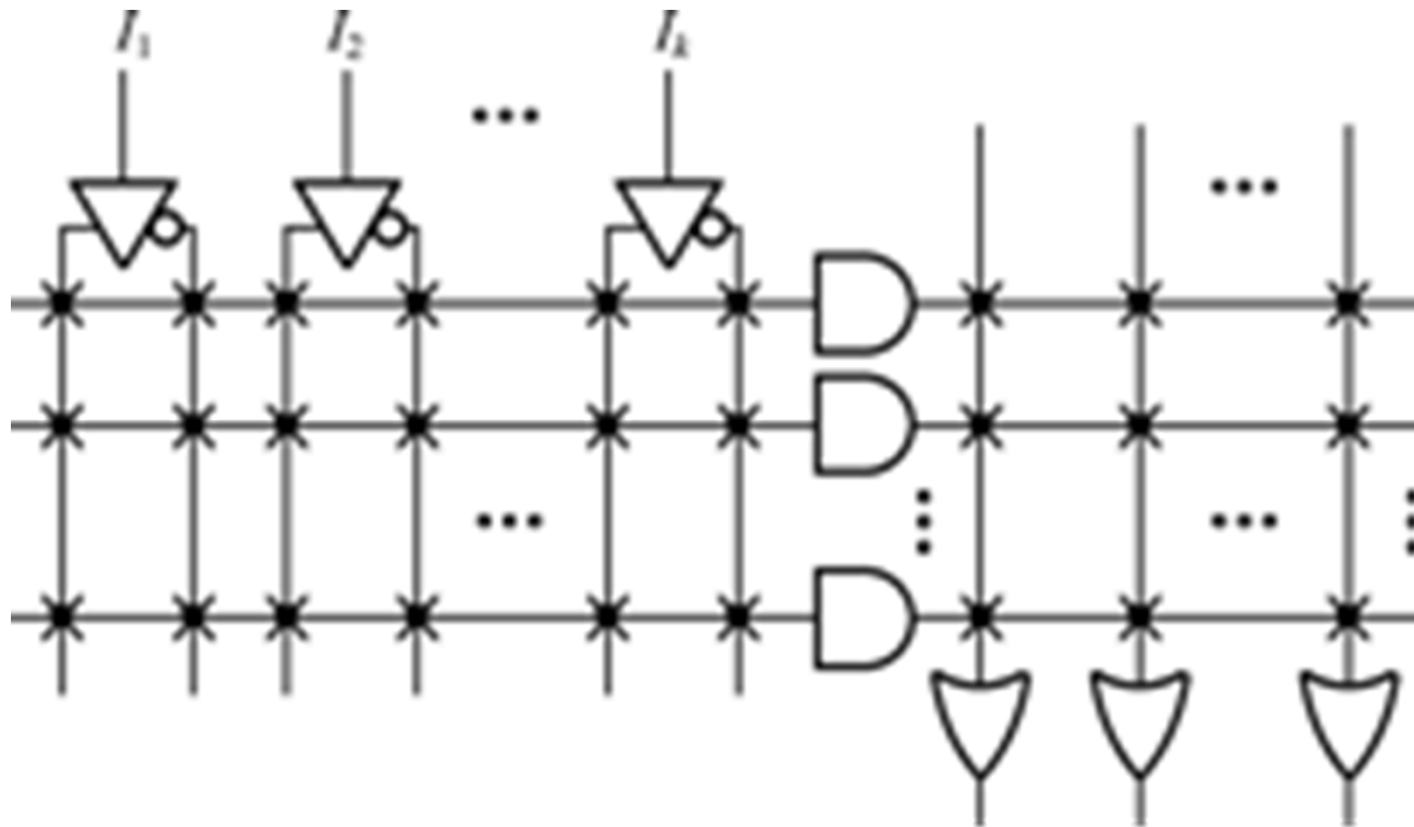
$$\begin{aligned}
 C_2 &= \overline{B_8}/\overline{B_4}/\overline{B_2}B_1 + \overline{B_8}/\overline{B_4}B_2\overline{B_1} + \overline{B_8}B_4/\overline{B_2}B_1 + \overline{B_8}B_4B_2\overline{B_1} + B_8/\overline{B_4}/\overline{B_2}B_1; \\
 C_4 &= \overline{B_8}/\overline{B_4}B_2/\overline{B_1} + \overline{B_8}/\overline{B_4}B_2B_1 + \overline{B_8}B_4/\overline{B_2}B_1 + B_8/\overline{B_4}/\overline{B_2}/\overline{B_1} + B_8/\overline{B_4}/\overline{B_2}B_1; \\
 C_2 &= \overline{B_8}B_4/\overline{B_2}/\overline{B_1} + \overline{B_8}B_4B_2/\overline{B_1} + \overline{B_8}B_4B_2B_1 + B_8/\overline{B_4}/\overline{B_2}/\overline{B_1} + B_8/\overline{B_4}/\overline{B_2}B_1; \\
 C_1 &= \overline{B_8}B_4/\overline{B_2}B_1 + \overline{B_8}B_4B_2/\overline{B_1} + \overline{B_8}B_4B_2B_1 + B_8/\overline{B_4}/\overline{B_2}/\overline{B_1} + B_8/\overline{B_4}/\overline{B_2}B_1.
 \end{aligned}$$

表 4.1 代码转换电路真值表

输入编码 (8421)				输出编码 (2421)			
B_8	B_4	B_2	B_1	C_2	C_4	C_2	C_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	1



现场可编程逻辑阵列 (FPLA)

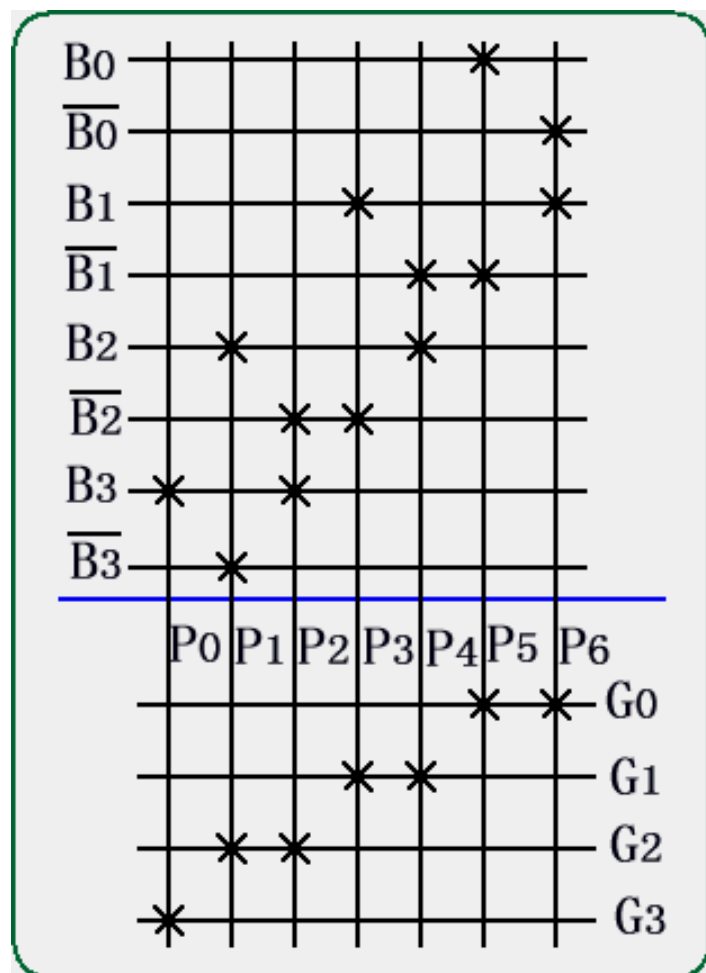


和PROM区别，与项可以自己定义一些一般项，而不是产生所有最小项

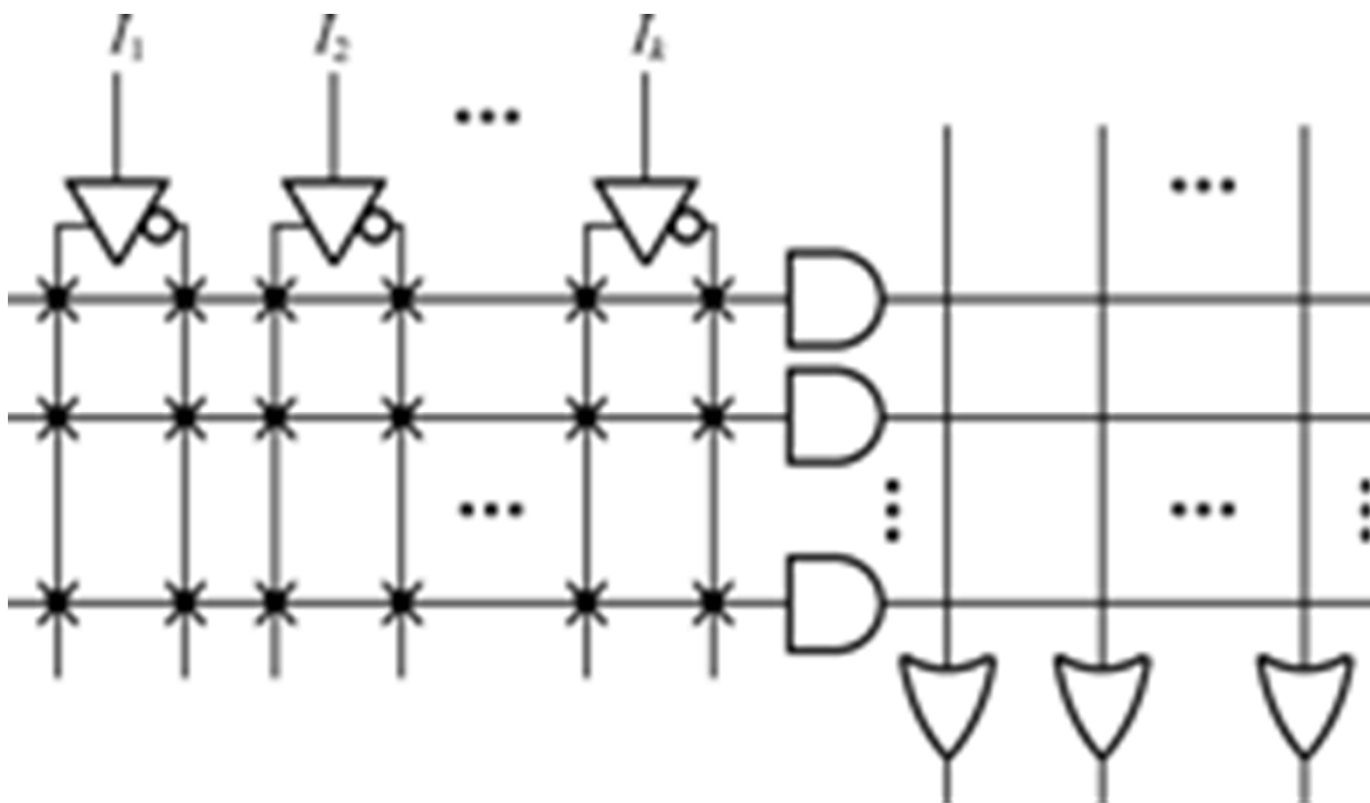
设计一个4位二进制码到Gray码的代码转换电路。

二进制码				Gray码			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

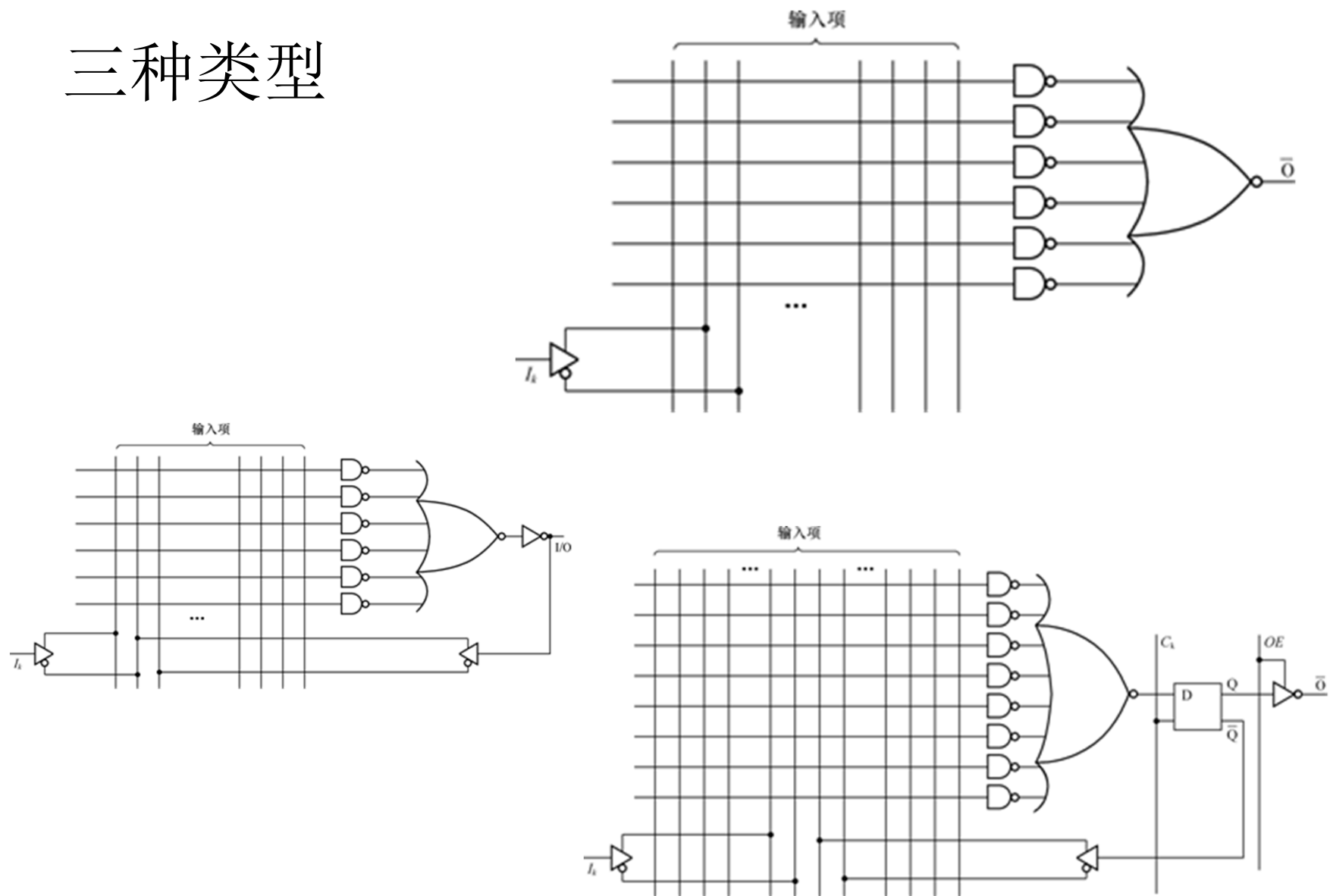
- 阵列图如下如所示：



可编程阵列逻辑 (PAL)

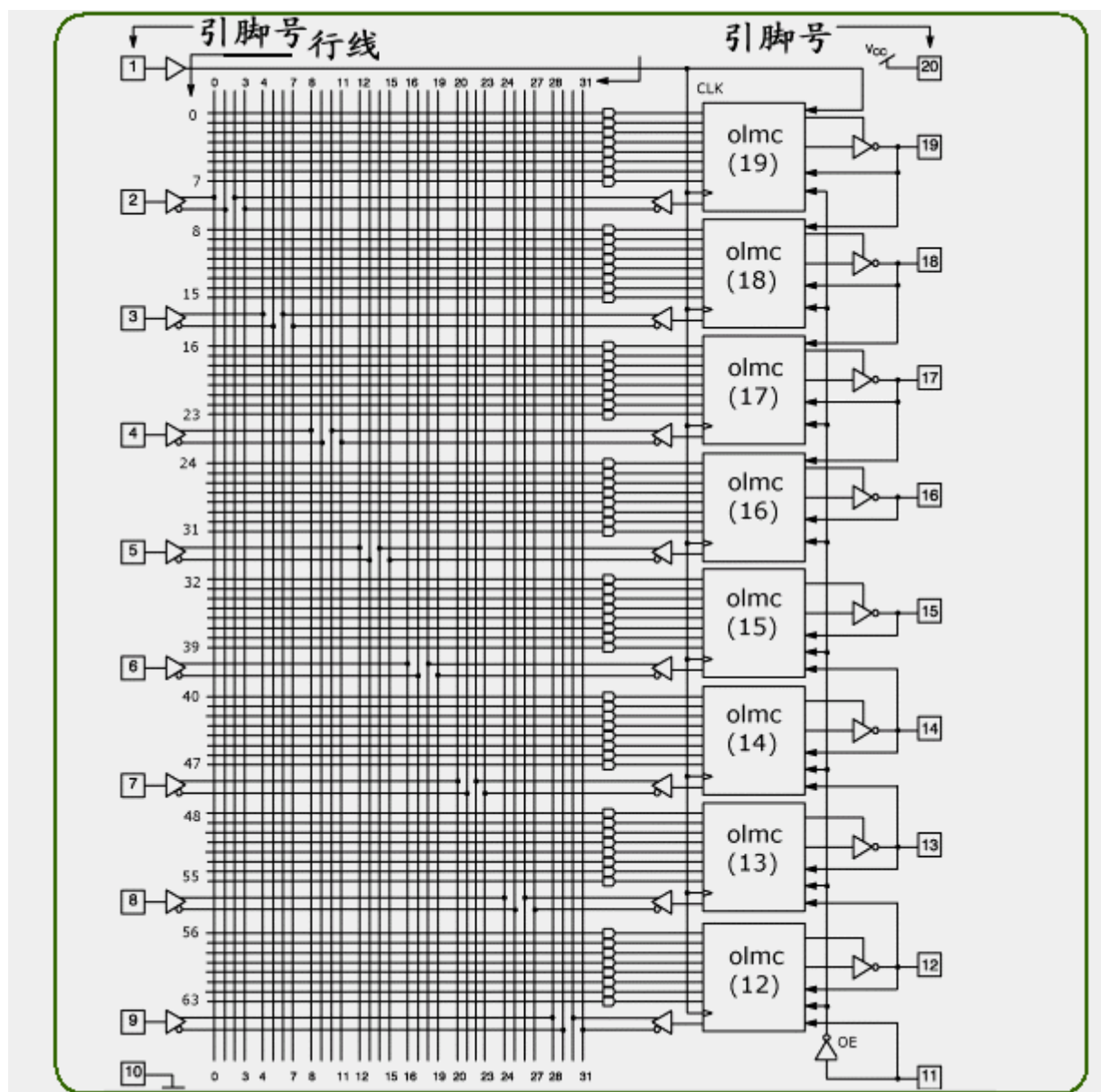


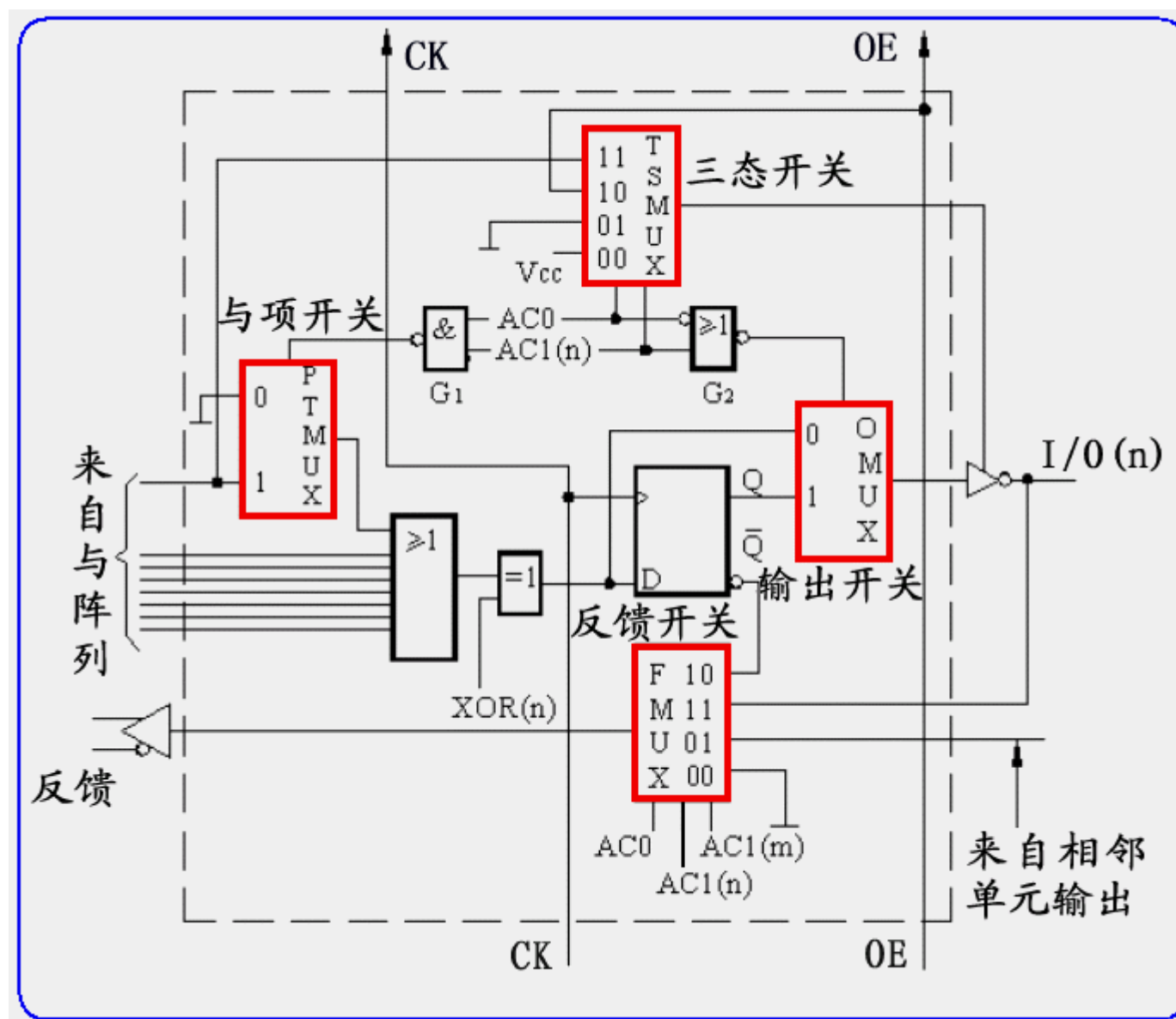
三种类型



通用阵列逻辑GAL

- 与可编程，或不可编程
- 输出部分比较灵活，加入了OLMC，其中含异或门等，可以产生F或其反函数，给设计带来好处（特别是原函数与项过多时，可以用反函数）
- 多种模式和组态





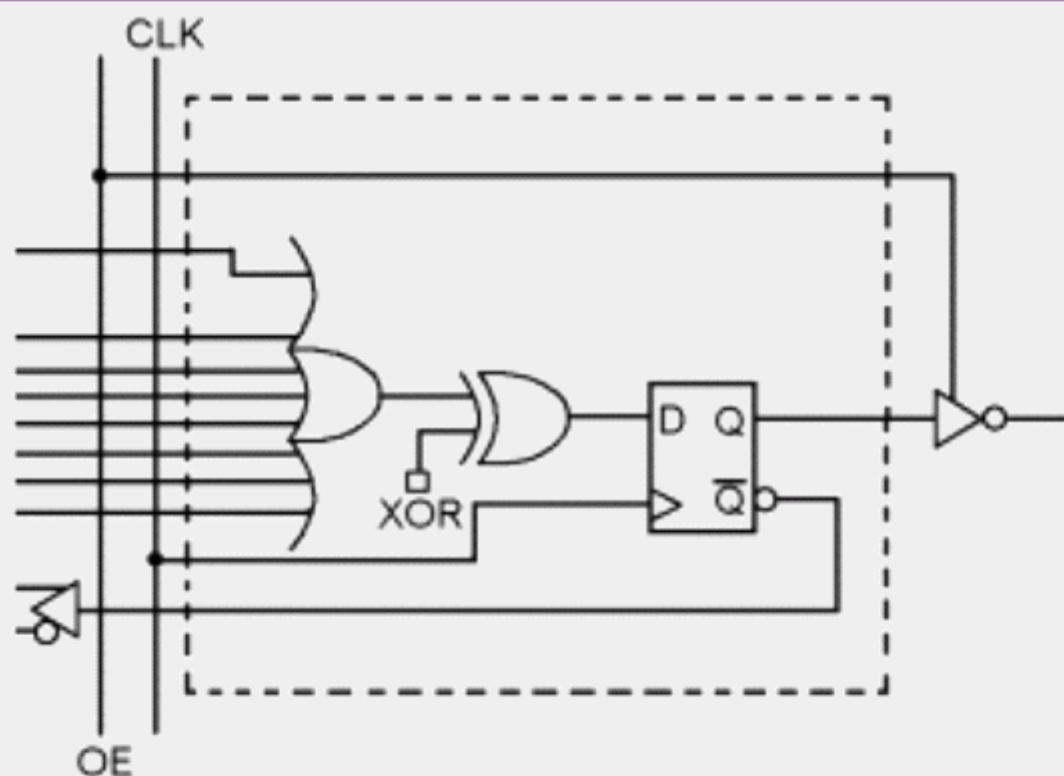
GAL的工作模式和逻辑组态

OLMC的工作模式与引脚定义方程的关系

输出方程类型	输出定义方程式	工作模式
A型	引脚名:=逻辑方程式	SYN=0 AC0=1 寄存器模式
B型	引脚名=逻辑方程式 引脚名.OE=逻辑方程式	SYN=1 AC0=1 复杂模式
C型	引脚名=(逻辑方程式)	SYN=1 AC0=0 简单模式

三种模式和七种组态的关系

工作模式	逻辑组态	
寄存器模式	(1) 寄存器输出组态	(2) 组合输出组态
复杂模式	(3) 组合I/O组态	(4) 纯组合输出组态
简单模式	(5) 无反馈组合输出组态	(6) 有反馈组合输出组态
	(7) 邻级输入组态	



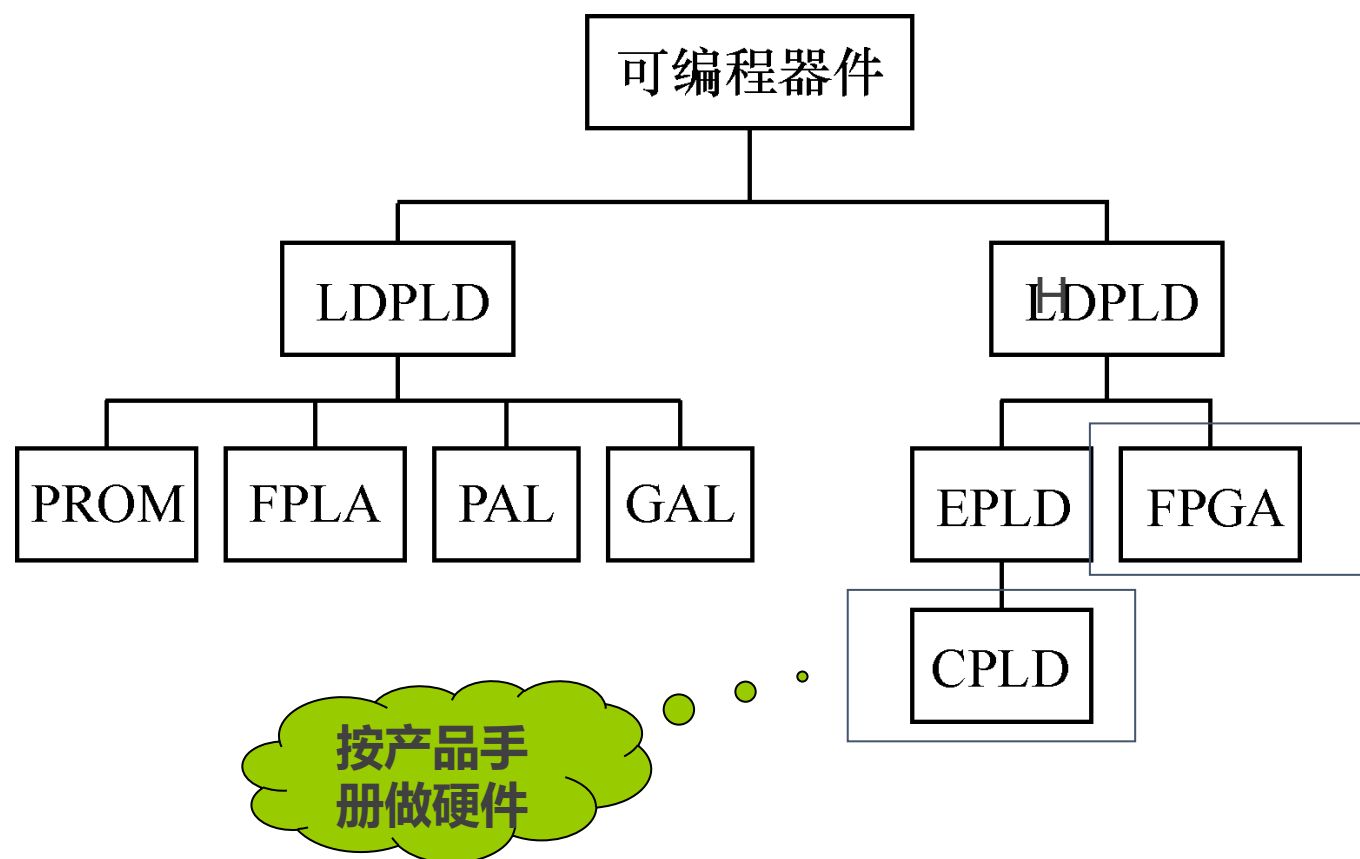
寄存器模式下的寄存器输出组态

$\text{SYN} \cdot \text{AC0} \cdot \text{AC1} (n) = 010$, OLMC全部为寄存器输出。CK作同步时钟、OE作使能端。

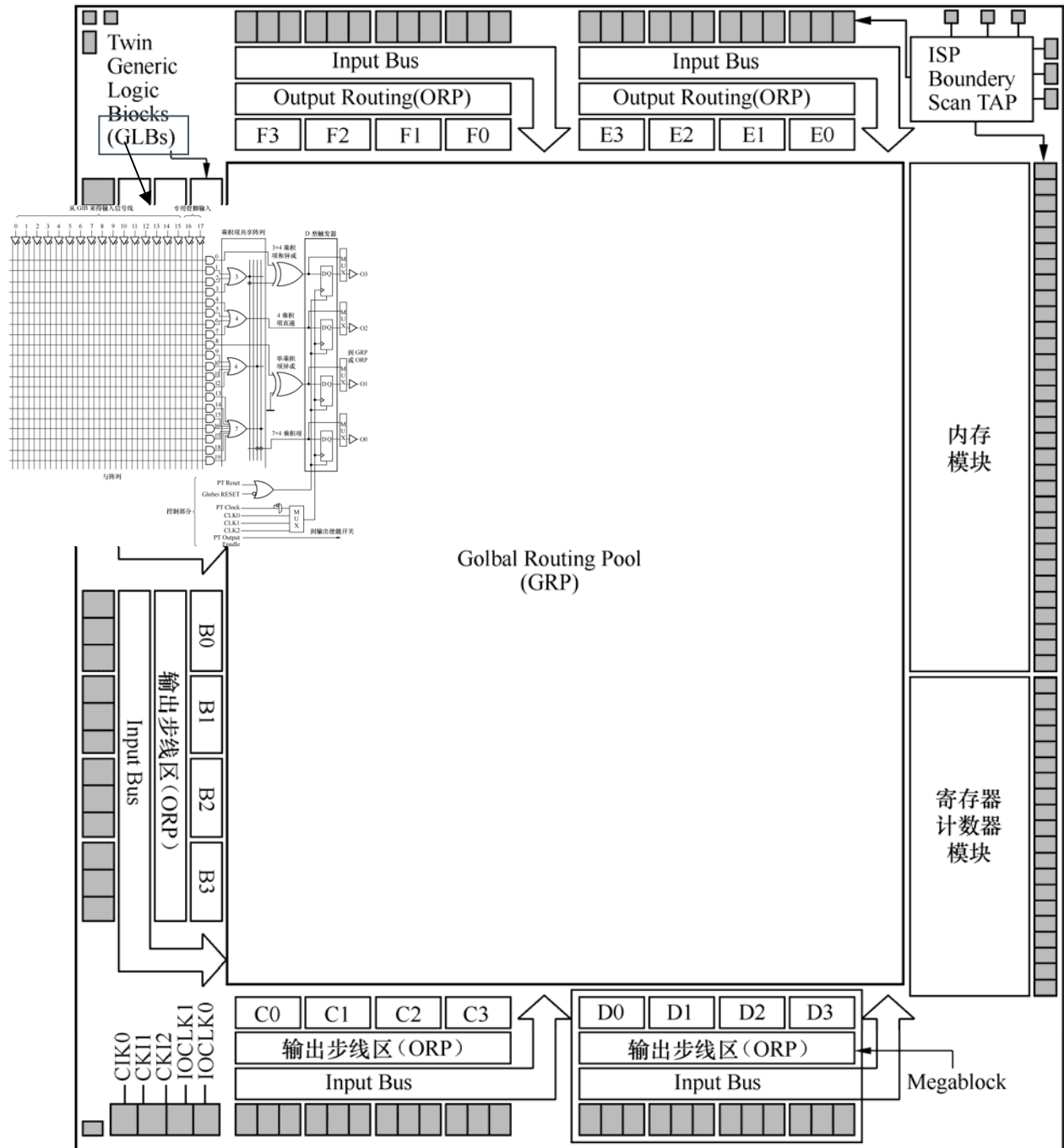
设计过程

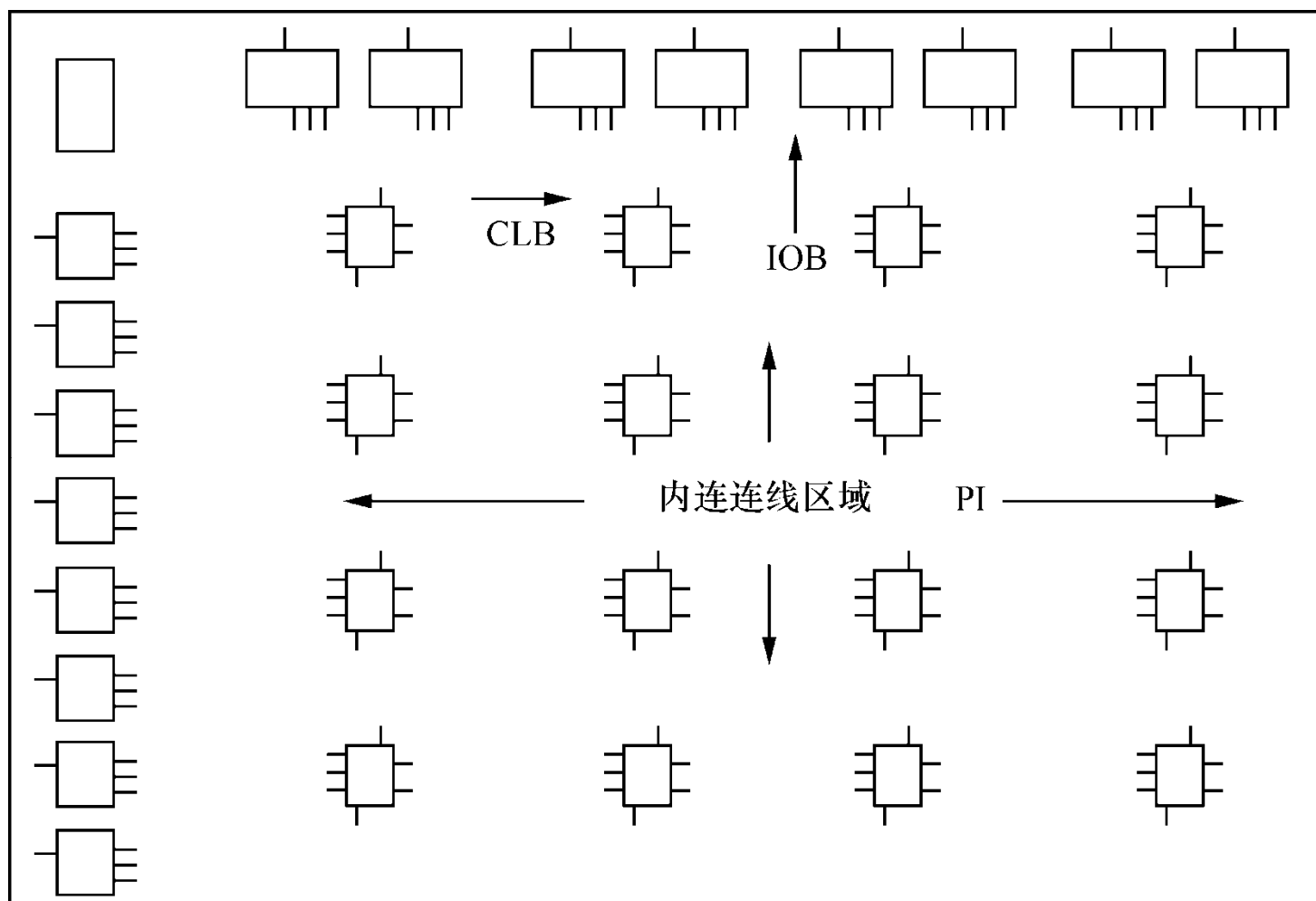
- (1) 优化 (**Optimization**) 过程: 它实现逻辑函数自动简化并将逻辑函数转化成最适合在可编程逻辑器件中实现的形式;
- (2) 合并 (**Merging**) 过程: 设计中因设计模块结构化问题可能产生多个设计文件, 其合并过程将把这些设计文件合并成一个实现逻辑功能部件设计的网表文件, 以达到层次设计平面化;
- (3) 分割 (**Partitioning**) 过程: 将逻辑功能设计划分成多个可以适合于器件内部逻辑资源实现模块, 以实现逻辑功能独立且资源共享;
- (4) 布局 (**Placement**) 过程: 该过程是器件结构工艺的嵌入。它自动将通过分割后的各个功能模块放置到硅片上具体位置, 以达到资源部件相互连线最少和方便的目的;
- (5) 布线 (**Routing**) 过程: 软件包利用布线资源完成器件内部各个功能模块间传输信号的连接, 因此设计者确定的资源布局和设计复杂程度将对信号连接成功率带来影响, 一旦布线过程失败, 就需要修改原输入或设计结构, 以便解决布线失败的问题。
- (6) 数据文件生成 (**JEDEC或Bit Stream Generation**) 过程: 数据文件是供可编程器件实现设计者逻辑功能实现的文本, 通过编程电缆该数据文件被下载到可编程器件中。可编程器件内部“与-或”阵列将根据该文件产生相应熔丝过程, 在CPLD分类下的可编程器件称为熔丝图文件, 而FPGA分类下的可编程器件称为位流文件。

两种常用的HDPLD可编程逻辑器件



一般使用公司提供的开发器来进行硬件开发，没有开发器基本无人能操作





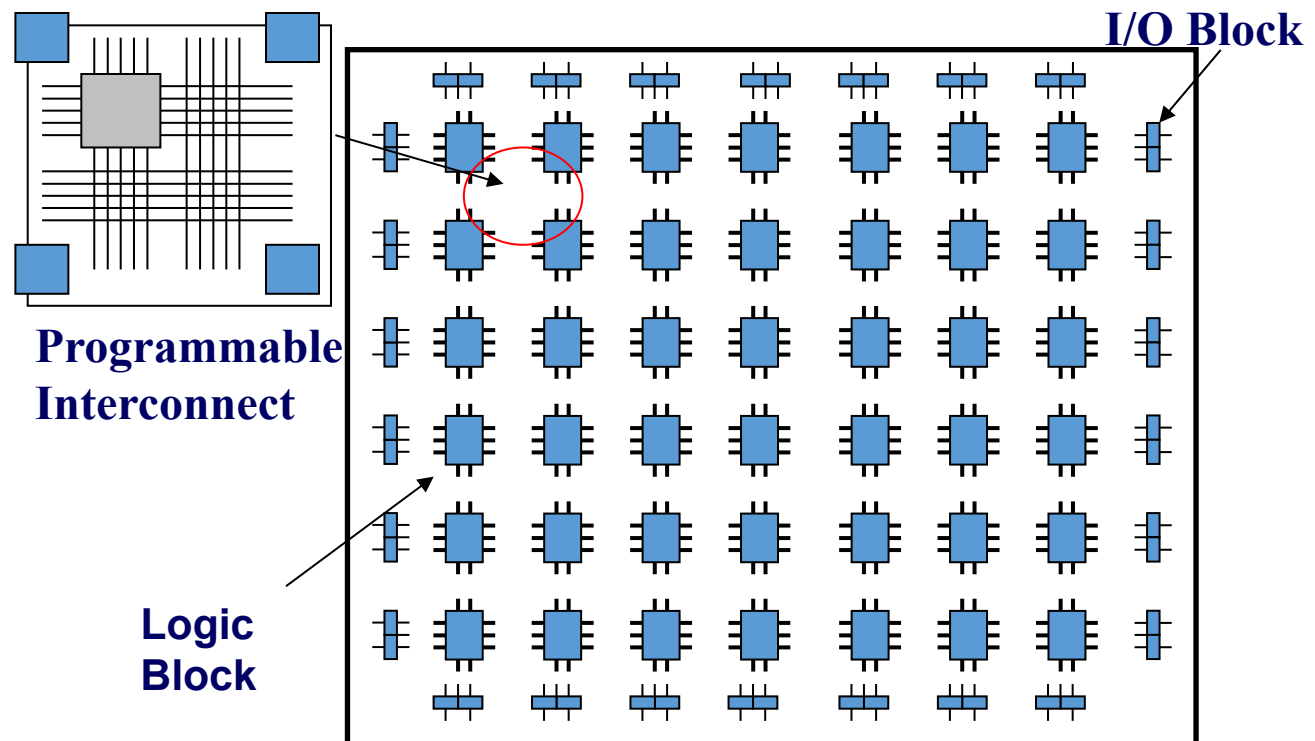
每个单元也是可编程的

FPGA 的概念

- **概念：一种集成度较高的PLD器件，FPGA称为现场可编程门阵列 (*Field Programmable Gate-Array*)**
- ***lattice* 的FPGA器件、*xilinx*的FPGA器件、*Altera*的FPGA器件**

FPGA 的结构

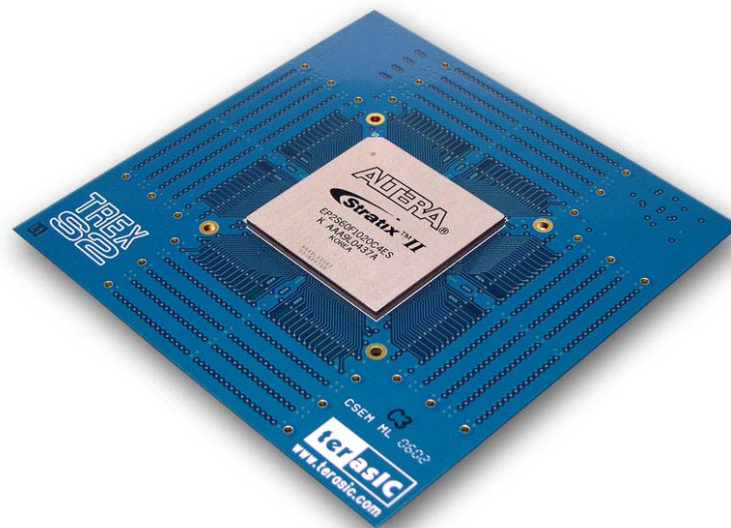
主要由三部分组成：可配置逻辑块 (*configurable logic block, CLB*)，输入输出模块 (*I/O BLOCK, IOB*) 和布线通道 (*Routing Channels*)



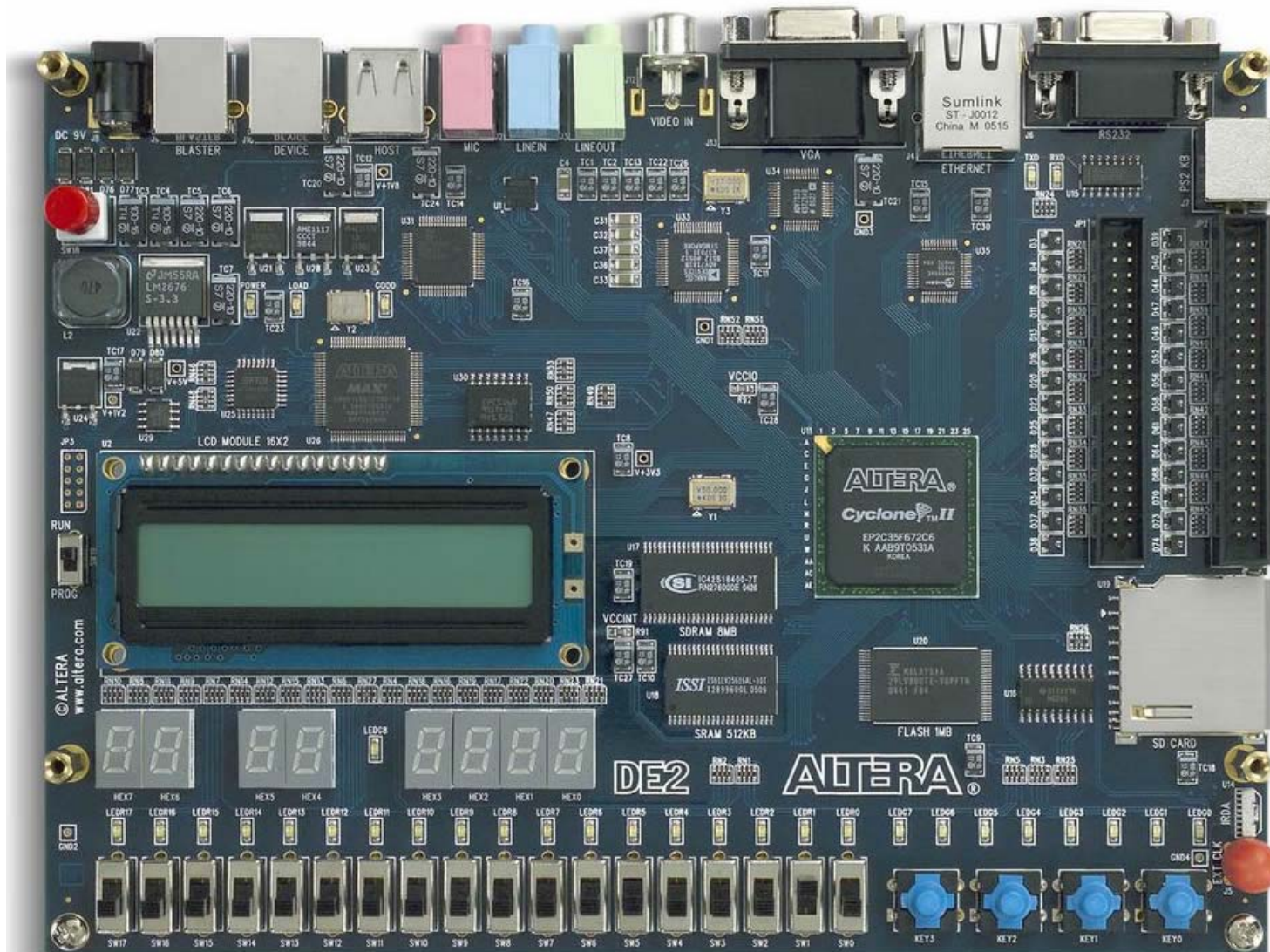
FPGA 的结构及特点



1、Xilinx 的Virtex 5



2、Altera 的stratix ii

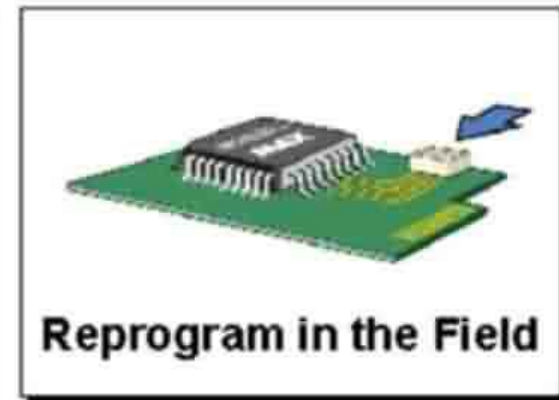
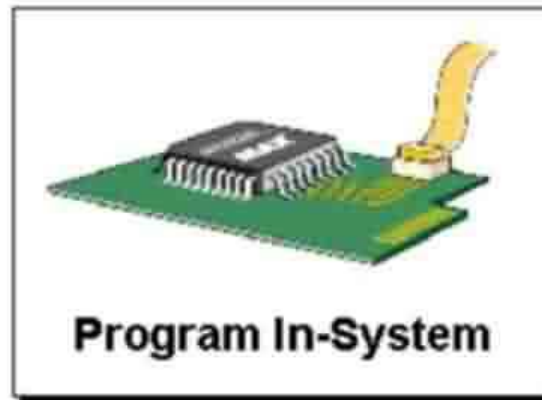
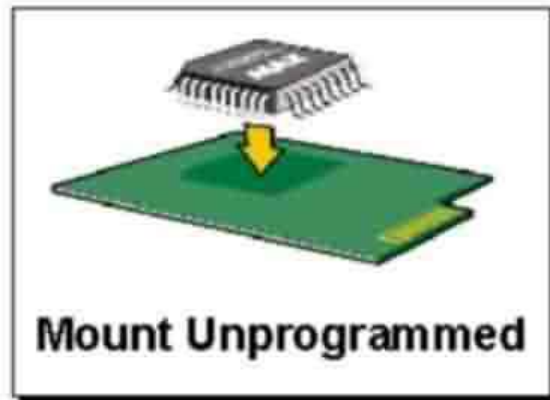


Altera 的 DE2, cyclone II FPGA

ISP (in system programmable)

所谓“在系统编程”指的是：对器件、电路板或整个电路子系统的逻辑功能可随时进行修改重构的能力。它允许用户“在系统中”编辑和修改逻辑，给使用者提供了在不修改系统硬件设计的条件下重构系统的能力和硬件升级能力，使硬件修改变得像软件修改一样方便，系统的可靠性因此而提高。

ISP (in system programmable)



1.将PLD/FPGA焊在PCB板上 2.接好编程电缆

3.现场烧写PLD芯片

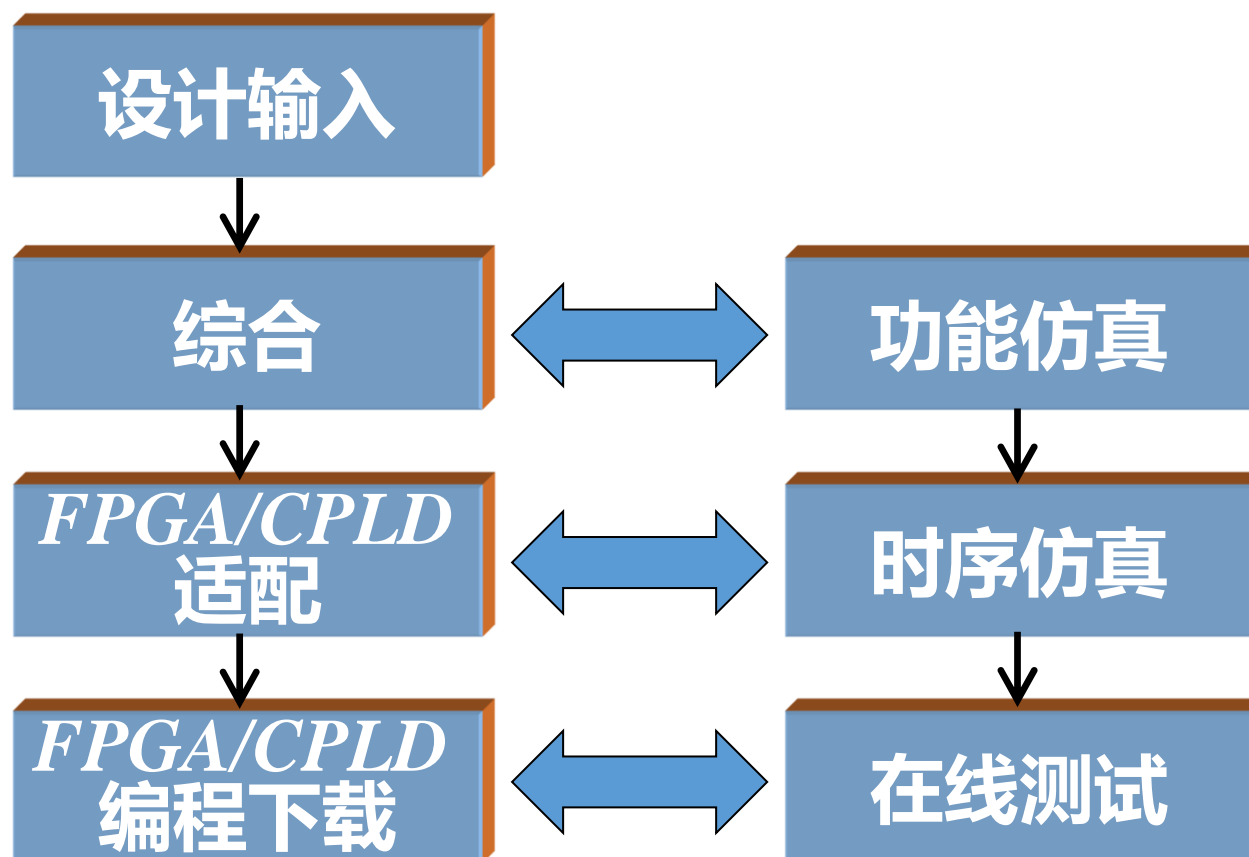
CPLD/FPGA 能做

夸张的讲, *CPLD/FPGA* 能完成任何数字器件的功能, 上至高性能 *CPU*, 下至简单的电路, 都可以用 *CPLD/FPGA* 来实现。利用 *CPLD/FPGA*, 工程师可以通过传统的原理图输入法, 或硬件描述语言自由的设计一个数字系统。通过软件仿真, 我们可以事先验证设计的正确性。在 *PCB* 完成以后, 还可以利用 *CPLD/FPGA* 的在线修改能力, 随时修改设计而不必改动硬件电路。使用 *CPLD/FPGA* 来开发数字电路, 可以大大缩短设计时间, 提高系统的可靠性。*CPLD/FPGA* 的这些优点使得 *CPLD/FPGA* 技术得到飞速的发展, 同时也大大推动了 *EDA* 软件和硬件描述语言(*HDL*)的进步。

CPLD/FPGA 数字系统的实现步骤

使用 *CPLD/FPGA* 实现数字系统的
步骤和要点：**设计输入、综合、适配、
仿真、编程。**

CPLD/FPGA 数字系统设计流程



CPLD/FPGA 实现步骤—设计输入

设计输入 (*design entry*)：将设计者所设计的电路以开发软件要求的某种形式表达出来，并输入到相应软件中的过程。

- 1、原理图输入：**图形化的表达方式，使用元件符号和连线来描述设计。适合描述连接关系和接口关系，而描述逻辑功能比较烦琐。
- 2、HDL文本输入：**逻辑描述能力强但描述接口和连接规则不如图形方式直观。

CPLD/FPGA 实现步骤—综合

综合：将较高层次的设计描述自动转化成较低层次的描述的过程。

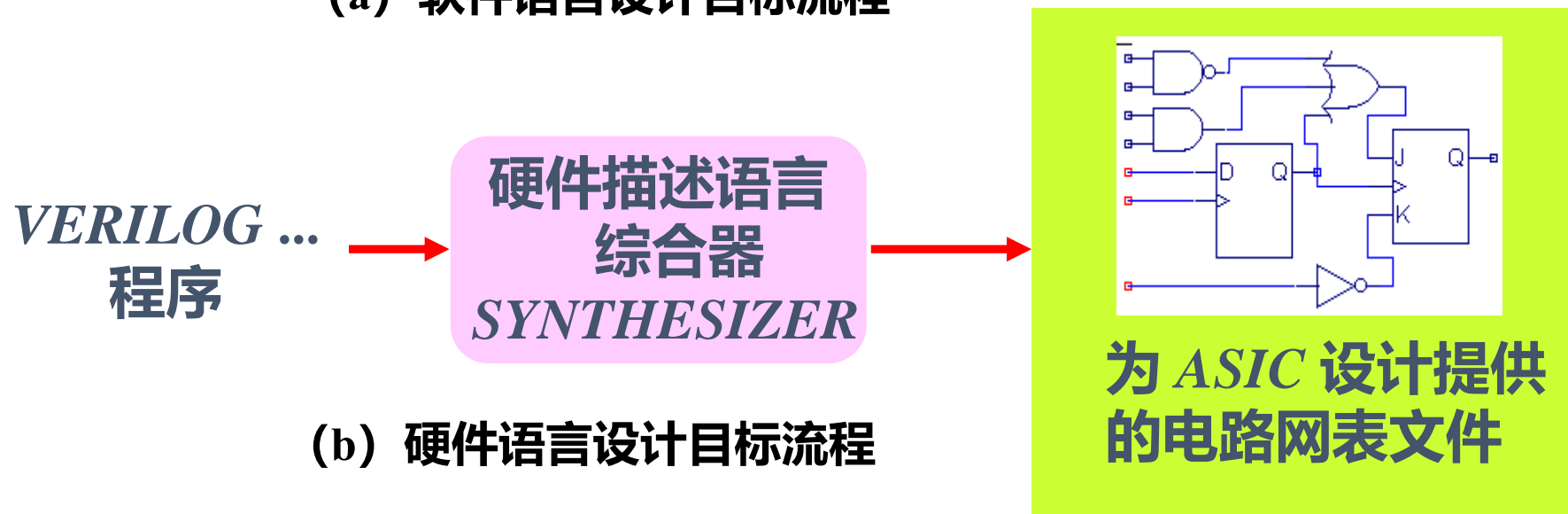
- 1、行为综合：**将算法表示、行为描述转换到 *RTL*（寄存器传输）级描述。
- 2、逻辑综合：**将 *RTL* 级描述转换到逻辑门级描述。
- 3、版图综合：**将逻辑门级表示转换到版图表示，或转换到 *PLD* 器件的配置网表表示。

综合器是能够自动实现上述转换的软件工具，是能将原理图或 *HDL* 语言描述的电路功能转化为具体电路结构网表的工具。

软件编译器和硬件综合器功能比较



(a) 软件语言设计目标流程



(b) 硬件语言设计目标流程

CPLD/FPGA 实现步骤—适配

适配：通过适配器（也称结构综合器）将由综合器产生的网表文件配置于指定的目标文件中并最终产生可下载文件。

适配器产生的重要文件：

- 适配报告（包括内部资源的耗费情况、设计布尔方程等）
- 输出文件（面向其它EDA工具的输出文件）
- 适配后的仿真模型，包括延迟信息等，以便进行精确的时序仿真。
- 器件编程文件（对CPLD器件而言，产生熔丝图文件，即JEDEC文件；对FPGA器件则产生Bitstream位流数据文件）。

CPLD/FPGA 实现步骤—仿真与编程

仿真与编程：

1 仿真：对所设计电路的功能的验证。

功能仿真：不考虑信号延迟等因素的仿真，又叫前仿真。

时序仿真：在选择了具体的器件并完成了布局布线后进行的并包含定时关系的仿真。

2 编程：把适配后生成的编程文件装入到PLD器件中的过程。

在系统编程、专用的编程器编程

通常将对基于EEPROM工艺的非易失结构PLD器件的下载称为编程 (*Program*)，将基于SRAM工艺结构的PLD器件的下载称为配置 (*Configure*)。




常用的 EDA 设计工具

- **集成开发工具**
- **逻辑综合器**
- **仿真工具**



集成CPLD/FPGA 开发工具

软件	简介
 MAX+PLUS® II	<u>MAX+plus II</u> 是 <u>Altera</u> 的集成开发软件， <u>MAX+plus II</u> 最后版本是 10.2
 QUARTUS® II	<u>Quartus II</u> 是 <u>Altera</u> 继 <u>MAX+plus II</u> 后的新一代开发工具，适合大规模 FPGA 的开发。
 ISE ALL THE SPEED YOU NEED	ISE 是 Xilinx 公司 FPGA/CPLD 的集成开发软件
 ispLEVER	<u>ispLEVER</u> 是 Lattice 公司的集成开发工具，该软件同时集成了许多第三方专业工具

逻辑综合器

软件	简介
	<u>Synplify</u> Pro/ <u>Synplify</u> 是 Synplicity 的 <u>Verilog</u> HDL/VHDL 综合软件，使用广泛。
	FPGA Compiler II 是 Synopsys 公司的 <u>Verilog</u> HDL/VHDL 综合软件。
	Leonardo Spectrum 是 Mentor 的子公司 Exemplar Logic 出品的综合软件，并作为 FPGA Advantage 软件的一个组成部分。

仿真工具

软件	简介
 ModelSim	ModelSim 是 Mentor 的子公司 Model Technology 的一个出色的 VHDL/ Verilog HDL 混合仿真软件，属于编译型仿真器，仿真速度快。
 NC-Verilog/NC-VHDL/NC-Sim Verilog-XL	这几个软件都是 Cadence 公司的 Verilog HDL/VHDL 仿真工具，其中 NC-Verilog 用于对 Verilog 程序进行仿真；NC-VHDL 用于 VHDL 仿真；而 NC-Sim 则能够对 Verilog HDL/VHDL 进行混合仿真。
 VCS/Scirocco	VCS 是Synopsys公司的 Verilog HDL 仿真软件，Scirocco 是Synopsys的 VHDL 仿真软件。