

# 计算机组成和体系结构

## 第十四讲

四川大学网络空间安全学院

2020年6月1日

封面来自techlive.com

# 版权声明

---

课件中所使用的图片、视频等资源版权归原作者所有。

课件原创内容采用 [创作共用署名—非商业使用—相同方式共享4.0国际版许可证\(Creative Commons BY-NC-SA 4.0 International License\)](#) 授权使用。

Copyright@四川大学网络空间安全学院计算机组成与体系结构课程组，2020



# 上期内容回顾

---

- RISC指令集与CISC指令集
- 费林分类(Flynn's Taxonomy)
- 并行计算及多处理器体系结构
  - 超标量体系结构，超长指令字体系结构，矢量体系结构
  - 处理器/内存互连网络
  - 统一内存访问（UMA）与非统一内存访问（NUMA）
- 其它并行处理体系结构：
  - 数据流计算、神经网络和脉动阵列

# 本期学习目标

---

- 了解常见的嵌入式硬件类型和开发方法
  - 标准处理器
  - 了解可重构硬件PAL、 PLA、 FPGA的原理
  - 了解面向专用芯片设计的硬件描述语言和设计开发流程
- 了解嵌入式操作系统和软件开发的特点

# 中英文缩写对照表

英文缩写	英文全称	中文全称
SoC	System-on-Chip	片上系统
PLD	Programmable Logic Device	可编程逻辑设备
PLA	Programmable Logic Array	可编程逻辑阵列
PAL	Programmable Array Logic	可编程阵列逻辑
FPGA	Field Programmable Gate Array	现场可编程逻辑门阵列
ASIC	Application-specific Integrated Circuit	专用集成芯片
VHDL	Very-High-Speed Integrated Circuit Hardware Description Language	高速集成芯片硬件描述语言



# 嵌入式系统

# 嵌入式设备



图片来源：<https://www.quora.com/What-is-an-embedded-system>

# 嵌入式设备特点

---

- 嵌入式设备和通用计算机相比，资源比较有限
- 内存和能耗的利用率非常重要
- 软硬件的功能划分不固定
- 嵌入式系统开发者通常深入了解系统硬件细节

# 嵌入式设备分类

---

按照核心控制组件的可编程程度进行分类：

- 标准处理器 (Off-the-shelf, 即买即用)
- 可重构处理器 (Configurable)
- 全定制处理器 (Fully Customizable)

# 标准处理器

- 控制组件内部的连线方式固定，硬件功能固定
- 按照复杂程度可以分为微控制器（micro controller）和片上系统（system-on-chip, SoC）
- 现在的标准处理器大多数是过去的通用处理器



68HC系列：36元



8051系列：4.5元

摩托罗拉68HC12的原型6800芯片曾经是第一代苹果计算机的处理器

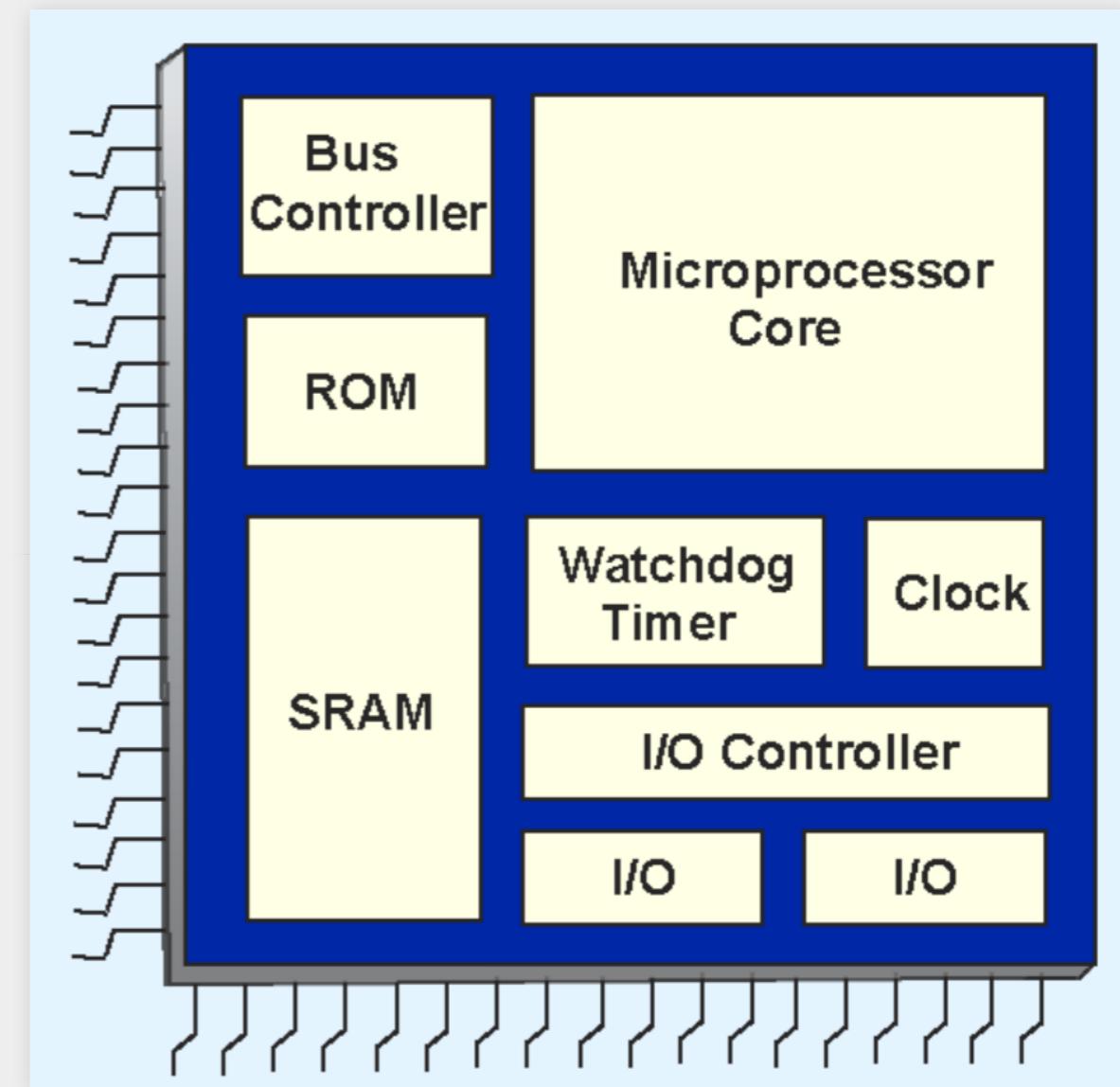
Intel 8051芯片的原型8086处理器曾经是第一代IBM个人计算机的处理器

图片来源：taobao.com

# 微控制器示例

微控制器的一般组成部分包括：

- CPU、存储器、I/O端口和端口控制器、系统总线、时钟和看门狗定时器（Watchdog Timer）
- 其中看门狗定时器是嵌入式系统专用的组件



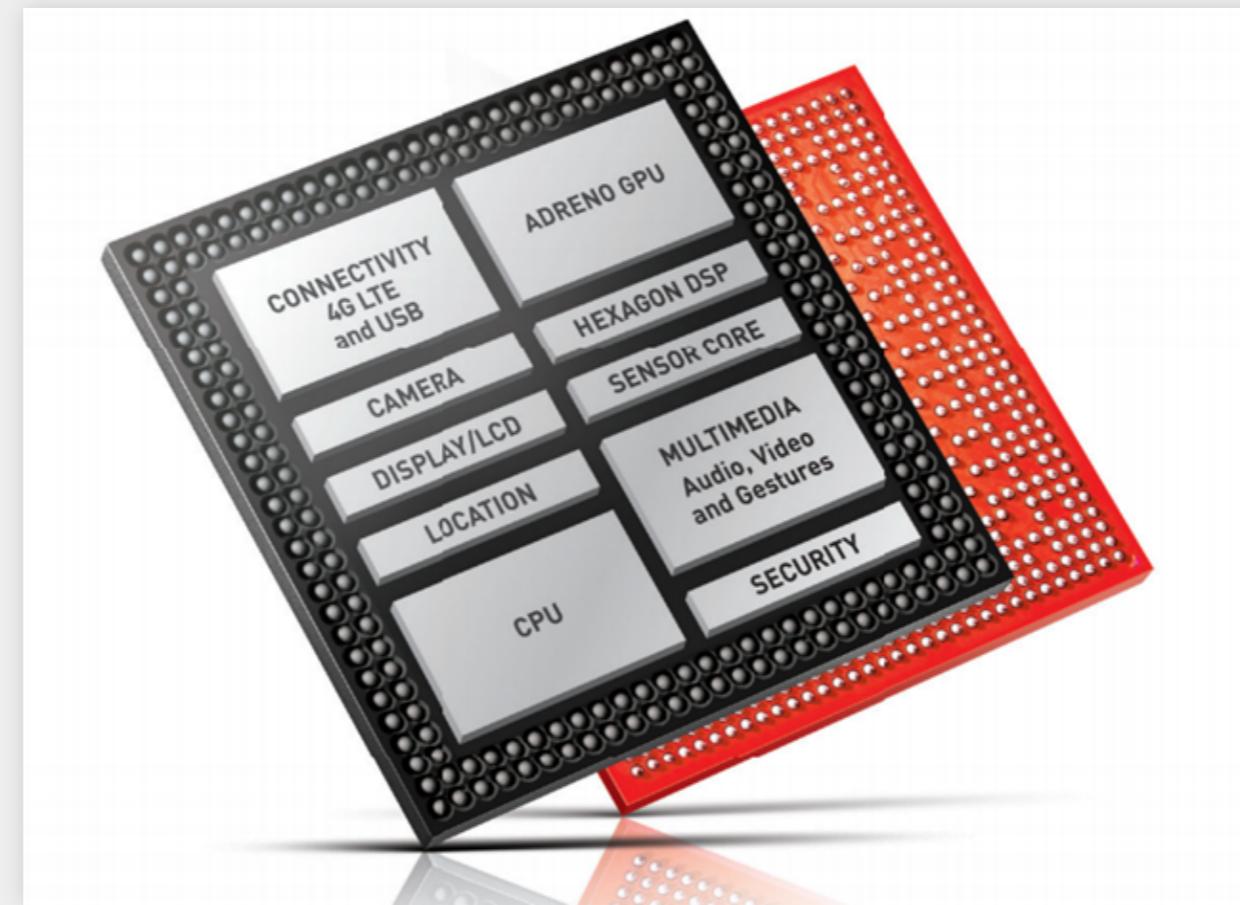
# 看门狗定时器的工作原理

---

- 看门狗定时器提供了一种自动检测错误、进行系统重置的机制
- 每隔一段时间，看门狗定时器的计数器减1
- 应用程序执行完成后重置看门狗定时器的计数器到初始值
- 保证任何应用程序的执行时间都不会超过看门狗定时器设定的时间

# 片上系统

- 片上系统通常包括多个微处理器以及互连电路等
- 特定的处理器执行特定的计算任务



图片来源：<https://www.digitalcitizen.life/soc-system-on-chip>

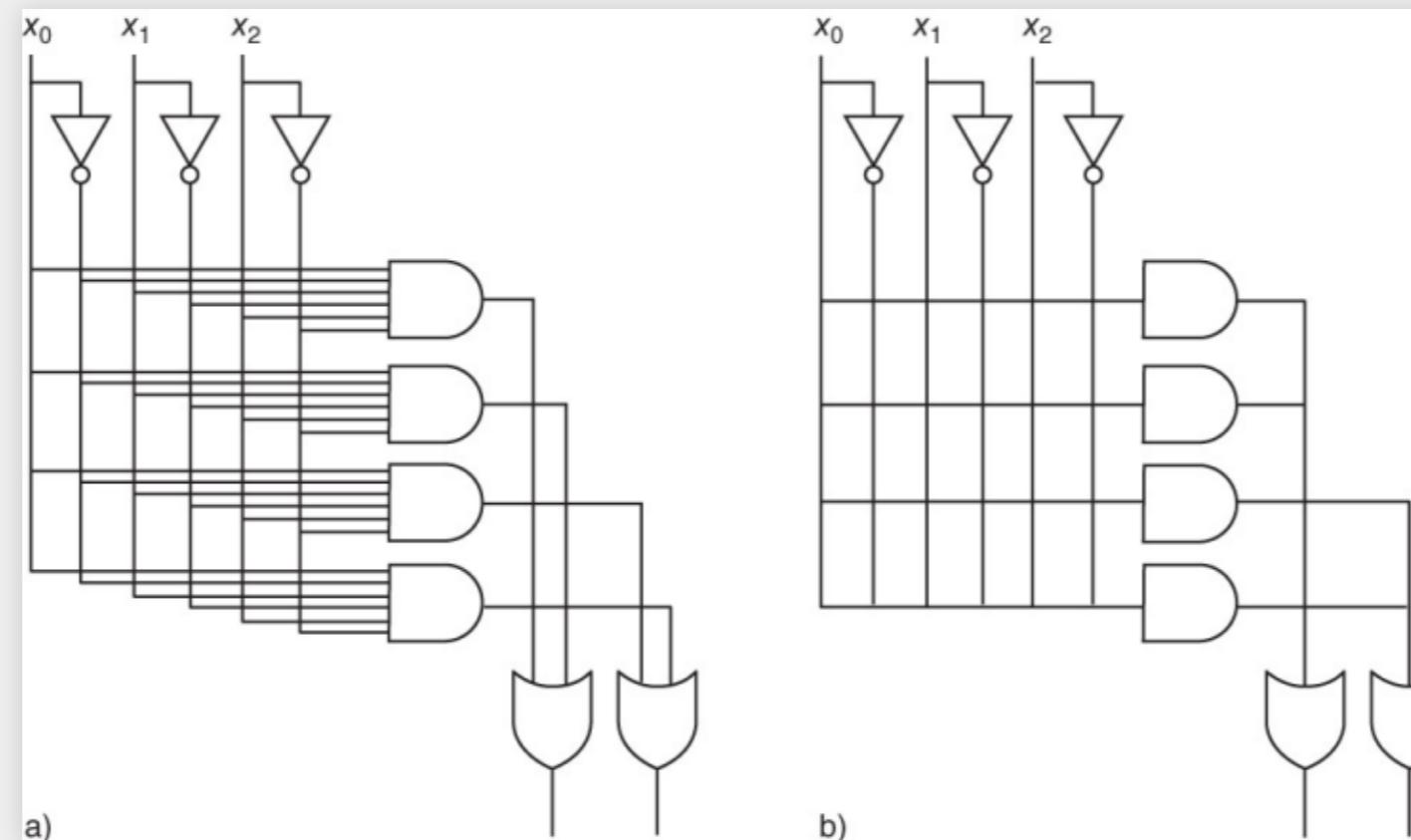
# 可重构硬件

---

- 当一些特殊的应用无法使用现有标准处理器（高效）实现的时候，需要实现新的硬件，此时一般有两种选择：采用可编程逻辑设备（Programmable Logic Device, PLD），或者设计专用芯片（Application-specific Integrated Circuit, ASIC）
- ASIC通常比PLD速度更快、尺寸更小，但是设计、制造成本更高，通用性更差
- PLD的三种实现方式：
  - 可编程阵列逻辑（Programmable Array Logic, PAL）
  - 可编程逻辑阵列（Programmable Logic Array, PLA）
  - 现场可编程门阵列（Field Programmable Gate Array）

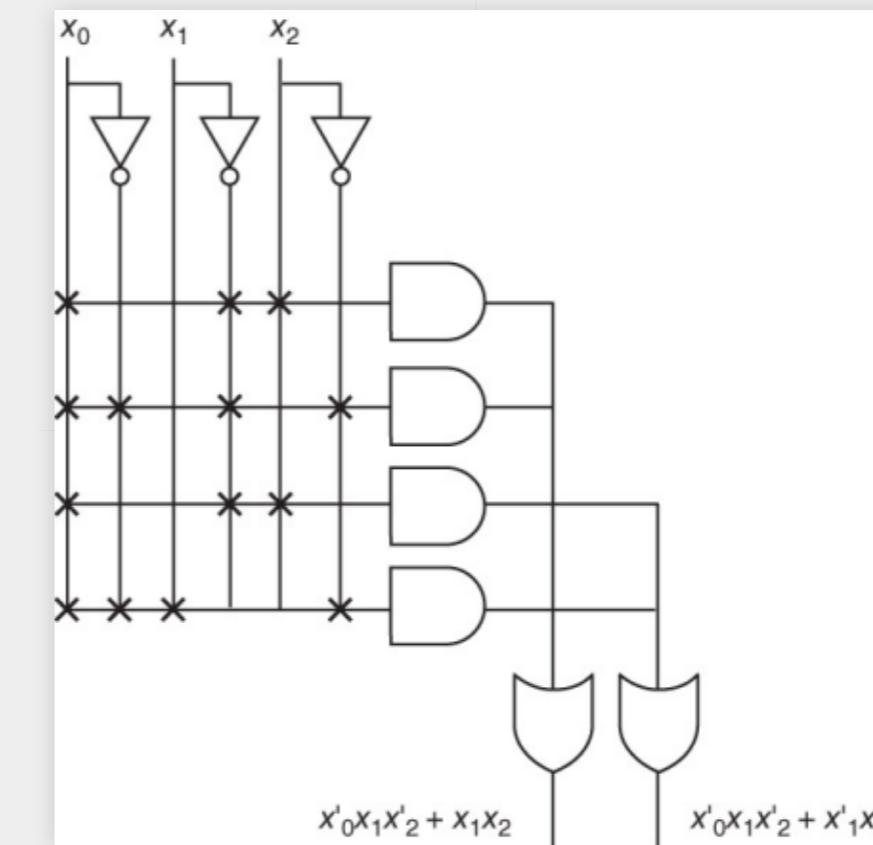
# 可编程阵列逻辑

- PAL芯片由**可编程**的与门阵列和**固定**的或门阵列构成
- 电路的可编程性通过熔断电路或者拨动双掷开关控制
- 熔断有两种形式：熔断断开和熔断连接，本课程采用熔断断开方式



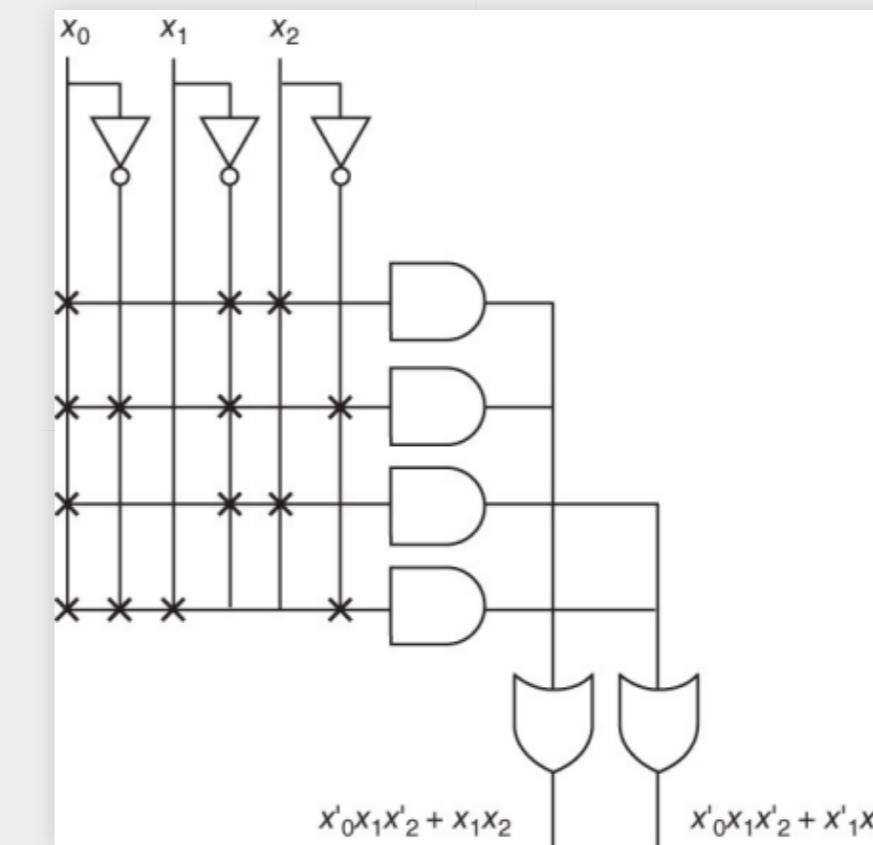
# 如何定制可编程阵列逻辑

- 假设熔断表示电路断开
- 每个与门的电路输出是所有未熔断的电路信号的乘积
- 每个或门的电路输入是对应的与门电路输出的和



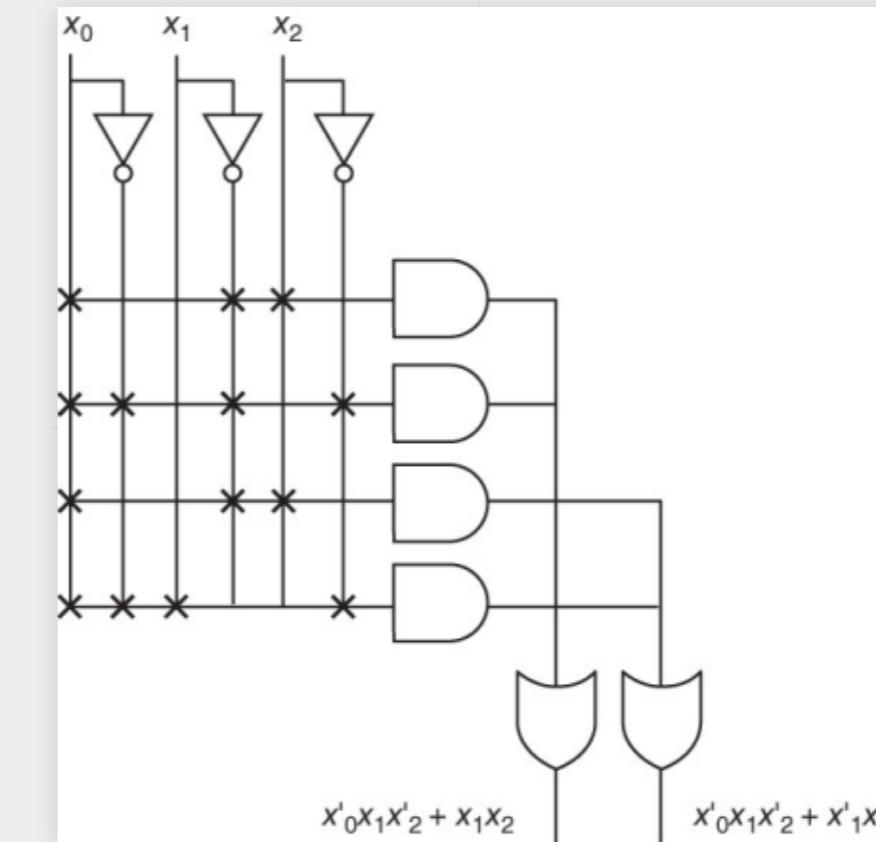
# 如何定制可编程阵列逻辑

- 假设熔断表示电路断开
- 每个与门的电路输出是所有未熔断的电路信号的乘积
- 每个或门的电路输入是对应的与门电路输出的和



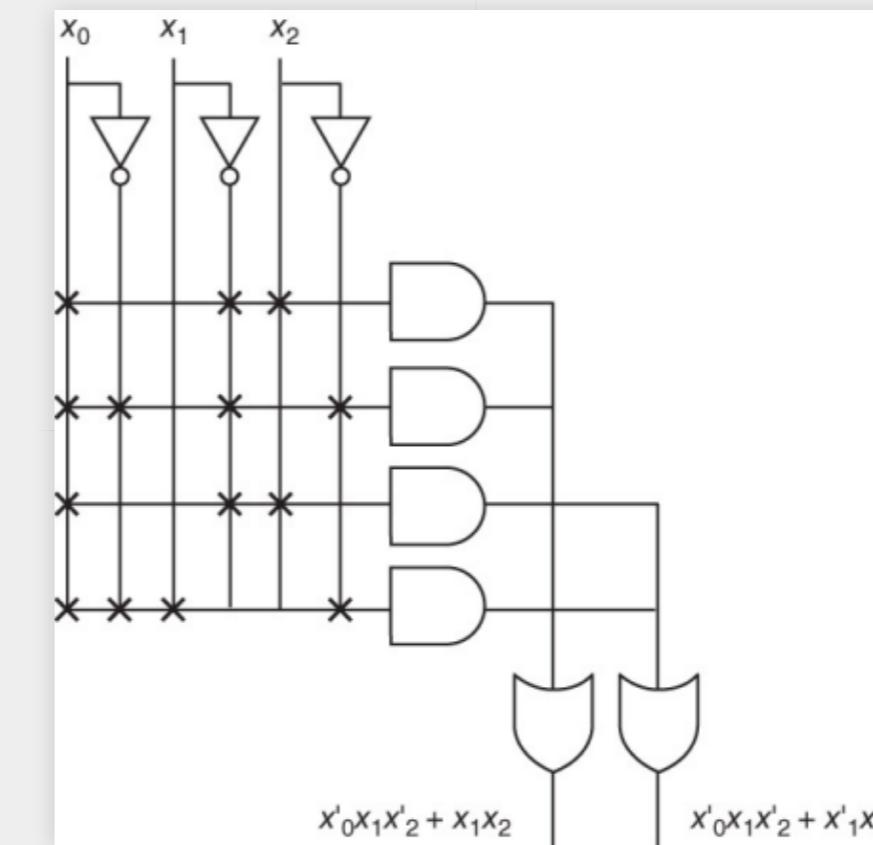
# 如何定制可编程阵列逻辑

- 假设熔断表示电路断开
- 每个与门的电路输出是所有未熔断的电路信号的乘积
- 每个或门的电路输入是对应的与门电路输出的和
- 与门1:  $x'_0 x_1 x'_2$



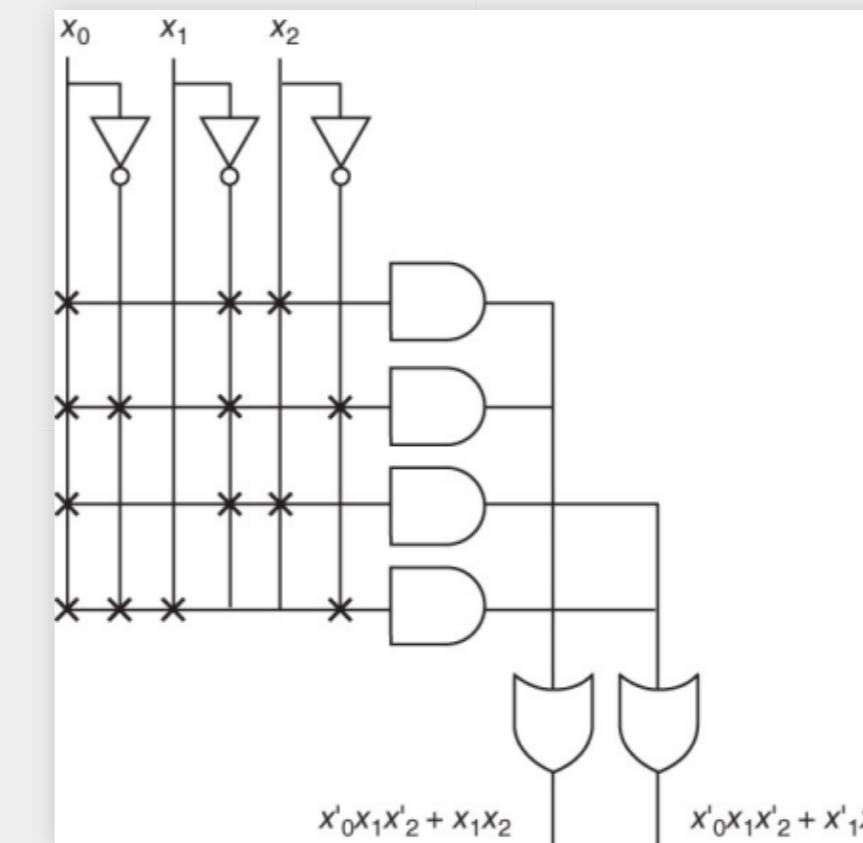
# 如何定制可编程阵列逻辑

- 假设熔断表示电路断开
  - 每个与门的电路输出是所有未熔断的电路信号的乘积
  - 每个或门的电路输入是对应的与门电路输出的和
  - 与门1:  $x'_0 x_1 x'_2$
  - 与门2:  $x_1 x_2$



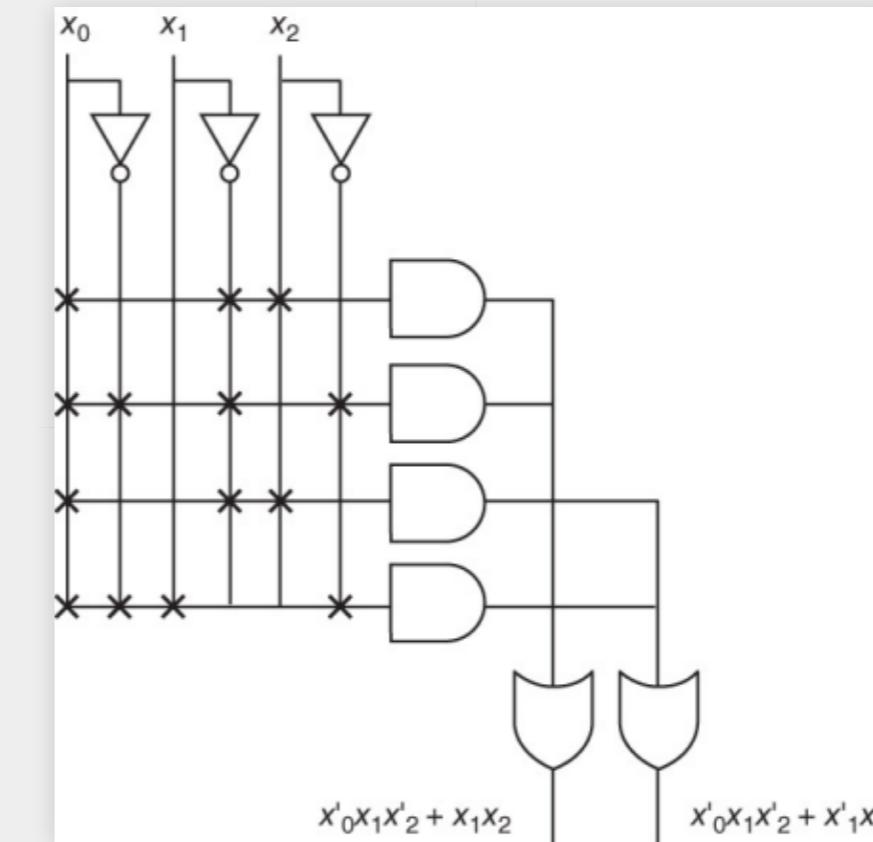
# 如何定制可编程阵列逻辑

- 假设熔断表示电路断开
- 每个与门的电路输出是所有未熔断的电路信号的乘积
- 每个或门的电路输入是对应的与门电路输出的和
- 与门1:  $x'_0 x_1 x'_2$
- 与门2:  $x_1 x_2$
- 与门3:  $x'_0 x_1 x'_2$



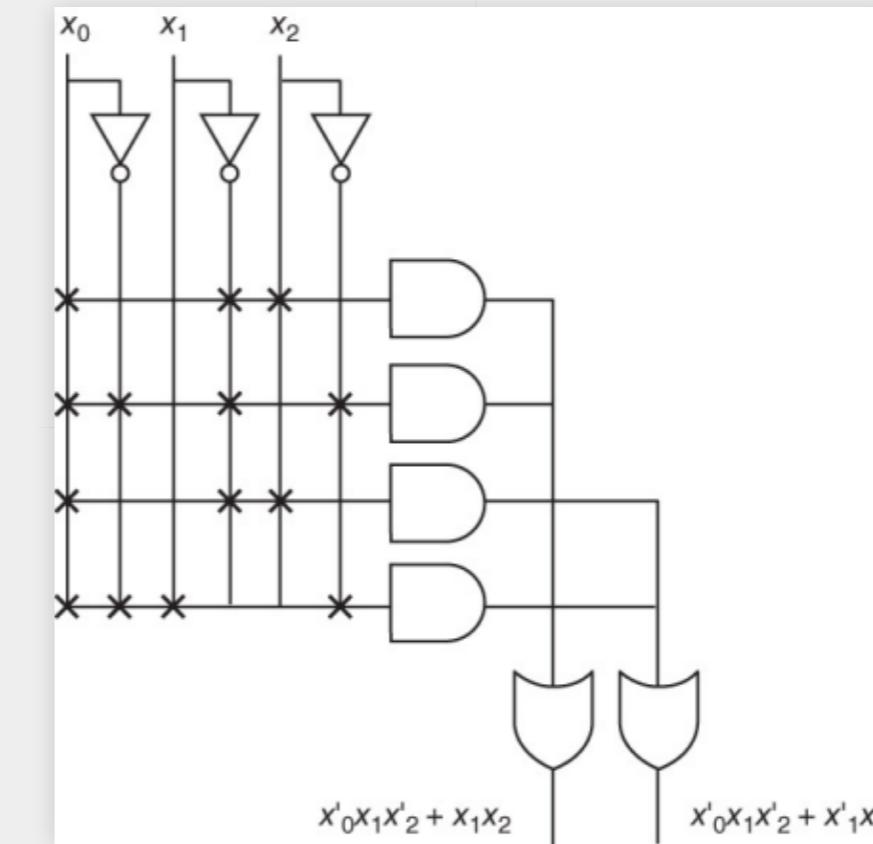
# 如何定制可编程阵列逻辑

- 假设熔断表示电路断开
- 每个与门的电路输出是所有未熔断的电路信号的乘积
- 每个或门的电路输入是对应的与门电路输出的和
- 与门1:  $x'_0 x_1 x'_2$
- 与门2:  $x_1 x_2$
- 与门3:  $x'_0 x_1 x'_2$
- 与门4:  $x'_1 x_2$



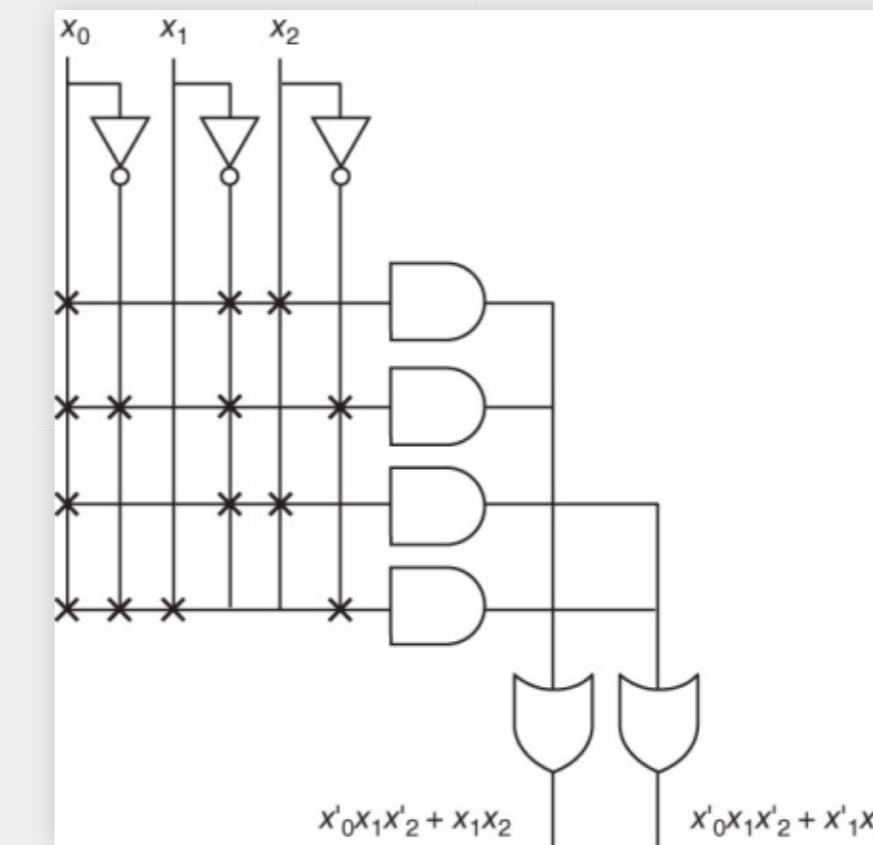
# 如何定制可编程阵列逻辑

- 假设熔断表示电路断开
- 每个与门的电路输出是所有未熔断的电路信号的乘积
- 每个或门的电路输入是对应的与门电路输出的和
- 与门1:  $x'_0 x_1 x'_2$
- 与门2:  $x_1 x_2$
- 与门3:  $x'_0 x_1 x'_2$
- 与门4:  $x'_1 x_2$
- 或门1:  $x'_0 x_1 x'_2 + x_1 x_2$



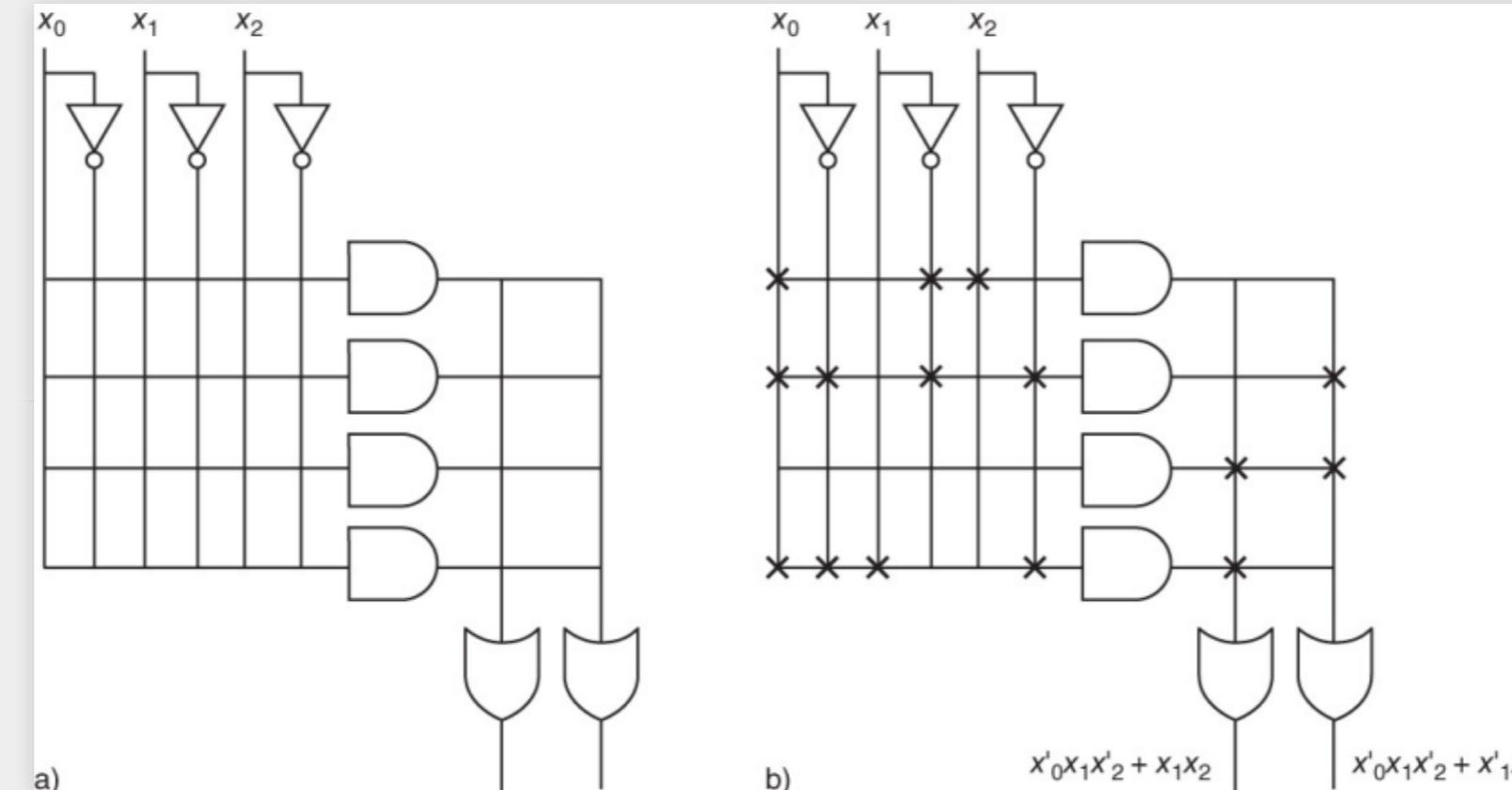
# 如何定制可编程阵列逻辑

- 假设熔断表示电路断开
- 每个与门的电路输出是所有未熔断的电路信号的乘积
- 每个或门的电路输入是对应的与门电路输出的和
- 与门1:  $x'_0 x_1 x'_2$
- 与门2:  $x_1 x_2$
- 与门3:  $x'_0 x_1 x'_2$
- 与门4:  $x'_1 x_2$
- 或门1:  $x'_0 x_1 x'_2 + x_1 x_2$
- 或门2:  $x'_0 x_1 x'_2 + x'_1 x_2$



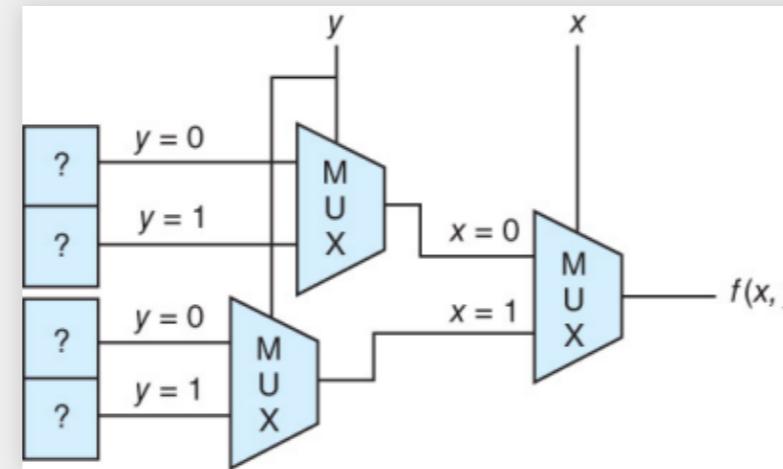
# 可编程逻辑阵列

- PLA采用**可编程**的与门阵列和**可编程**的或门阵列
- PLA比PAL更加灵活，但是速度更慢，成本更高
- 计算方法类似，或门的输出是未熔断电路连接的与门输出的和



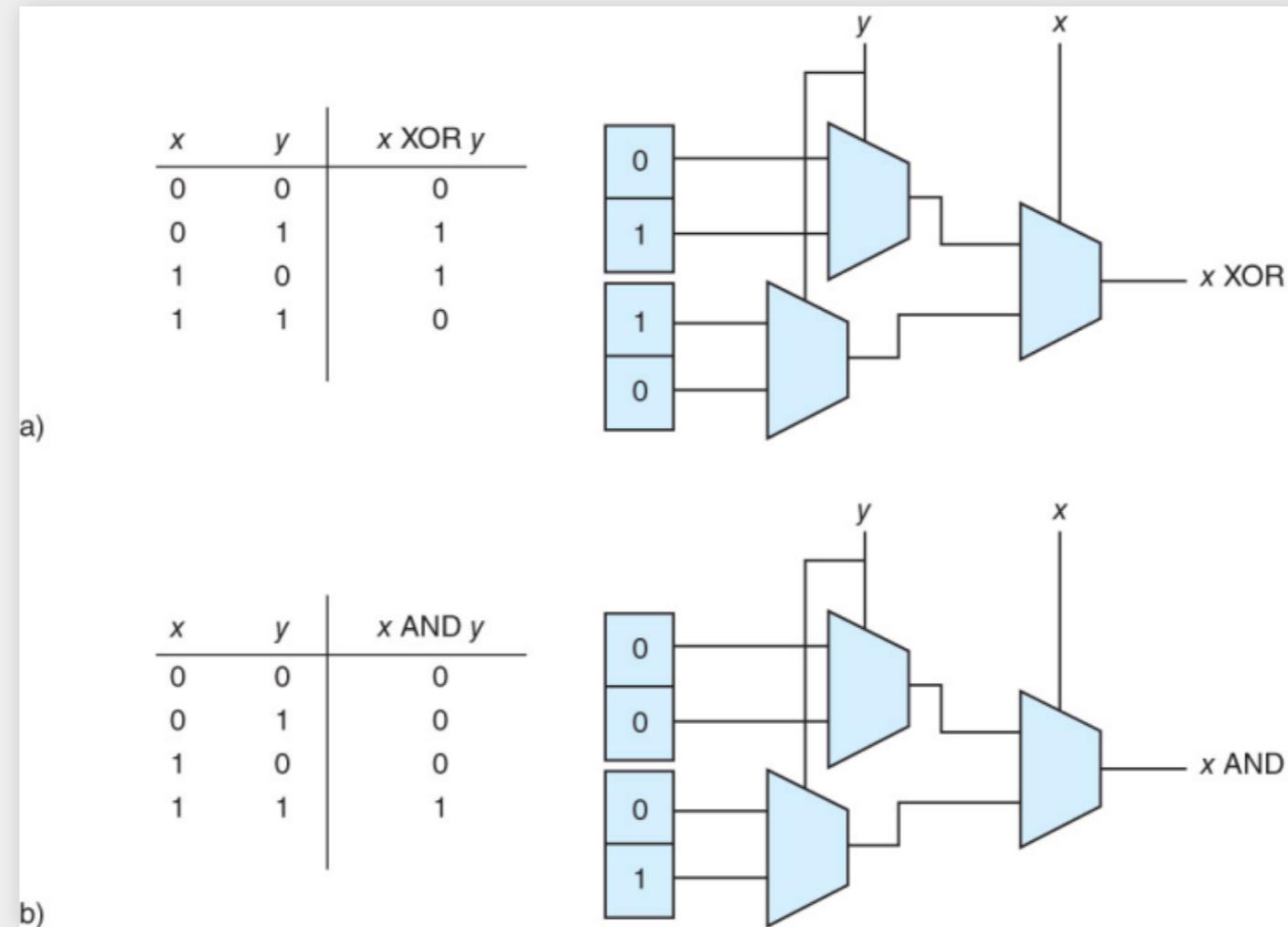
# 现场可编程逻辑门阵列

- FPGA使用查找表代替芯片内的电路连线
- FPGA由可读写的存储单元和多路复用器构成
- 根据输入的电路信号选择对应的存储单元中的值作为输出

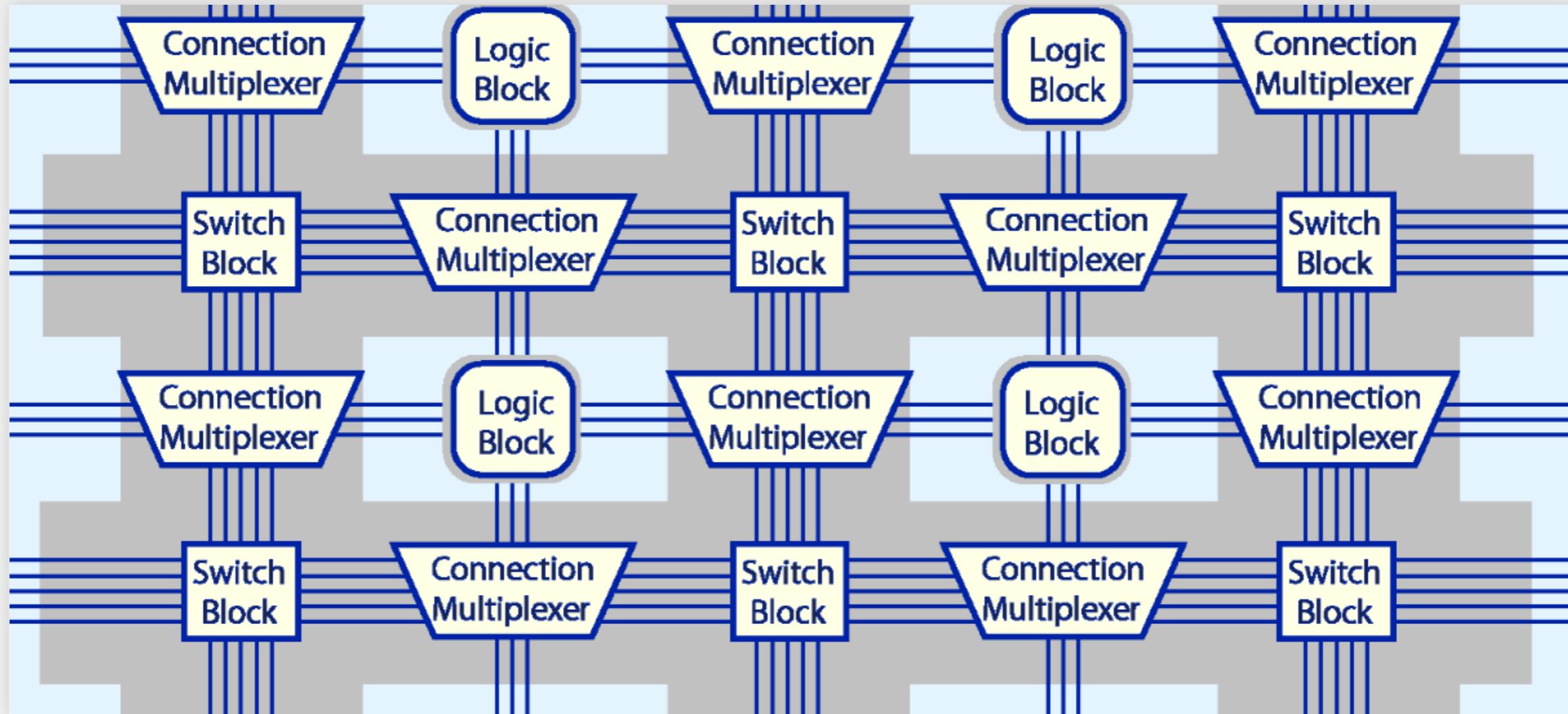


# FPGA计算单元电路分析

- 真值表和存储单元内的数值一一对应

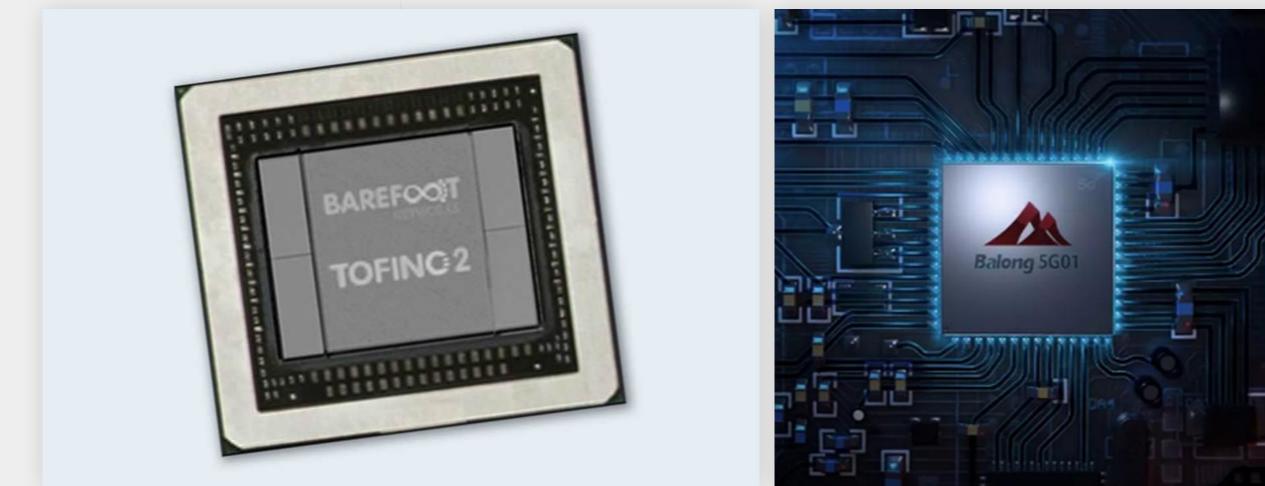


# FPGA电路排布



# 定制专用芯片

- 实际中，嵌入式系统可能会遇到功能或性能无法被标准处理器或可重构硬件满足的情况
- 这种情况下只能采用定制专用芯片（Application-specific Integrated Circuit, ASIC）



图片来源：<https://www.eweek.com/networking/barefoot-expands-reach-with-tofino-2-network-chips>, <http://tech.sina.com.cn/it/2018-02-26/doc-ifyrwsqi2308449.shtml>

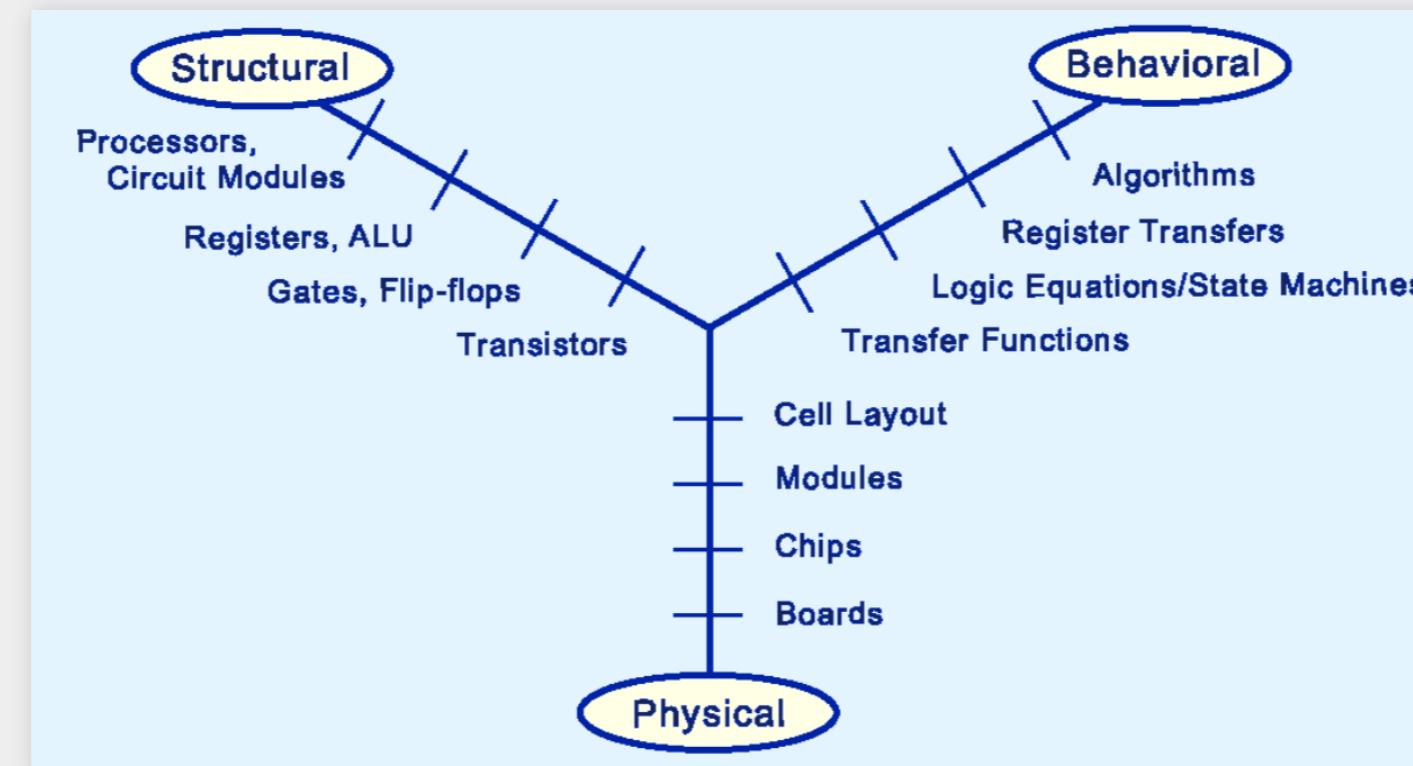
# ASIC设计

---

- 芯片设计是一个非常复杂的过程
- 通常需要考虑三个方面：
  - ASIC需要实现哪些功能
  - ASIC需要哪些组件来实现对应的功能
  - 如何对组件之间进行电路连接，提高执行速度、降低能耗

# Y-CHART

- Dannie Gajski提出使用三个维度描述芯片设计
- 每个坐标轴代表一类细节的层次，越靠近原点抽象层次越低
- 芯片设计的过程通常是自外向内的
- ASIC的设计过程需要考虑三个坐标轴，该过程被称为**综合**(Synthesis)



# 硬件描述语言

---

- 硬件描述语言 (Hardware Description Language, HDL)  
使得芯片开发者可以在较高层次进行芯片设计
- HDL可以在算法层面对硬件功能进行描述
- 生成称为网表 (netlist) 的描述文件，交给电路布局软件生成对应的电路连接
- 比较主流的HDL包括Verilog和VHDL

# VERILOG

---

- 1983年提出，1995年成为IEEE标准IEEE 1264-2001

```
module counter(rst, clk, cnt);
    input rst, clk;
    output reg [3:0] cnt;
    always @ (posedge clk)
        if(rst)
            cnt <= 4'b0000;
        else
            cnt <= cnt + 1'b1;
endmodule
```

# VHDL

- 1983年提出，1987年成为IEEE标准IEEE 1097-2002

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is

generic(n: natural :=2);
port(clock: in std_logic;
      clear: in std_logic;
      count: in std_logic;
      Q: out std_logic_vector(n-1 downto 0)
);
end counter;

architecture behv of counter is
```

# 系统级硬件设计语言

---

- 随着电路规模的增大，Verilog和VHDL难以支持集成芯片设计
- 抽象程度更高的工具使得开发者可以关注芯片的系统和功能，自动生成对应的电路、信号和连线
- 标志性的系统级设计语言包括SystemC和SpecC

# SYSTEMC

- SystemC是一个基于C++的领域专用语言

```
#include "systemc.h"

SC_MODULE (first_counter) {
    sc_in_clk      clock;           // Clock input of the design
    sc_in<bool>    reset;          // active high, synchronous Reset input
    sc_in<bool>    enable;         // Active high enable signal for counter
    sc_out<sc_uint<4>> counter_out; // 4 bit vector output of the counter

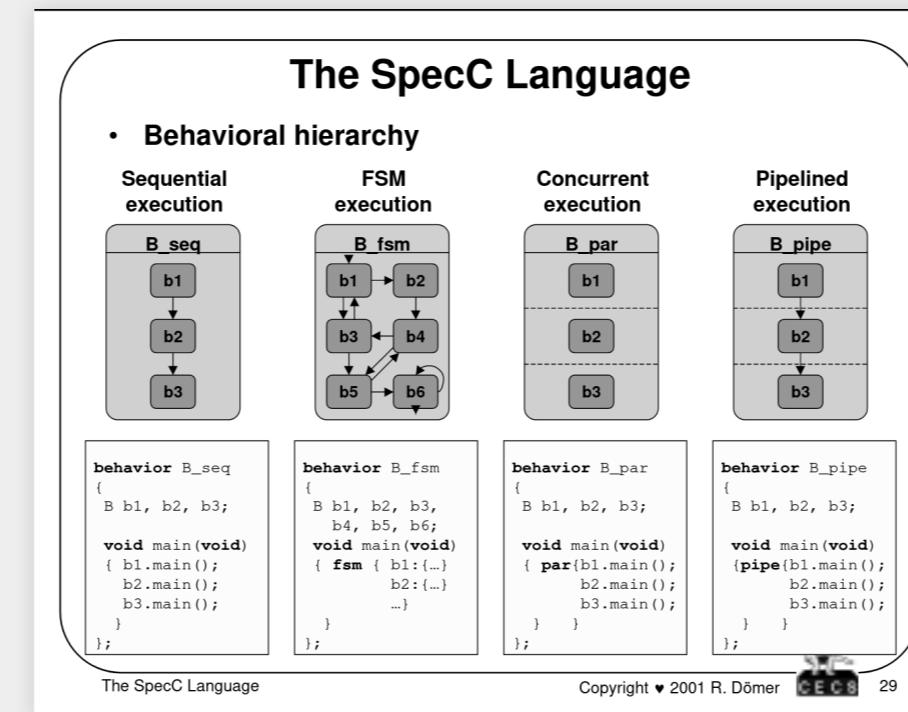
    sc_uint<4>    count;

    void incr_count () {
        if (reset.read() == 1) {
            count = 0;
            counter_out.write(count);
        } else if (enable.read() == 1) {
```

代码来源：[http://www.asic-world.com/code/systemc/first\\_counter.cpp](http://www.asic-world.com/code/systemc/first_counter.cpp)

# SPECC

- SpecC是基于ANSI C语言开发的扩展语言
- 加入了许多用于描述硬件行为的扩展



图片来源：<http://www.cecs.uci.edu/~specc/>

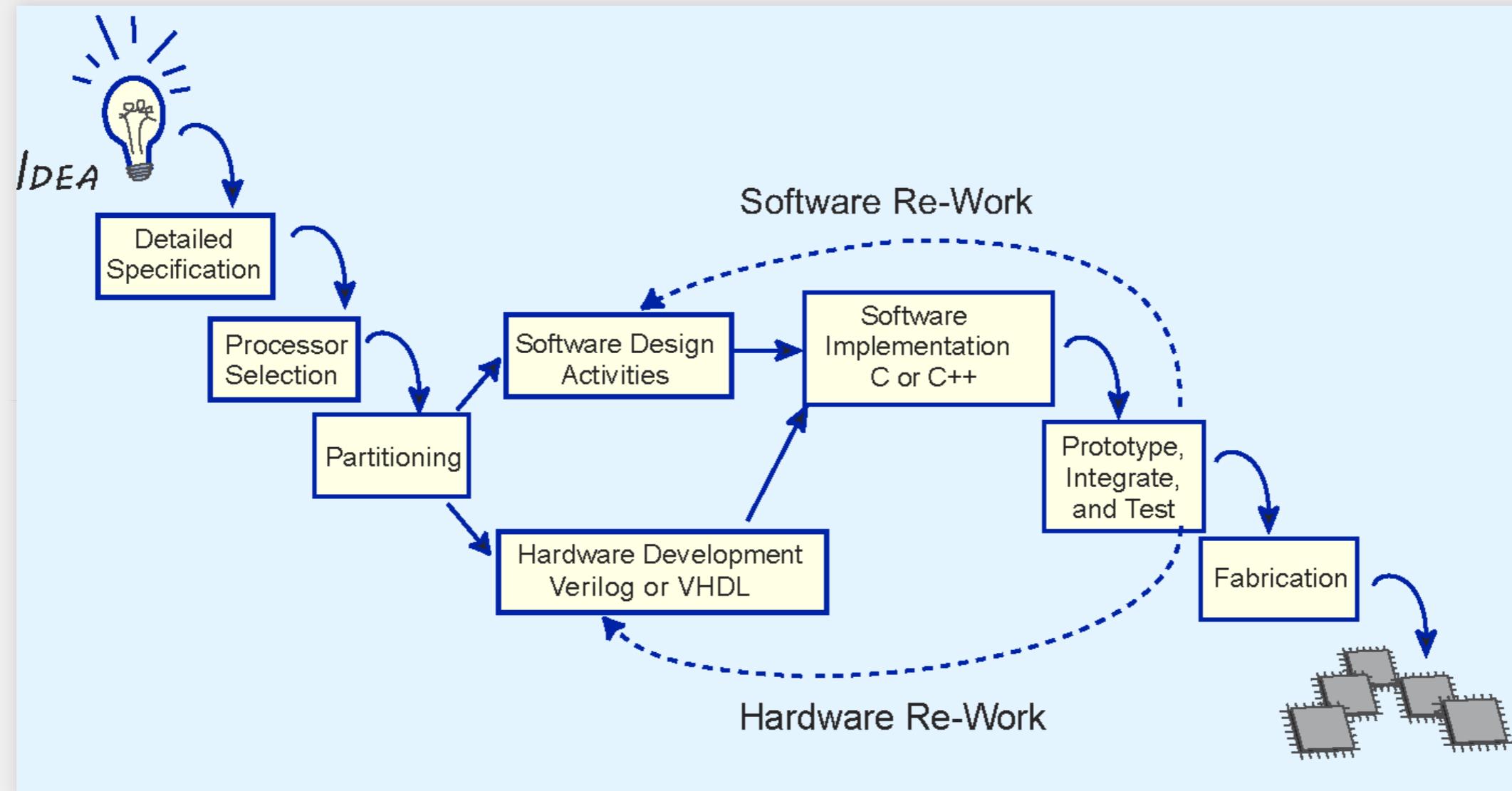
# 传统嵌入式系统开发流程

---

传统的嵌入式系统开发设计流程如下：

- 根据功能描述生成详细的说明文档
- 选择采用合适的处理器或者设计制造一个合适的处理器
- 对系统进行软硬件划分
- 对软硬件部分进行开发
- 搭建原型并进行测试

# 传统嵌入式系统开发流程示意图

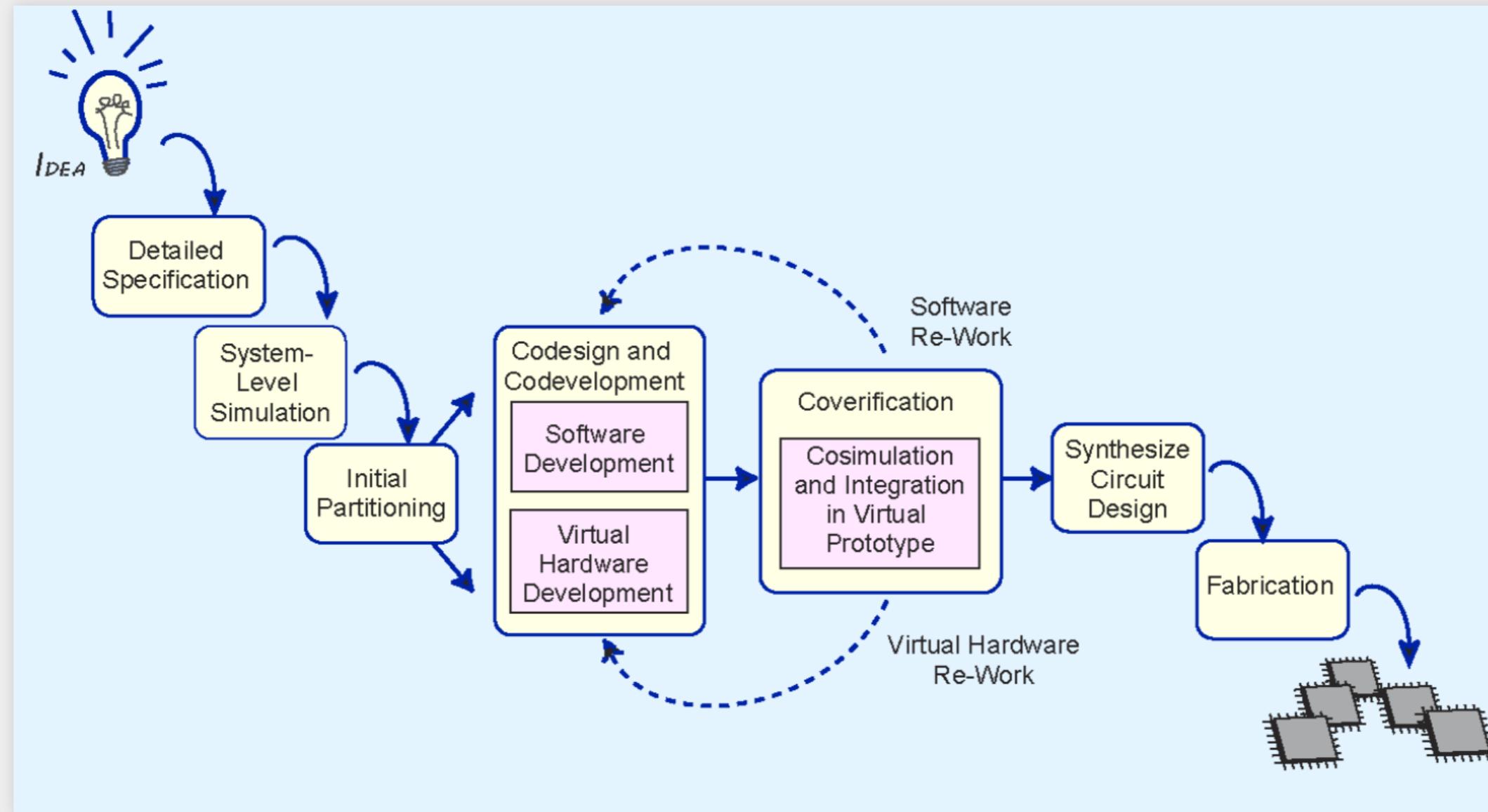


# 协同嵌入式系统开发流程

---

- SystemC和SpecC等系统设计语言引入了协同开发
- 硬件开发和软件开发可以采用同样的编程语言
- 协同开发团队持续进行软件和硬件的设计开发
- 协同开发加快了开发周期，有助于提高产品质量

# 系统嵌入式系统开发流程示意图



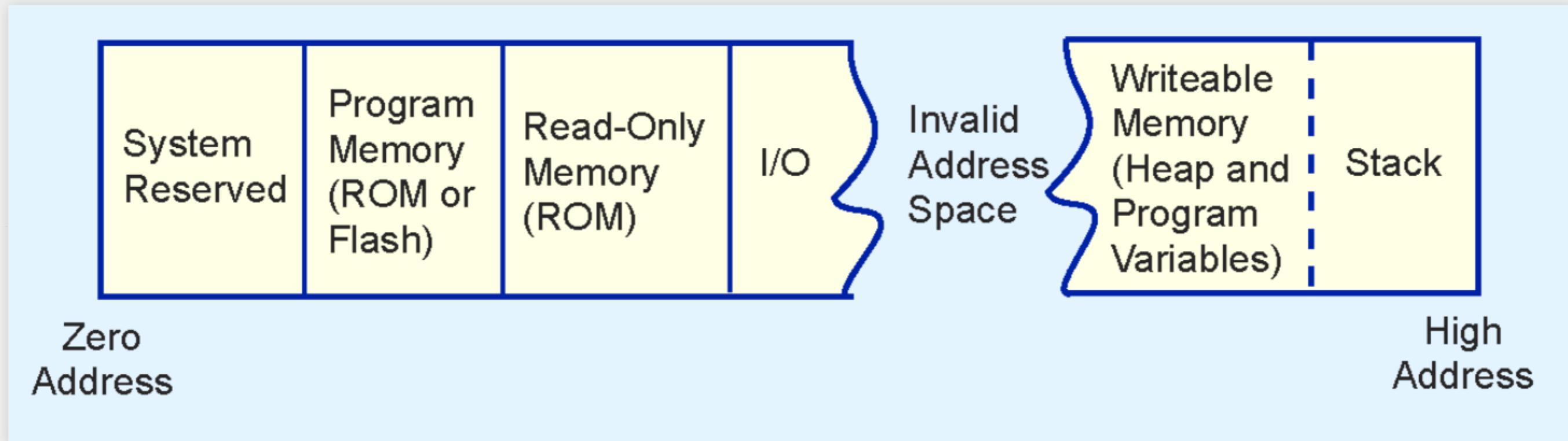
# 嵌入式系统的软件

---

- 嵌入式系统和通用计算机在资源、实时性要求上有许多不同
- 其中，内存限制是嵌入式设备最主要的约束之一
  - 嵌入式系统中的内存总量有限
  - 出于实时性的需求，通常不能采用虚拟内存

# 嵌入式系统的内存分布

- 嵌入式系统中可能包含多种类型的内存
- 存储空间可能不是连续的
- 为了避免内存泄露，一些程序避免使用动态内存分配



# 嵌入式系统的操作系统

---

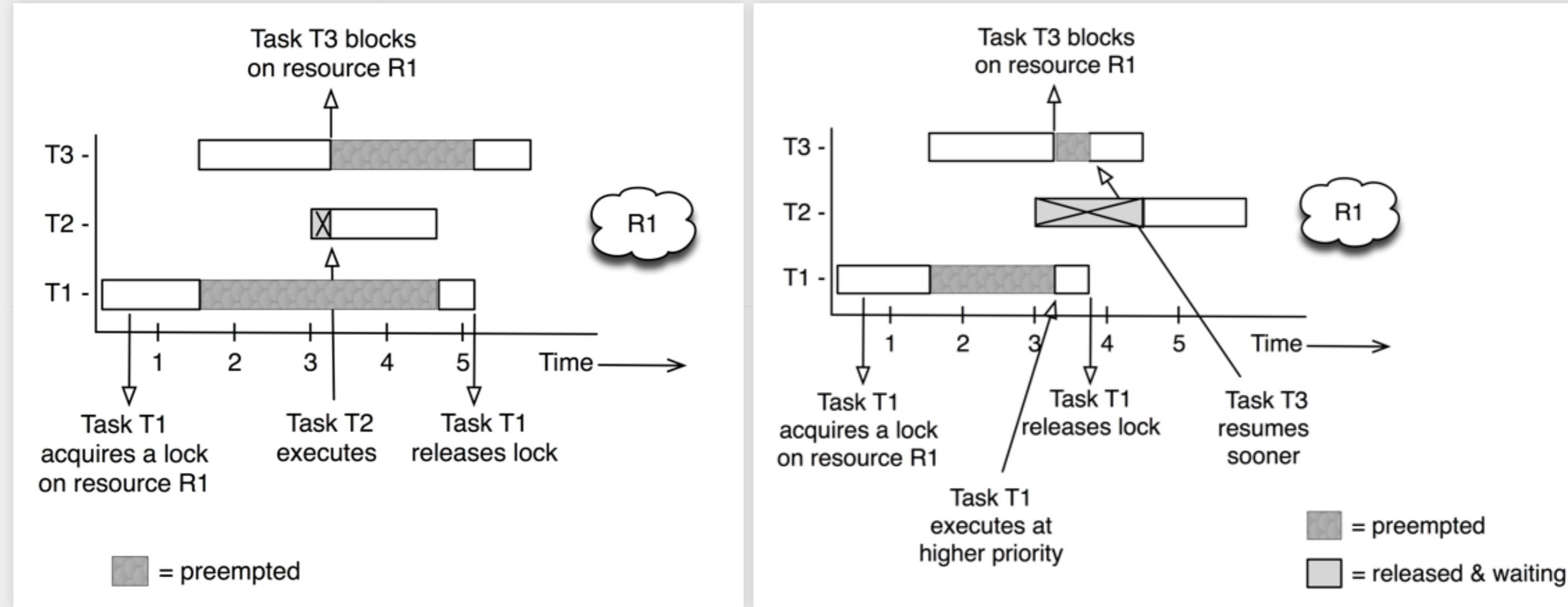
- 选择嵌入式操作系统主要的性能指标包括响应性和操作系统所需的存储空间
- 嵌入式设备需要频繁响应物理事件，这些事件通常通过中断的形式发送给操作系统
- 响应性的主要指标包括**上下文切换事件**和**中断延迟**
- 中断延迟（Interrupt Latency）指产生中断到执行中断处理程序第一条指令的时间

# 支持高响应性的中断系统

---

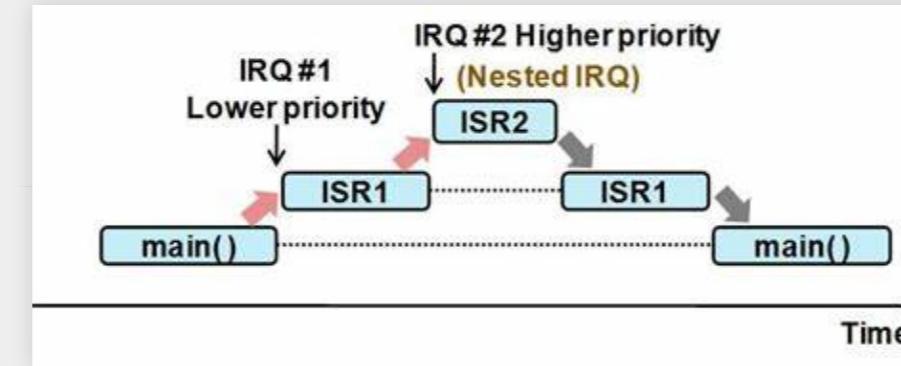
- 为了快速响应高优先级的外部事件，必须采用抢占式的任务调度
- 但是抢占式任务调度可能出现**优先级反转问题**
- 优先级反转1：
  - 高优先级的任务依赖低优先级的任务
  - 低优先级任务占用了高优先级任务的关键资源
  - 解决方案：优先级继承。出现优先级反转时，所有任务优先级置为同一级别
- 优先级反转2：
  - 处理低优先级任务时关闭了中断导致高优先级任务无法执行
  - 解决方案：中断嵌套。允许高优先级抢占低优先级中断的执行

# 优先级反转和优先级继承



图片来源：<https://www.drdobbs.com/jvm/what-is-priority-inversion-and-how-do-yo/230600008>

# 中断嵌套



图片来源：<https://letanphuc.net/2015/03/stm32f0-tutorial-3-external-interrupts/>

# 嵌入式操作系统的其它指标

---

- 不同嵌入式设备的功能和存储空间各不相同
- 嵌入式操作系统通常是模块化的，不同嵌入式系统可以根据需求进行定制
- 为了保证通用性，需要操作系统遵循一定的标准接口和协议
- 例如IEEE 1003.1-2001标准POSIX（Portable Operating System Interface）是一个标准化的UNIX技术规范
- 可用的嵌入式操作系统包括Linux、QNX、嵌入式Windows等

# 嵌入式系统软件开发特点

---

- 嵌入式软件开发受制于硬件资源条件约束，例如存储空间受限等
- 同时也受到实际复杂环境的影响：事件的触发情况可能非常复杂
- 由于嵌入式设备的更新可能比较困难，对软件测试的要求更高
- 一些工作引入形式化语言对软件的功能进行验证

# 第十四讲结束

# 本期内容总结

---

- 了解常见的嵌入式硬件类型和开发方法
  - 标准处理器
  - 了解可重构硬件PAL、PLA、FPGA的原理
  - 了解面向专用芯片设计的硬件描述语言和设计开发流程
- 了解嵌入式操作系统和软件开发的特点

# 扩展阅读

---



# Q & A