LoongArch CPU 设计实战

第6天：Linux系统适配与构建

教师：杨建磊、万 寒

助教：苏阳、周振源

北京航空航天大学 计算机学院

2025年8月2日 四川 成都

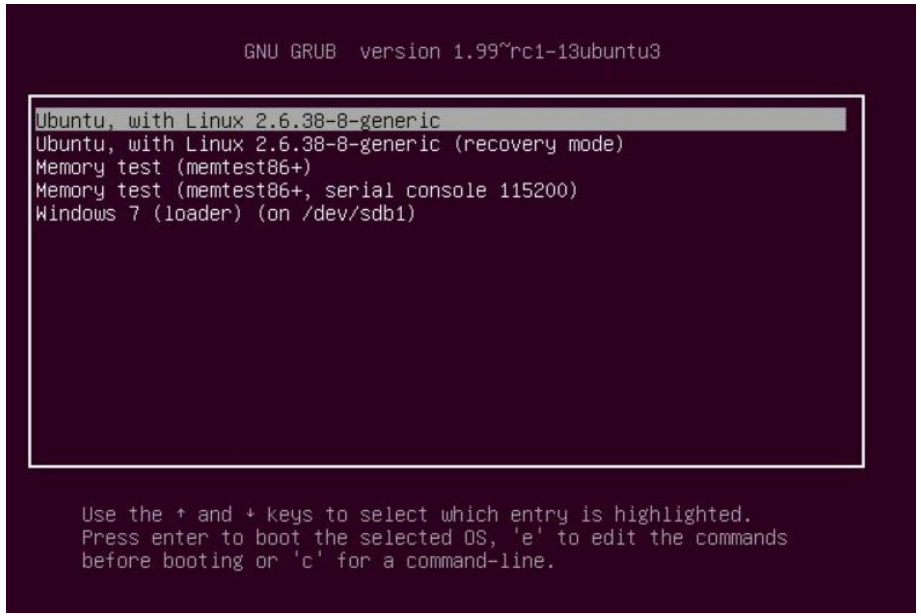
Day6 目录

1.1 运行操作系统所需要的支持

- 你可能听过BIOS和GRUB

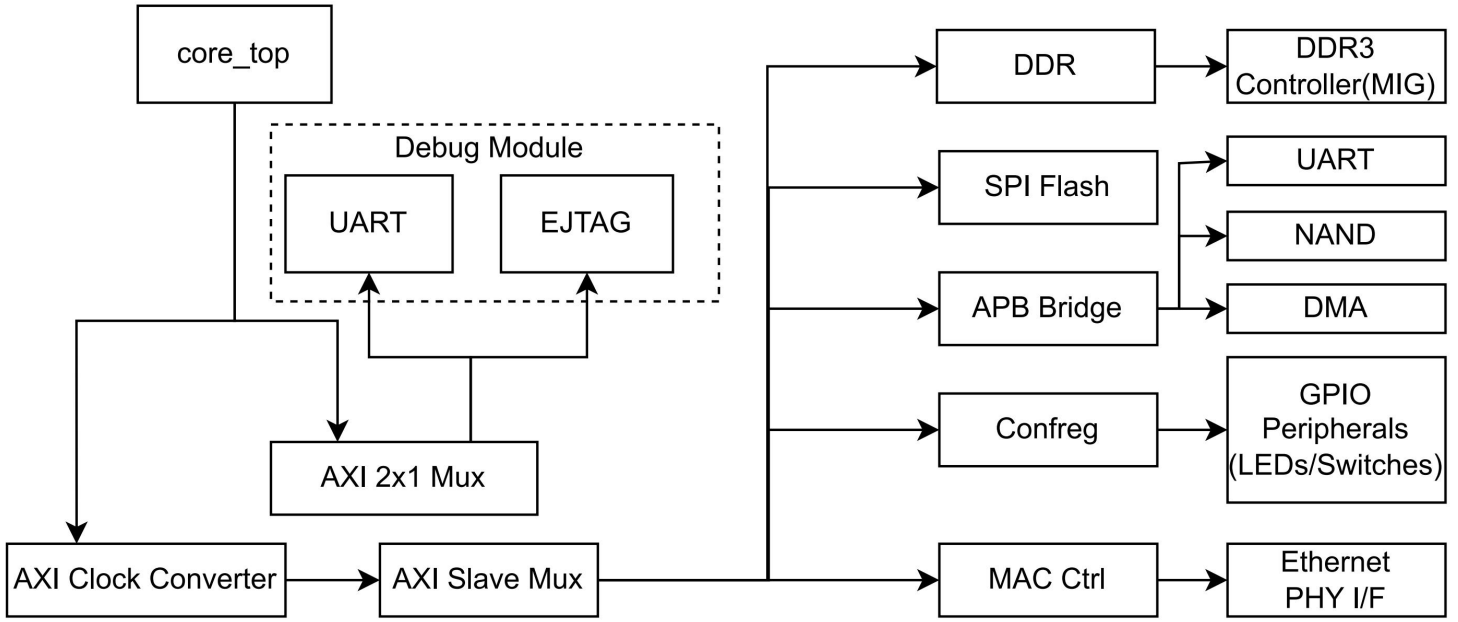
开机的时候，电脑在干什么？

  - BIOS是主板上预装的固件，是计算机开机后首先运行的最底层程序
    - 检查并初始化硬件设备
    - 加载和执行引导加载程序（GRUB）
  - GRUB是一个引导加载程序（Bootloader），用于操作系统引导
    - 被BIOS调用后，负责加载Linux/Windows
    - 支持多系统启动（双系统）
    - 提供引导菜单，配置操作系统



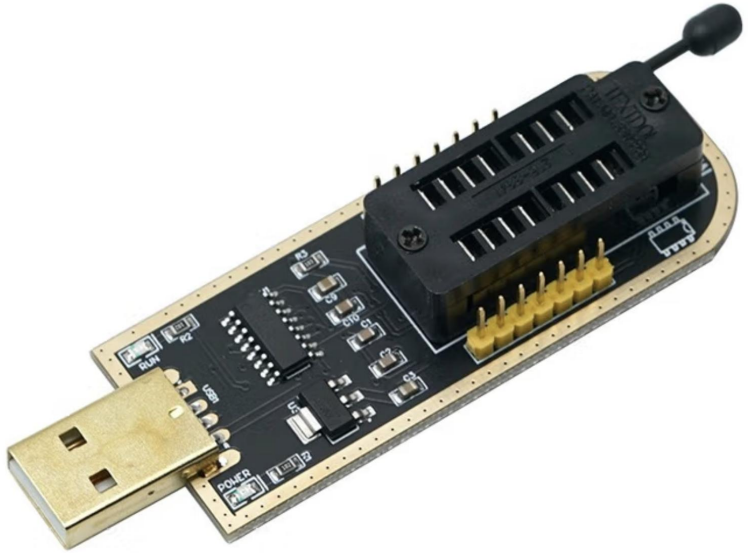
1.1 运行操作系统所需要的支持

- 芯片上电
  - SPI Flash存放Bootloader，如PMON、UBoot
  - 硬件开始运行第一条指令
- 硬件初始化
  - 从NAND中读取内核和rootfs，加载到RAM（DDR3）
- 启动内核
  - 运行操作系统



1.2 Chiplab启动内核

- Bootloader烧写
  - 烧写PMON文件或UBoot文件（.bin）到可插拔SPI Flash上
- 处理器烧写
  - 通过JTAG烧写处理器的bit流文件到FPGA
- 加载内核
  - FPGA上电，运行PMON/UBoot
  - 搭建tftp服务器加载内核或从SD卡中加载内核
  - 运行内核



- Chiplab配套相应串口bit流文件，可以连接电脑直接通过串口将PMON传输到FPGA的SPI Flash中
- 但若使用其他SoC，则需要自行烧录

1.3 Linux初始化过程

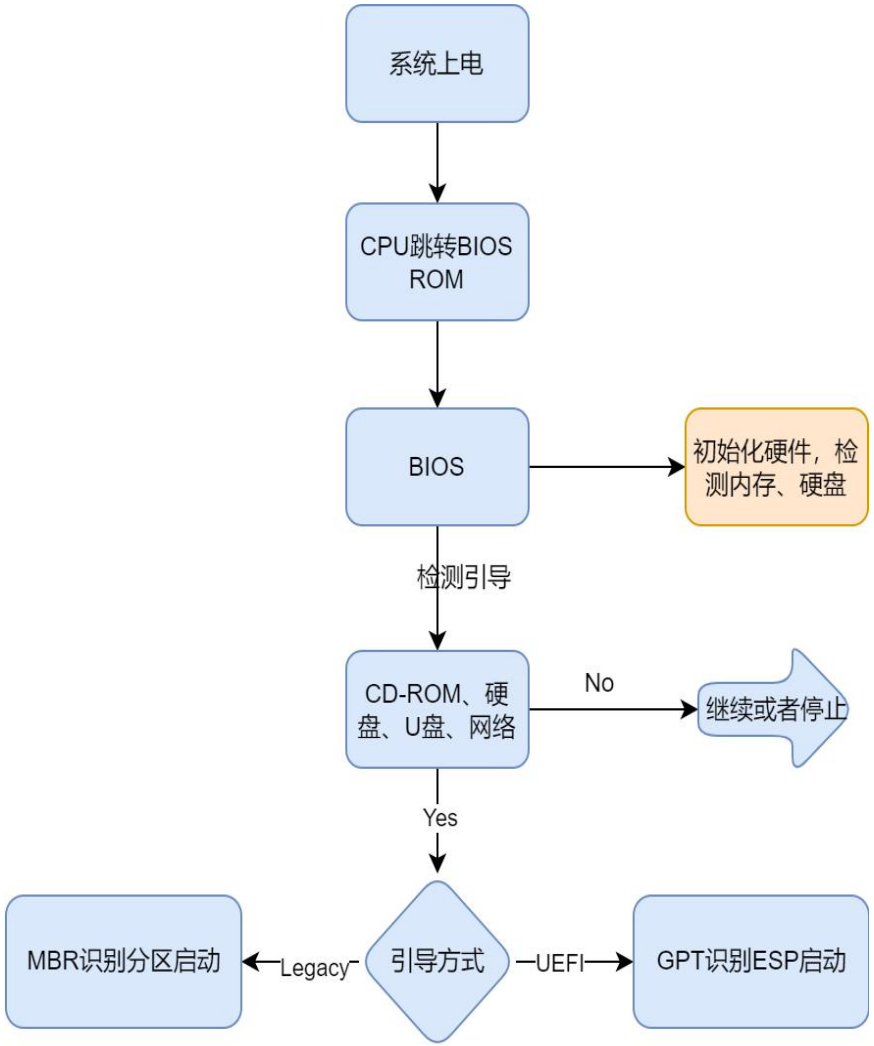
- 我是操作系统，我在哪？
  - 我在哪？——开机自检（CPU、硬盘）
  - 我能使用什么？——初始化硬件，加载驱动
  - 需要哪些准备工作？——启动系统服务、守护进程（网络、声音、systemd...）
  - 接下来要干什么？——挂载根文件系统，等待用户进程启动

为什么每次开机都要等半天?

```
         Starting /etc/rc.local Compatibility...
[  OK  ] Started ACPI event daemon.
         Starting LSB: automatic crash report generation...
         Starting LSB: Record successful boot for GRUB...
         Starting LSB: MD monitoring daemon...
[  OK  ] Started FUSE filesystem for LXC.
         Starting System Logging Service...
[  OK  ] Started Regular background program processing daemon.
         Starting LSB: Set the CPU Frequency Scaling governor to "ondemand"...
         Starting Auto import assertions from block devices...
[  OK  ] Started Permit User Sessions.
[  OK  ] Started /etc/rc.local Compatibility.
[  OK  ] Started Auto import assertions from block devices.
[  OK  ] Started System Logging Service.
         Starting Terminate Plymouth Boot Screen...
         Starting Hold until boot process finishes up...
[  OK  ] Started LSB: automatic crash report generation.
[  OK  ] Started LSB: MD monitoring daemon.
[  OK  ] Started LSB: Set the CPU Frequency Scaling governor to "ondemand".
[  OK  ] Started Terminate Plymouth Boot Screen.
[  OK  ] Started Hold until boot process finishes up.
[  OK  ] Started LSB: daemon to balance interrupts for SMP systems.
         Starting Set console scheme...
[  OK  ] Started Set console scheme.
         Starting Authenticate and Authorize Users to Run Privileged Tasks...
[  OK  ] Started LSB: Record successful boot for GRUB.
[  OK  ] Started Authenticate and Authorize Users to Run Privileged Tasks.
[  OK  ] Started Accounts Service.
[  OK  ] Started LXD - container startup/shutdown.
```

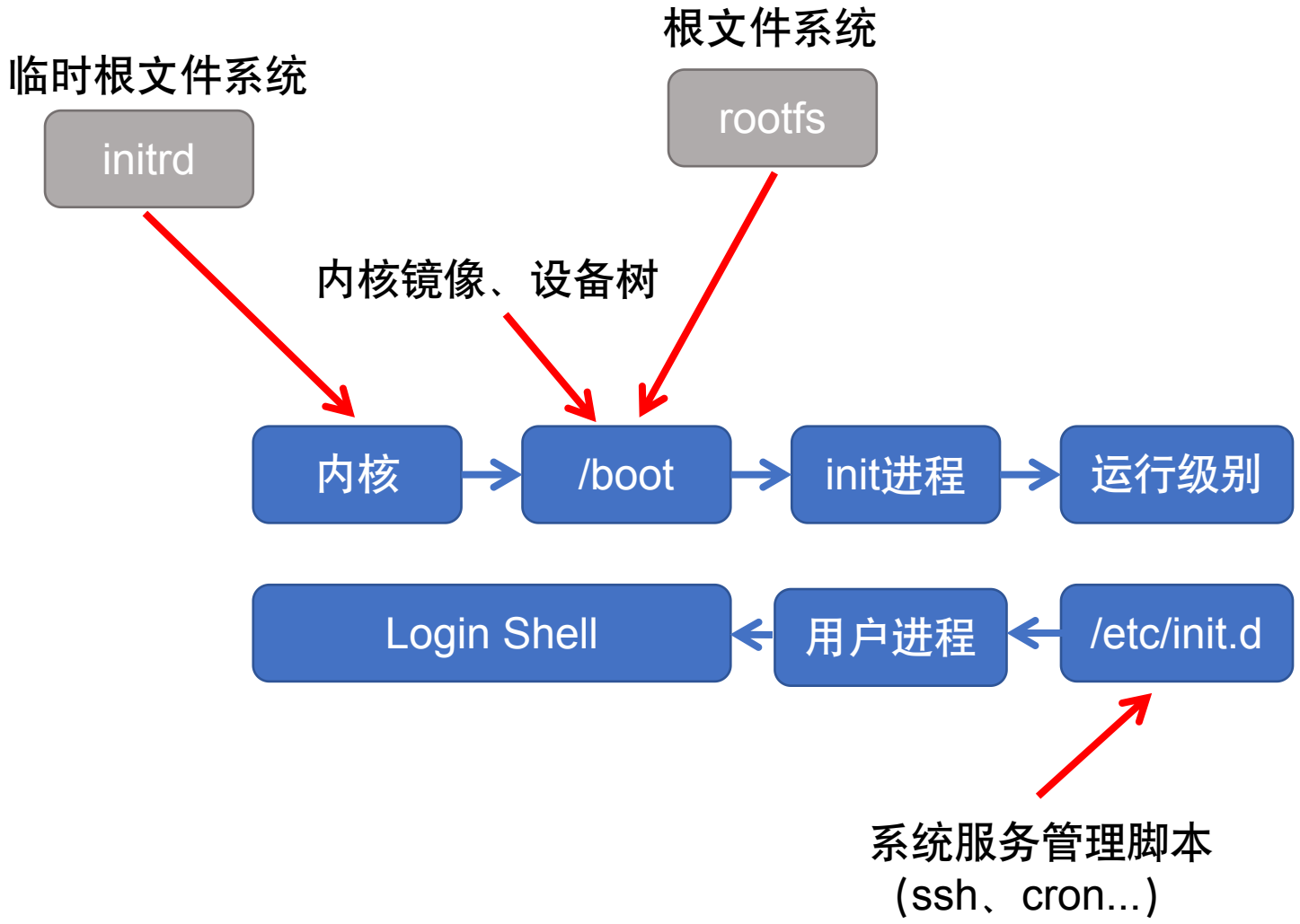
1.3 Linux初始化流程

- 上电和硬件初始化
  - 上电：芯片主板被供电
  - 处理器执行固化启动代码（ROM）
  - 初始化时钟、寄存器
- 第一阶段Bootloader
  - 初始化SDRAM、分区、存储设备基本启动等
  - 读取后续固件和Bootloader主体
- 第二阶段Bootloader
  - 初始化外设和更复杂的硬件
  - 加载Linux内核和设备树
  - 加载initrd，设置启动参数
  - 跳转到Linux内核入口



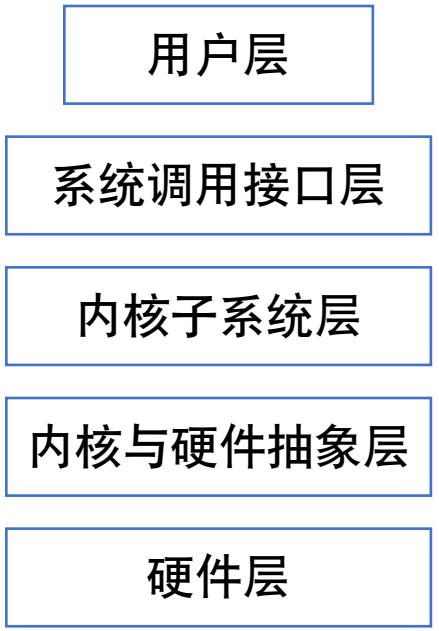
1.3 Linux初始化流程

- 内核启动
  - 初始化CPU
  - 初始化内存
  - 注册驱动
  - 挂载根文件系统
- 启动第一个用户进程
  - /sbin/init



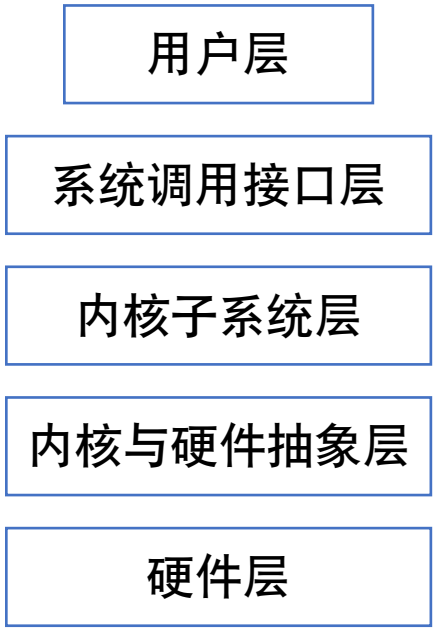
1.4 Linux分层抽象

- 用户层
  - 用户通过insmod/modprobe加载驱动模块（.ko文件）
  - 用户空间请求最终通过系统调用进入内核空间
- 系统调用接口层
  - 系统调用处理诸如insmod，加载模块，打开设备文件等操作
- 内核子系统层&驱动层
  - 根据insmod请求解析驱动模块，将模块加载到内核空间
  - 驱动注册后成为受管理的驱动对象
  - 设备模型，Linux通过总线-设备-驱动三层模型抽象管理硬件



1.4 Linux分层抽象

- 总线与匹配层
  - 内核在注册驱动和设备时把两者链入各自的链表，总线层负责轮训匹配
  - 依据compatible字符串等匹配信息，与设备树/总线上的设备自动配对
  - 匹配成功后，内核调用probe函数
- 内核与硬件抽象层
  - probe函数执行时，操作IO地址空间、寄存器、中断等，对实际硬件进行配置和初始化
- 硬件层
  - 实际配置和启动硬件，物理设备被初始化后，可以响应和处理来自上层的操作



1.5 Linux设备树

- SoC中的外设地址如何被操作系统感知？——设备树
  - 设备树（dts文件）是一种结构化、平台无关的数据格式，用于描述SoC各种硬件的信息，如外设类型、地址、中断信号、时钟等
  - 内核启动时会读取设备树，将其中的信息注册给相应的驱动，驱动通过设备树获知硬件具体细节
  - 各外设的reg地址段和长度需要与SoC完全对照
  - 驱动需要能接收设备树参数（比如compatible，reg，interrupts）

```
/* dts设备树片段 */
soc {
    #address-cells = <1>;
    #size-cells = <1>;

    ethernet@6000 {
        compatible = "yourvendor,ethernet-ctrl";
        reg = <0x6000 0x60>;
        interrupts = <5>;
        /* 其他配置 */
    };
};
```

1.5 Linux设备树

- SoC中的外设地址如何被操作系统感知？——设备树
  - 内核匹配compatible字符串
    - 设备树匹配表
    - 驱动自动识别
    - 自动导出
  - 定义Platform Driver结构体
    - 设备初始化
    - 设备释放
    - 设备树匹配
    - 简化注册

```c
/* 匹配compatible字符串 */
static const struct of_device_id ethernet_of_match[] = {
    { .compatible = "yourvendor,ethernet-ctrl", },
    { /* others */ }
};
MODULE_DEVICE_TABLE(of, ethernet_of_match); // 内核自动导出驱动所支持的硬件ID信息
```

```c
/* Platform Driver结构体 */
static struct platform_driver ethernet_driver = {
    .probe = ethernet_probe,      // 设备就绪时调用
    .remove = ethernet_remove,    // 设备移除时调用
    .driver = {
        .name = "yourvendor-ethernet",
        .of_match_table = ethernet_of_match, // 连接设备树
    },
};
module_platform_driver(ethernet_driver); // 简单注册宏
```

1.5 Linux设备树

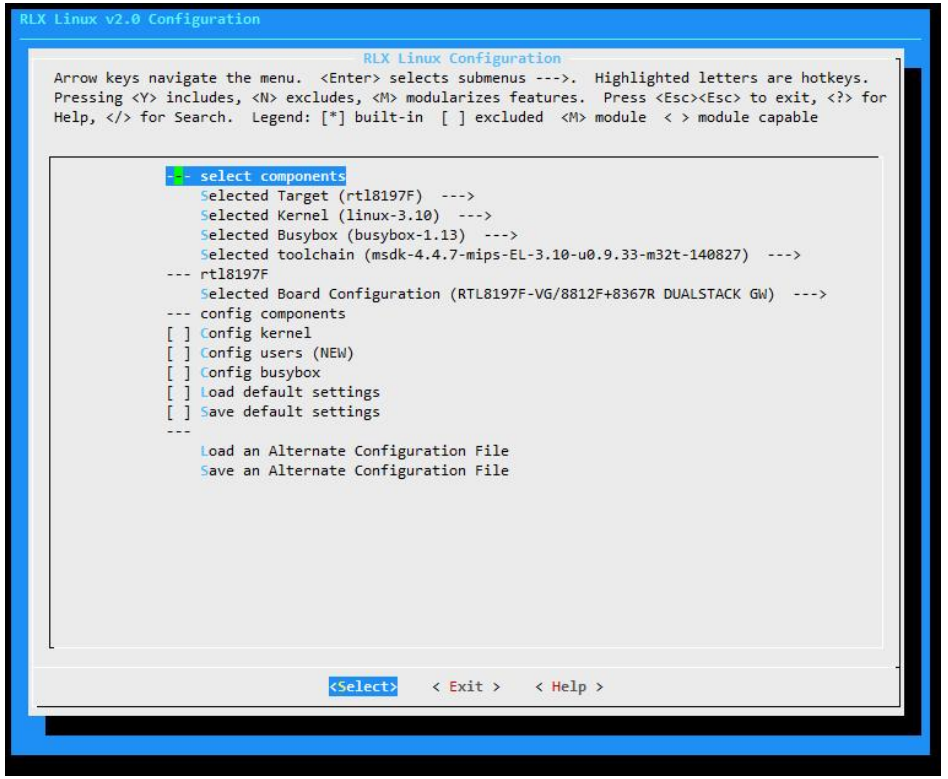
- SoC中的外设地址如何被操作系统感知？——设备树
  - 内核驱动获取dts参数并硬件初始化
    - 获取设备资源信息
    - IO内存映射
    - probe初始化入口

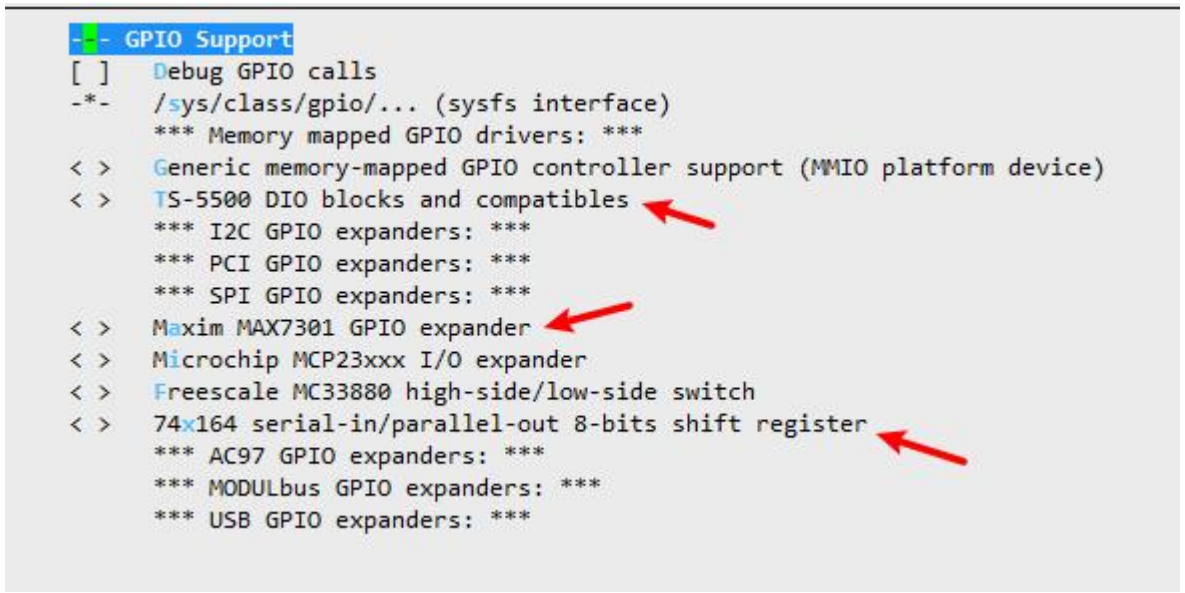
```
/* probe函数获取dts参数、硬件初始化 */
static int ethernet_probe(struct platform_device *pdev) {
    struct device *dev = &pdev->dev;
    struct resource *res;
    void __iomem *base;

    // 获取reg (物理地址、长度), 映射IO
    res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    base = devm_ioremap_resource(dev, res);
    ...
}
```

1.6 Kconfig

Kconfig的由来和作用

- 内核支持数千种硬件和功能模块，开发者需要灵活选择和定制
- Kconfig用于解决管理大量配置项、提升定制化编译效率

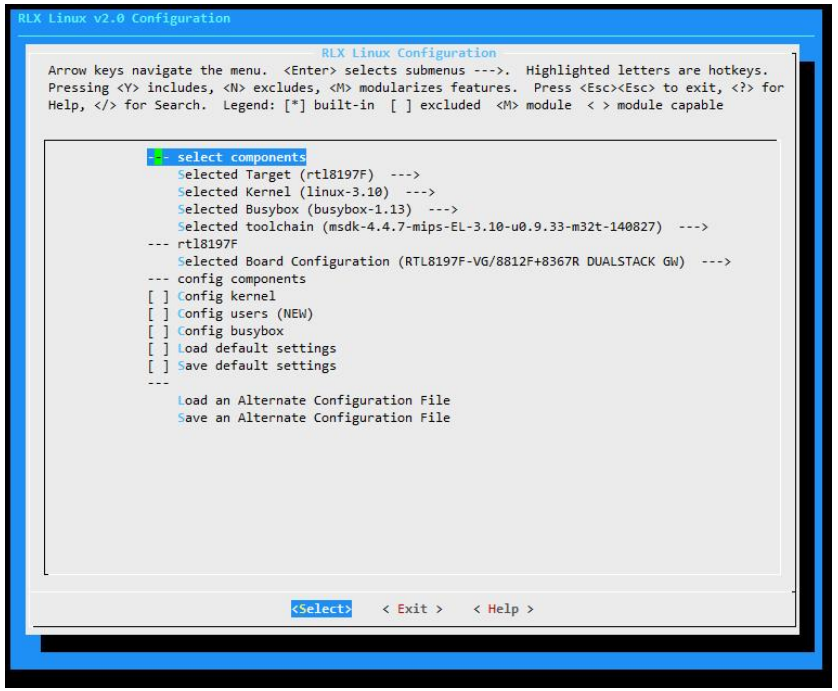




1.6 Kconfig

- Kconfig如何编写
  - 在驱动目录 drivers/mydrv(example)创建Kconfig文件
  - 编辑Kconfig文件

```
// 单独罗列
config MYDRV_DEBUG
    bool "Enable debug for my driver"
    default n
    help
      Select this option if you want to enable debug output for my driver.

// 归属到某个menu
menu "My Driver options"
config MYDRV_DEBUG
    bool "Enable debug for my driver"
    default y
endmenu
```



1.6 Kconfig

- Kconfig如何编写
  - 主Kconfig包含新添加的Kconfig
  - source "drivers/mydrv/Kconfig"
  - menuconfig感知配置项
- Kconfig的高级优化
  - 其他类型（int、string、hex）
  - 依赖关系（只在某条件下显示）
  - 互斥关系（出现一个不能出现另外一个）

```
// logic in C
#ifdef CONFIG_MYDRV_DEBUG
    printk("Debug message...\n");
#endif
```

```
// 其他类型
config MYDRV_TIMEOUT
    int "Timeout value"
    default 100
    range 10 1000

// 依赖关系
config MYDRV_DEBUG
    bool "Enable debug"
    depends on MYDRV

// 互斥关系
config FOO
    bool

config BAR
    bool
    depends on !FOO
```

1.6 Kconfig

- defconfig
  - Linux内核使用的默认配置文件，用于预设内核配置选项
  - 其作用在于：
    - 为特定平台/板级/芯片/应用场景准备了一键默认配置
    - 快速生成工作.config文件，为后续的make编译提供基础

Kconfig：你可以选择什么
defconfig：特定场景的推荐配置选择集合
.config：最终会用的配置集合（可手工修改）

linux / arch / loongarch / configs / la32rmega_defconfig

gmlayer0 LoongArch32r: add support for Lain/EULA Chip and megaSoC

Code Blame 336 lines (336 loc) · 7.67 KB

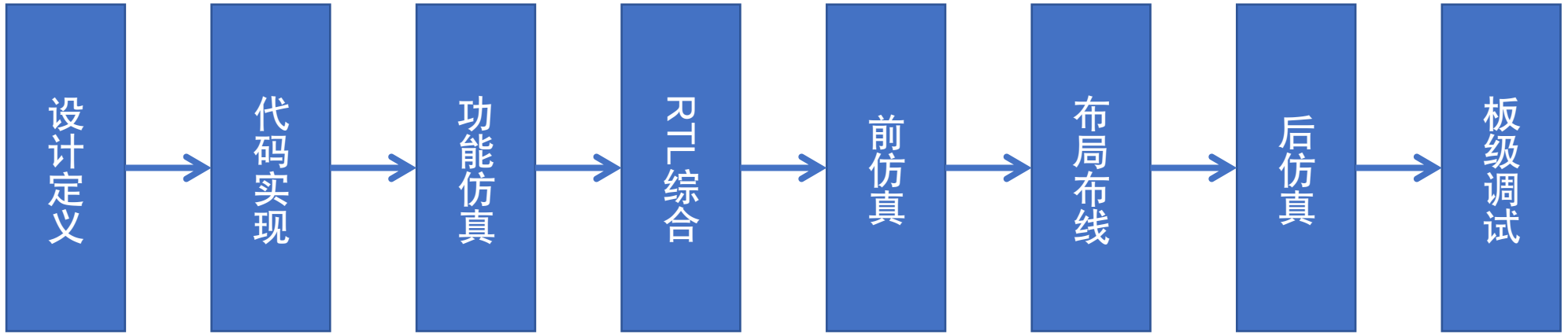
```
1   CONFIG_SYSVIPC=y
2   CONFIG_NO_HZ=y
3   CONFIG_HIGH_RES_TIMERS=y
4   CONFIG_BPF_SYSCALL=y
5   CONFIG_PREEMPT=y
6   CONFIG_BSD_PROCESS_ACCT=y
7   CONFIG_BSD_PROCESS_ACCT_V3=y
8   CONFIG_LOG_BUF_SHIFT=18
9   CONFIG_MEMCG=y
10  CONFIG_CFS_BANDWIDTH=y
11  CONFIG_RT_GROUP_SCHED=y
12  CONFIG_CGROUP_PIDS=y
13  CONFIG_CGROUP_FREEZER=y
14  CONFIG_CGROUP_DEVICE=y
15  CONFIG_CGROUP_CPUACCT=y
16  CONFIG_CGROUP_PERF=y
17  CONFIG_CGROUP_BPF=y
18  CONFIG_CHECKPOINT_RESTORE=y
19  CONFIG_SCHED_AUTOGROUP=y
20  CONFIG_SYSFS_DEPRECATED=y
21  CONFIG_RELAY=y
22  CONFIG_EXPERT=y
23  CONFIG_USERFAULTFD=y
24  CONFIG_PERF_EVENTS=y
```

Day6 目录

2.1 综合的基本概念

什么是综合？

- 综合（Synthesis）是EDA工具（如Synopsys DC、Vivado等）自动将Verilog/VHDL描述转为门级网络结构的过程
- 综合分为两大过程：RTL综合和后端物理实现



2.2 RTL->门级电路

RTL综合

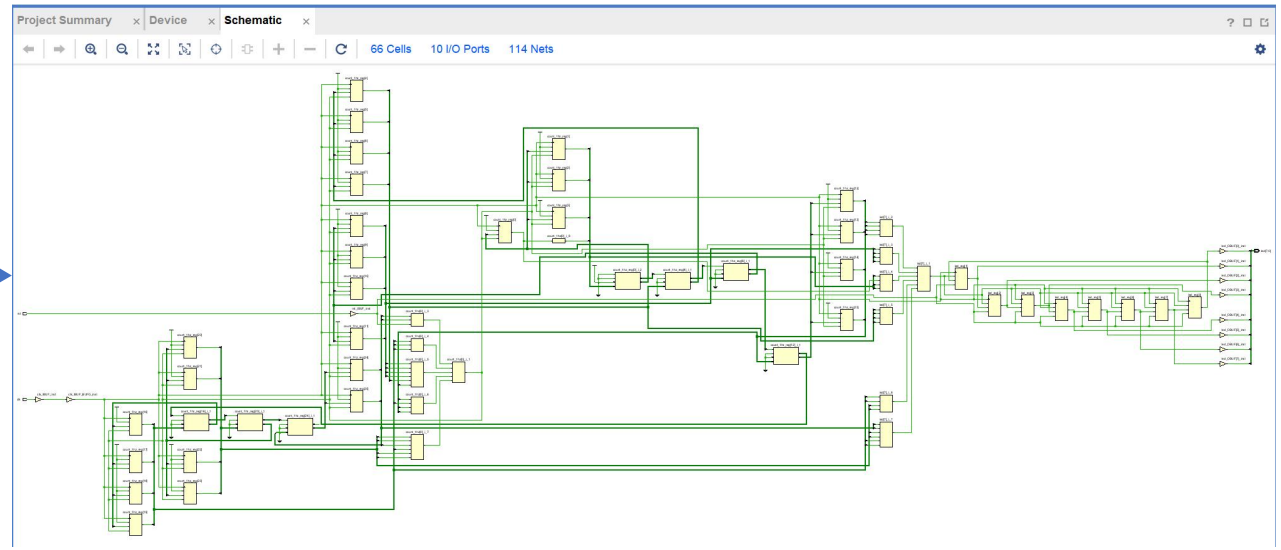
- RTL综合是把可仿真的、行为级Verilog/VHDL自动翻译成门级电路网表，即将用代码描述的电路逻辑转成标准门和触发器之间的网络连接表
- 其主要任务包含：
  - 语法检查，硬件化转化（如always@(posedge clk)转化为D触发器）
  - 逻辑优化（合并、重构、平衡等）
  - 应用设计约束（如时钟频率、IO、面积等）
- 输入输出：
  - 输入：RTL级Verilog/VHDL、约束（SDC、时序等）
  - 输出：门级网络（通常为Verilog网表文件和约束文件），可以进行综合后仿真

2.2 RTL->门级电路

LED.v

```
1  module LED (
2      input                   clk,
3      input                   btn,
4      output reg [7:0]        led
5  );
6
7  reg [31:0] count_1hz;
8  wire rst;
9  parameter TIME_CNT = 50_000_000;
10
11 assign rst = btn;    // Use button for RESET
12
13 always @(posedge clk) begin
14     if (rst)
15         count_1hz <= 0;
16     else if (count_1hz >= TIME_CNT)
17         count_1hz <= 0;
18     else
19         count_1hz <= count_1hz + 1;
20 end
21
22 always @(posedge clk) begin
23     if (rst)
24         led <= 8'b0000_1111;
25     else if (count_1hz == 1) begin
26         led <= {led[6:0], led[7]};
27     end
28 end
29 endmodule
```

synthesis



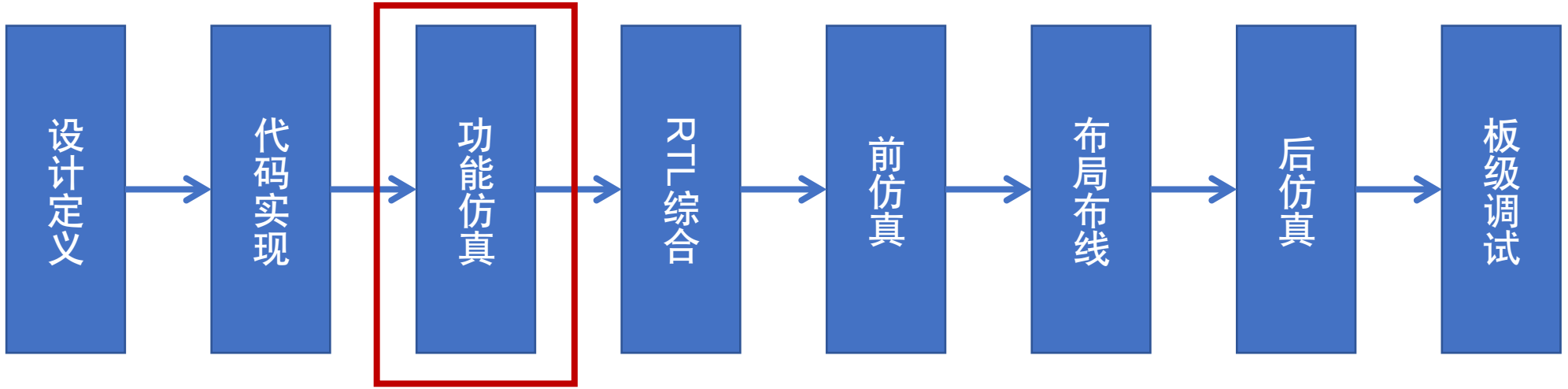
2.3 门级电路->芯片物理版图

后端物理实现

- 后端物理实现是将门级电路网表实现为芯片物理版图，即将逻辑连接关系和功能变成具体每个门、连线、管脚在芯片上的真实位置和形状
- 其主要任务包含：
  - 芯片/FPGA时钟树综合（CTS）、布局（placement）、布线（route）
  - 物理优化：减少拥塞、布线合法性、时序收敛、功耗完整性、DRC/LVS等物理规则检查
  - 版图生成和制造检查
- 输入输出：
  - 输入：门级网表、工艺库、物理/时序/IO约束、芯片封装信息
  - 输出：GDSII等最终的物理版图数据

2.4 处理器测试

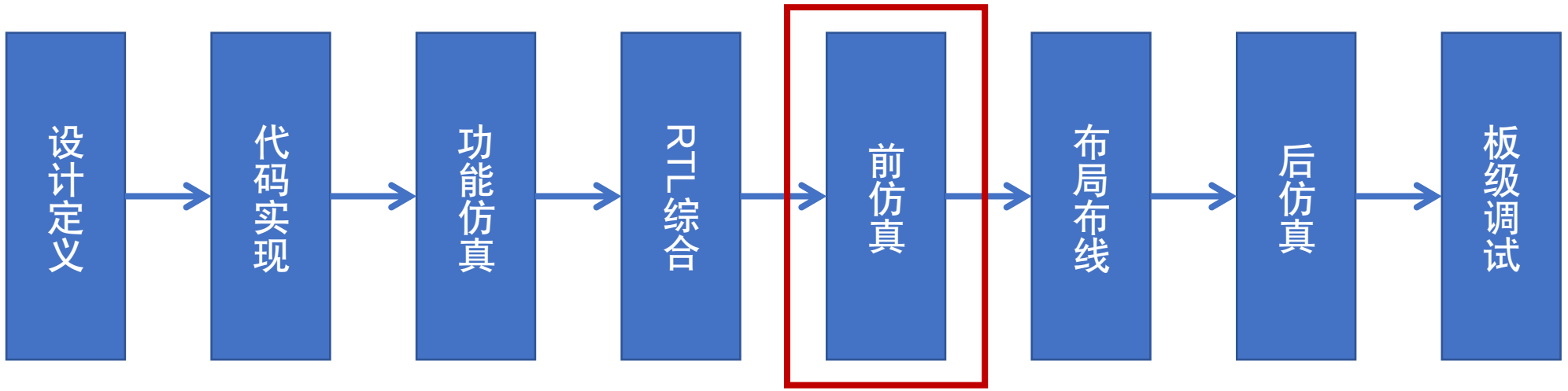
- 综合过程中的三类测试
  - 功能仿真（作为学生用的最多的）
  - 前仿真
  - 后仿真（跑得最慢，环境最真实的）



面对RTL级代码（Verilog/VHDL源代码）
验证代码逻辑是否满足设计功能要求，检查功能正确性
不考虑时序延迟，仅验证行为

2.4 处理器测试

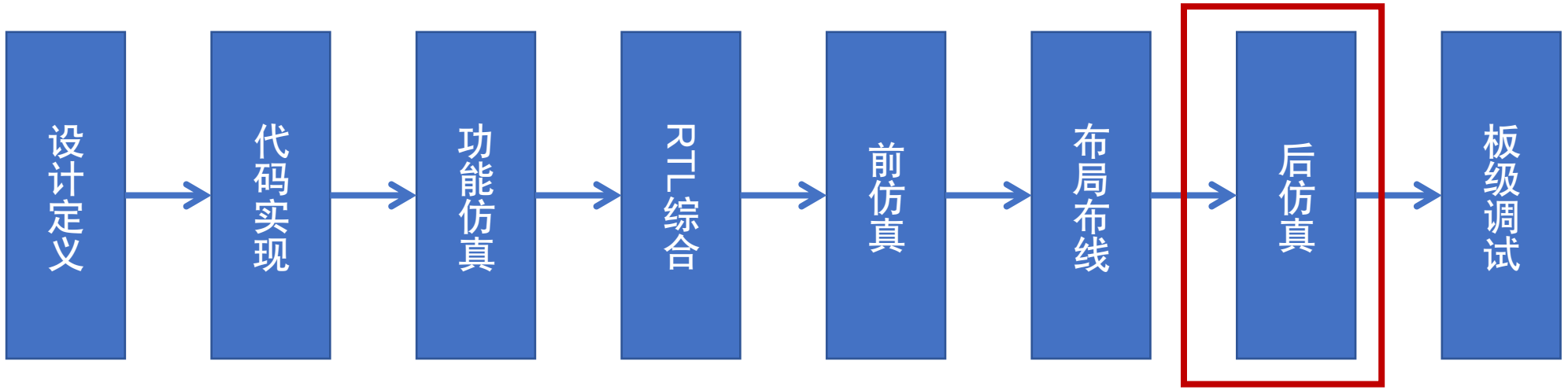
- 综合过程中的三类测试
  - 功能仿真（作为学生用的最多的）
  - 前仿真
  - 后仿真（跑得最慢，环境最真实的）



面对门级网表（.vg/.v/.edf文件）
验证RTL代码综合后，与原先设计功能是否一致
逻辑层面和RTL等效，但门级结构已固定，不考虑时序延迟

2.4 处理器测试

- 综合过程中的三类测试
  - 功能仿真（作为学生用的最多的）
  - 前仿真
  - 后仿真（跑得最慢，环境最真实的）



面对门级网表+标注精确时序的SDF文件
验证实际电路的时序和功能，排查setup/hold等时序违例
带有详细时延，仿真量激增

Day6 目录