华为面试数字芯片提纲

1、 时序逻辑/ 组合逻辑

同步逻辑/ 异步逻辑

同步时钟/ 异步时钟

2、亚稳态的概念/ 可能出现的场合和抑制手段

3、异步 FIFO 原理/ FIFO 的深度计算/ 同步 FIFO/ 要求自己写异步 FIFO 代码/ 自己尝试添加约束

4、双端口 RAM 的实现/ 要求自己写双端口 RAM

5、两级触发器同步/ 应用场合/ MTBF 的计算以及 2-FF 的计算

6、握手机制

7、同步复位/ 异步复位/ 异步复位同步释放/ 三者比较/ 代码

8、恢复时间/ 去除时间/

9、触发器组成/ 锁存器组成/ 区别

10、脉冲检测电路（非同步时钟如何处理，快到慢，慢到快，同频，长脉冲化为单时钟短脉冲）

11、可综合和不可综合概念/ timescale 的理解

12、组合逻辑环路概念

13、静态时序分析 STA/ 动态时序分析/ 二者对比

14、建立时间/ 保持时间/ 数据到达时间/ 数据需求时间/ 建立时间裕量/ 保持时间裕量

15、四类时序路径及起点终点/ 完整的时序路径包含三部分/ 建立时间检查，保持时间检查/ 恢复时间检查，去除时间检查/ 检查违例的解决方案

16、时序约束/ 时钟约束/ 输入输出约束/ 时序例外约束/ 多周期路径约束(Multicycle paths)/ 伪路径约束(False paths)/ min-max 约束/ bus skew

17、竞争冒险/ 消除方法

18、状态机分类/ 状态转移图/ 一段两段三段式状态机区别特征（优缺点，本质特征，状态数差别，输出改变时间差别，状态机也分为同步状态机（有 CLK 控制输出）和异步状态机）/ 思考具体的状态机实例（饮料售卖）

19、系统级/ 算法级/ RTL 级/ 门级/ 开关级

20、阻塞赋值/ 非阻塞赋值

21、function/ task/ repeat/ while/ for

22、详细研究对于运算数为 X/Z 以及逻辑运算中有 X/Z 的情况下结果分别如何/ reg 和 wire 缺省值/ 关系符 ===以及 !==的具体用法/ 一元简约运算符用法/ 位运算数位不对齐的情况/ 算数运算操作数的位数对不齐的情况运算结果以及被赋值的数位不够的情况/ 运算数存在有符号数的情况

23、Case /if 用法对比/ casez/ casex

24、Verilog 常见编码风格/ 串并转换/ 乒乓操作/ 流水线/ 面积速度互换/ 逻辑复用/ 流水线设计全加器，注意细节，注意位宽匹配/ 流水线乘法器（用逻辑门）/ 超前进位加法器

25、奇数分频/ 偶数分频/ 小数分频/ N+0.5 分频/ PLL 原理

26、Verilog 仿真常用关键词和用法/ 常用原语

27、FPGA 内部资源以及结构/ FPGA 烧写模式/ FPGA 和 CPLD 比较

28、提高系统工作频率的方法/

29、验证的概念/ 形式验证/ 常用验证方法

30、CMOS 功耗类别/ 数字芯片设计低功耗方案

31、SPI/ I2C/ RS232/ RS485/ UART 常见协议以及电平标准/ 波特率/ 比特率

32、DDR/ SDRAM/ SRAM/ DRAM/ ROM 原理速度功耗价格容量等等比较

33、数字滤波器/ 模拟滤波器/ 传递函数/ 如何画数字滤波器框图

34、二进制数运算/ 定点数浮点数/ 源码反码补码/ 相关数据的运算

35、逻辑代数表达式/ 化简公式/ 最大项最小项/ 卡诺图化简/ 真值表/ 状态转移图

36、CMOS 与非门/ 或非门/ 反相器（优良特性）/ 三态门/ 线与逻辑/ 传输门/ MOS 工作原理

37、施密特触发器

38、AD/DA 基本原理和结构

39、Serdes 接口知识/ 预加重/ 去加重/ 均衡/ 常用编码方式/ 时钟数据恢复(CDR)原理，参考时钟作用/

华为面试数字芯片提纲解答以及要点备注

1、 时序逻辑/ 组合逻辑

时序逻辑电路主要由组合逻辑电路和触发器等记忆元件组成，输出不仅取决于当前输入，还和电路原来状态有关（这里原来状态是指的中间的触发器等记忆元件部分的端口值也可能会影响到当前的电路总的输出）；组合逻辑电路主要由组合逻辑器件构成，电路无存储元件，输出仅仅取决于当前的输入。

同步逻辑/ 异步逻辑

同步逻辑是指的电路驱动时钟是有固定因果关系的逻辑，异步逻辑是电路驱动时钟之间没有固定因果关系的逻辑。通常同步逻辑电路的驱动时钟来源于同一个时钟源产生的相同时钟或者相位差可预知的不同时钟，这里不包括相位差在一定时钟周期内（例如 1000clk）依然无法确定公共周期的同源时钟；异步逻辑的电路驱动时钟来自不同源时钟或者相位差关系不确定的时钟。

同步时钟/ 异步时钟

同步时钟指的是时钟源来自于同一个时钟源生成的时钟，其相位差可计算预知，这里不包括来自同一时钟源但是在一定时钟周期内（例如 1000clk）依然无法确定公共周期的时钟，通常为由 PLL 生成的非整数分频或者倍频时钟。异步时钟是指的相位差无法预测的时钟，通常来自于不同的时钟源。

2、亚稳态的概念/ 可能出现的场合和抑制手段

亚稳态是指触发器的建立时间或者保持时间不能满足时触发器的输出介于 0 或者 1 两者之间的某个不稳定态。理想的触发器在时钟到来时刻采样数据，但现实之中时钟边沿通常具有一定斜率，因此数据需要在时钟到来的前后各一段时间内保持稳定，使得触发器能够准确采样数据，否则就可能出现亚稳态。

亚稳态通常出现在同步时钟驱动电路的异步复位信号来临和消失的时候，或者是异步时钟域之间数据传递的时候。前者解决方案是采用同步复位（通常会导致电路逻辑资源占用面积增加）/或者是异步复位同步释放方式避免亚稳态；后者通常考虑用异步 FIFO，双端口 RAM，双触发器，或者是握手机制等方法进行同步。此外还有例如降频，使用速度较快的触发器，改善时钟质量也会有一定效果。

3、异步 FIFO 原理/ FIFO 的深度计算/ 同步 FIFO/ 要求自己写异步 FIFO 代码/ 自己尝试添加约束

（H:\Desktop\Hardware_study\Async_FIFO）

异步 FIFO 主要用于实现异步时钟域之间的数据传输。异步 FIFO 由以下几部分构成。

- BRAM/DRAM 组成的缓冲区。用于缓存数据流，其深度的设定需要根据输入输出数据的时钟差别以及最大连续输入/输出数据量确定。
- 读写指针。其变化需要根据 FIFO 的空满状态以及当前的读写请求指令共同确定。当缓存为空则不可读，读指针不变；当缓存为满不可写，写指针不变。
- 空满状态标志。由于异步 FIFO 工作在不同时钟域，因此对空满状态的判断依赖于不同时钟域的读指针和写指针。为了便于区分“快一圈”的现象，可以考虑将指针位宽多设置一位，当最高位相同时候，读指针等于写

指针认为是读空，当最高位不同的时候，读指针等于写指针认为是写满；

- 同步电路。由于两者读写指针比较是在不同时钟域下进行，因此为了避免亚稳态需要进行同步设计。此处通常采用格雷码进行比较（传输端首先 BIN TO GRAY，经过两级触发器在目的端同步，然后 GRAY TO BIN，进行下一步的比较，得出空满状态），保证一次只有一位数据变化，利用格雷码结合两级触发器进行同步后，可以严格保证至少数据不会出错（起码是原地踏步，不会造成满状态写入的情况），中间加的两级触发器会对实际状态做延迟比较（避免了空读和满写），也属于保守预估，最多是不是真空/真满情况下告知空满，但是不会导致数据出错，属于保守的方法。
- FIFO 深度计算，计算主要考虑最坏的情况，例如 100wclk 里面进来 80 个这种，需要考虑 200 个时钟内连续进来 160 个的情况，然后计算深度也是根据“平均进来一个出去几个，然后乘以最大连续量即可得到 FIFO 深度”这样的方式考虑最大深度。

同步 FIFO 由于驱动时钟同步，因此可省去中间同步器以及格雷码编码机制，其作用原理和异步 FIFO 类似。

4、双端口 RAM 的实现/ 要求自己写双端口 RAM（H:\Desktop\Hardware_study\Async_Dual_port_ram）

FPGA 内部 RAM 资源分为 Block RAM 和 Distribute RAM，前者一般用于大量数据的缓存，后者多用于小部分数据缓存。就速度而言由于 Distribute RAM 利用 FPGA 内部 Slicem 的 LUT 存储资源，少量存储的话速度比较快，但是大量存储会对布线造成影响，难以保证时序。

FPGA 可以配置 Single-port-ram, Simple-port-ram, True-port-ram 等形式的 RAM，读写模式有 write-first, read-first, no-change 等模式，write-first 表明输出端口的数据和写入的数据相同，read-first 表明输出端口的数据为当前地址之前存储的数据，no-change 表明输出数据为写入之前一刻的数据不变；

Single-port-ram 同时只能进行读或写操作；True-port-ram 可同时进行读写操作，但是要避免地址冲突，不能两个端口同时对一个地址写入，Simple-port-ram 实际上是 True-port-ram 只开启了 A 端口的写入和 B 端口的输出。

5、两级触发器同步/ 应用场合/ MTBF 的计算以及 2-FF 的计算

两级触发器通常用于异步时钟域之间的单比特信号传输，通常是控制信号。其抑制亚稳态传播的原理并不是避免亚稳态的发生，也无法避免出错信号的继续传输，而是尽量减少亚稳态传播的概率。根据触发器 MTBF（平均故障时间间隔）计算，一级触发器的 MTBF = (e^(tmet/c1) / c2*f*a) 两级触发器 MTBF = (MTBF1)*(MTBF2)，相当于不稳定态在第一级触发器后被阻断了传播。如果条件更苛刻可考虑三级触发器。

6、握手机制/ 代码（H:\Desktop\Hardware_study\Async_Dual_port_ram）

握手机制通常用于数据传输速率要求不高但要求准确的场合，两边的握手信号都需要各自时钟域的同步器进行同步。当接收端经过同步电路接收到 req 信号后锁存总线数据，然后发出 ack 信号，ack 经过同步电路后到达发送端，发送端接受后撤销 req 信号，接收端也撤销掉 ack 信号，一次握手完成。

7、同步复位/ 异步复位/ 异步复位同步释放/ 三者比较/ 代码

同步复位指的是时钟有效沿来临的时候进行复位操作，同步复位使得电路为同步电路，能够利于仿真，缺点是要求复位信号要大于一个周期，否则无法保证成功复位，此外综合出来的电路实际上并没有利用器件的复位端，而是在输入端插入额外的逻辑电路，这样增加了额外的逻辑资源消耗。

异步复位是指的在任何时候只要复位信号有效即可复位，不占用额外的逻辑资源，充分利用器件的复位引脚，但是由于来临和结束时间未知，容易引起亚稳态现象。

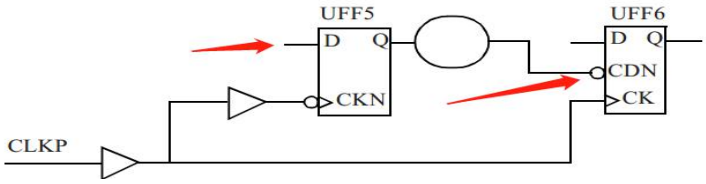
对于 FPGA 的 Flip-Flop，一般有同步复位，置位引脚 RST/S 和异步复位，置位引脚 CLR/PRE，对于同步复位，需要引入额外的 MUX 电路控制 RST/S，对资源有一定的占用。

比较理想的方式是异步复位同步释放，既充分利用了器件的复位引脚，又能够有效避免亚稳态的产生。

8、恢复时间/ 去除时间/ positive or negetive or global skew/ jitter/

Recovery time（恢复时间）是指的异步控制信号（例如复位信号）在被断言后到下一个时钟沿的最短时间

Recovery time is the minimum amount of time required between the release of an asynchronous signal from the active state to the next active clock edge. （类比建立时间）（Example: The time between the reset and clock transitions for a flip-flop. If the active edge occurs too soon after the release of the reset, the state of the flip-flop can be unknown.）



Removal time（去除时间）是指异步控制信号在时钟沿后需要保持稳定的时间。Removal time specifies the minimum amount of time between an active clock edge and the release of an asynchronous control signal.（类比保持时间）这个异步控制信号可以来自于“异步复位同步释放产生的复位信号，如下图”。

positive or negetive or global skew 分别表明相比 latch clock edge 的延迟为正/负/ 最大延迟减去最小延迟

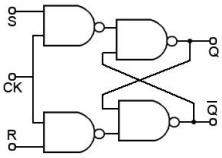
Clock Jitter 指的是实际周期和理想周期之间产生的偏差，jitter 通常由时钟发生器电路，噪声，电源变化引起。

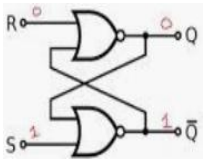
9、触发器组成/ 锁存器组成/ 区别

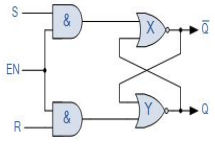
触发器通常包括 SR 触发器，JK 触发器，D 触发器，T 触发器；SR 锁存器由与非门或者或非门组成，SR 触发由时钟控制逻辑加上双与非门/或非门组成，还有门控 SR 触发器将时钟换为使能即可。四状态分别为保持/0/1/不允许。

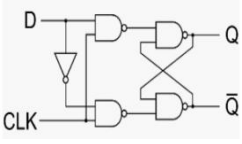
不允许态主要是全 1/全 0 同时翻转为 0/1 不能确定哪个逻辑元件最先变成 0/1，从而导致下一时刻不定态。

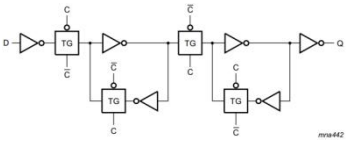
D 触发器由时钟边沿触发，锁存数据，D 触发器是 FPGA 中的主要元件，T 触发器作用是翻转信号。











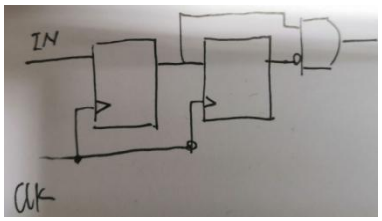
触发器对边沿敏感，锁存器对电平敏感。

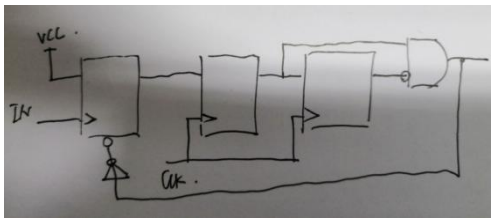
10、脉冲检测电路（非同步时钟如何处理，快到慢，慢到快）

对于慢时钟到快时钟的脉冲检测电路，由于脉冲长度一般大于接收端一个时钟周期，可以利用两级触发器结合一个与门和反相器电路实现单周期脉冲的传输，如图 1；

对于快时钟到慢时钟区域的检测电路，由于脉冲长度一般较小，可能导致慢时钟区域检测不到，因此可借助脉冲信号作为触发器的时钟信号，然后利用接收端信号重新复位该触发器信号即可，如图 2；

对于相同时钟的脉冲检测电路，可以利用两级触发器结合与门和非门电路，类似方案一





11、可综合和不可综合概念/ timescale 的理解

可综合是指的代码能够被 EDA 工具映射为具体的逻辑电路，能够在硬件端实现的语句；不可综合指的无法被映射为具体的电路在硬件端，通常用于仿真阶段。

timescale 是指的时间维度的常量，用于仿真阶段，由时间最小单元和时间精度两部分组成，例如 timescale 1ns/100ps，意思是时间最小延迟单元为 1ns，按照 100ps 的精度进行换算，#5.22 代表延迟 52*100ps，如果是 1ns/1ps 则代表延迟 5220ps，如果遇到一个模块包含其他模块的情况则精度按照最小的精度计算。精度越高会导致仿真的复杂度越大，仿真需求时间也就越久。

12、组合逻辑环路概念

组合逻辑需要避免生成环路，即组合逻辑输出端不经过任何时序逻辑就反馈到输入节点形成的环路，这样会产生振荡和毛刺等现象，而且会出现无法预知的结果。这样的组合逻辑环路的功能完全依赖于逻辑元件的延迟和布线延迟，具有很大的不确定性。

13、静态时序分析/ 动态时序分析/ 二者对比

静态时序分析(STA)通过遍历系统中所有路径来计算每条路径是否满足时序要求。无需外部信号激励的输入，只需要按照设计要求作出对应约束即可。静态时序分析不能分析电路逻辑功能是否满足需求，只能测试设计是否满足时序要求，系统能否在要求的时钟速率下正常运行。静态时序分析的速度较快。

动态时序分析是指的对系统生成测试向量并查看输出结果是否满足设计要求的方法。动态时序分析能够验证系统的逻辑功能，但难以通过输入向量测试到所有路径，而且随着输入信号数量增加，验证复杂度会越来越大。

14、建立时间/ 保持时间/ 数据到达时间/ 数据需求时间/ 建立时间裕量/ 保持时间裕量

建立时间(setup time)是指的时钟边沿到来之前数据保持稳定的最少时间。

保持时间(hold time)是指的时钟边沿到来后数据保持稳定的最少时间。

数据到达时间(data arrival time) = Tlatch + Tclka + Tcq + Tlogic

建立时间的数据需求时间(data required time/setup) = Tcapture + Tclkb - Tsu - Tuncertainty

保持时间的数据需求时间(data required time/hold) = Tlatch + Tclkb + Thd - Tuncertainty (Tuncertainty-还是+存疑)

建立时间裕量(setup slack) = (data required time - data arrival time) = T - Tsu - (Tcq + Tlogic) + Tskew

保持时间裕量(hold slack) = (data arrival time - data required time) = Tskew + Thd - (Tcq + Tlogic)

15、四类时序路径及起点终点/ 完整的时序路径包含三部分/ 建立时间检查，保持时间检查/ 恢复时间检查，去除时间检查/ 检查违例的解决方案

Start point - All input ports/pins or clock ports/pins of sequential cells are considered as start points.

End points - All output ports/pins or D pin of sequential cells are considered as end points.

A. 输入端口到第一级寄存器的数据输入 D 端之间路径

B. 最后一级寄存器输出 Q 端到输出端口之间路径

C. 系统内部寄存器的时钟端口 CLK 到下一级寄存器的数据输入端口 D 之间路径

D. 输入端口经过组合逻辑到输出端口之间路径

一个完整的时序路径包括源时钟路径（时钟端到当前寄存器的时钟输入端），数据路径（当前寄存器时钟输入端到下一级寄存器数据输入端），目的时钟路径（时钟端到下一级寄存器的时钟输入端）。

建立时间检查主要看数据路径的最大延迟路径，两级寄存器时钟 skew 最小的情况，满足建立时间需要有: T + Tskew - Tsetup > (Tcq + Tlogic)

保持时间路径主要看数据路径的最小延迟路径，两级寄存器时钟 skew 最大的情况，满足保持时间需要有: (Tcq + Tlogic) > Thd + Tskew

恢复时间检查和去除时间检查比较类似建立时间和保持时间检查，检查方式是一样的，主要看这个异步控制信号（通常是经过“例如异步复位同步释放生成的的复位信号”后传递给寄存器的复位信号）到达的时间是否和时钟边沿满足一定的时间关系，否则可能导致复位失败或者控制失败。

对于建立时间违例，可以考虑对较大的组合逻辑插入寄存器增加一级流水线，或者是考虑降低时钟频率，或者是考虑优化组合逻辑使之延迟降低；对于保持时间违例可以考虑插入 buffer 或者两级反相器对数据路径实现延迟。

16、时序约束/ 时钟约束/ 输入输出约束/ 时序例外约束/ 多周期路径约束(Multicycle paths)/ 伪路径约束(False paths)/ min-max 约束/ set_bus_skew

时序约束是指的对电路的时序提出要求，并在时序约束的基础上检验系统电路是否能够满足设计需求。

- 时钟约束是指的对系统中出现的时钟信号添加的限定条件，主要有以下类别：

1）Primary clock(通过引脚输入系统)　creat_clock -name clk_in -period 10 -waveform {0 5} [get_ports clk]

2）Virtual clock(并未连接到系统的任何端口，只是用来做输入信号的参考，而该信号按照这个虚拟时钟的速率传输，只是这个时钟也没有接到系统任何部位。可以理解为上游芯片的一个数据时钟，只是这个时钟没有接入系统)。tcl 语言描述为 create_clock -name virtual_clk -period 10

3）Generated clock(通过 PLL 或者内部逻辑，如计数分频生成的时钟) example1: create_generated_clock -name clk_div2 -source [get_ports clkin] -divide_by 2 [get_pins clk_div2]; example2: create_generated_clock -name clk_div_mul -source [get_pins mmcm0/clkin] -multiply_by 4 -divide_by 3 [get_pins mmcm0/clkout];

4）Clock groups(主要对异步时钟组之间约束，避免对异步时钟之间进行无效的时序分析，例如两个独立的时钟以及他们生成的时钟网络之间，或者是虽然来自同一个时钟源但是生成的时钟在 1000clk 内无法确定公共周期的时钟之间) set_clock_groups -name async_clk -asynchronous -group clk1 -group clk2; set_clock_groups -name clkunexclusive -group clk0 -group clk1;

5）Clock latency(由于 EDA 工具通常会自动计算内部网络的时钟延迟，因此通常只需要提供器件外部时钟走线延迟即可) set_clock_latency -source -early 0.2 [get_ports sysclk]; set_clock_latency -source -late 0.5 [get_ports sysclk]

6）Clock jitter(包含 input_jitter 和 system_jitter) set_input_jitter sysclk 0.3; set_system_jitter 0.11

7）Clock uncertainty(为 特 定 的 时 钟 或 者 时 钟 之 间 约 束) set_clock_uncertainty 0.1 [get_clocks clk]; set_clock_uncertainty 0.2 from [get_ports clk1] to [get_clocks clk2]

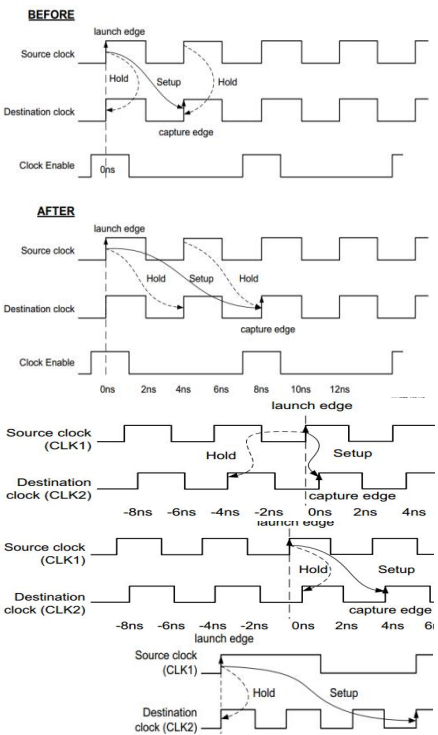
- 输入输出约束

1）set_input_delay(用于确定和系统某个时钟相关的输入信号的外部路径延迟，通常取决于外部器件输出的最后一级寄存器的 Tcq 和中间走线延迟。Consequently, the input delay value can be positive or negative, depending on the clock and data relative phase at the interface of the device，这个相关的时钟可以来自于系统的某个时钟或者虚拟时钟) example1: set_input_delay -clock sysclk [get_ports din]; example2: set_input_delay -clock sysclk -max 4 [get_ports din] example3: create_clock -name clk_virtual - period 10 set_input_delay -clock clk_virtual -min 2 [get_ports din2]

2）set_output_delay(用于确定和系统某个时钟相关的输出信号的外部路径延迟，通常取决于外部走线延迟和下游芯片第一级输入寄存器的建立时间和保持时间) example: set_output_delay -clock sysclk -min 2 [get_ports dout]

- 时序例外包含多周期路径约束，伪路径约束，最大最小延迟约束

1）Multicycle paths(多周期路径通常指的由于系统的特定设计，重新修改建立时间和保持时间检查方式，通常不像 EDA 工具默认的单周期的检查，多周期路径约束通常定义新的时序检查方式)



A) 相同时钟下由于使能信号等原因导致的多个周期读取。

对于相同时钟，无需确定-start -end，因为参考的时钟都是一样的，只需要重新确定建立时间检查和保持时间检查的关系。由于 setup relationship 同时影响 hold relationship，因此修改建立时间的同时也要修改保持时间关系。如图 set_multicycle_path N -setup -from [get_pins a] -to [get_pins b]; set_multicycle_path N-1 -hold -from [get_pins a] to [get_pins b] (其中 N 为建立时间检查的移动周期数)

B) 相同周期时钟但是有相移，因此修改建立时间检查关系，保持检查会随之移动。set_multicycle_path 2 -setup -from [get_clocks clk1] to [get_clocks clk2]。

如果是负相移（相移很小）的话，无需约束，情形相当于同时钟无相移。

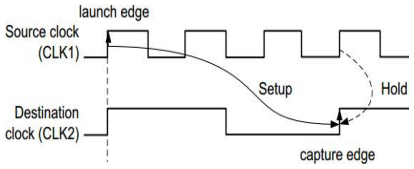
C) 慢时钟到快时钟。set_multicycle_path 3 -setup -from [get_clocks clk1] -to [get_clocks clk2]

Set_multicycle_path 2 -hold -end -from [get_clocks clk1] -to [get_clocks clk2]

D) 快时钟到慢时钟。set_multicycle_path 3 -setup -start -from [get_clocks clk1] -to [xxx] set_multicycle_paths 2 -hold -from[get_clocks clk1] to [get_clocks clk2];

Table 5-3: **To define a multicycle path with a Setup of N**

Scenario	Multicycle Constraints
Same clock domain or between synchronous clock domains with same period and no phase-shift	set_multicycle_path N -setup -from CLK1 -to CLK2 set_multicycle_path N-1 -hold -from CLK1 -to CLK2
Between SLOW-to FAST synchronous clock domains	set_multicycle_path N -setup -from CLK1 -to CLK2 set_multicycle_path N-1 -hold -end -from CLK1 -to CLK2
Between FAST-to SLOW synchronous clock domains	set_multicycle_path N -setup -start -from CLK1 -to CLK2 set_multicycle_path N-1 -hold -from CLK1 -to CLK2



- False paths(伪路径约束) 对于一些永远不可能起作用的路径或者无需进行时序分析的路径可以设置为 False path （例如异步时钟之间已经经过两级同步器同步则无需分析，例如上电瞬间内容确定则无需分析） example: set_false_path -from [get_clocks clk1] to [get_clocks clk2]
- Min-max(用于约束输入端口到输出端口的最大延迟以及异步信号之间的最大延迟(意思是即使两个时钟域以经通过同步器或者 FIFO 进行同步，依然可能需要一个最大路径延迟约束)) example1: set_max_delay 10 -from [get_ports din] -to [get_ports dout] example2: set_max_delay -from [get_pins a] -to [get_pins B]
- Set_bus_skew(用于对异步时钟域多位数据传输路径延迟进行约束，通常用于格雷码转换，多 bit 数据传输) 由于传输是多位的 bit，因此每个 bit 之间的路径差异需要被约束，通常是约束一个目的时钟周期。example: set_bus_skew -from [get_cells gray*] -to [get_cells gray_sync*] 2.5，其中*代表这个寄存器的每个 bit；同时为了保证源时钟和目的时钟之间数据正常传输，需要进一步设置 sex_max_delay 使得这个数据路径的延迟不能超过一定时间（通常是一个源时钟周期）set_max_delay -datapath_only -from [get_cells gray*] -to [get_cells gray_sync*] 5

17、竞争冒险/ 消除方法

竞争是指的由于逻辑和走线延迟差异导致不同信号到达输入端的时间不同，冒险是指的由于不同信号到达输入端时间差异导致输出信号存在短时不稳定的现象，也叫输出毛刺。

- 消除互补乘积项，例如(A~ + B)*(A + C)需要消除 A~A
- 增加冗余项，例如 AB+A~C，当 B=C=1 的时候是 A+A~，因此可以考虑增加 BC 项，这样就确保输出无毛刺。
- 输出端并联电容器，能够是的毛刺的上升下降沿变得缓慢，从而抑制毛刺被后级电路读取的概率。
- 将组合逻辑电路转变为时序逻辑，因为触发器对毛刺边沿不敏感，可以有效减少毛刺对后级电路影响。

18、状态机分类/ 状态转移图/ 一段两段三段式状态机区别特征（优缺点，本质特征，状态数差别，输出改变时间差别，状态机也分为同步状态机（有 CLK 控制输出）和异步状态机）/ 思考具体的状态机实例（饮料售卖）

Mealy 状态机：输出取决于当前状态和输入，输出可以在输入发生改变之后立刻响应，具有异步输出的特点，Mealy 由于结合了当前输入信息和状态信息，因此状态数量更少。

Moore 状态机：输出仅仅取决于当前状态，和 Mealy 相比 Moore 机首先根据输入信息更新状态，然后在下一个时钟根据当前状态决定输出，因此速度响应比 Mealy 慢一拍，但是具有可以同步输出的特点。

设计状态机首先考虑设计功能需求，然后根据运作流程列举可能的状态并根据转移条件列状态转移表转移图。状态的编码可以考虑采用格雷码（适用于连续跳变）或者是独热码，

一段式状态机将状态转移和输出集中在一个 always 块内书写，电路为时序逻辑不会产生毛刺，但是书写代码冗长且不易修改，可维护性差；

两段式状态机将状态转移和输出分放在两个 always 块中，前一个用时序逻辑进行状态调转，第二个用组合逻辑控制下一状态的计算和组合逻辑输出，容易产生毛刺。这种写法具有最优的面积和时序性能，但是由于是组合逻辑输出因此增加了到下一级寄存器的输出延迟。

三段式状态机分为状态调转模块，状态转移条件判断模块，输出逻辑模块。三段式状态机将组合和时序逻辑分开，易于维护和综合。

饮料机编写(自己编的题目)，可以首先考虑实现的功能，输入应该包括时钟信号，复位信号，输入的钱，输入的饮料选择，以及是否有当前饮料存货；输出应包括找零，饮料输出，无饮料提示等信息。中间状态应包括空闲状态，提示无饮料，输入钱多，输入钱正好，输入钱相等，找钱，出饮料，提示继续输入钱等状态，基于上述可能的状态进行状态转移图绘制，然后编写状态转移组合逻辑以及输出逻辑。“H:\Desktop\Hardware_study\Sail_machine”

19、系统级/ 算法级/ RTL 级/ 门级/ 开关级

Verilog HDL 语言自顶向下通常分为系统级，算法级，RTL 级（寄存器传输级），门级，开关级；

系统级描述语言提供的高级结构和所能实现的性能

算法级描述算法运行的模型，以上两种描述级别一般不涉及具体实现细节，不考虑是否能转化为硬件结构。

RTL 级用于描述数据如何在寄存器之间流动和传输

门级用于描述逻辑门之间的连接模型

开关级描述的是器件晶体管规模的具体连接和信号流动模型

前三个级别表述的是行为级，后两个级别分别代表逻辑级和电路级

20、阻塞赋值/ 非阻塞赋值

阻塞赋值是指的执行当前语句的时候阻塞其他语句的执行，因此阻塞赋值的执行具有一定的顺序性；非阻塞赋值是指的一次激活操作来临后会首先计算所有非阻塞语句表达式的右值，在激活操作结束后统一赋值给左边变量，执行无先后顺序，当前语句的赋值操作不会阻塞其他语句的赋值操作，因而叫做非阻塞赋值。在设计代码的时候通常对组合逻辑采用阻塞赋值，对时序逻辑采用非阻塞赋值，对阻塞赋值和非阻塞赋值分开在不同的 alway 块中实现。

21、function/ task/ repeat/ while/ for

function 用于执行一段的功能电路，由组合逻辑组成，至少一个输入，无输出。返回一个输出值（缺省值为 1bit 寄存器数据），输入输出的类型可以自己定义。function 可以驱动和使用全局变量，内部定义的为局部变量。函数

不能包含 always 块，内部可以调用其他函数但是不能调用 task。function 不能包括延时，因此执行时间为 0;

```
function [SIZE:0]GRAY_TO_BIN;   //定义函数 ID，后续调用函数也是通过 ID(variable1, variable2...)完成
    input[SIZE:0]GRAY;      //定义输入端口
    reg[SIZE:0]cnt;         //定义静态变量
        begin
        GRAY_TO_BIN[SIZE] = GRAY[SIZE];   //开始内部逻辑内容
        for (cnt = SIZE; cnt > 0; cnt = cnt - 1)
            GRAY_TO_BIN[cnt-1] = GRAY[cnt - 1] ^ GRAY_TO_BIN[cnt];
            end
endfunction   //结束定义
```

task 用于执行一段特定功能电路，用法和 function 类似，不过 task 可以有任意数量输入和输出接口甚至都没有都可以，task 只能用于 always 块内，不能用作 assign，task 可以调用别的 task 和 function 甚至是自己。Task 可以包含延时语句，包含延时语句后为不可综合。

```
task adder;     // 定义任务 ID
    input a,b,cin;
    output sum,cout;
    begin
      sum = a^b^cin;
      cout = (a&b) | (a & cin) | (b & cin);
    end
    endtask

task mul;   // 定义任务 ID
    mul_out = a * b;   //能够使用全局的变量
endtask

always@(*) begin     adder(a,b,cin,sum,cout);   mul();      end   //调用方式
```

repeat 和 while 和 for 用法类似，内部均为组合逻辑运算，用于 always 块内，循环次数需要在综合之前确定。由于循环语句实际上可以通过计数器实现，而且循环结构弊端是展开后资源浪费，因此不适合高速设计。

22、详细研究对于运算数为 X/Z 以及逻辑运算中有 X/Z 的情况下结果分别如何/ reg 和 wire 缺省值/ 关系符 ===以及 !==的具体用法/ 一元简约运算符用法/ 位运算数位不对齐的情况/ 算数运算操作数的位数对不齐的情况运算结果以及被赋值的数位不够的情况/ 运算数存在有符号数的情况

对于与运算，只要有 0 则为 0，无 0 有 X/Z 结果均为 X；对于或运算，只要有 1 则为 1，无 1 有 X/Z 均为 X；对于与非和或非门，结果类似与和或只是加了一级非；对于异或门同或门，只要有 X/Z 均为 X；对于非门，X 非为 X，Z 非为 Z；在进行基本算术运算（+-*/%）的时候，如果一个操作数为 X 则运算结果也为 X；对于关系运算符（除了===和!==）有一个为 X 则返回不定值 X；

reg 缺省在不同编译器下是不一样的，有的是全 1 有的是全 0，不过硬件上的缺省值都是 0。对于 reg 的初始化最好是通过 rst 实现，这样时序可控且复位后自动初始化；wire 是线网型，必须连接到寄存器端；

===和==!主要用在仿真语句中，用于比较含有 Z/X 的运算数，如果用在综合中会被替换为==和!=。

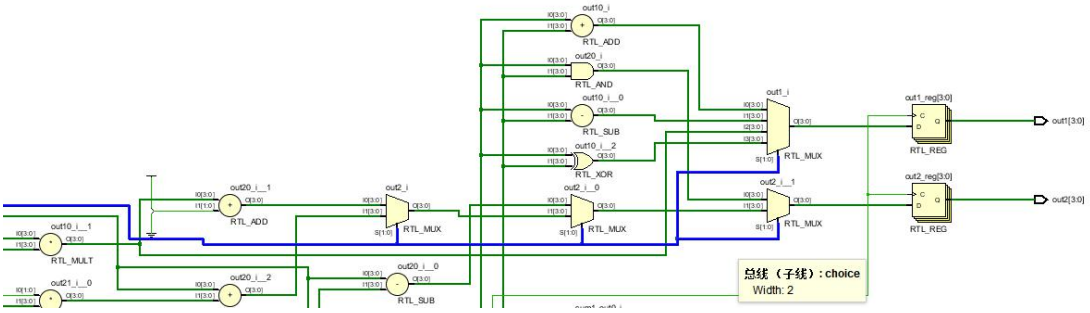
一元简约运算符和位运算符一样，只是用做单目运算符，运算过程为：首先将操作数最低位和次低位运算，然后将结果再和次次地位运算，以此类推直到最高位； example: assign k = ∑ ((k = sum[0]&sum[1]) & sum[2])。

对于位运算对不齐的情况，会对位数少的高位补零然后继续运算。

对于算数运算位数不齐的情况，如果是最大位数是左值，则对右端操作数均高位补零至和左值相同位数进行运算；如果是最大位数是右值，则右边操作数均高位补零运算，然后将运算结果的低位赋值给左值（实际上已经出错，因此务必保证左值的位数大于或者等于右值的运算结果的位数）。

运算数均为有符号数才按照有符号运算处理，否则均按照无符号运算处理。声明有符号数可以通过定义时加上 signed，例如 reg signed [1:0]a,b; 此时如果运算 a * b 则按照有符号乘法运算，否则基本算数运算均按照无符号运算。

23、Case /if 用法对比/ casez/ casex



case 语句的各级没有优先级分别，一般通过一个 MUX 控制每一级的数据输出（如上图），因此 case 语句速度较快，但是一般会占用较大面积；if 语句有先后优先级（如上图），一般是通过分散的 MUX 电路来控制信号向前流动，一般优先级高的 MUX 放在最后一级，优先级低的通过一级一级的 MUX 将信号往前传输，因此延迟较大，一般将延迟最大的关键信号路径放在第一级避免出现时序不满足的情况，此外由于 if 是一级一级向前传递信号，因此 if 级数越多延迟越大，不适合在高速场合使用过多的级数。

casez 语句用于分支表达式为含有 X 或者 Z 的情况，对于 casez 语句，对于某一位是 Z 的情况，这一位会被忽略，即认为 2'b0z 和 2'b01 等价；对于 casex 语句，会把 x/z 忽略，认为 2'b0x 2'b0z 2'b01 均等价；其余情况是必须完全一致才可以，caze 语句则认为必须完全一致则为 true，例如 2'b0x 和 2'b0x 才能等价。

24、Verilog 常见编码风格/ 串并转换/ 乒乓操作/ 流水线/ 面积速度互换/ 逻辑复用/ 流水线设计全加器，注意细节，注意位宽匹配/ 流水线乘法器（用逻辑门）/ 超前进位加法器

Verilog 代码编写重点兼顾面积和速度两方面，面积即资源消耗量，速度即系统运行的最高频率。根据实际需求合理权衡两者的关系。乒乓操作和流水线以及串并转换可以提高系统运行速度，但是增加了资源消耗；并串转换或者是逻辑复用降低了资源消耗，但是同时也降低了最高运行时钟。

流水线操作是通过将较长的组合逻辑拆分，中间插入寄存器缓存中间计算量，从而避免了较长组合逻辑带来的延迟，提高了系统运行时钟。流水线作业的时候无需等待后端操作全部完成，只需要按照一定的时序像流水一样输入数据，中间各个模块资源得到了充分利用而没有空余时间，提高了系统的工作效率。流水线操作实质上是一种面积换速度的做法。

流水线设计的时候需要注意适当缓存前级变量，例如设计流水线加法器的时候每一级计算都要将输入的数据通过寄存器缓存，防止后来的数据将之前的数据“冲掉”。

超前进位加法器是为了避免进位等待而设计的一种全加器。由于后一位的进位和运算都需要等待前一位的运算结果和进位结果，因此会造成较大延迟，超前进位加法器能够去除这一等待时间，提高运算效率。原理如下：

下面给出 4 比特超前进位链的推导。首先，对于 1 比特全加器，

$$S_i = x_i \oplus y_i \oplus C_i \tag{11-3}$$

$$C_i = x_i y_i + (x_i + y_i) C_{i-1} \tag{11-4}$$

从中可以看出，如果两个输入 x、y 都为 1，则进位输出肯定为 1；如果 x、y 有一个为 1，则进位输入等于下一级的进位输入。定义 P=x+y，G= xy，则 4 比特超前进位链的逻辑如下所列：

$$C_0 = C_{in}$$

$$C_1 = C_0 + P_0 G_0 = C_0 + P_0 G_{in}$$

$$C_2 = C_1 + P_1 G_1 = C_1 + P_1(G_0 + P_0 G_{in}) = G_1 + P_1 G_0 + P_1 P_0 G_{in}$$

$$C_3 = C_2 + P_2 G_2 = C_2 + P_2(G_1 + P_1 G_1) = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 G_{in}$$

$$C_4 = C_3 + P_3 G_3 = C_3 + P_3(G_2 + P_2 G_2) = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 G_{in}$$

$$C_{out} = C_4 + P_4 G_4$$

可以看出，各个进位彼此独立产生，将进位级联传播的串行特性取消掉，因此通过超前进位链可以减少进位所产生的延迟。

25、奇数分频/ 偶数分频/ 小数分频/ N+0.5 分频/ PLL 原理

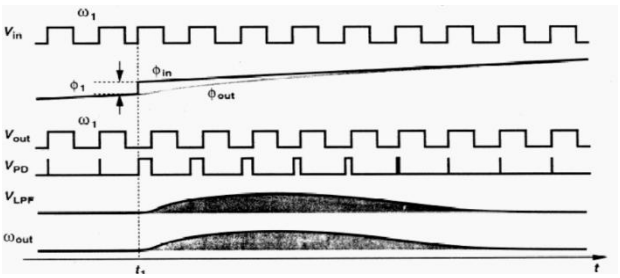
奇数分频可以通过分别利用上升沿和下降沿得到两个分频时钟，将这两个时钟进行相与或相或运算即可令半个周期多出半个原时钟，从而实现技术分频的目标。对于 N 分频可以利用上升沿和下降沿分别产生(N-1)/2N 占空比周期为 N 的时钟然后相或；或者是利用上升沿和下降沿分别产生(N+1)/2N 占空比周期为 N 的时钟然后相与；

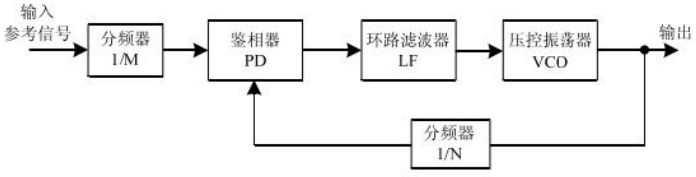
偶数分频就是常规的计数器分频即可，利用计数器计数 N/2 - 1 翻转即可。

小数分频可以通过模 X 计数器得到，但是这样得到的分频通常周期是不稳定的，也就是占空比不定。例如需要 8.9 分频，可以设置一个 16bit 计数器，每次加(2^16/8.9)，然后以最高位为时钟输出。但是这样得到的时钟质量不高，不建议用于高速场合，高速的非整数比例时钟建议用 PLL 实现。

N+0.5 分频可以通过两个计数器分别在上升沿和下降沿计数实现；通过判断上升沿计数=下降沿计数=0.5N，进行翻转，然后继续判断上升沿计数=N,下降沿计数=N+1，再次翻转同时清零，得到对应的时钟。

PLL 由鉴相器(PD)，环路滤波器(LPF)，压控振荡器(VCO)以及分频器组成。其中分频器主要用于倍频操作。如果分频器 M 用于参考时钟输入端，分频器 N 用于 VCO 输出反馈端，则最终锁定的频率为 F*(N/M)；鉴相器用于比较输入的时钟波形和参考时钟波形的差异，内部通常由一个逻辑运算，例如异或运算（多用于方波信号的锁相电路），或者是模拟乘法器运算（(acos(wt+u1)*bcos(wt+us) = ab/2 *(cos(2wt+u1+u2) + cos(u1 - u2))，利用低通滤波去除二倍频率项，只留下 ab/2 * cos(u1-u2)，多用于输入端均为正弦信号的锁相场合）反应相位差，会将二者差异反馈为电压信号输出，下一级通常是环路滤波器，作用是对输出的去除高频噪声分量，加到 VCO 处，从而控制输出的振荡频率。VCO 同时具有积分作用，能够对相位差进行积分。当频率被“锁定”后鉴相器输出的差异在很小范围内波动，VCO 的控制电压相应保持不变，从而达到稳定频率的目的。如果有外界干扰导致频率失锁，后续会继续重复上述操作，使得输出频率稳定。





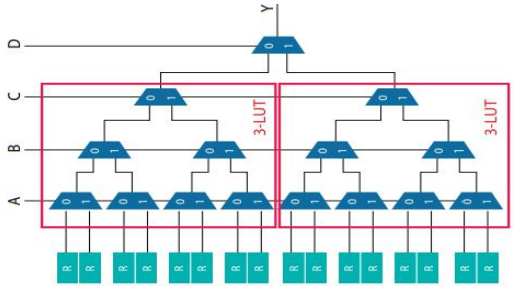
26、Verilog 仿真常用关键词和用法/ 常用原语

27、FPGA 内部资源以及结构/ FPGA 烧写模式/ FPGA 和 CPLD 比较

以 XILINX 7 SERIES FPGA 为例，FPGA 内部资源主要分为可编程逻辑块(CLB)，块 RAM(BRAM)，集成 DSP，集成高速收发器，可编程 IO，集成 ADC 模块和时钟管理单元(MMCM/PLL)组成。

CLB(Configurable logic block)内部集成有两个 SLICE，或者是两个 SLICEL(LUT 不可配置为 Distribute RAM/shift register)，或者是一个 SLICEL 和一个 SLICEM(LUT 可配置为 Distribute RAM)；CLB 和外部的 Switch matrix 连接，用于信号的外部输入输出；每个 SLICEL/M 内部有 4 个 6-input LUT(SLICEM 类型的除了 A1-A6 地址输入还有读写使能输入和时钟输入)和 8 个 Flip-flop(触发器，具有使能/时钟/同步置位复位/异步清除置位/数据输入 D/输出 Q 等接口)，一个进位链(carry logic)用于加法运算或者乘法运算的进位逻辑控制，两个 FMUX7 和一个 FMUX8 可用于将 LUT 配置为 4:1/8:1/16:1 的 MUX；每个 SLICEM 可将 4 个 LUT 配置为 256bit(4*2^6)的 Distribute RAM，或者是 128bit 的 Shift register；同一个 CLB 的两个 SLICEL/M 之间无直接连接，会有外部的开关矩阵分别与其连接，每个 SLICEL/M 会通过进位链同上方和下方对应同一列的 SLICEL/M 连接。

FPGA 内部的 LUT 实际上是由 SRAM+MUX 组成，例如一个 4-input LUT，实际上是由 16 个 SRAM 存储 bit 加上(8+4+2+1)=15 个二输入 MUX 组成。对于 SLICEM 的 LUT，其内部 SRAM 还可以作为 distribute RAM 使用。



FPGA 烧写模式有 7 种，通过配置 BANK 的 M0:M2 决定烧写模式的选择。JTAG/Master BPI/Master SPI/Master Serial/Slave Serial/Master SelectMAP/Slave SelectMAP/；对于多个 FPGA 可采用菊花链方式下载程序。

FPGA 和 CPLD 相比

1、前者是基于 SRAM 的结构，掉电之后内部结构清除，上电的时候通过外部 FLASH 读取配置信息，CPLD 基于 EEPROM，掉电后内部结构不清除，无需上电重新配置信息，安全性更好；理论上 FPGA 可以烧写无数次，但是 CPLD 烧写次数有限。

2、前者内部包含大量 CLB，能够实现复杂的逻辑功能，是一种适合时序电路设计的结构，后者内部组合逻辑较多，更适合于组合逻辑设计；前者由于规模庞大，时序分析精确度不高；后者内部采用固定长度金属线连接各类功能模块，具有时间可预测性。

3、价格上前者更贵，能够实现更复杂的功能，后者多用于重复性要求不高的场合，功能实现相对简单。

28、提高系统工作频率的方法/

乒乓操作/流水线操作/逻辑复制/串并转换 这些基本的方法属于面积换速度提高工作频率的方法；此外可以通过优化组合逻辑结构，通过时序分析找到延时最大的路径并进行拆分，例如插入寄存器做流水线处理，从而提高系

统工作频率。此外由于计数器是通过加法进位链实现，因此在状态机中避免使用过大的计数器，可以减小延迟。

29、验证的概念/ 形式验证/ 常用验证方法

30、CMOS 功耗类别/ 数字芯片设计低功耗方案

CMOS(Complementary Metal-Oxide-Semiconductor 互补金属氧化物半导体)，由于其只在状态切换的时候消耗电流（指的较大的电流，这里不讲静态电流），因此具有优良的低功耗特性。

静态功耗：CMOS 在逻辑稳定的时候都有一个 MOS 处于截止状态，因此 VCC 和 GND 之间通路应该是“断开的”，理论上没有电流通过，但是由于处于截止状态的 MOS 的模型实际上是一个反偏的二极管，因此存在反偏 PN 结漏电流，此外还包括 MOS 管截止时候存在的漏源极之间的亚阈值电流，这两部分电流指的就是静态功耗。也即逻辑不发生反转保持稳定的时候依然存在的电流功耗。

动态功耗分为两部分，一部分是逻辑状态切换的时候，由于 PMOS 和 NMOS 短时导通导致的 VCC 和 GND 之间短时短路（短路功耗），另一部分是状态切换的时候，NMOS 导通 PMSO 闭合对负载电容充电，PMOS 导通 NMOS 断开对负载电容放电导致的动态电流（开关功耗，这部分是主要部分）。

主要的低功耗设计措施如下：（P(switch) = C*(VCC^2)*f）这部分是所有功耗中占比最大的）

1）Clock gating，由于时钟翻转是开关功耗中的重要组成部分，因此对不经常使用的电路部分可以通过逻辑门控关闭其时钟输入信号，避免无效的能耗浪费。需要注意的是，门控时钟可能影响时钟信号的质量。

2）Power gating，对于不经常使用的模块对其关闭，可以包括外部电源关闭和内部关闭。Power gating 由于每次需要安全的启动和关闭（就像电脑开关机要先存储数据一样）因此会影响运行速度和响应速度。

3）对于减小动态功耗中的亚阈值电流可以通过提高阈值电压（采用 HVT 或者 RVT）实现，但是这个方法会导致电路速度降低。

4）可以采用多路电源方案，通过降低电源电压能够有效减小功耗。对于速度较高的场合可以采用较高电源电压，对于速度要求不高的场合采用低电源电压。

5）降低系统运行频率也可减小功耗。

31、SPI/ I2C/ RS232/ RS485/ UART 常见协议以及电平标准/ 波特率/ 比特率

SPI 有四种工作模式，分别由空闲时刻时钟电平 0/1 和上升沿还是下降沿读取数据决定；CPOL=0 表示空闲时钟极性为 0，上升沿为前沿，下降沿为后沿，CPOL=1 表示空闲时时钟为 1，下降沿为前沿，上升沿为后沿；CPHA=0 表示 out 端数据在前一个时钟的后沿改变，而 in 端在当前时钟的前沿读取数据，该数据保持稳定到当前时钟的后沿；CPHA=1 表示 out 端数据在当前时钟的前沿改变，而 in 端在当前时钟的后沿读取数据。SPI 可以工作在 MHZ 级别，属于全双工通信。

I2C 属于半双工通信，两线制，工作频率在 KHZ-MHZ 级别。工作的时候 SCL 空闲时为高电平，当 SDA 拉低表明传输开始，数据只能在 SCL 的低电平期间改变，在 SCL 的高电平期间数据被读取。一次数据传输完毕后发送端释放 SDA，接收端接管 SDA 线，发出一个 ACK（低信号）表明收到数据。整个传输过程截止需要在 SCL 拉高后 SDA 也从

低拉高，表明传输过程结束。

UART 为一种通信协议，支持全双工/半双工/单工。空闲时为高电平，开始传输数据的时候先把电平拉低，生成一个开始位 0，然后根据一定的波特率发送若干 bit 数据，结束后发送一个奇偶校验 bit（可选择是否有无）和一个或者多个停止 bit（高电平）。同步操作也是根据起始位做接收端对其，只要能在一个发送周期内数据对齐即可保证整个传输过程数据对齐。

RS232 的电平逻辑为负逻辑，(-3)-(-15)为正，(+3)-(+15)为负电平，其他未定义。RS232 内部传输协议通常为 UART（严格来说不一样),RS232 和 485 一样都是一种电平协议，不仅可以用来传输 UART，当然主要传输的是 UART。RS232 仅支持点对点。传输距离一般为几米到几十米，特殊电缆可以达到百米级别。

RS485 为差分信号，支持一对多，差分电压最小 200mv 即可被识别为正常逻辑，可接受电压范围是-5V 到+12V。RS485 仅仅是一种接口协议，但是并没有规定其中的通讯协议。传输距离可以达到 KM 级别。

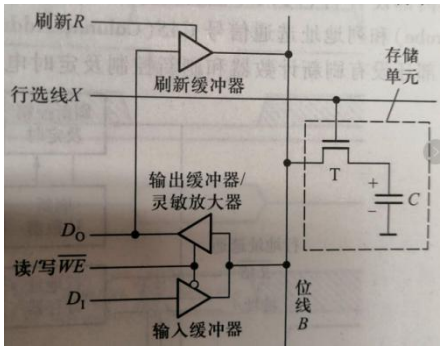
波特率计算：例如一帧数据包括 1bit 起始位，8bit 数据位，1bit 停止位，每秒发送 240 帧数据，则波特率为 240buad，比特率为 240*10=2400bps。

32、DDR/ SDRAM/ SRAM/ DRAM/ ROM 原理速度功耗价格容量等等比较

首先比较 DRAM 和 SRAM，前者为动态随机存储器，内部存储原理是基于电容存储电荷来存储 bit 信息，结构简单价格便宜。由于电容存在漏电流因此需要定期刷新操作保持数据。功耗比 SRAM 要高，有同步 DRAM 和异步 DRAM 之分，取决于是否有统一的时钟参考；SRAM 内部有多个 transistors 组合成锁存器进行数据存储，结构较为复杂，无需定期刷新即可存储数据，速度比 DRAM 快，适合于充当高速 Catch，价格较贵，容量小于 DRAM。

对于 SRAM，首先通过行选择线选择一行中的单元，然后通过列地址选择线开启列数据通道。对于读操作，IO 直接读取锁存器中数据，对于写操作，IO 输出电路驱动能力较大，因此会将原有的锁存器中数据覆盖。

对于 DRAM，存取实际上是通过电容实现。由于电容存在漏电流，因此需要增加刷新电路，如图。



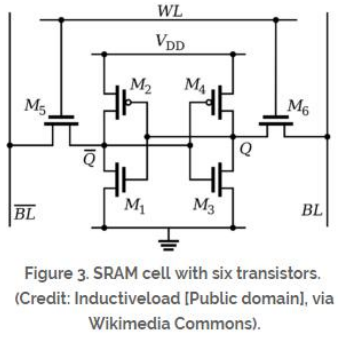
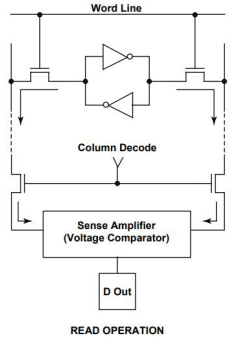
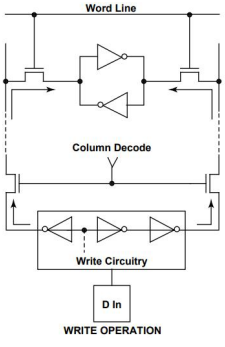
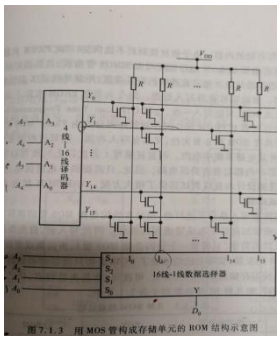


Figure 3. SRAM cell with six transistors. (Credit: Inductiveload [Public domain], via Wikimedia Commons).







对于 ROM，首先经过地址译码器选中某一行，该行所有 MOS 导通，然后选中某一列，交叉点如果有 MOS 则数据为 0，否则数据为 1，原理如图。

DDR SDRAM(Double data rate synchronous dynamic random access memory)全称双倍数据速率同步动态随机存储器。相比 SDRAM，在同样的时钟速率下由于采取上升沿下降沿读写因此数据带宽提高一倍。DDR3 有地址线/控制线/数据线，其中地址线行列复用（A10 自刷新控制）。DDR3 上电后首先复位，然后初始化配置相关信息，例如选择突发模式，终端电阻选择等，然后进行终端电阻校准（ZQ），终端电阻集成在内部用于抑制信号反射。之后进行 writing level，用于校准 DQS 和 CK 的相位信息；之后就开始进入工作模式：

RAS(行地址选通脉冲)，CAS(列地址选通脉冲)

TRCD：行激活到读写指令之间间隔延迟（发出 BANK 地址和行地址认为是行激活命令）

TCL：读写指令发出到数据输出之间的延迟（包含 CL(CAS latency)和 AL，AL 是信号放大时间）

TRP：pre-charge 预充电指令时间（开启新的一行 ROW 的时候需要预充电操作）

33、数字滤波器/ 模拟滤波器/ 传递函数/ 如何画数字滤波器框图

34、二进制数运算/ 定点数浮点数/ 源码反码补码/ 相关数据的运算

正数的原码，反码，补码相同，负数的补码等于除了最高位符号位不变，其余位取反加一。做有符号数减法运算可以先对被减数求补码然后做加法运算即可。注意由于符号位不能表示数值，因此要防止溢出。

浮点数：分为符号位，指数位，尾数位。注意尾数位均省略了最高位的 1，因为“归一化”后均为 1.xxxx 所以直接省略。对于 32bit 单精度浮点数，符号位 1+指数位 8+尾数位 23(隐藏了最高位 1)。当指数位位 01111111(127)表明后面乘数为 2^0=1，此时如果符号位是 1，尾数是 1000000_00000000_00000000，则该数是+1.5((1+1*0.5)*2^0)=1.5。可表示范围是-2^127 到 2^128(当尾数全为 1 的时候实际上 1.11111....约等于 2，因此正数最大范围要乘 2 因此是 2^128，同理，负数最小范围被乘以 1 因此是 2^127)。对于可表示精度，由尾数位数决定，因此有效位数是 2^23=8388608(最高位 1 对精度无影响因此不考虑，尾数位按照 23 计算)一共七位有效数字。双精度浮点数的尾数为 52+1bit，指数位数是 11，符号位 1bit，道理同上。双精度浮点数精度为 2^52=4.5*10^15，因此有效数字位数约为 15-16 位。

定点数就是小数点位置保持不变的数据，表示范围有限。

35、逻辑代数表达式/ 化简公式/ 最大项最小项/ 卡诺图化简/ 真值表/ 状态转移图

逻辑代数表达式可以通过真值表写出，将真值表中输出项为 1 的输入项相加（输入为 0 取非，输入为 1 取自身）。花间逻辑表达式可以通过摩根定律，卡诺图进行化简。最小项最大项分别表示输入信号的不同组合的乘积与和（具体见课本）。最小项和最大项为取非关系。

36、CMOS 与非门/ 或非门/ 反相器（优良特性）/ 三态门/ 线与逻辑/ 传输门/ MOS 工作原理

PN 结：P 硅中掺杂带正电的离子，N 中掺杂带负电的离子。PN 结背靠背会使得 N 中负电荷（多子）漂移到 P 中，而 P 中正电荷（多子）会“漂移到”N 中，形成 N 到 P 的内建电场。这个内建电场叫做空间电荷区，阻止了多子的进一步扩散。此时如果外加电场（P 接+N 接-）削弱这个内建电场，多子就会进一步扩散形成电流，叫做正偏；如果外加电场（P 接-N 接+）增强了这个内建电场，多子漂移会进一步被削弱，但是这有利于少子的漂移，从而形成了反向电流，但是由于少子有限，因此这个电流非常小。

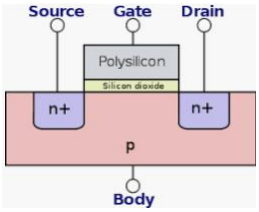
CMOS 门电路出现或运算则需要并联 MOS 管，出现与运算则需要串联 MOS 管。

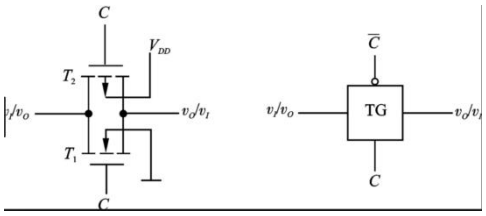
CMOS 反相器由一个 PMOS 和 NMOS 串接组成，只有在逻辑切换的时候才会导通，因此消耗电流很小。

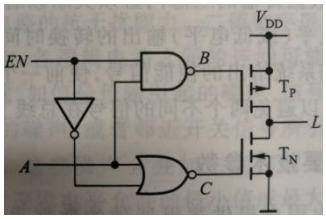
三态门在使能为 0 的时候输出高阻，使能为 1 的时候正常输出。

线与逻辑是利用 OC/OD 门，将漏极输出/集电极输出直接连接，只要有一路为 0 则输出为 0，全为 1 才输出 1。

传输门是两个并联的 PMOS 和 NMOS 组成，两个栅极分别接相反信号，PMOS 的衬底为 N 硅，接最高电位，反偏防止正向电流，NMOS 衬底为 P 硅，接最低电位，反偏。当栅极信号为 0（设 NMOS 栅极接信号本身,PMOS 接反相信号），两个管子都截止，电路断开；当栅极信号为 1，电路导通。







对于 MOS 管，以 N 沟道增强型 MOS(NMOS)为例，栅极不加正电压的时候 P 衬底中电子为少子，两个 N 硅之间无导通通路。当栅极加正电压，栅极和 P 衬底之间形成电场吸引少子电子到两个 N 硅之间，随着电压增大，电子集聚较多，N 硅之间形成 N 型薄层（反型层），电路导通，由于存在压降，当电流增大，右边 N 硅附近电势较高，削弱了从栅极指向 P 衬底的电场，因此电子减少，继续增大电流则形成“夹断层”，此后电流将不再增加，压降主要集中在右边 N 硅附近。

37、施密特触发器

38、AD/DA 基本原理和结构

39、Serdes 接口知识/ 预加重/ 去加重/ 均衡/ 常用编码方式/ 时钟数据恢复(CDR)原理，参考时钟作用/