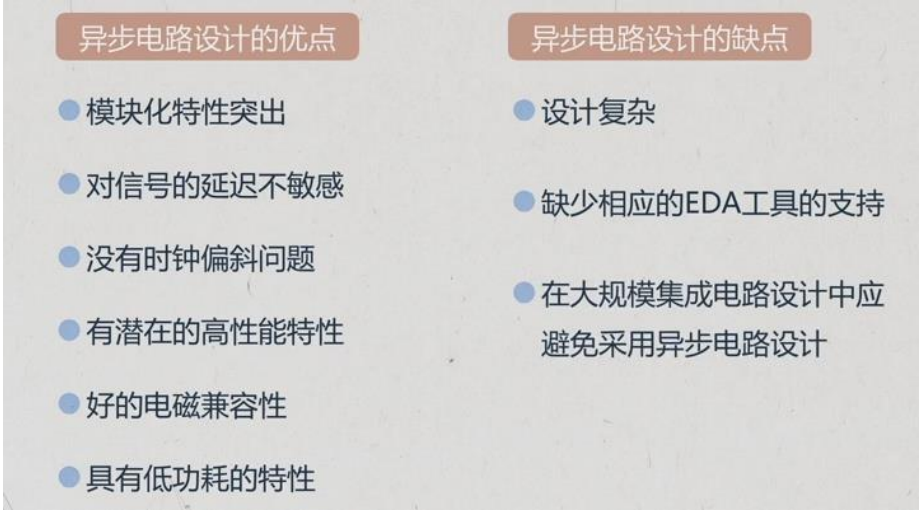
一、异步电路设计的优缺点



二、同步电路设计的优缺点

1.优点：

（1）在同步设计中，EDA 工具可以保证电路系统的时序收敛，有效避免了电路设计中竞争冒险现象。

（2）由于触发器只有在时钟边缘才改变取值，很大限度地减少了整个电路受毛刺和噪声影响的可能。

2.缺点：

（1）触发器距离时钟源点的距离不同，会产生时钟偏斜（clock Skew）、时钟抖动（clock jitter）。

（2）时钟树综合，需要加入大量的延迟单元，使得电路的面积和功耗大大增加。

三、亚稳态

1.定义：亚稳态是指触发器无法在某个规定时间段内达到一个可确认的状态。当一个触发器进入亚稳态时，无法预测该单元的输出电平，也无法预测何时输出才能稳定在某个正确的电平上。

2、单 bit 信号亚稳态的应对措施：

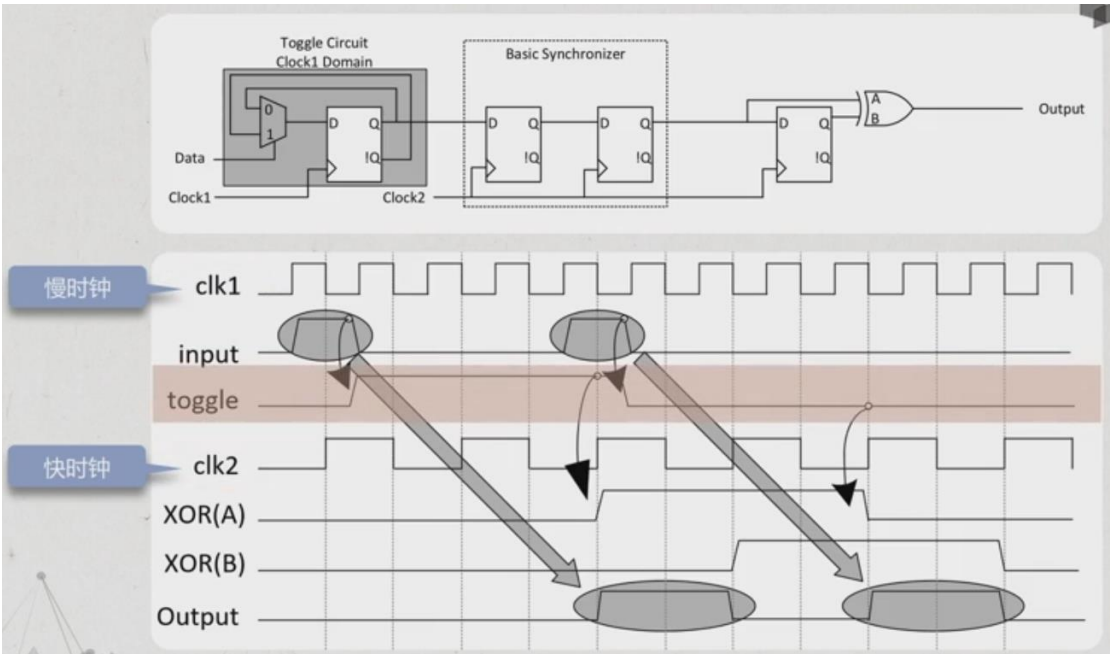
（1）亚稳态不能从根本上消除，但可以通过采取一定的措施使其对电路造成的影响降低；

（3）慢到快：双锁存器电平同步器（多级寄存器）：一个信号在过渡到另一个时钟域时，如果仅仅用一个触发器将其锁存，那么时钟采样的结果将可能是亚稳态，这时用双寄存器锁存数据，将会降低亚稳态出现的概率。优点：结构简单、易于实现；缺点：增加了触发器延迟、当快时钟域转到慢时钟域时，容易造成慢时钟采样丢失。所以该方法常用于慢时钟域转到快时钟域。

（4）快到慢：采用同步器：

①边沿检测同步器检测一个单 bit 信号的边沿。适用于慢时钟采样快时钟：输入数据的宽度必须比一个接受时钟周期加上一个同步触发器的 hold 时间要长，最安全的就是两个同步周期宽度。

②脉冲检测同步器：从某个时钟域去除一个单时钟宽度脉冲，然后再新的时钟域中建立另一个单时钟宽度的脉冲。适用于快时钟采样慢时钟：



三种同步器比较

Type	应用	输入	输出	限制
电平检测	同步电平信号，时钟域任何时钟域的传输	电平	电平	输入信号必须保持两个接受时钟周期宽度，每一次同步之后，输入信号必须恢复到无效状态
边沿检测	检测输入信号的上升沿和下降沿，适用于低频时钟域向高频时钟域传输	电平或脉冲	脉冲	输入信号必须保持两个接受时钟周期宽度
脉冲检测	同步单周期脉冲信号，适用于高频时钟域向低频时钟域传输	脉冲	脉冲	输入的脉冲时间的距离必须保持两个接收时钟周期以上

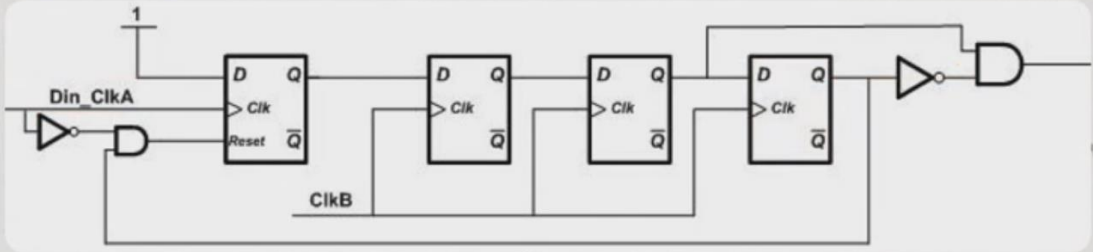
PS:（1）信号从快时钟域到慢时钟域过渡时，慢时钟将可能无法对变化太快的信号实现正确采样；

（2）同步器对两个时钟之间的关系要求很严格，而“结绳法”适合任何时钟域的过渡。

结绳法：将快时钟信号的脉冲周期延长，等到慢时钟同步采样后再“结绳”，还原为原来的脉冲周期宽度。如下图。

结绳法优缺点：结绳法可以解决快时钟域向慢时钟域过渡的问题，且其适用的范围很广，但结绳法的实现较为复杂，且效率不高，在对设计性能要求较高的场合应该慎用。

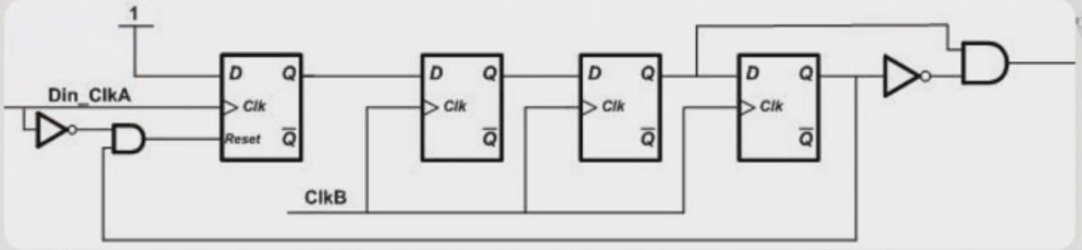
- 利用数据的边沿作时钟（图中上升沿）。（可以将脉冲无限延长，直到可以采集到数据，然后复位，要考虑产生数据的频率）。



说明：

- 这种结绳法的原理是，数据作为Din_clkA，即当数据有上升沿(0->1)时，寄存器1的输出将会稳定在高电平，此时等待ClkB采样；当ClkB完成采样后，寄存器4会输出高电平，若此时Din_clkA为低电平，那么即可完成复位，开始下一次采样等待。

- 这里需要注意的是当数据来临(即上升沿)时，ClkB域需要等待3个ClkB才会在寄存器4输出并完成输入端的复位，所以Din_ClkA如果变化较快，即持续时间短于3个ClkB，也就是Din_ClkA频率大于ClkB的1/3，那么这时Din_ClkA的变化将无法被采样到，因为ClkB域需要3个ClkB才能完成采样，并且此时Din_ClkA必须是低电平才能复位，采用异步复位。



在慢时钟采样快时钟的时候，结绳法适合采样数据较少即脉冲间隔较大的控制信号。即脉冲间隔Ta>3Tb；即等待3个clkB时钟后，完成复位，才允许下一个输入脉冲。

四、跨时钟域的时候，“快时钟到慢时钟”与“慢时钟到快时钟”之间的区别

（1）慢到块：只需要考虑亚稳态的问题。

（2）快到慢：除了亚稳态，还需要考虑慢时钟的采样速率问题，因为采样的条件需要满足奈奎斯特采样定律，即采样频率低于信号最高频率的 2 倍时，是无法完整采样的。

五、多比特指示信号跨时钟域的传输

（1）多个控制信号跨时钟域仅仅通过简单额同步器同步可能不安全，因为信号可能会有 skew，导致多信号可能不会再同一时刻起作用。解决方法：让多个信号合成一个信号，同步到另一个时钟域中；如果遇到不能合并的情况，需要加入采样控制信号，确保另一个时钟域能够正确采样。

（2）使用格雷码传输也能很好地降低亚稳态出现的概率

二进制数据转换为格雷码：从最右边第一位开始，依次将每一位与左邻一位进行异或，

作为对应格雷码该位的值，最左边的一位不变。

格雷码转二进制码：从左边第二位起，将每位与左边一位解码后的值异或，作为该位解码后的值，最左边一位不变。

六、多比特数据流跨时钟域的传输。

（1）数据流与指示信号的区别：数据流大多具有连续性（背靠背传输），数据流要求信号具有较快的传输速度。

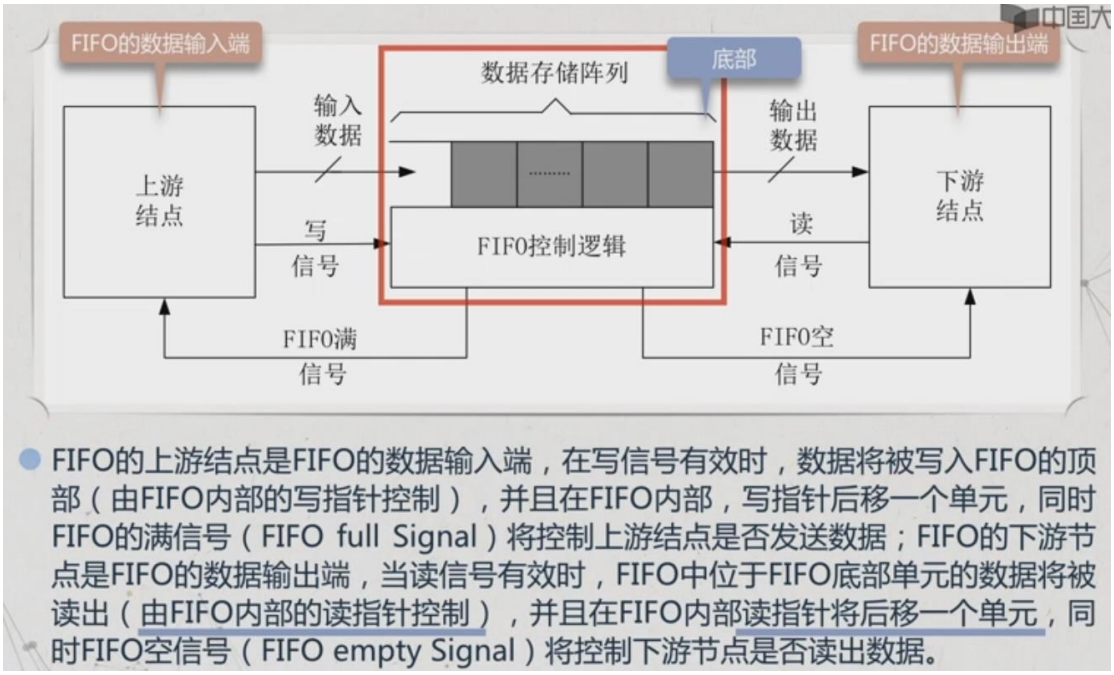
（2）数据流的跨时钟域传输：通常采用 SRAM 和异步 FIFO 进行传输。

七、FIFO

1．定义：FIFO（first in first out）是一种先入先出的存储结构，与普通存储器（RAM）的区别在于它没有外部读写地址线。使用非常简单

2、使用方式：顺序写入数据，顺序读出数据，数据地址由内部读写指针自动加 1 完成，不能像 RAM 那样可以有地址线决定读取或写入某个指定的地址。

3、异步 FIFO：在 IC 设计的过程中，模块与模块之间的多时钟情况已经不可避免，数据在不同时钟域之间的传输很容易引起亚稳态，此时需要使用异步 FIFO 来跨时钟域。



4、用途

（1）异步 FIFO 的读写可以采用相互异步的不同时钟（即跨时钟域）

（2）对于不同宽度的数据接口也可以使用 FIFO。例如单片机的 8 位数据输出，DSP 是 16 位数据输入，在单片机和 DSP 连接时，可以用 FIFO 来进行数据匹配。

5、FIFO 的参数

（1）宽度：FIFO 一次读写的数据位；

（2）深度：FIFO 可以存储多少个 N 位的数据，深度是一个重要的参数，太深会导致资源浪费，太浅会导致数据量不够深；

（3）满标志：FIFO 已满或者将要满时由 FIFO 的状态电路送出的一个信号，来阻止 FIFO 继续执行写操作，防止数据溢出（overflow）。

（4）空标志：FIFO 已空或将要空时由 FIFO 的状态电路送出的一个信号，来阻止 FIFO 的读操作继续从 FIFO 中读出数据而造成无效数据的读出（underflow）。

（5）读时钟：读操作所遵循的时钟，在每个时钟沿来临时读数据。

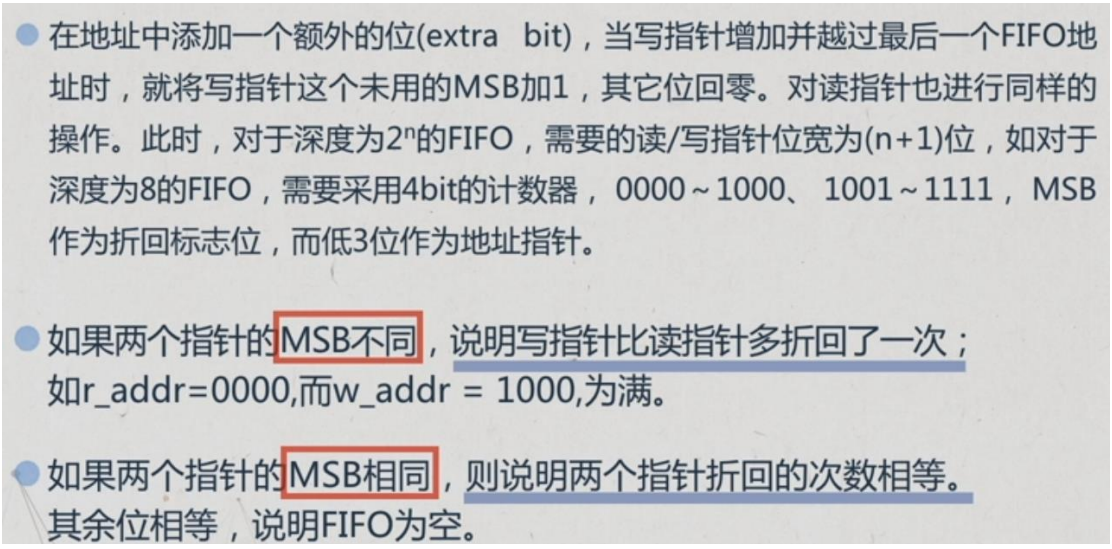
（6）写时钟：写操作所遵循的时钟，在每个时钟沿来临时写数据。

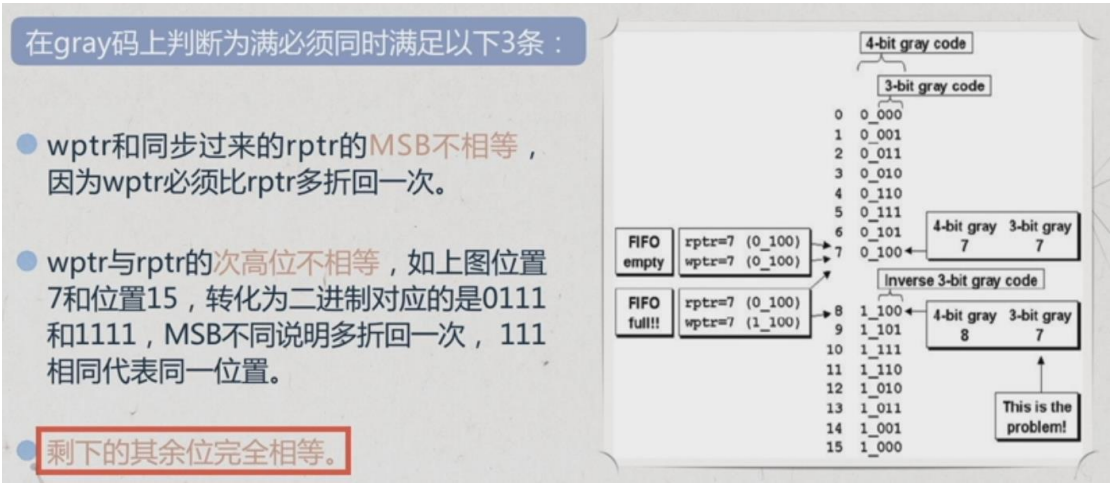
6、FIFO 的设计

（1）空满标志是任何 FIFO 设计的关键，遵循的原则是“写满而不溢出，能读空而不多读”

（2）空满状态的判断：读写指针相等

（3）区分空状态还是满状态：





（4）FIFO 深度的设计：

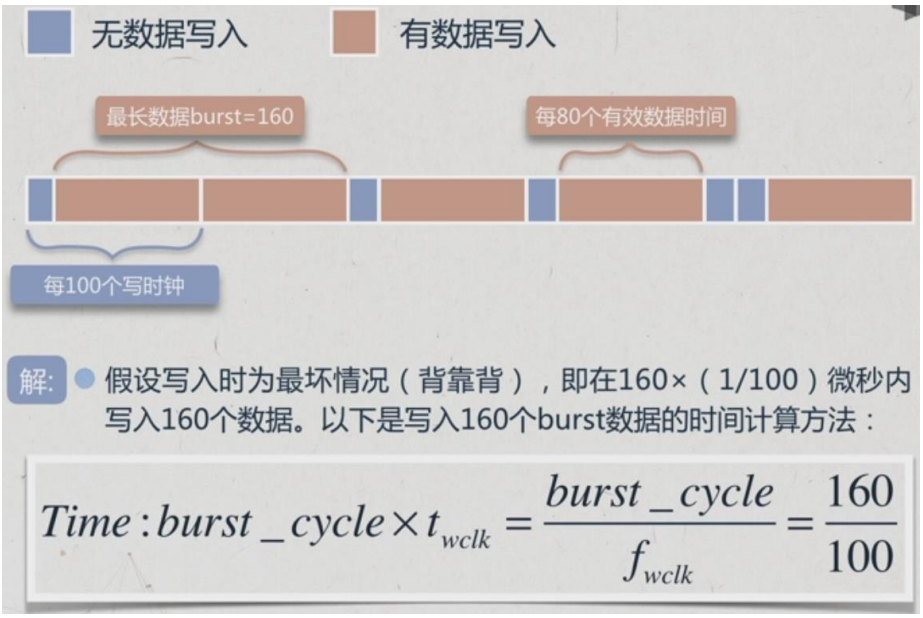
写时钟频率（wclk）、读时钟频率 rclk、写数据时每 B 个时钟周期内会有 A 个数据写入 FIFO、读时每 Y 个时钟周期会有 X 个数据读出 FIFO：此时 FIFO 的最小深度是（要考虑重载时的情况，即写数据背靠背）

$$depth = 2A - \frac{2A}{wclk} \times \left(rclk \times \frac{X}{Y} \right)$$

$$depth = burst\_length - \frac{burst\_length}{wclk} \times \left(rclk \times \frac{X}{Y} \right)$$

- 其中(burst_length/WCLK)表示这个burst的持续时间，(RCLK*(X/Y))表示读的实际速度，两者的乘积自然就是这段时间读出的数据量。
- burst_length表示这段时间写入的数据量。

例题：假设 FIFO 的写时钟为 100MHz，读时钟为 80MHz。在 FIFO 输入侧，每 100 个写时钟，写入 80 个数据；读数据侧，假定每个时钟读走一个数据，问 FIFO 深度设置多少可以保证 FIFO 不会上溢出和下溢出？



解：假设写入时为最坏情况（背靠背），即在160×（1/100）微秒内写入160个数据。以下是写入160个burst数据的时间计算方法：

$$Time: burst\_cycle \times t_{wclk} = \frac{burst\_cycle}{f_{wclk}} = \frac{160}{100}$$

解：在这段时间内只能读出160×（1/100）×80个数据

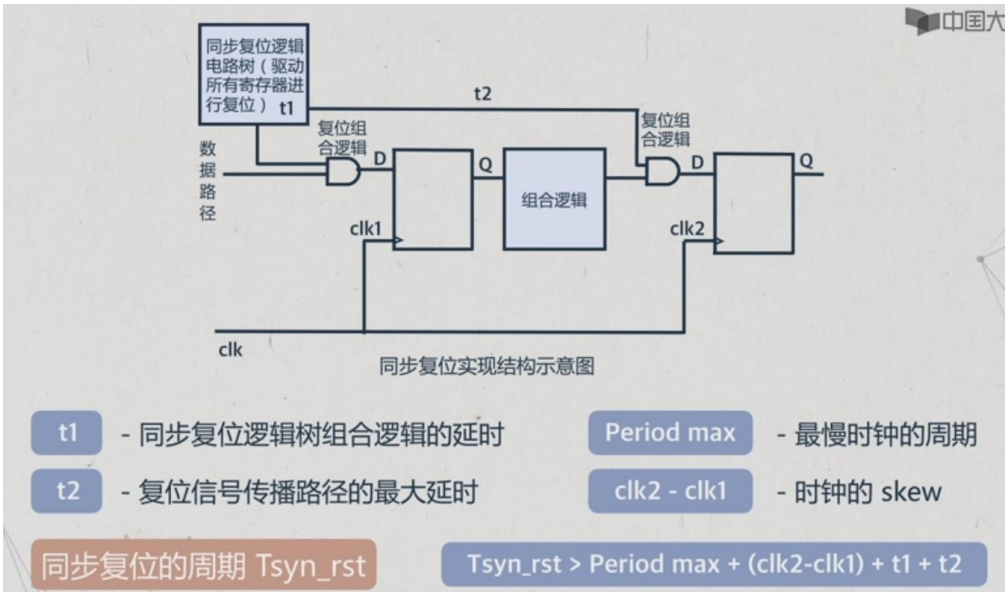
$$data\_num_{read} = \frac{Time}{t_r} = \frac{Time}{\frac{1}{f_r}} = Time \times f_r = \frac{160}{100} \times 80$$

$$depth = burst - data\_numread = 160 - \frac{160}{100} \times 80 = 32$$

八、同步复位异步释放电路

1、复位电路是每个数字逻辑电路中最重要的组成部分之一，逻辑电路中的任何一个寄存器、存储器结构和其他逻辑单元都必须要附加复位逻辑电路，以保证电路能够从错误状态

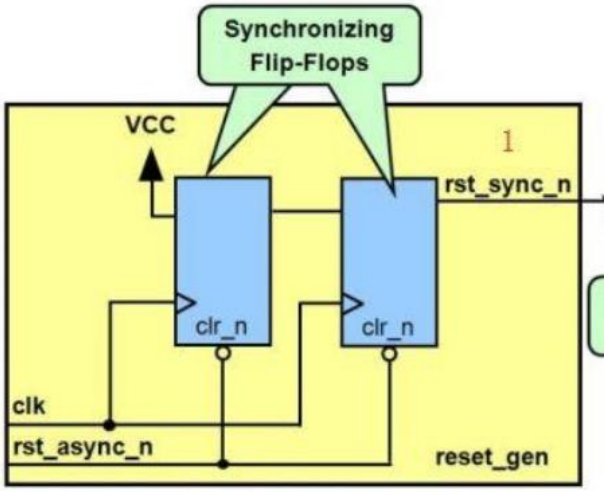
中恢复，可靠地工作。时序电路需要有复位信号，组合电路不需要。
复位电路的功能：
（1） 仿真的时候使电路进入初始状态或者其他预知状态；
（2） 对于综合实现的真实电路，通过复位使电路进入初始状态或者其他预知状态
2、同步复位
（1）定义：当复位信号发生变化时，并不立即生效，只有当有效时钟沿采样到已变化的复位信号后，才对所有的寄存器进行复位。
（2）优点：有利于仿真器的仿真；可以使所涉及的系统成为 100%的同步时序电路，有利于时序分析，综合出来的 fmax 一般较高；由于同步复位信号只在时钟有效电平到来时才有效，所以可以滤除高于时钟频率的毛刺。
（3）缺点：复位信号的有效时长必须大于时钟周期，才能真正被系统识别并完成复位任务，同时还要考虑，诸如 clk skew，组合逻辑路径延时，复位延时等因素；由于大多数的逻辑器件的目标库内的 DFF 都只有异步复位端口，所以，倘若采用同步复位的话，综合器就会在寄存器的数据输入端口插入组合逻辑，这样就会耗费较多的逻辑资源。

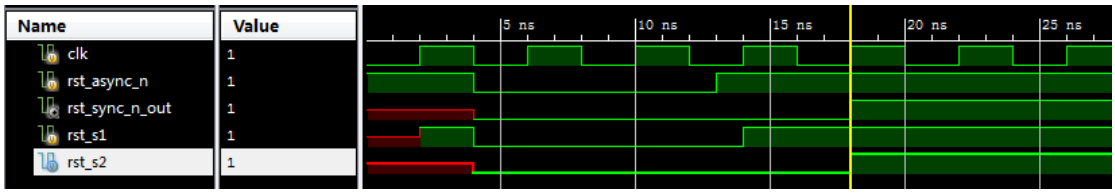


3、异步复位
（1）定义：异步复位是指当复位信号有效沿到达时，无论时钟沿是否有效，都会立即对目标（如寄存器、RAM 等）复位。
（2）优点：大多数目标器件库的 DFF 都有异步复位端口，因此采用异步复位可以节省资源；设计相对简单；异步复位信号识别方便，而且可以很方便的使用 FPGA 的全局复位端口 GSR。
（3）缺点：在复位信号释放的时候容易出现问题。具体就是说：倘若复位释放时恰恰在时钟有效沿附近，就很容易使寄存器输出出现亚稳态；复位信号容易受到毛刺的影响
4、异步复位，同步释放
（1）定义：
异步复位：复位信号可以直接不受时钟信号影响，在任意时刻只要是有效电平就可以复位（复位信号不需要和时钟同步）；
同步释放：让复位信号取消的时候，必须跟时钟信号同步（正好和时钟同沿）
（2）优点：只要复位信号一有效，电路就处于复位状态；即使是端脉宽复位也不会丢失；复位的撤离是同步信号，因此有良好的撤离时序和足够的恢复时间。

（3）电路设计：如图所示，当复位信号到来时，直接进行复位，但是复位信号释放输出时，会通过两级寄存器缓存，然后同步释放。代码设计、电路图与仿真波形如下

```
//////////////////////////////////////////////////
module RST(
   input rst_async_n,
   input clk,
   output rst_sync_n_out
    );

reg rst_s1;
reg rst_s2;
always @ (posedge clk, negedge rst_async_n)
if (!rst_async_n) begin
rst_s1 <= 1'b0;
rst_s2 <= 1'b0;
end
else begin
rst_s1 <= 1'b1;
rst_s2 <= rst_s1;
end
assign rst_sync_n_out = rst_s2;
endmodule
```



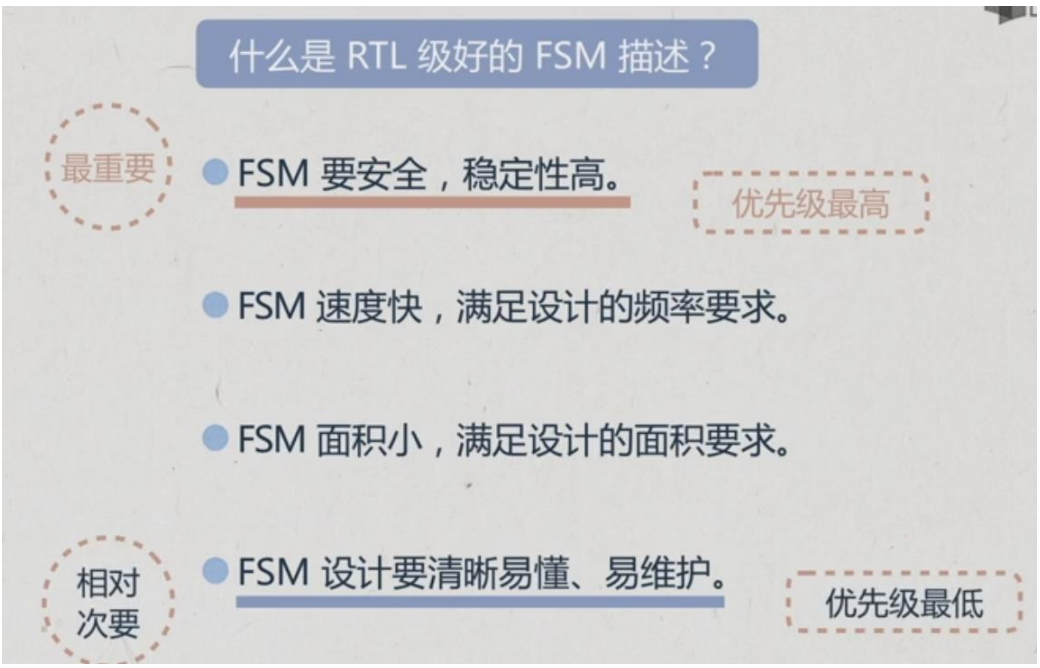


九、状态机 FSM

（1）概述：对具有逻辑顺序或时序规律事件的一种描述方法，状态机的目的都是要控制某部分电路，完成某种具有逻辑顺序或时序规律的电路设计。

（2）应用思路：如果一个电路具有时序规律或者逻辑顺序，我们就可以自然而然地规划处状态，从这些状态入手，分析每个状态的输入，状态转移和输出，从而完成电路功能；明确电路的输出关系，这些输出相当于状态的输出。

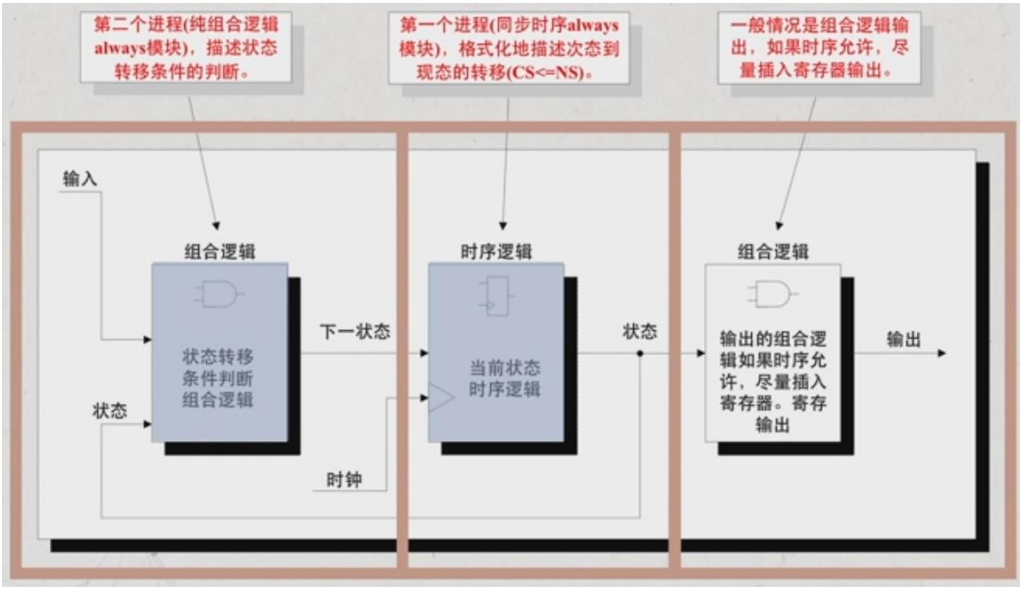
（3）状态机（FSM）正确描述方式：FSM 要安全，稳定性高，速度快，面积小，设计需要清晰易懂、易维护。



（4）两段式状态机：

一个 always 模块采用同步时序描述状态转移

另一个 always 模块采用组合逻辑判断状态转移条件，描述状态转移规律



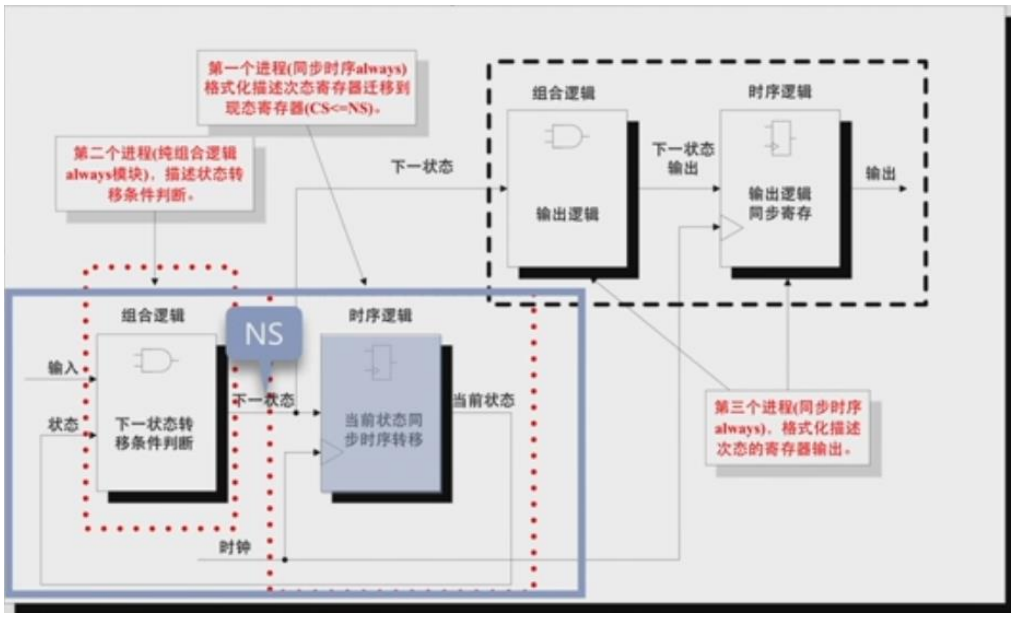
两段式状态机的缺点：其输出一般使用组合逻辑描述，而组合逻辑易产生毛刺等不稳定因素。解决方法：如果在时序允许的情况下插入一个额外的时钟节拍来寄存 FSM 的组合逻辑输出，可以有效地消除毛刺。但有时设计并不允许插入额外的节拍。所以只能采用三段式 FSM 的描述方法（使用同步时序逻辑寄存 FSM 的输出）。

（5）三段式 FSM

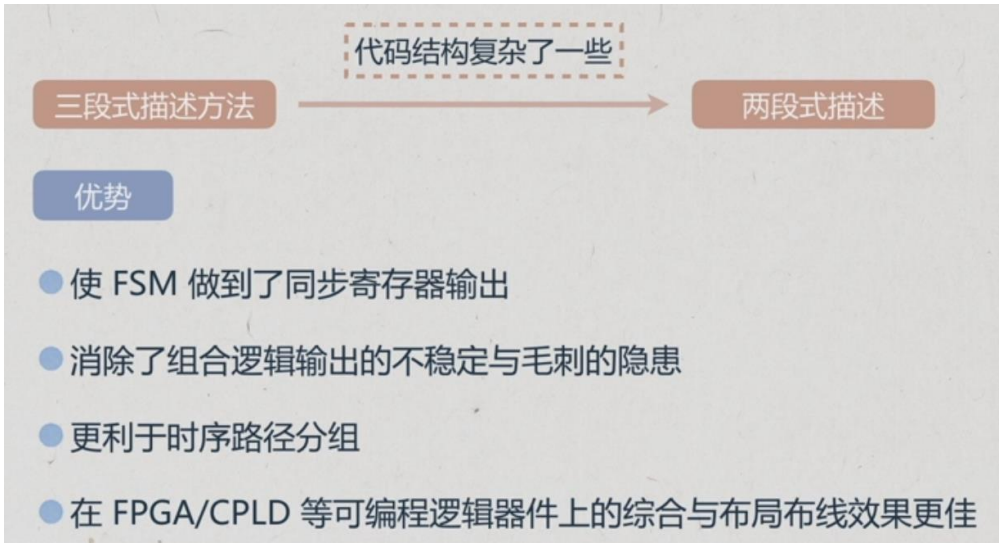
第一段（时序逻辑）是格式化描述次态寄存器迁移到现态寄存器（CS<=NS）

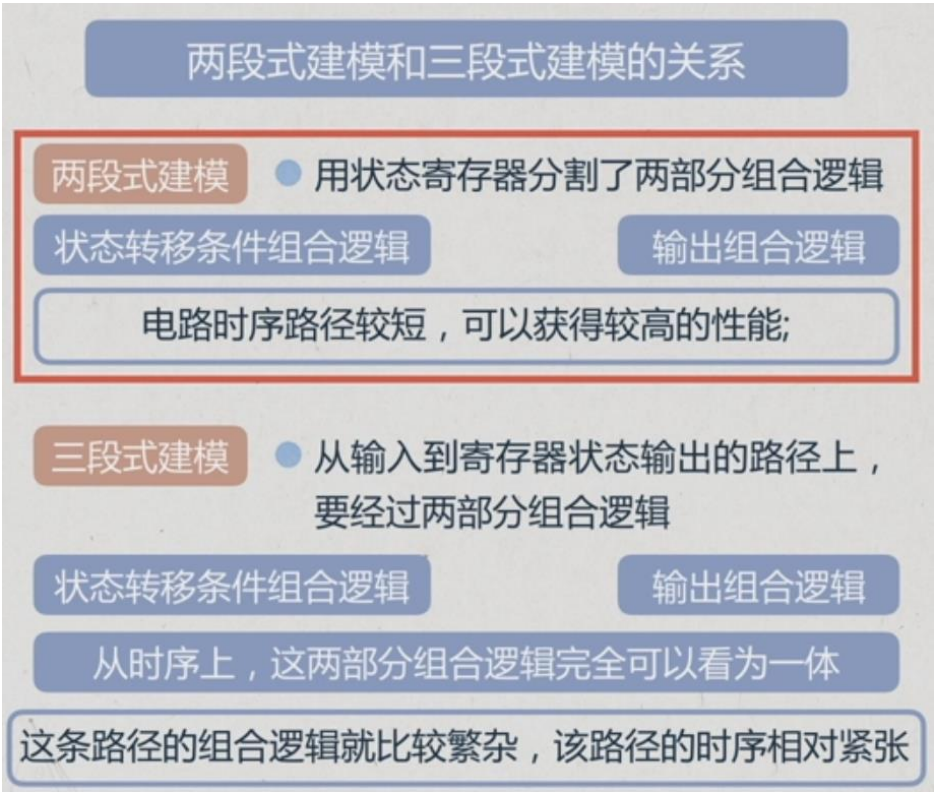
第二段（组合逻辑）是描述状态转移条件判断（case（CS））；

第三段（时序逻辑）是格式化描述次态的寄存器输出（case（NS））



（6）三段式与两段式 FSM 的区别





3种 FSM 描述方法比较表

比较项目	一段式描述方法	两段式描述方法	三段式描述方法
推荐等级	不推荐	推荐	最优推荐
代码简洁程度（对于相对复杂的 FSM 而言）	冗长	最简洁	简洁
always 模块个数	1	2	3
是否利于时序约束	不利于	利于	利于
是否有组合逻辑输出	可以无组合逻辑输出	多数情况有组合逻辑输出	无组合逻辑输出
是否利于综合与布局布线	不利于	利于	利于
代码的可靠性与可维护度	低	高	最好
代码风格的规范性	低，任意度较大	格式化，规范	格式化，规范

十、静态时序分析（STA）

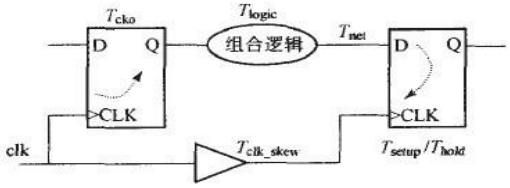


图3　寄存器的建立/保持时间分析

寄存器的建立时间余量 $T_{slack,setup}$ 为：

$$T_{slack,setup} = T_{period} - (T_{cko} + T_{logic} + T_{net} + T_{setup} - T_{clk\_skew}) \quad (1)$$

寄存器的保持时间余量 T_{hold} 为：

$$T_{hold} = T_{cko} + T_{logic} + T_{net} - T_{hold} - T_{clk\_skew}) \quad (2)$$

（1）定义：

Setup time:建立时间,在时钟沿到来之前,数据必须稳定的时间,如果建立时间不满足,无法将值打入寄存器。

Hold time：保持时间，在时钟沿到来之后，数据必须稳定的时间，如何保持时间不满足，无法将值打入寄存器。

STA 分析的专属名词：

时钟周期：Tpd、Tclk

建立时间：Tsetup、Tsu

保持时间：Thold、Thd

两个寄存器之间的组合逻辑延迟：Tlogic、Tcomb

寄存器传输延时（时钟—>输出）：Tco、Tcko

时钟抖动 jitter：Tjitter，对时钟有害

时钟网络延时 skew：Tskew，对时钟有益

静态时序分析黄金公式：

$$（1）\ T_{pd}+T_{skew}>T_{cko}+T_{\log ic}+T_{setup}+T_{jitter}$$

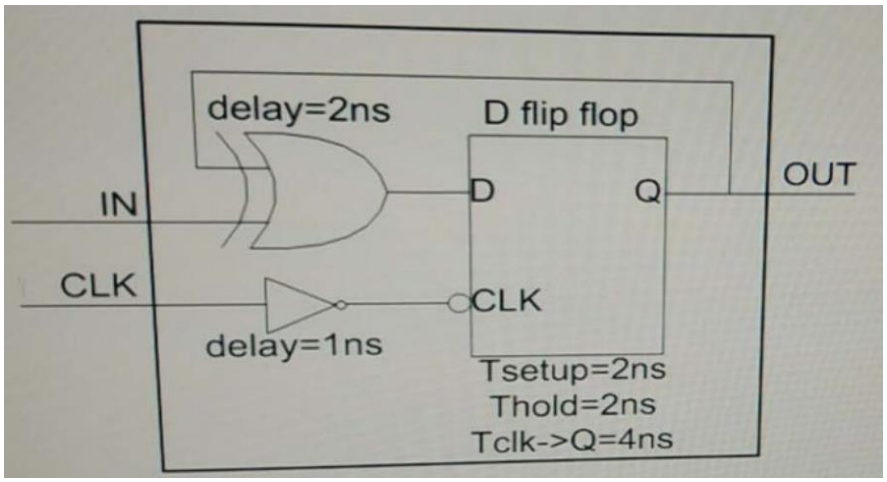
$$（2）\ (T_{setup}+)T_{cko}+T_{\log ic}>(T_{setup}+)T_{hold}+T_{skew}$$

关于这两个公式的正确理解：

（1）在一个时钟周期以内，数据的传输延时（经过寄存器的传输延迟，组合逻辑延迟，建立时间）不能太大，超过一个时钟周期将会导致数据无法不能及时到达下一个寄存器；

（2）图中其实简略了 setup time，当两边加上 setup time 以后，则会发现规律，即数据传输延时（Tsetup+tcko+Tlogic）不能太小，Thold+Tsetup 被称为数据采样窗口期，即数据传输延时不能小于数据采样窗口期，即此窗口期传输的数据应该保持不变，否则会进入亚稳态。

例题：下图中的电路，器件延时如图标注，将框内电路作为一个寄存器，其有效建立时间（setup time）和保持时间（hold time）分别是多少？



解析：时钟信号早于数据到达，对于 setup time 有害，而对于 hold time 是有益的。根据 setup time 和 hold time 的定义，在时钟到达整个寄存器时，应该这样分析：

Setup time：内部寄存器的 setup time 为 2ns，由于有数据路径延迟和时钟延迟，我们以时钟到达外部寄存器的端口为时间终点，则数据在此之前需要保持的时间为 Tsetup +datadelay-

clkdelay=2+2-1=3ns；

Holdup time：内部寄存器的 hold time 为 2ns，由于有数据路径延迟和时钟延迟，我们以时钟到达外部寄存器的端口为时间起点，则数据在此之后需要保持的时间为 Tsetup - datadelay+clkdelay=2-2+1=1ns；

（2）PVT 与保持时间、建立时间之间的关系：

PVT 指的是 process、Voltage、temperature，在 IC 制造工艺中，设计的参数本身存在一些偏差，而且芯片很难在温度恒定不变的情况下运行。

V：晶体管的延时取决于饱和电流，而饱和电流取决于供电电压。

T：对于一个管子，当温度升高，空穴/电子的移动速度会变慢，使延时增加，而同时温度的升高也会使管子的阈值电压降低，较低的阈值电压意味着更高的电流，因此管子的延时减小。而通常温度升高对空穴/电子移动速度的影响会大于对阈值电压的影响，所以温度升高管子的延时呈增加趋势。

在实际情况下，STA 需要考虑这三个因素对保持时间和建立时间的影响。温度和电压都是会对组合逻辑延迟产生一定的影响，延迟增大则会减少建立时间裕度，延迟减小会减少保持时间裕度。

Tsetup<=Tclk+Tskew-Tcq-Tcomb

Thold<= Tcq+Tcomb-Tskew

升温后，延迟增大，建立时间有问题

降温后，延迟减小，保持时间有问题

升压后，延迟减小，保持时间有问题

降压后，延迟增大，建立时间有问题

在分析建立时间是否满足时序要求时要使用慢速模型，即高温+低压+慢速模型；

分析保持时间是否满足时序要求时要使用快速模型，即低温+高压+快速模型。

（3）异步复位信号的时序分析：

recovery 恢复时间：在有效的时钟沿来临前，异步复位信号保持稳定的最短时间（类似于 Tsetup）；

removal 移除时间：在有效的时钟沿来临后，异步复位信号保持稳定的最短时间（类似于 Thold）；

十一、有符号数和无符号数运算总结

1、位宽截断：无论是有符号数还是无符号数，位宽大的数赋值给位宽小的数，数据就会被截断，截断的规则就是从低位开始取，被截断的是高位。

2、位宽展宽：对于无符号数来说，将原位宽赋给低位，高位补 0；对于有符号数来说，将原位宽赋给低位，高位补符号位。

3、数值运算：数值运算包含下列三种情况

（1）两个有符号数运算：有符号数运算按照补码运算

（2）两个无符号数运算：直接运算

（3）有符号数和无符号数进行运算：把有符号数看做无符号数，直接按照无符号数进行运算。

4、赋值：

将一个数或计算结果 A(有符号或者无符号)赋值给另一个数 B，根据位宽不同有以下三种情况：

（1）当位宽相同的时候直接赋值，无论 B 是有符号或者无符号，对赋值结果没有影响，只是把二进制位完全不变的赋值进来。

（2）当 B 的位宽比 A 小的时候，位宽截断如前面所说，从低位开始截取

（3）当 B 的位宽比 A 大的时候，A 的二进制码先填充在 B 的低位；高位填充分三种情况：

若 A,B 都是有符号数，B 的高位用 A 的最高位填充。

若 A,B 都是无符号数，B 的高位用 0 填充

若 A 是有符号数，B 是无符号数，B 的高位用 A 的最高位填充

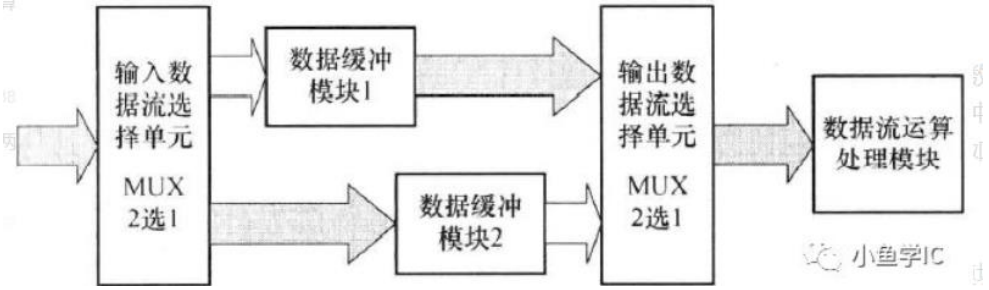
若 A 是无符号数，B 是有符号数，B 的高位用 0 填充

总结：若 A 是有符号数，则 B 高位空余用 A 的最高位填充，若 A 是无符号数，B 的高位用 0 填充

PS:负数用二进制数表示时一般用补码形式表示。如-5 用四位有符号数表示为 4'b1011。

十二、乒乓 buffer

1. 其本质是两个同样大小的单口 SRAM 背靠背组成的一种电路结构，其工作方式就是一乒一乓，即一个在读的时候另一个在写，然后互换读写状态。该 buffer 放在数据通路的中间，在大部分时候都能保证一块 RAM 收上游的数据，一块往下游发数据，一读一写并行操作。



工作过程如下：

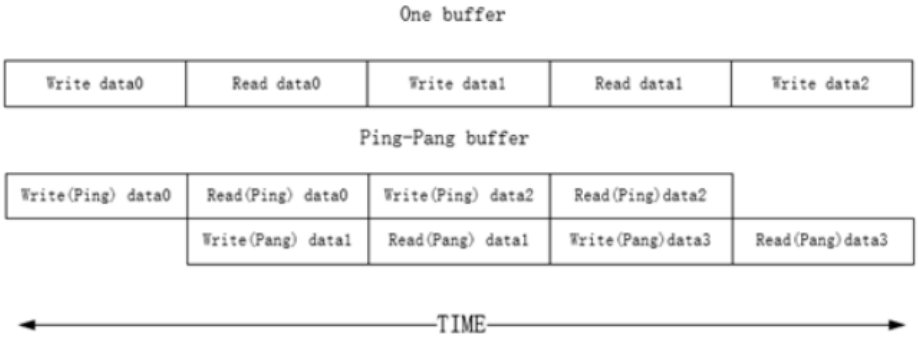
第一段乒乓缓冲周期：将数据流缓存到数据缓冲模块 1。

第二段乒乓缓冲周期：将数据流缓存到数据缓冲模块 2，并从数据缓冲模块 1 读取数据送入数据流运算处理模块。

第三段乒乓缓冲周期：将数据流缓存到数据缓冲模块 1，并从数据缓冲模块 2 读取数据送入数据流运算处理模块。

接下来就是上面描述的第二段、第三段乒乓缓冲周期的循环往复。

2. 优缺点：该方法是利用加倍存储资源的方法来提高读写速度的方法。当然最大的代价就是面积和功耗的开销。下图是一个 buffer 的读写方式和 Ping-Pong buffer 的读写方法，可以看出，乒乓 buffer 可以很高地提高数据读写效率。



十三、串并转换电路

1、 串行输入、并行输出（SIPO）：

输入是单 bit 数据，输出是 N bit 数据，设置一个 N bit 寄存器，只需在每个时钟周期内

将输入数据放入低位，然后进行移位。最后设定一个计数器，在计数到 N-1 时，将 N bit 数据进行输出。

2、 并行输入、串行输出（PISO）：

输入是 N bit 并行数据，将 N 为并行数据暂时寄存在一个 N 位的寄存器中，然后通过移位寄存器的移位，依次输出到一位输出端口既可以实现并串转化。

3、串并转换的核心思想是面积与速度之间的相互转换，在设计中常根据具体的设计指标来实现串并转换。

十四、逻辑综合

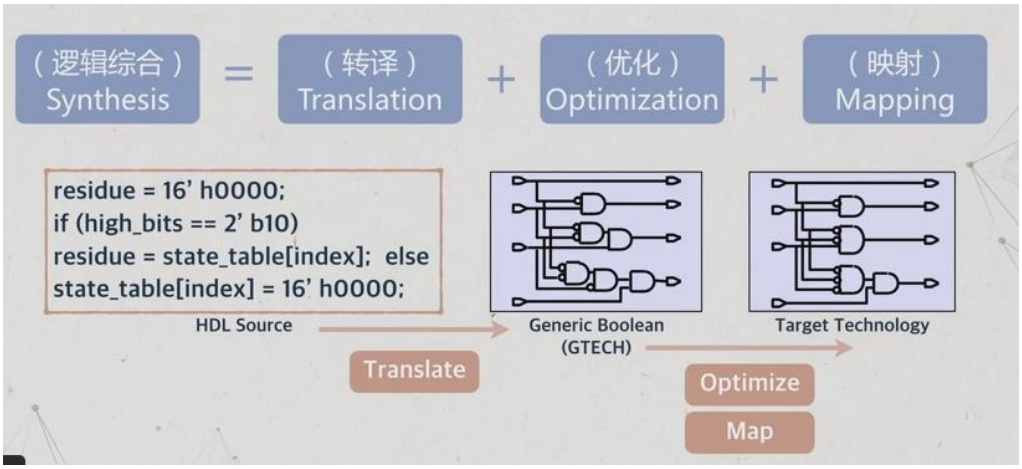
1、 概述

（1） 综合是前端模块设计中的重要步骤之一，综合的过程是将行为描述的电路、RTL 级的电路转换到门级的过程。

（2） Design Compiler 是 Synopsys 公司用于做电路综合的核心工具，它可以方便地将 HDL 语言描述的电路转换到基于工艺库的门级网表。

（3） 逻辑综合的目的：决定电路门级结构、寻求时序和面积的平衡、寻求功耗与时序的平衡、增强电路的测试性。

2、 逻辑综合的三个阶段：



（1）转译——将 HDL 代码根据数据库转换成相应的门级数据库，这个数据库跟工艺库是独立无关的。

（2）优化——添加相应的约束：如工作频率、面积、功耗等

（3）映射——指定工艺库，最终形成该工艺库对应的门级网表

3、 设计对象

指的是待综合的对象

（1） Design：整个需要待综合的电路对象；

（2） Port：整个 Design 最外部的输入、输出端口；

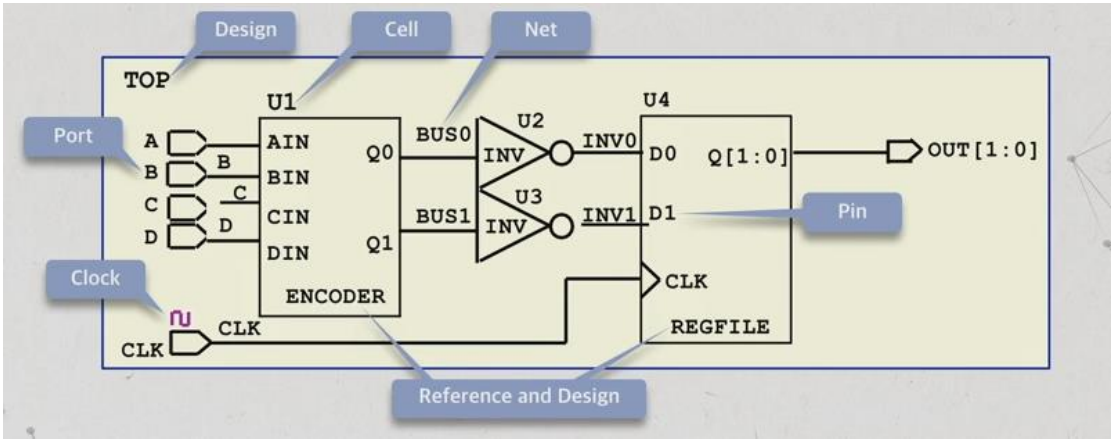
（3） Clock：时钟信号作为电路中非常重要的端口，一定要单独处理；

（4） Cell：被例化的模块；

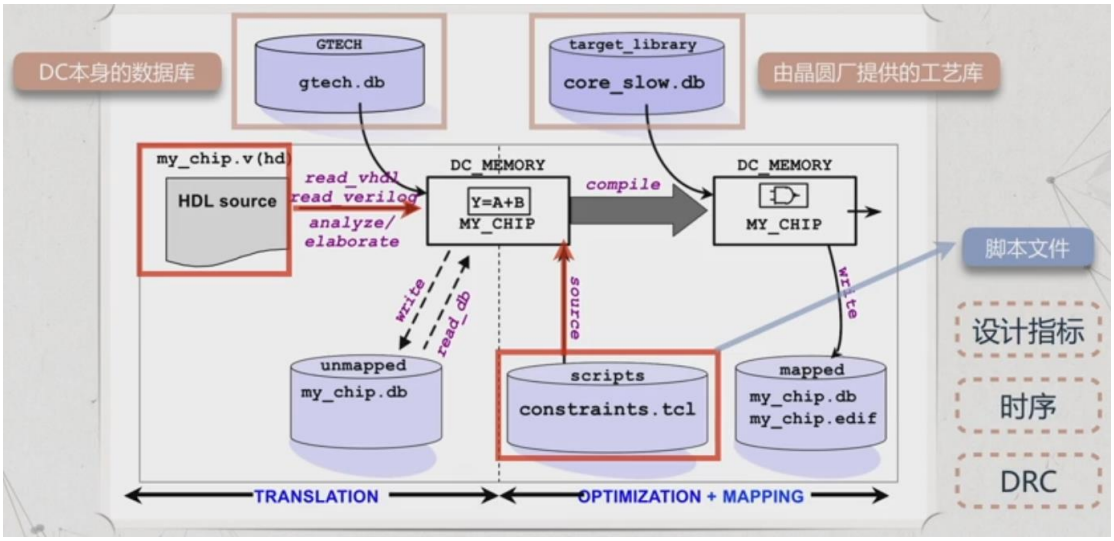
（5） Reference：原电路模块；（例化 reference 后的模块叫做 cell）

（6） Pin：cell 的引脚；

（7） Net：内部连线。



4、 DC 流程



（1）HDL_source：原始的 HDL 代码输入

（2）GTECH：DC 本身的数据库，用于转译过程；

（3）Target_library:晶圆厂提供的工艺库，用于映射过程；

（4）Scripts:脚本文件，用于优化过程，施加约束主要对电路的设计指标，时序和 DRC 等要求。

整个综合过程分为四个部分：预综合过程，施加设计约束，设计综合，后综合过程

（1） 预综合过程：为综合做准备得步骤：

a) DC 启动：启动方式主要有四种，dc_shell 命令行方式、dc_shell-t 命令行方式（tcl 语言）、design_analyzer 图形方式、design_version 图形方式，图形启动方式是建立在命令行方式之上的；启动 DC 时会生成日志文件，command.log 和 view_command.log 用来记录用户在使用 DC 是执行的命令和设置的参数，而 filenames.log 用来记录 dc 访问过的目录，包括库、源文件，在退出 DC 是会自动删除。日志文件可以对错误进行追溯。

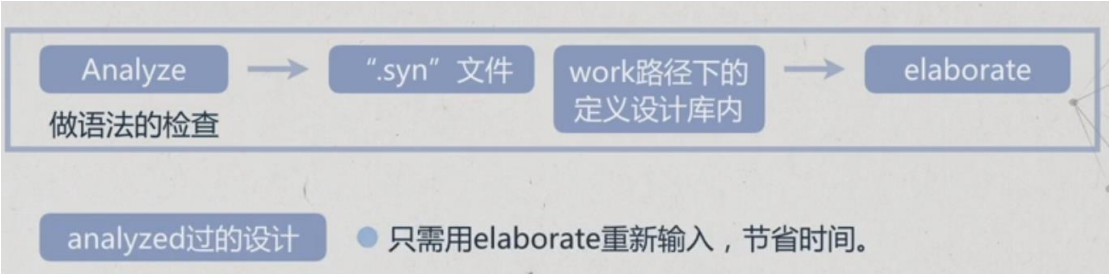
b) 读入设计文件：DC 的 read 指令支持读入多种硬件描述的格式：.db、.v、.vhd。但是 dcsh 工作模式读取不同的文件格式只需要带上不同的参数，而 TCL 的工作模式读取不同的文件格式需要使用不同的命令。

举例：read -format Verilog[db、vhdl etc] file //dcsh 的工作模式

Read_db file.db//tcl 工作模式读取 db 格式；

Read_Verilog file.v//tcl 工作模式读取 Verilog 格式；

读取源程序的另一种方式是 analyze&elaborate（推荐使用这种方式去读取 Verilog 和 VHDL 文件），如下图所示，analyze 做完语法检查以后的设计文件，可以直接用 elaborate 重新输入。但是 Read 不可以这么做。



两种读取文件的区别：

类别	analyze&elaborate	read
格式	verilog 或VHDL	verilog 、VHDL、EDIF、db等所有格式
用途	综合verilog 或VHDL的RTL设计	读网表，设计预编译
设计库	用-library选项定义设计库名，存储 ".syn" 文件	用缺省的设置，不能存储中间结果
Generics(vhdl)	可以对parameter进行操作	不能对parameter进行操作
Architecture(vhdl)	可以进行结构化的操作	不可用

当所有需要综合的模块全部读完以后，用 link 指令来将模块或者实体连接起来。

c) 设置各种库文件；

i. DC 设计中需要用到四种库文件：目标库（target_library），链接库(link_library)，符号库(symbol_library)，算术运算库(synthetic_library)

Set target_library "my_tech.db"

Set link_library "*my_tech.db"

Set symbol_library

1. target_library 是综合后电路网表要最终映射到的库，一般是.db 格式，可以由文本格式的.lib 转化而来，目标库包含了各个门级单元：行为、引脚、面积、时序信息、功耗方面的参数。DC 在综合时就是根据 target_library 中给出的单元电路的延迟信息来计算路径的延迟，并根据各个单元延时、面积和驱动能力的不同选择合适的单元来优化电路；
2. link_library 设置模块或者单元电路的引用，对于 DC 可能用到的库，我们都需要在 link_library 中指定，其中也包括要用到的 IP。链接库和目标库的区别主要是：链接库主要是付费 IP、存储器、IO、PAD 之类的 IP 库，而目标库更多的是标准单元。设置 link_library 的时候需要加 "*"。
3. symbol_library 定义了单元电路显示的 Schematic 的库，用来查看电路的原理图，如果不需要查看原理图可以不设置。
4. synthetic_library 包含了一些高性能运算器的库，如超前进位加法器，一

般需要更高级的 license 才能使用。

d) 创建启动脚本文件；

e) 读入设计文件；

f) DC 中的设计对象；

g) 各种模块的划分；

h) Verilog 编码

（2） 施加设计约束：

a) 设计面积约束；

Set_max_area 100//施加一个最大面积 100 单位的约束，

单位的定义有三种可能的标准：1）第一种是将一个二输入与非门的大小作为单位 1；第二种是以晶体管数目规定单位；第三种则是根据实际的面积。

b) 设计时序约束；

i. 时序路径分为四种，输入到寄存器的路径；寄存器到寄存器之间的路径；寄存器到输出的路径；输入直接到输出的路径。

create_clock -period 10 [get_ports Clk]//定义一个 10ns 的时钟

set_dont_touch_network [get_ports Clk]//综合时不对 clk 信号优化

set_input_delay -max 4 -clock Clk [get_ports A]//定义输入最大延迟

set_output_delay -max 5.4 -clock Clk [get_ports B]//定义最大输出延迟

c) 设计规则约束（DRC 约束）

set_max_fanout 50 //设定扇出数，表示单元输入引脚相对负载的数目

set_max_capacitance //设定最大负载电容

set_max_transition //设置最大转化时间

d) 设计环境约束（PVT 对电路的影响）：单元的延时会随着温度的上升而增加，随着电压的上升而减小，随着工艺尺寸的增大而增大。工作条件有最差，最好，典型的情况。综合时考虑最好和最差两种，其中最差用于建立时间的时序分析，最好用于保持时间的时序分析。

set_load 5 [get_ports OUT1]

set_driving_cell 允许用户可以自行定一个实际的外部驱动 cell

set_wire_load_model -name 160KGATES //设置连线延迟

在定义完环境属性之后，我们可以使用下面的几个命令检查约束是否施加成功

check_timing 检查设计是否有路径没有加入约束

check_design 检查设计中是否有悬空管脚或者输出短接的情况

write_script 将施加的约束和属性写出到一个文件中，可以检查这个文件看看是否正确，这个脚本很重要，后续都要以这是约束脚本为基准。

（3） 编译：DC 会选择一种编译策略来执行编译，编译策略分两种，top down 和 bottom up。在 top-down 策略中，顶层设计和字设计在一起编译，所有的环境和约束设置针对顶层设计，虽然此种策略自动考虑到相关的内部设计，但是此种策略不适合与大型设计，因为 top down 编译策略中，所以设计必须同时驻内存，硬件资源耗费大。在 bottom up 策略中，子设计单独约束，当子设计成功编译后，被设置为 dont_touch 属性，防止在之后的编译过程中被修改，所有同层子设计编译完成后，再编译之上的父设计，直至顶层设计编译完成。Bottom up 策略允许大规模设计，因为该策略不需要所有设计同时驻入内存。

（4） 产生报告并解决问题

DC 使用命令 report_timing 来报告设计的时序是否满足目标。时序报告的主要内容：表

头、数据发射路径、数据捕获路径、时序结果。

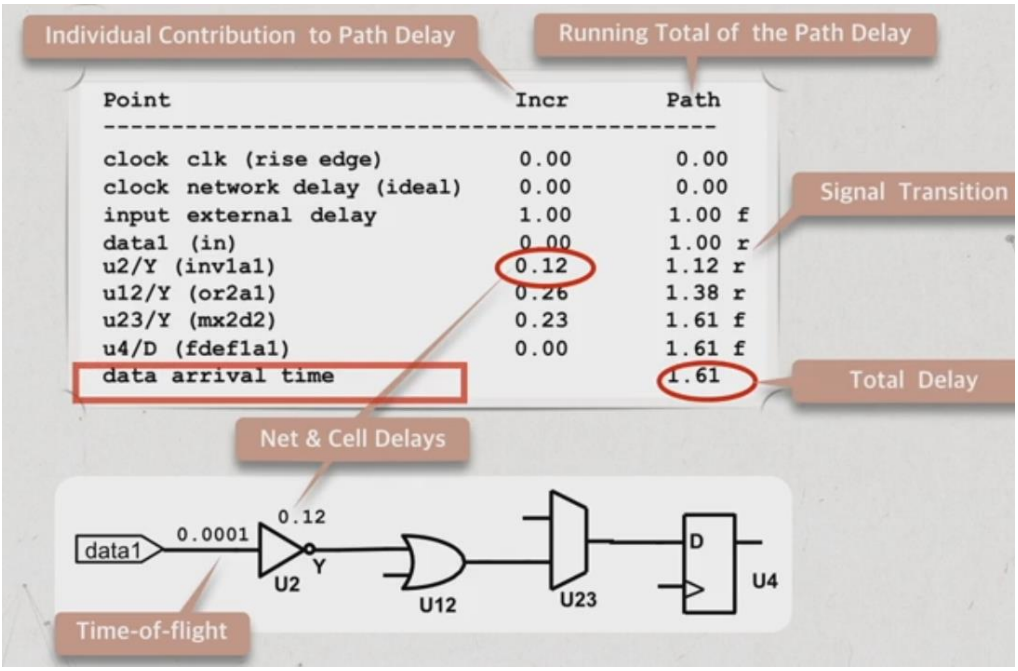
```
****************************************
Report : timing
         -path full
         -delay max
         -max_paths 1
Design : TT
Version: 2000.05
Date   : Tue Aug 29 18:22:38 2000
****************************************

Operating Conditions: slow_125_1.62    Library: ssc_core_slow
Wire Load Model Mode: enclosed

  Startpoint: data1 (input port clocked by clk)
  Endpoint: u4 (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  TT                 5KGATES               ssc_core_slow
```

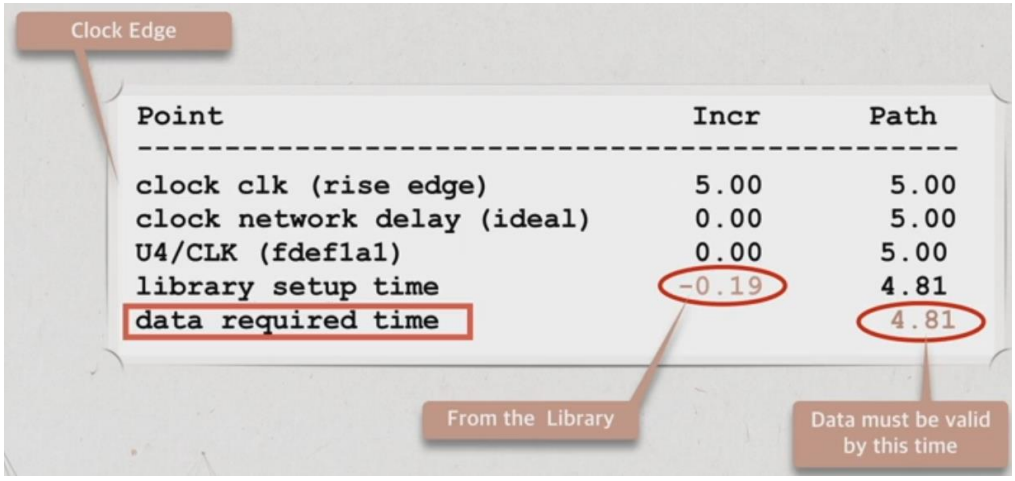
运行环境

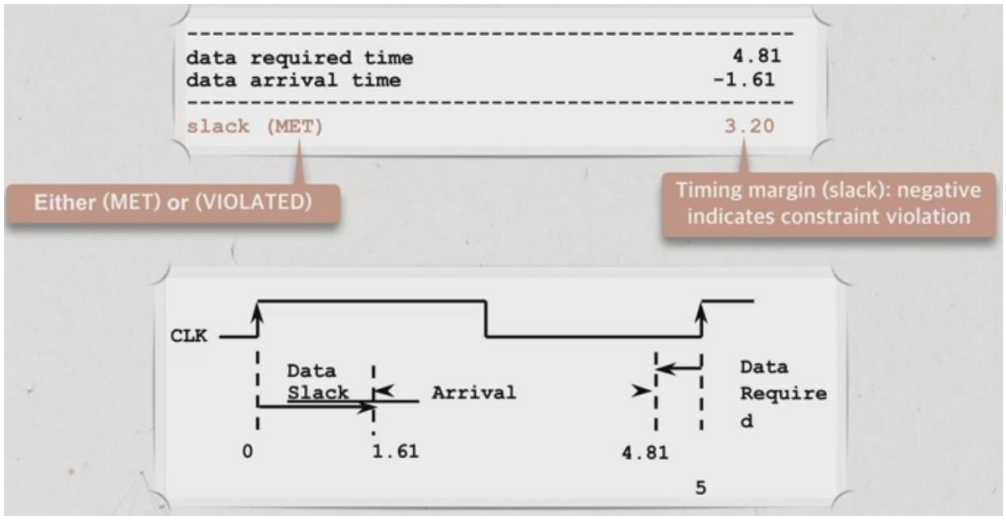
关键路径

max表示建立时间分析，min表示保持时间分析

线性负载模型







（5） 存储设计数据：DC 需要手动存储设计数据（网表、延时信息等数据文件）。

（3）TCL 示例

TCL脚本

```
sh date       //显示开始时间
remove_design –designs  //移除DC中原有的设计
//下面是库的设置，对应图形界面操作的2
############################
#set library              #
############################
set search_path [list *********]
set target_library  { tt.db }
set link_library    { * tt.db  }
set symbol_library  { tt.sdb }
```

```
//下面是屏蔽一些warning信息，DC在综合时遇到这些warning时就把它们忽略，不会报告这些信息，VER-130，VER-129等是不同warning信息的编码，具体含义可以查看帮助
############################
#void warning Info       #
############################
suppress_message  VER-130
suppress_message  VER-129
suppress_message  VER-318
suppress_message  ELAB-311
suppress_message  VER-936
```

```
//读入example1.v文件，对应于图形界面的3
################################
#read&amp;link&amp;Check design#
################################
read_file  -format verilog  ~/example1.v

#analyze –format verilog ~/example1.v
#elaborate EXAMPLE1

current_design EXAMPLE1   //把EXAMPLE1指定为当前设计的顶层模块
uniquify
check_design
```

```
//设置一些变量
##############################
#   define IO port name     #
##############################
set clk [get_ports clk]  //设置变量clk的值是[get_ports clk]，在下面的代码中若出现$clk字样，则表示引用该变量的值，即用[get_ports clk]代替$clk。
set rst_n [get_ports rst_n]

set general_inputs [list a b c]
set outputs [get_ports o]
```

```
//设置约束条件，对应于图形界面的4
##############################
#    set_constraints        #
##############################
//设置时钟约束，对应于图形界面的4.1
#1 set constraints for clock signals
create_clock -n clock $clk -period 20 -waveform {0 10}  //创建一个周期为20ns，占空比为1的时钟
set_dont_touch_network [get_clocks clock]
set_drive 0 $clk   //设置时钟端口的驱动为无穷大
set_ideal_network [get_ports clk]  //设置时钟端为理想网络
```

```
//设置复位信号约束，对应于图形界面的4.2
#2 set constraints for reset signals
set_dont_touch_network $rst_n
set_drive 0 $rst_n
set_ideal_network [get_ports rst_n]
```

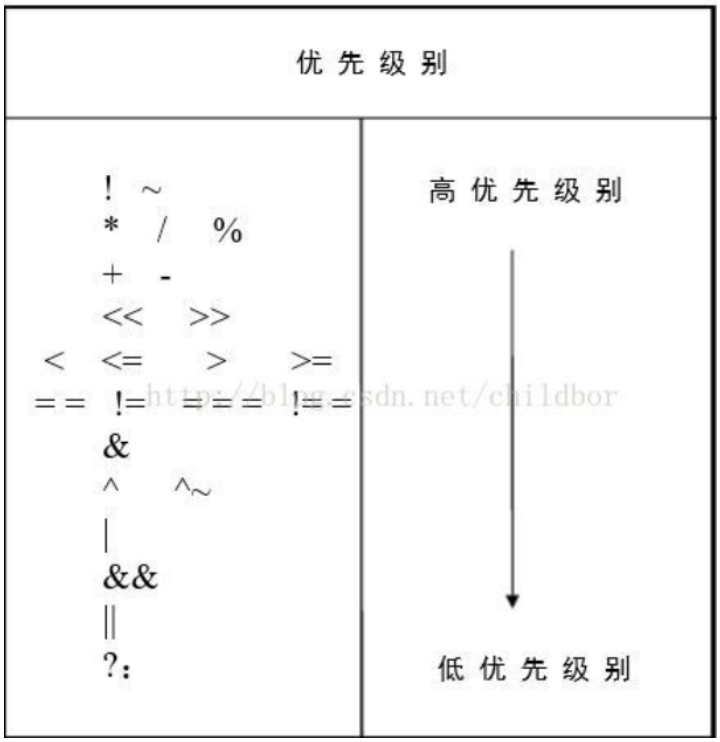
```
//设置面积约束和设计约束，对应图形界面的4.5
#5 set design rule constraints
set_max_fanout 4 $general_inputs
set_max_transition 0.5 [get_designs
"EXAMPLE1"]
#6 set area constraint
set_max_area 0
```

```
//综合优化，对应图形界面的5
##############################
#   compile_design         #
##############################
compile -map_effort medium

//保存文件，对应图形界面的7
############################
# write *.db and *.v     #
############################
write -f db -hier -output ~/EXAMPLE1.db
write -f verilog -hier -output ~/EXAMPLE1netlist.v
write_sdf -version 2.1 ~/EXAMPLE1.sdf   //保存反标文件
```

```
//产生报告并保存，对应图形界面的6
############################
# generate reports       #
############################
report_area > EXAMPLE1.area_rpt  //把报告面积的文件保存
成EXAMPLE1.area_rpt文件，运行完脚本以后可以查看该文件。
report_constraint -all_violators > EXAMPLE1.constraint_rpt
report_timing > EXAMPLE1.timing_rpt
sh date  //显示结束时间
```

（5）DC 的优化策略：优化目的主要是 timing 和 area 约束，以满足用户对功能，速度和面积的要求。

十五、逻辑运算优先级



十六、浮点数转化为定点数

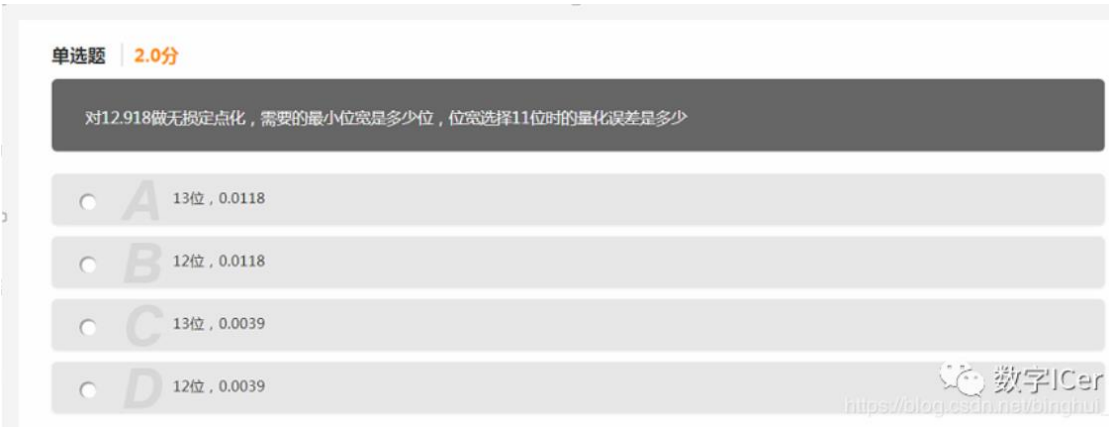
（1） 浮点数：小数的位置不确定，理解为小数点的位置在浮动；定点数

（2） 浮点数量化为定点数的格式为：m 位定数数，n 个小数位，无符号位，那么 n 被称为量化系数；

（3） 对于小数位而言，如果小数位所占的位数越多，则精度就越高，小数点后有 n 位，那么其表示的最大精度为 1/(2^n)；而对于整数位而言，整数位所占的位数越多，

则能表示的值越大。

（4） 无损定点化：无论有多少小数位，2 进制编码的精度都是以 5 结尾的，因此 2 进制编码并不能完全无损的表示任意小数，但是根据数学上误差的概念，只要误差小于精度的一半，就可以认为是“无损”的了。



分析：12.918 用二进制数表示，整数部分需要 4 位，假如用 8 位来表示小数，则量化精度是 1/256=0.00390625，如果小数位用 8 位表示，则 0.918*2^8=235.008，用整数 235 表示，量化误差为.008/256=0.00003125,明显量化误差小于量化精度的一半，所以 12 位能够满足无损定点化；如果使用 7 位来表示小数（11 位表示数据），则量化精度是 1/128=0.0078125，0.918*128=117.504，四舍五入用 118 表示，量化误差为 0.496/128=0.003875,不满足无损定点化。

十七、系统任务和系统函数

1.系统函数和系统任务不可以在综合工具或者布线工具中运行。实际上，它们智能在 Verilog 仿真器中运行，仅仅对代码仿真有意义，综合工具和布线工具将忽略所有的系统任务和函数。

2.函数不能调用任务，但是任务可以调用函数。

3.Verilog 中的预先定义的函数与任务类型：

（1） 显示任务；

$display 是显示任务，通常用来显示变量值、字符串和仿真时间等信息。

如：$display (“The value of ABC is %d”, ABC)//该语句显示当前 ABC 变量的值。

（2） 文件输入/输出任务；

系统函数$fopen 用于打开一个文件，并返回一个整数的文件指针。然后，$fdisplay 就可以使用这个文件指针在文件中写入信息。写完会后，则可以使用$fclose 系统关闭这个文件。

例如：

```
Integer Write_Out_File;//定义一个文件指针
Write_Out_File = $fopen (“Write_Out_File.txt”);
$fdisplay (Write_Out_File, “@%h \n %h”, Mpi_addr, Data_in);
$fclose (Write_Out_File);
```

（3） 时间标度任务：

（4） 模拟（仿真）控制任务：

$finish 表示使仿真器退出；

$stop 表示使仿真器挂起。

（5） 时序验证任务：

$setup 系统任务用来检查建立时间；

$hold 系统任务用来检查保持时间；

$time 系统函数用来返回一个 64 位的模拟时间。

（6） PLA 建模任务：

（7） 随机建模任务：

（8） 实数变换函数：

（9） 概率分布函数：

$random 系统函数可以用来返回一个 32 位的有符号整型随机数。

十八、面积优化和时序（速度）优化

1、 时序优化的常用方式，一般为了提高系统运行速度：

（1） 流水线设计：在关键路径中插入寄存器达到优化时序的目的；

（2） 寄存器配平（重定时 retiming）；

（3） 关键路径优化：一般减少关键路径上的延时；

（4） 迟滞信号后移：延时较大的信号放在后面，缩短这个信号的路径长度

（5） 消除代码的优先级

（6） 并行化处理：树形结构

2、 面积优化的常用方式，一般为了提高资源利用率同时降低功耗要求：

（1） 串行化：串行化可以有效减少资源的利用，相应的会牺牲一定的速度

（2） 资源共享：对于一些可以复用的资源，尽可能的进行复用

（3） 逻辑优化：消除冗余的逻辑

十九、毛刺的成因，无毛刺时钟切换

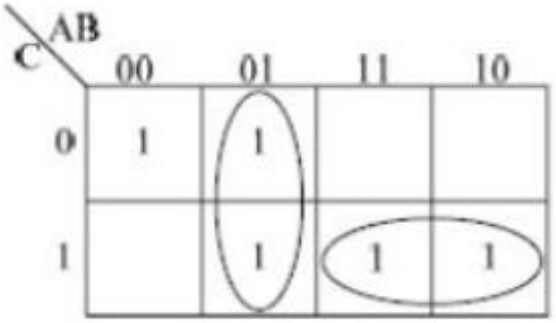
1.定义：

a) 竞争：由于组合逻辑中存在的连线延迟和门延迟，导致每个信号高低电平转换需要的时间不同，导致同一个门的多个输入信号同时发生变化时，变化后的信号到达门电路输入端口的时间存在差别，这种现象叫做竞争。

b) 冒险与毛刺：由于竞争会导致门电路输出一些错误的尖峰信号，这些尖峰信号被称为毛刺。若电路输出会产生毛刺，那电路中存在冒险。冒险可分为静态冒险和动态冒险，静态冒险指输入发生变化而输出不应该发生变化时却产生毛刺的冒险，动态冒险是输入发生变化，输出也发生变化时产生毛刺的冒险。冒险还可分为功能冒险和逻辑冒险，功能冒险指多个输入同时发生变化在输出端产生毛刺的冒险，逻辑冒险指一个输入发生变化在输出端产生毛刺的冒险。

2.冒险的判断方法：

a) 卡诺图法：在卡诺图中如存在两个相切的素项圈，则电路存在冒险。



产生冒险的组合逻辑的卡诺图示意图

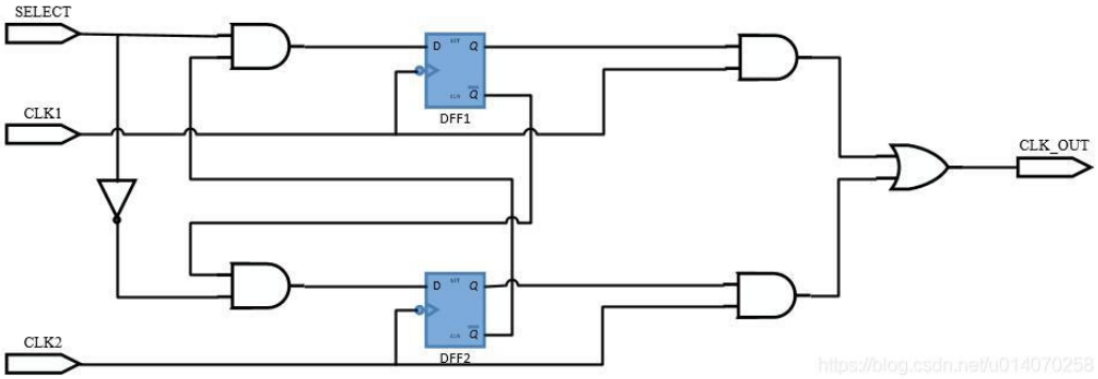
3.毛刺的消除方法：

a) 同步电路触发器对组合逻辑输出进行采用来消除毛刺对后级电路的影响；
b) 对逻辑表达式添加冗余项来消除逻辑冒险；
c) 用格雷码来代替普通二进制码变化从而消除功能冒险。
d) 在毛刺输入到下一个模块之前，通过滤波电路将毛刺滤除。毛刺一般是非常窄的脉冲，可以在输出端接一个几百微法的电容将其滤除掉。
e) 在下一级模块中，对输入信号进行采样，当信号保持稳定后，再进行操作

4．无毛刺时钟切换电路：

在芯片运行时，有时需要切换不同的时钟源，两个不同的时钟之间可能是完全不相关的，也可能是同源时钟。但是在切换的时候，如果未加切换电路，则极易产生毛刺，可能会引发逻辑错误。

（1） 同源时钟之间的相互切换电路的关键设计点在于，让时钟切换的时间发生在待切换成的低电平状态下。



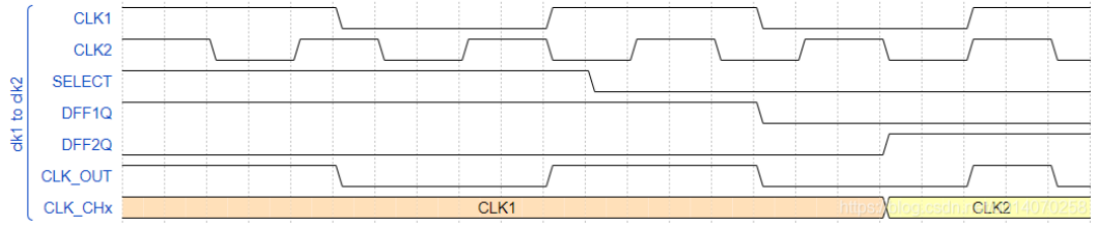
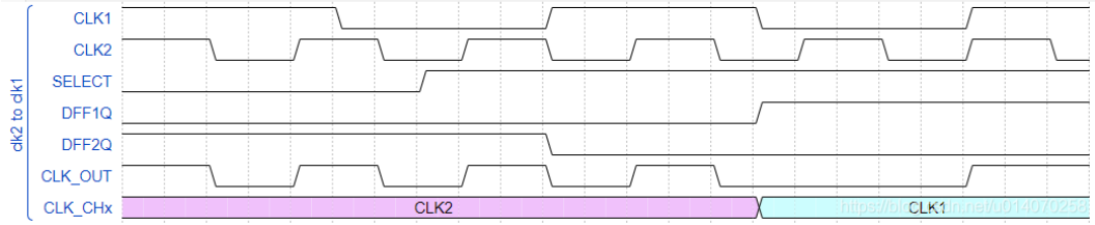
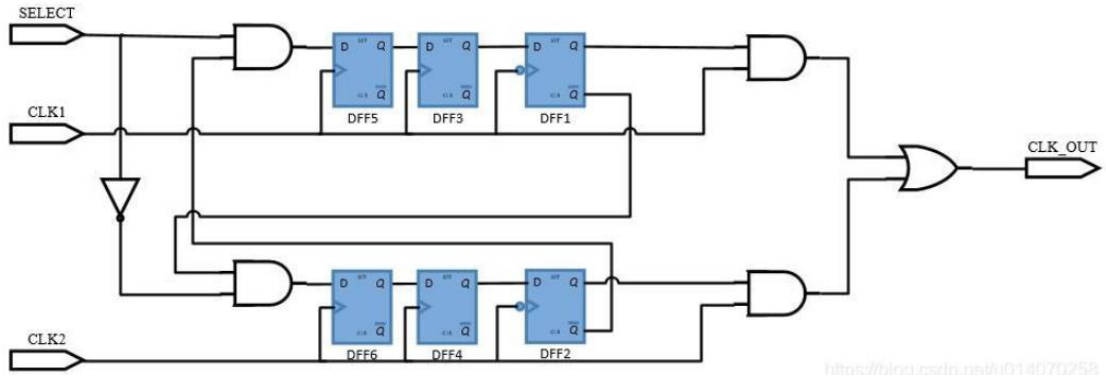
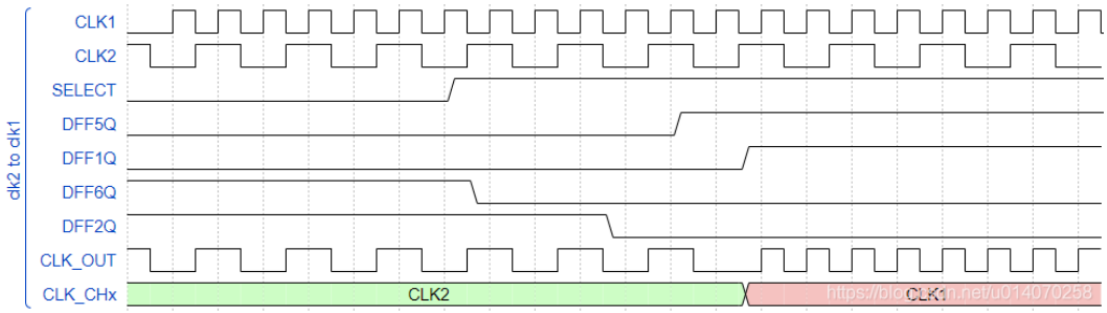


图 4 时钟1切换到时钟2时序图

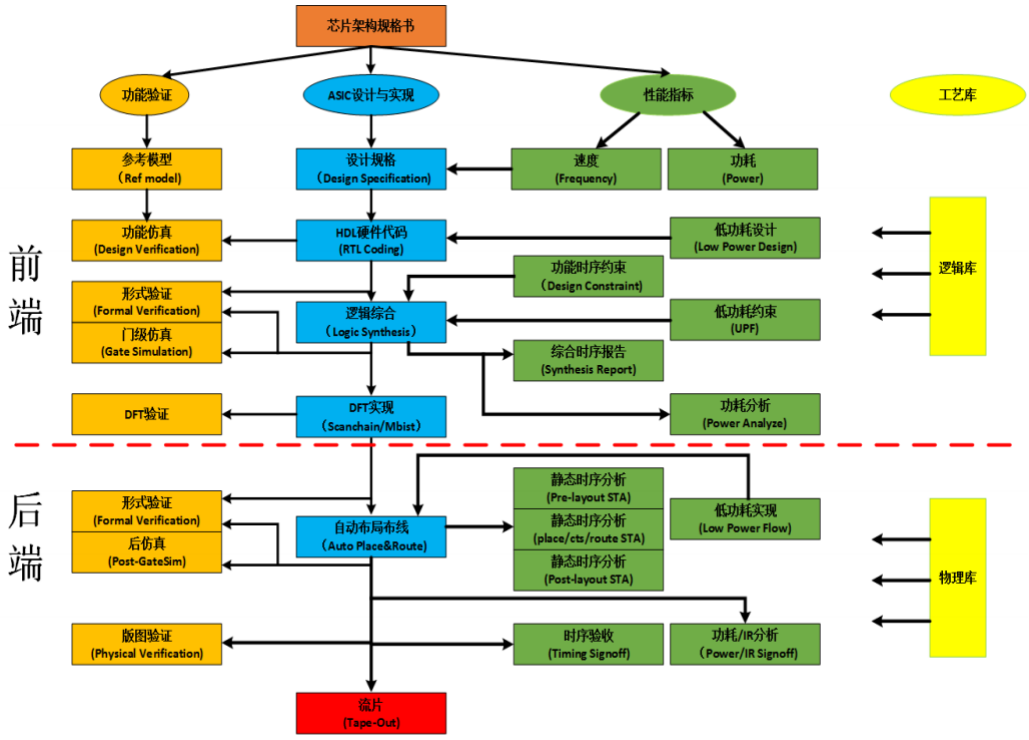


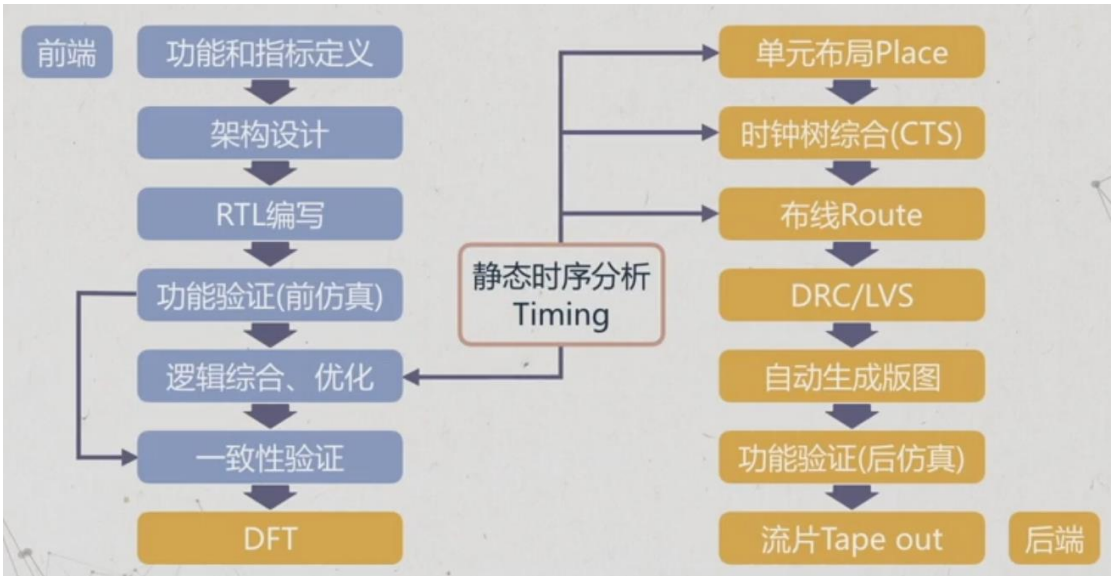
（2） 异步时钟信号的无毛刺切换：与同源时钟不同的是，异步信号还需要加上同步器来减少亚稳态。





二十、数字 IC 设计全流程





1. 确定项目需求

   首先做一款芯片需要有市场，一般公司会先做市场调研，比如最近市面上比较火的人工智能芯片，物联网芯片，5G 芯片，需求量都比较大。有了市场的需求我们就可以设计芯片的 spec 了。先由架构工程师来设计架构，确定芯片的功能，然后用算法进行模拟仿真，最后得出一个可行的芯片设计方案。

2. 前端设计

   （1）RTL 设计：

   RTL（register transfer level）设计：利用硬件描述语言，如 VHDL，Verilog，System Verilog，对电路以寄存器之间的传输为基础进行描述。

   （2）功能仿真：通常是有 DV 工程师来完成这部分工作，通过搭建 test bench，对电路功能进行验证。

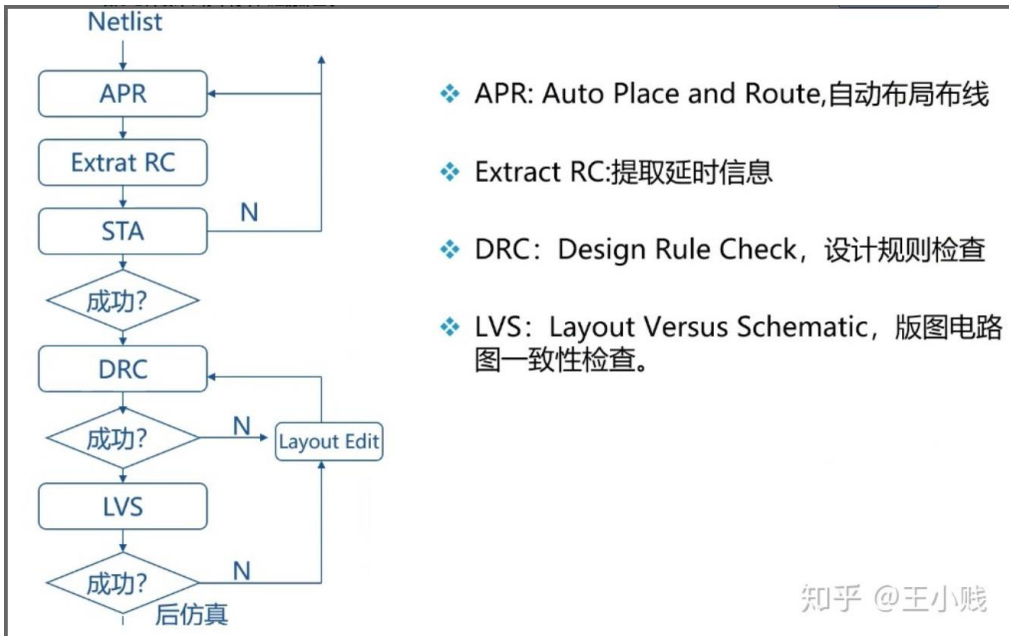
（3）逻辑综合：逻辑综合是将电路的行为级描述，特别是 RTL 级描述转化成为门级表达的过程。也就是将代码翻译成各种实际的元器件。DC compiler

（4）STA：（static timing analysis) 静态时序分析，也就是套用特定的时序模型，针对特定电路分析其是否违反设计者给定的时序限制。Prime time

整个 IC 设计流程都是一个迭代的过程，每一步如果不能满足要求，都要重复之前的过程，直至满足要求为止，才能进行下一步。

3　后端设计

一张图帮你了解后端知识（APR 布局布线—>Extrat RC 延时信息提取—>DRC 设计规则检查—>LVS 版图电路图一致性检查）



二十一、协议与接口

1、 DDR

2、 PCIE

3、 SPI

4、 IIC

二十二、FPGA 的基本结构与内部资源

1.目前绝大多数的 FPGA 是用 SRAM 工艺的查找表组成的结构，其内部资源主要包含有：可编程输入/输出块（IOB）可配置逻辑块(CLB)、嵌入式块 RAM(BRAM)、丰富的布线资源、底层内嵌功能资源、内嵌专用硬核资源等。

(1)、可编程输入/输出块：为了便于管理和适应多种电器标准，FPGA 的 IOB 被划分为若干个组（bank），每个 bank 的接口标准由其接口电压 VCCO 决定，一个 bank 只能有一种 VCCO，但不同 bank 的 VCCO 可以不同。只有相同电气标准的端口才能连接在一起，VCCO 电压相同是接口标准的基本条件。

(2)、可配置逻辑块：由查找表和可编程寄存器组成，查找表完成纯组合逻辑功能，内部寄存器可配置成触发器或锁存器，1 个 CLB 由两个 SLICE 组成，1 个 SLICE 由 4 个 5 输入

LUT，3 个 MUX，1 个 CARRY 和 8 个 FF 组成，CLB 是 FPGA 的精华所在，也是 FPGA 容量的主要代表。

(3)、嵌入式块 RAM：可以配置成单端口 RAM、双端口 RAM、内容地址存储器(CAM)以及 FIFO 等常用存储结构。

(4)、丰富的布线资源：布线资源连通 FPGA 内部的所有单元，而连线的长度和工艺决定着信号在连线上的驱动能力和传输速度。主要分为四类：全局布线资源、长线资源、短线资源、分布式布线资源。

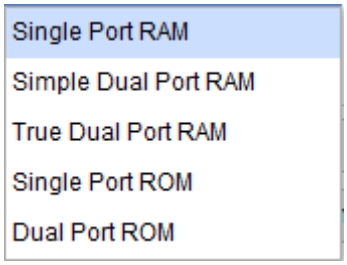
(5)、底层内嵌的功能单元：主要包括 DLL、PLL、DSP、CPU 等，现在越来越丰富的内嵌的功能单元，使得 FPGA 成为了系统级的设计工具，使其具备了软硬件联合设计的能力，逐步向 SOC 平台过渡。

(6)、内嵌专用硬核资源：内嵌的专用硬核是相对底层软核而言的，指 FPGA 处理能力强大的硬核，等效于 ASIC 电路。主要有乘法器、串并收发器、PCI-E、以太网控制器等。

二十三、RAM 资源与类型

1. 按照端口划分

根据 Xilinx 的 BRAM IP 划分为两大类：单端口和双端口 RAM（ROM 只能进行读操作，比 RAM 简单，不作介绍）



（1） 单端口 RAM：只有一个端口可以对存储数据进行读写访问，其中读写方式主要通过写使能（wea）来控制读写方式；

（2） 双端口 RAM：双口 ram 有两种，一种是简单双口 ram（simple Dual port），一种是真双口 ram（true Dual port），

简单双口 RAM 主要有两个端口，一个用于读，一个用于写，读写分开，互不干扰。

真双口 RAM 有两组读写端口，且两个端口都能同时对一个 memory 进行读写操作，但是为了避免发生冲突，通常两组端口不能同时对相同地址进行写操作。

2. 按照资源划分

在 FPGA 中，可以用作 RAM 的资源主要有两个：嵌入式块 RAM（block ram）和 SLICEM 中的 LUT（分布式 RAM），两者之间的不同点在于，BRAM 不能异步读取（即输入输出均在时钟上升沿进行），而 DRAM 可以（只要给出地址，即可输出数据）；但是 BRAM 可以作为真双口 RAM，而 DRAM 不可以。所以在设计上一般遵循一个原则，当用于较大的存储应用时，用 BRAM，而用于灵活的小型存储时，一般用分布式 RAM。

（1）Block ram 由一定数量固定大小的存储块构成的，使用 BLOCK RAM 资源不占用额外的逻辑资源，并且速度快。但是使用的时候消耗的 BLOCK RAM 资源是其块大小的整数倍。如 Xilinx 公司的结构中每个 BRAM 有 36Kbit 的容量，既可以作为一个 36Kbit 的存储器使用，也可以拆分为两个独立的 18Kbit 存储器使用。反过来相邻两个 BRAM 可以结合起来实现 72Kbit 存储器，而且不消耗额外的逻辑资源。

（2）只有成为 SLICEM 的逻辑块里的查找表才可以用做分布式 RAM。利用查找表为电路实现存储器，既可以实现芯片内部存储，又能提高资源利用率。分布式 RAM 的特点是可

以实现 BRAM 不能实现的异步访问。不过使用分布式 RAM 实现大规模的存储器会占用大量的 LUT，可用来实现逻辑的查找表就会减少。因此建议仅在需要小规模存储器时，使用这种分布式 RAM。

二十四、逻辑综合

二十五、时钟树综合

二十六、Verilog 不常见语法汇总

1.`ifdef……`endif 常用于 Verilog 中条件编译，可以出现在设计中的任何地方，也可以进行相互嵌套（`ifdef…`elsif…`elsif…else…`endif）。其通常和预编译指令`define（宏定义）配套使用，其功能主要是向编译器说明：如果使用`define 定义了称为‘FLAG’的宏，那么关键字`ifdef 会告诉编译器包含这段代码，直到下一个`else 或`endif。而`ifdef 只是告诉编译器，如果给定的名为 FLAG 的宏没有使用`define 指令，则将这段代码包含在下一个`else 或者`endif 之前（与`ifdef 恰恰相反）。

二十七、TCL 语言

1. 三种 TCL 语言：TCL 本身的，synopsis TCL 语言，用户自定义 TCL 语言

二十八、AHB 和 AXI 总线之间的异同

1. 相同点：都是高性能、高带宽传输的数据总线，都支持乱序发送，猝发传输，分割传输，流水线传输。
2. 不同点：
   （1） AXI 总线的读写通道是分开的，所以支持同时读写操作，而 AHB 总线没有分开的读写通道，所以只能分开读写；
   （2） AXI 总线支持非对齐操作，而 AHB 总线不支持；
   （3） AHB 一次突发传输必须在前次传输完成后才能进行，但是 AXI 总线只需要一次突发的首地址，可以连续发送多个突发传输首地址而无需等待前次突发传输完成，这样大大提高了总线的利用效率。