评论（1）

跨时钟域

简介

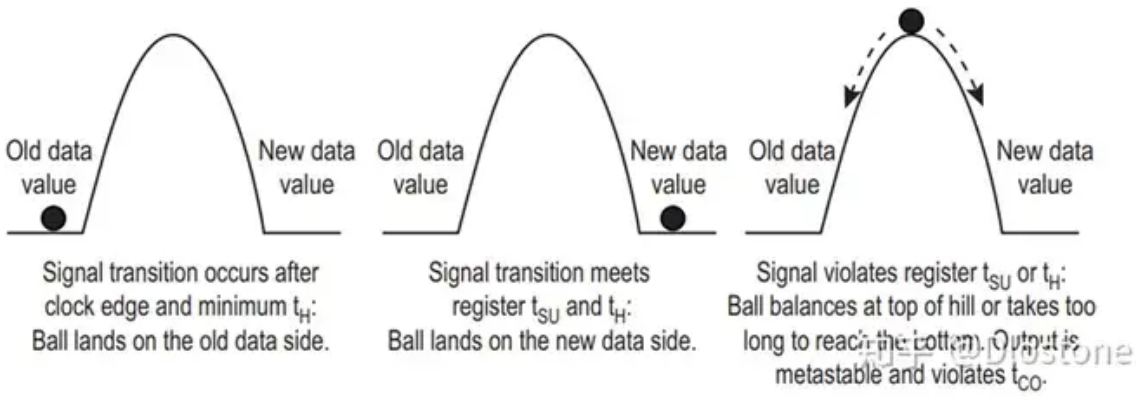
一个时钟源就是一个时钟域。

如果所有的时钟都来自同一个时钟源，那么时钟的频率和相位就是可知的，否则会出现同频不同相或者不同频的问题。

异步时序设计指的是在设计中有**两个或以上的时钟**， 且时**钟之间是同频不同相或不同频率**的关系。而异步时序设计的关键就是把数据或控制信号正确地进行跨时钟域传输。

亚稳态

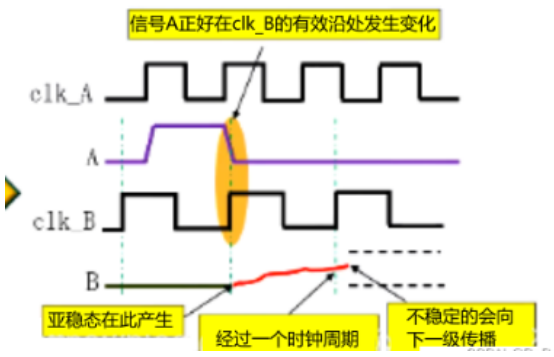
每一个**触发器**都有其规定的**建立(setup)和保持(hold)时间参数**, 在这个时间参数内， **输入信号在时钟的上升沿是不允许发生变的**。 如果在信号的建立时间中对其进行采样， 得到的结果将是不可预知的，即亚稳态。

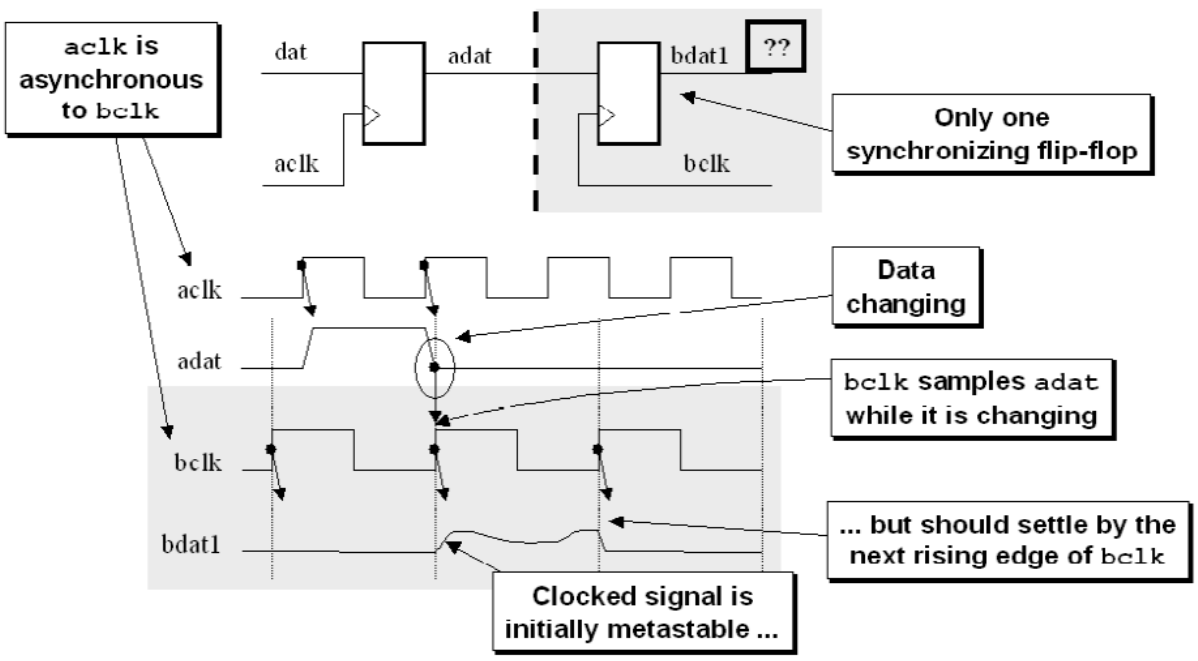


建立时间（setup time）：指触发器的时钟信号**动作沿到来以前**，数据稳定不变的时间。

保持时间（hold time）：指触发器的时钟信号**上升沿到来以后**，数据稳定不变的时间。

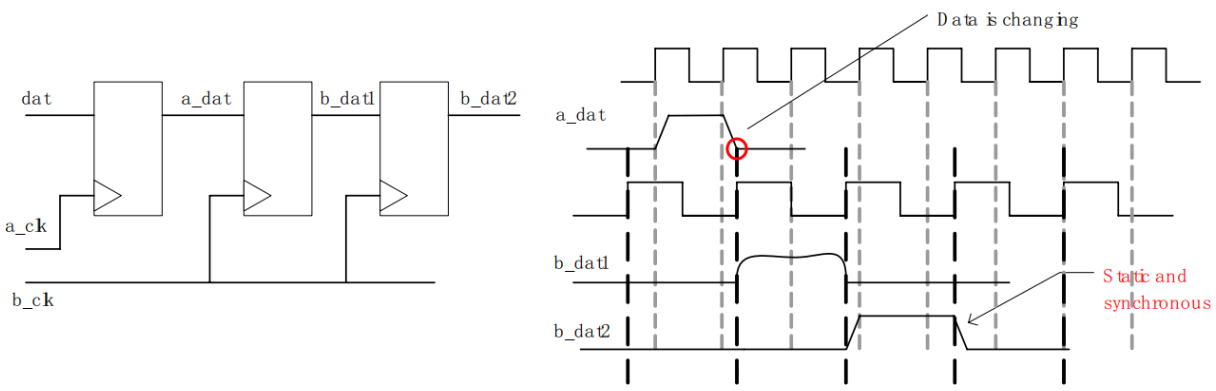
触发器的输出变化相比输入延迟了一个时钟周期，即“**打一拍**”。





单比特策略

双锁存器



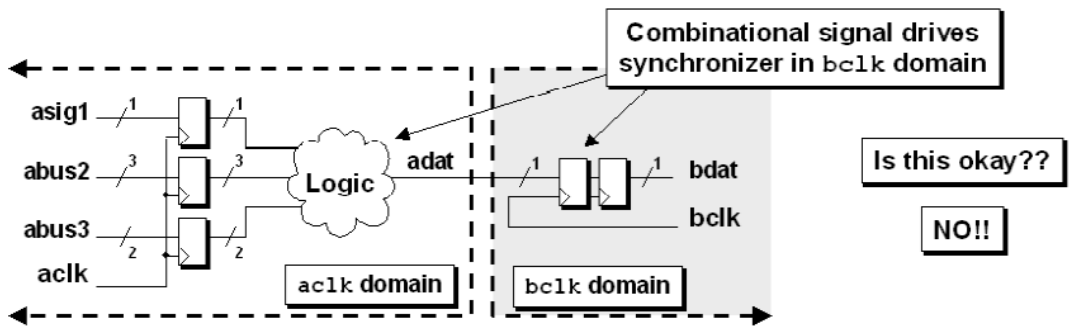
3-1 双锁存器法解决亚稳态问题

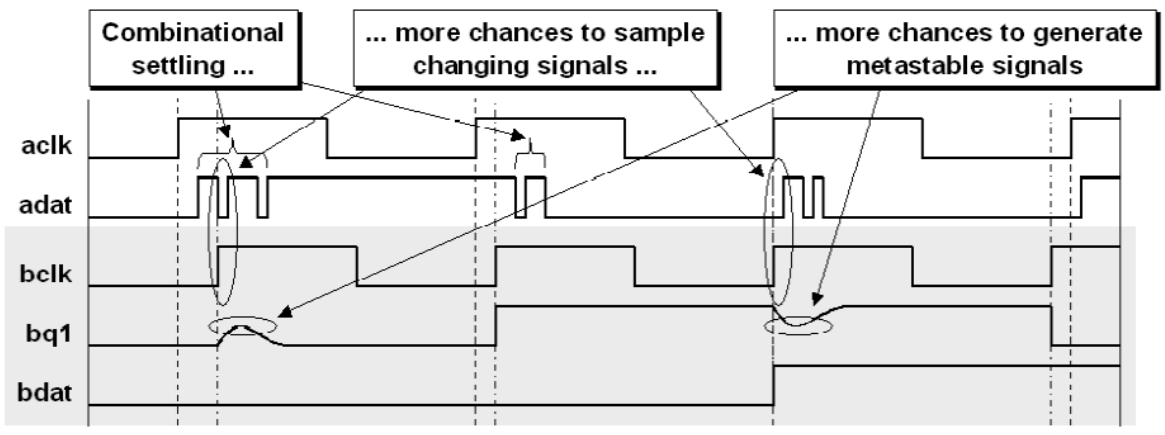
通过使用两个锁存器，就可以避免亚稳态的发生了。

存在的问题

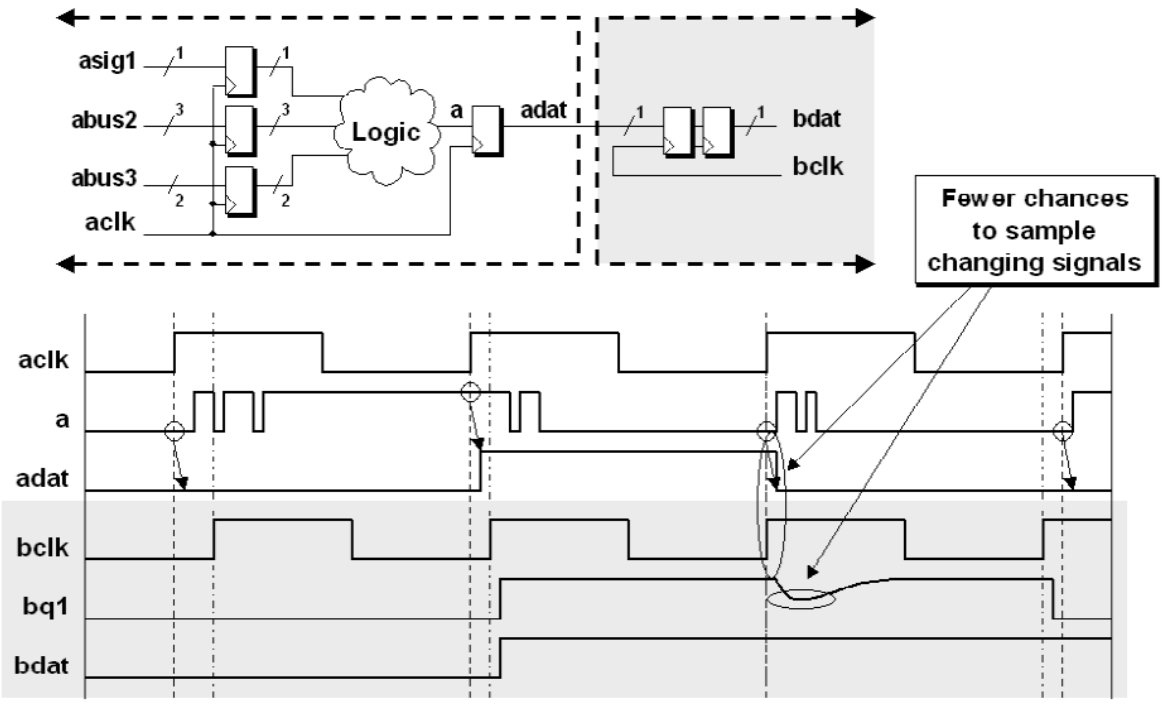
1）时钟域a与时钟域b之间不能存在组合逻辑。

因为组合逻辑的输出信号会产生毛刺（由于各路信号的到达时间不同等），因此如果直接将组合逻辑的输出作为时钟域b的输入，那么在我们无法确定在b时钟的上升沿是否可以精确的采集到a的每一个变化，具体错误如下图所示





因此如果想要b得到稳定的信号，就需要先用a时钟来稳定组合逻辑的输出，如下图所示

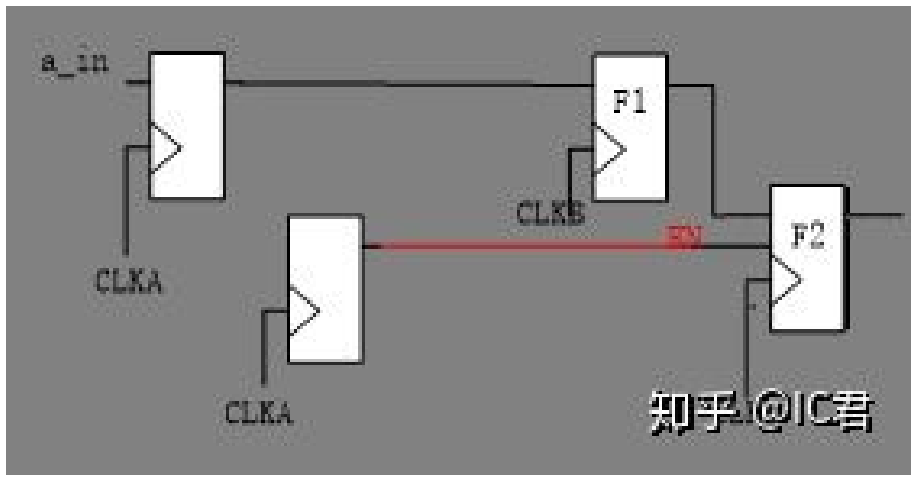


[图片]

张艺骞 5月2日 20:13

b_dat2 采第三条虚线的样时会不会带来亚稳态问题呢?

2）clock-gating enable信号没有经过异步处理



由于F2的使能是由CLKA控制的，无法保证在该信号的使能之后，CLKB的setup time和hold time仍然能满足，因此对clock gating信号也要做特殊处理。

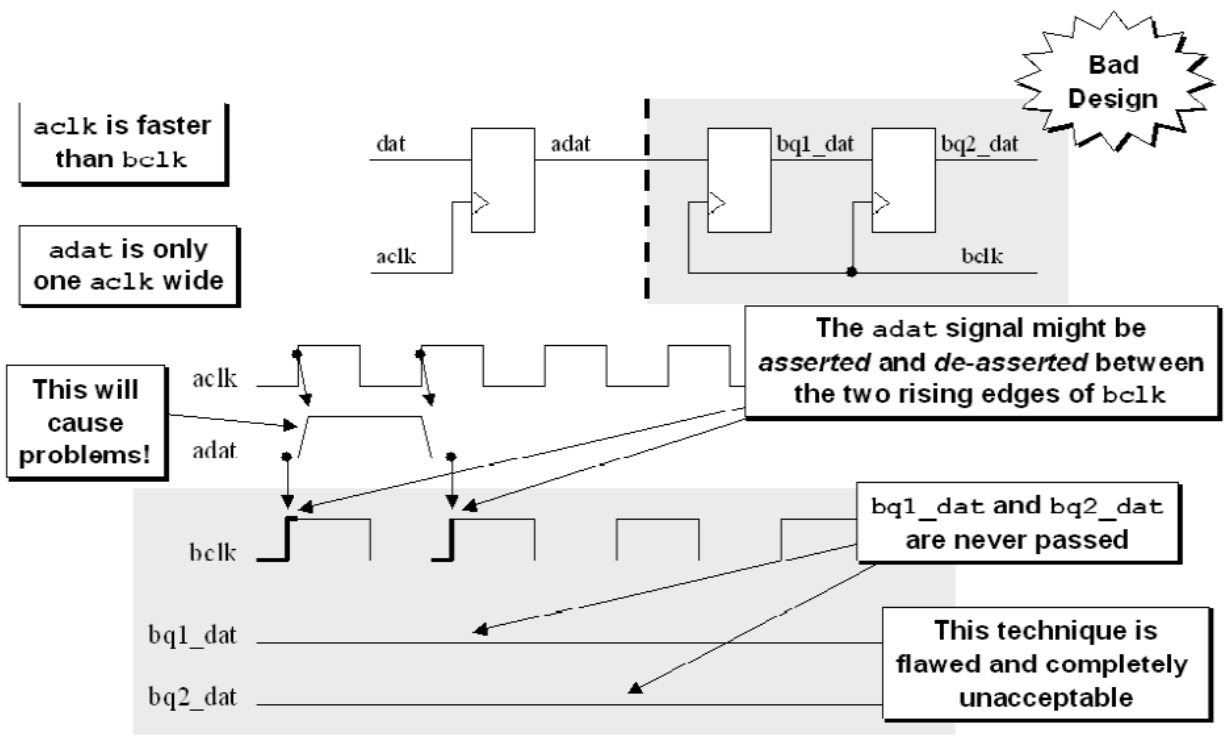
注：clock gating（时钟门控），是一种低功耗技术，用于节约电能。其中一种方法为通过寄存器实现使能信号，从而在**逻辑综合过程**自动被翻译为时钟门控。

3）慢采快问题

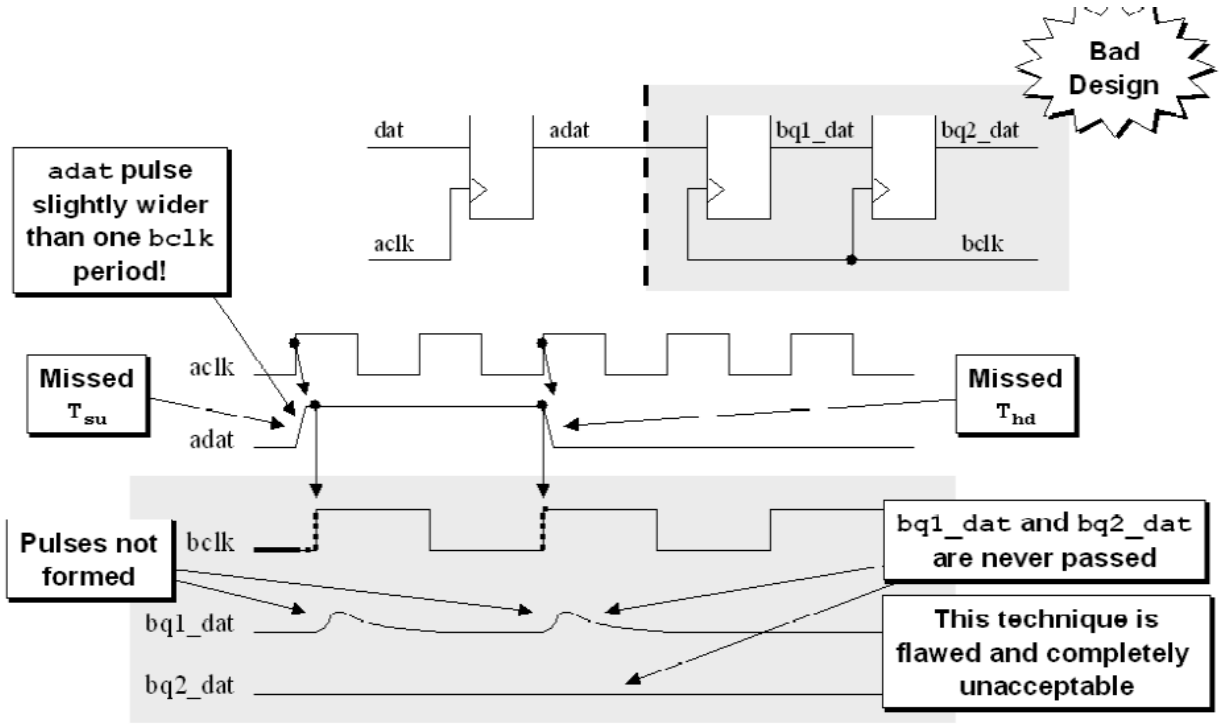
快采慢：如果较快的时钟域是较慢时钟域频率的1.5倍(或更多)，那么将较慢的控制信号同步到较快的时钟域通常不是问题，因为较快的时钟信号将采样较慢的CDC信号一次或多次。因此将快采慢比慢采快引起的潜在问题更少，所以对于快采慢的情况，使用简单的两个触发器同步器在时钟域之间传递单个Clock Domain Crossing (CDC)信号即可。（**直接后面用2个触发器即可**）

慢采快：而对于慢采快的情况则最好更加稳妥些，一般要求在**接收时钟域**中**采样信号**要保持**三个时钟边**沿的时间（"three edge" requirement），也就是1.5倍的采样时钟周期。一般来说1.25倍也够。

如果**采样信号维持时间过短**，则慢时钟域很可能会**漏采**：



即使采样数据维持时间**略长于**采样周期，同样会出现**漏采**的情况：



解决方法

1. 人为的控制被采样数据的维持时间，确保采样时钟可以采样到数据（即"three edge" requirement）
2. 通过反馈自动延长数据维持时间（不推荐）

多比特策略

解决思路主要有以下三点：

- 尽可能将这些信号合并成单bit
- Multi-cycle path （MCP). 使用同步信号。
- 使用Gray码

控制信号多比特

load和en信号可能因为发生了偏差（skew），从而永远无法同步上

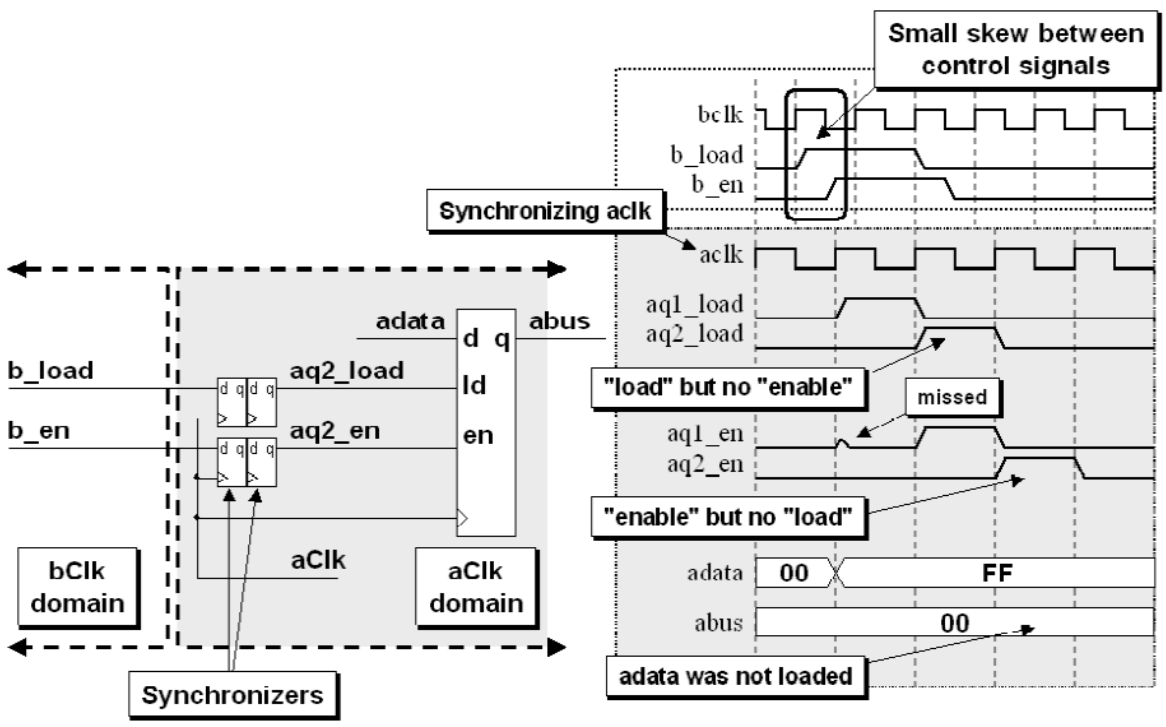


Figure 12 - Problem - Passing multiple control signals between clock domains

解决方法：**共用一个信号**就行了。删除其中一个信号

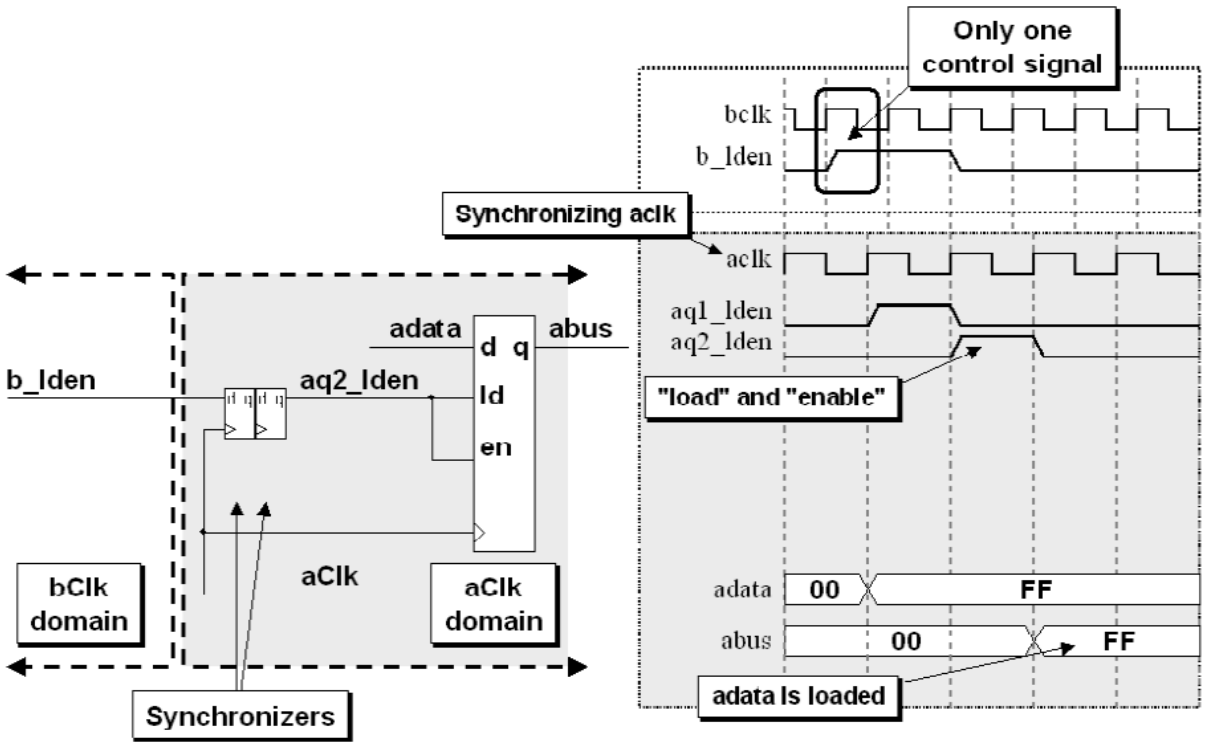
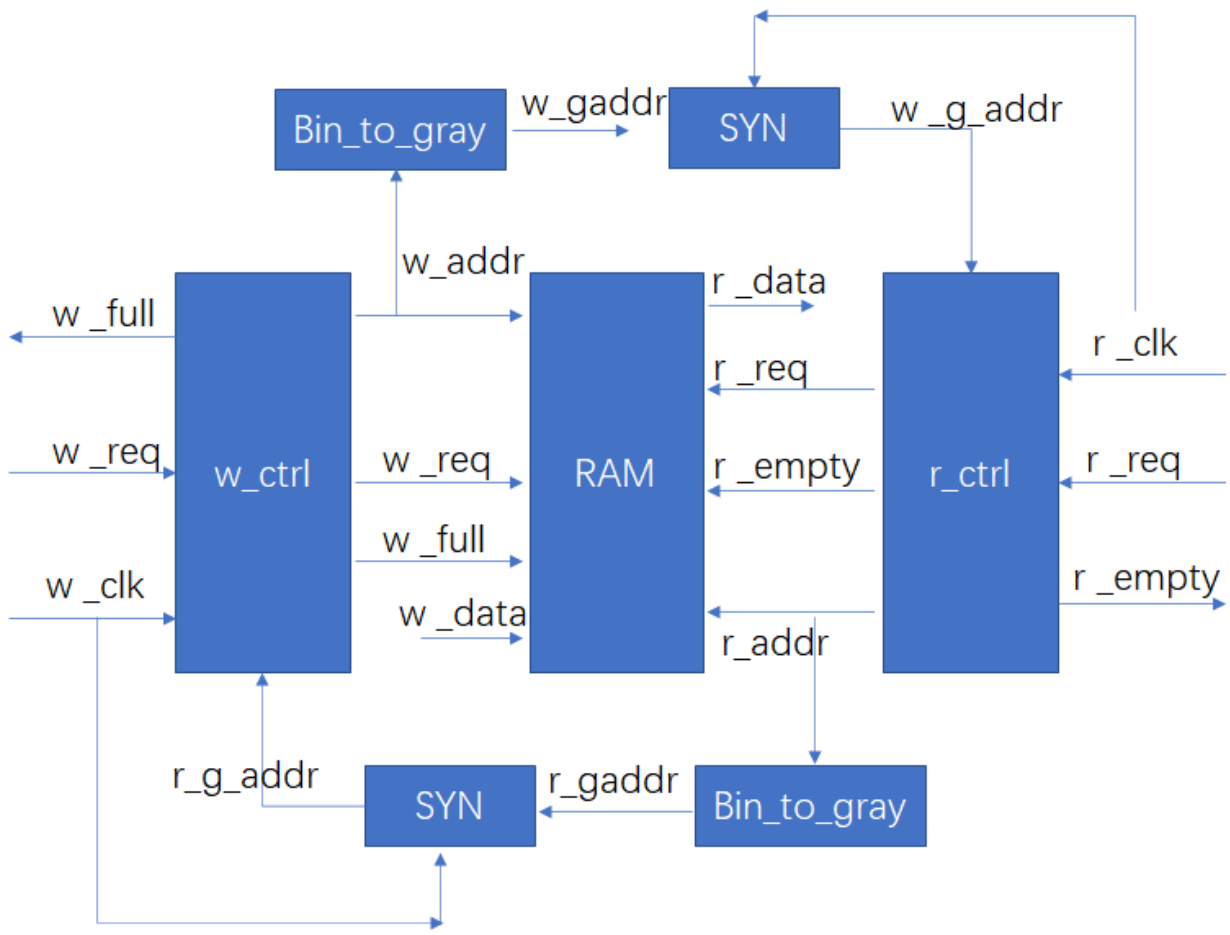


Figure 13 - Solution - Consolidating control signals before passing between clock domains

数据多比特（异步FIFO）

为了进行跨时钟域传输多比特数据，还可以采用异步FIFO/双口RAM

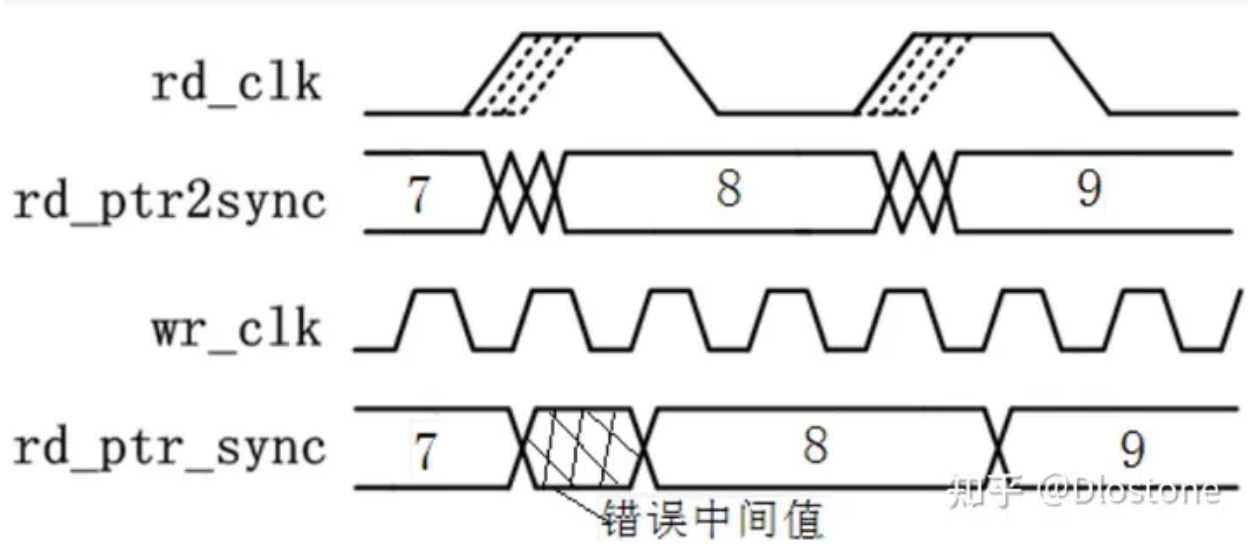
使用场景：在有**大量的数据**需要进行跨时钟域传输， 并且对数据**传输速度**要求比较高的场合 。



其中要解决的一个重要问题是如何判断FIFO内部是full还是empty。也就是比较读指针与写指针。由于读指针与写指针分别处在不同的时钟域下，因此存在一个跨时钟域传输的问题！

那是否可以直接使用2进制的地址进行跨时钟域比较呢？不可以！因为地址的多比特，在地址变化时，会存在多位比特都变化的情况，这样在采样时可能会出现问题，见下图：

如图，当指针从地址7（4‘b0111）跳变至地址8（4’ b1000）时， 4个bit位都会同时发生跳变，当同步时钟在此时采样时，将会采到错误的中间值，由于4个bit位同时跳变，这个中间值可能是任意值（4’ b1111、4’ b0000等），导致出现错误的空满信号。



为了解决这个问题，引入了**格雷码**，这种编码每次**从一个值变化到相邻的一个值**时，**有且仅有一位发生变化**；由于格雷码的这种特性，我们就可以将**多bit指针同步问题转化为单bit指针同步问题**，通过简单的双触发器进行同步操作，而不用担心由于发生亚稳态而出现数据错误中间值的情形。

特别注意，格雷码指针**适用于地址范围空间为2的整数次幂的FIFO**，如果FIFO深度不是2的整数次幂，地址指针转换为格雷码后，格雷码表示的最小地址与最大地址之间不是仅有一bit不同，这违背了我们选用格雷码的初衷。

二进制格雷码互相转换方法

二进制码转换为格雷码：

- 从最高有效位（MSB）开始，将第一个二进制位作为格雷码的第一个位。
- 从第二位开始，每一位的格雷码都是该位的二进制码与前一位的格雷码按位异或（XOR）得到的结果。

例如，将二进制码转换为格雷码的过程如下：

二进制码：1010　格雷码： 1111

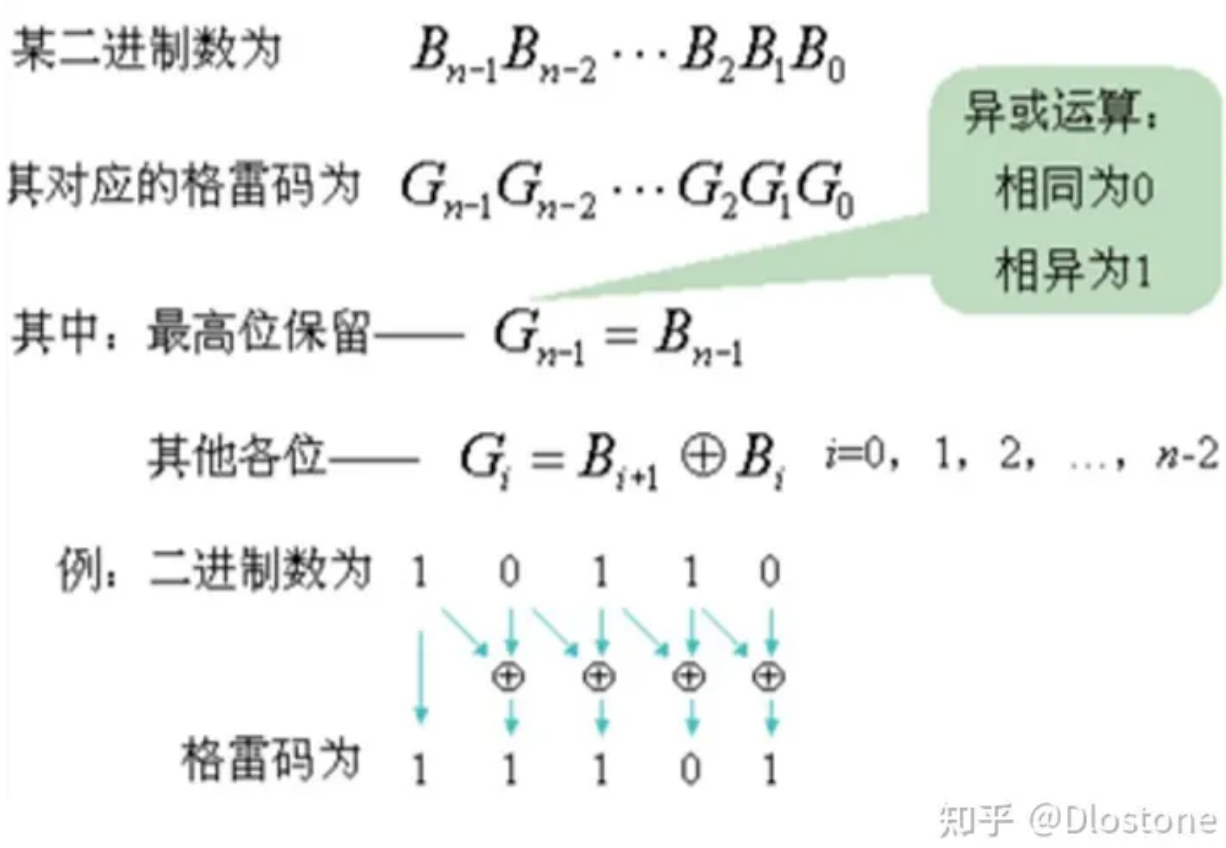
解释：（从高到低分别是第一位第二位，以此类推）

第一位：格雷码的第一位与二进制码的第一位相同。

第二位：格雷码的第二位是第二位二进制码（0）与前一位格雷码（1）异或得到的结果，即 0 XOR 1 = 1。

第三位：格雷码的第三位是第三位二进制码（1）与前一位格雷码（1）异或得到的结果，即 1 XOR 1 = 0。

第四位：格雷码的第四位是第四位二进制码（0）与前一位格雷码（0）异或得到的结果，即 0 XOR 0 = 0。



格雷码转换为二进制码：

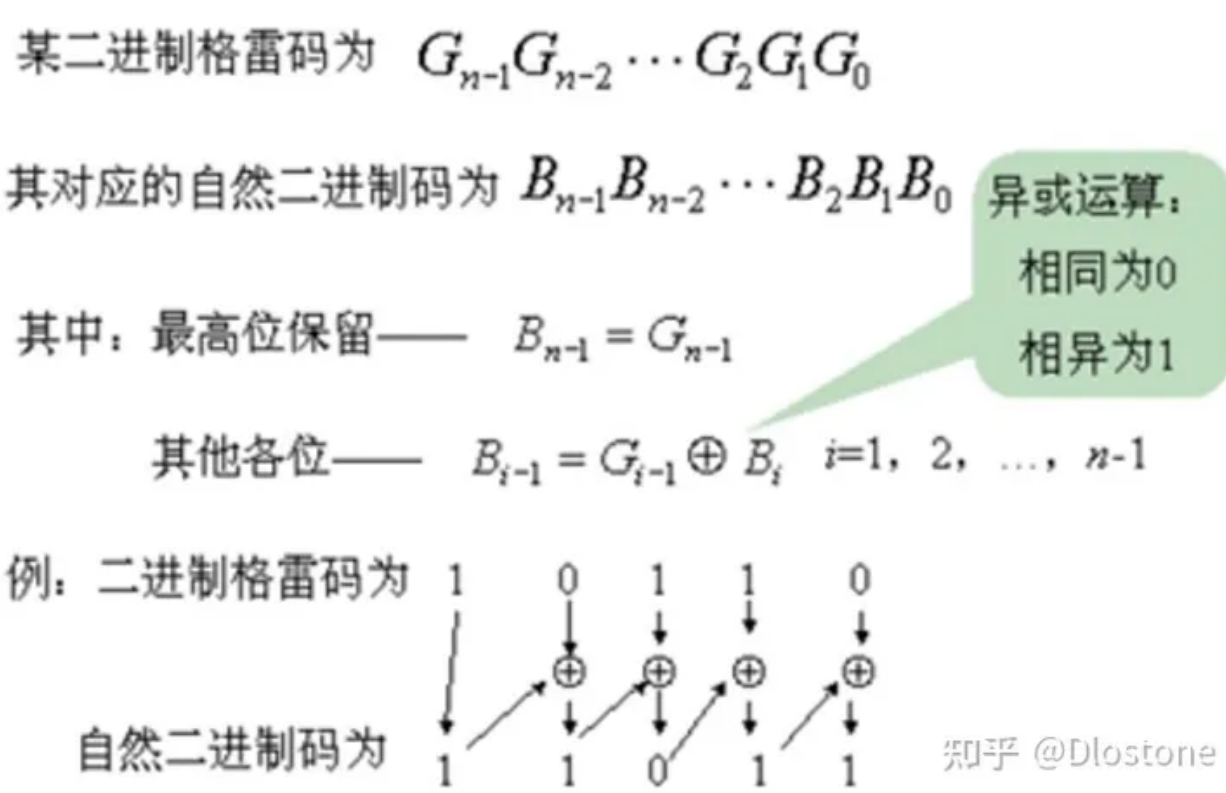
- 从最高有效位（MSB）开始，将第一个格雷码位作为二进制码的第一个位。
- 从第二位开始，每一位的二进制码都是该位的格雷码与前一位格雷码按位异或（XOR）得到的结果。

例如，将格雷码转换为二进制码的过程如下：

格雷码： 1111 二进制码：1010

解释：

第一位：二进制码的第一位与格雷码的第一位相同。
第二位：二进制码的第二位是第二位格雷码（1）与前一位的格雷码（1）异或得到的结果，即 1 XOR 1 = 0。
第三位：二进制码的第三位是第三位格雷码（1）与前一位的格雷码（0）异或得到的结果，即 1 XOR 0 = 1。
第四位：二进制码的第四位是第四位格雷码（1）与前一位的格雷码（1）异或得到的结果，即 1 XOR 1 = 0。



数据多比特（2深度异步FIFO）

把FIFO深度设置成2，这样地址的位宽只有1位，就不需要格雷码了。但深度太少，使用场景不好评价

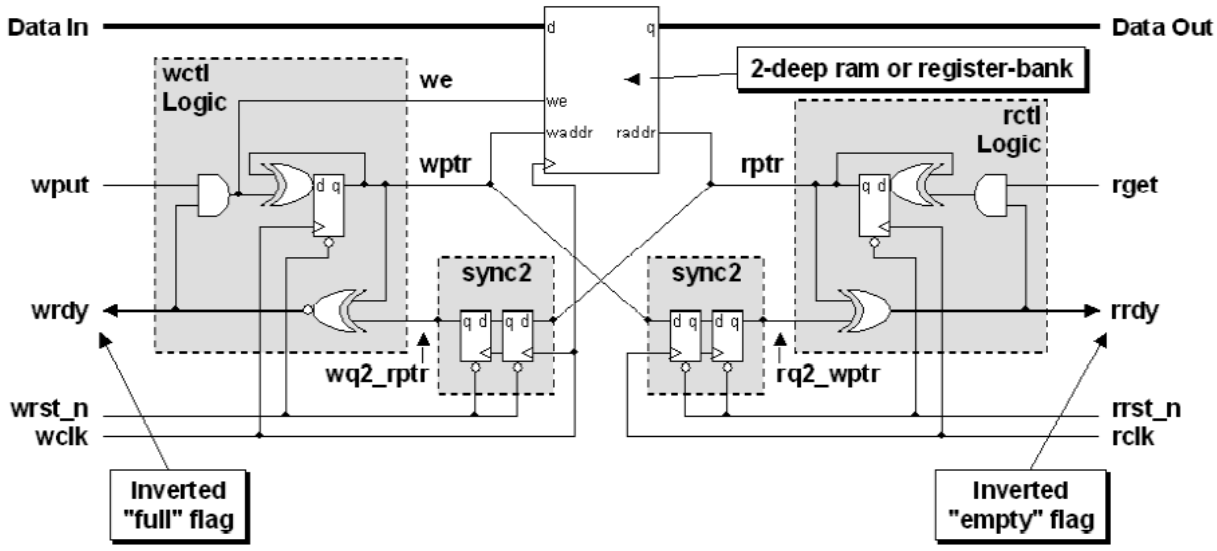


Figure 29 - 1-deep / 2-register FIFO synchronizer block diagram