跨时钟域

4.1 跨时钟域处理（20160620）

时钟对于 FPGA 就像我们的心脏，时刻控制着“跳动”的频率以及“血液”的流速；时钟域好比通过心脏的血液血型，不同血型的血液会产生排斥作用。在设计中建议时钟越少越好，好比于人有两个甚至更多的心脏，其内脏工作将会多么混乱。

但是某些情况下多时钟又不可避免，比如从 FPGA 外部输入的数据，其自带有个随路时钟，数据终归要在 FPGA 内部时钟域下处理，这来自外部的“血液”如何处理才能与内部的“血液”融合呢？配对及转换工作则是必不可少的，这就引入本节的主题：跨时钟域处理（Clock Domain Crossing）：

跨时钟域处理需要两方面的工作：1、设计者处理；2、 FPGA 工具（Vivado）处理。

1. 设计者处理

首先讲解一下如果不进行跨时钟域处理，会出现什么问题呢？如图1所示路径，QA 属于 CLKA 时钟域的数据输出，另一个时钟 CLKB 去捕获节点 REG A 的输出 QA，假定 CLKA 与 CLKB 是异步时钟，它们之间的相位并不固定，因此捕获过程中可能会出现建立冲突（setup violation）和保持冲突（hold violation），如图2所示，左右分别为发生建立冲突和保持冲突的情况。

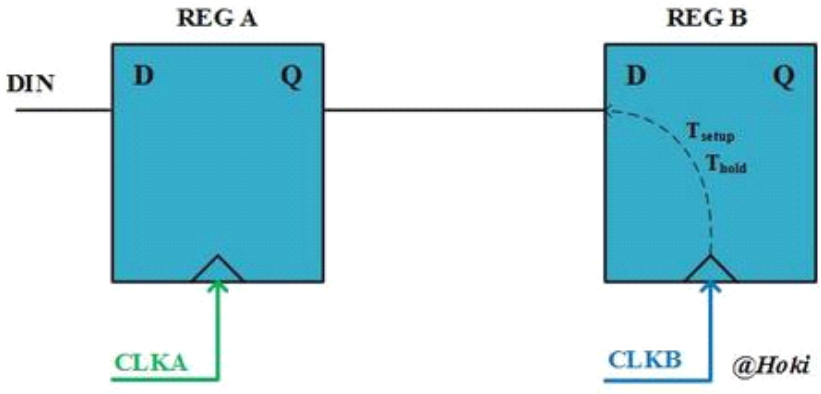


图1

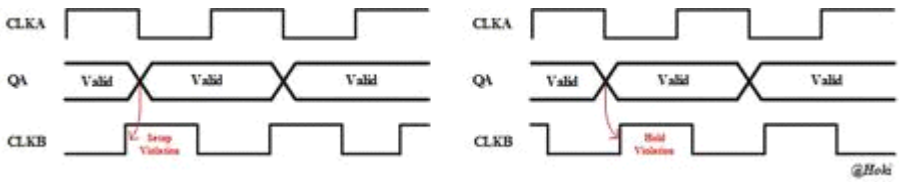


图2

当冲突出现时（我感觉整个人都不好了），会发生什么事情呢？在发生建立冲突或者保持冲突，捕获节点（REG B）会处于一个不定的状态，正常的状态是高电平或者低电平，而此时的状态停留在高电平和低电平的中间，无效的电平 X，称这个状态为**亚稳态**。

如图3所示，捕获节点输出保持在亚稳态，可能在整个时钟周期内都保持在亚稳态，由于不正确的状态，其后连接的逻辑在功能实现上就会出现问题，比如一个判断信号上升沿的逻辑，通常判断 D == HIGH && D_PREV == LOW（D 为信号当前电平状态，D_PREV 为信号上个时钟的电平状态）是否成立，而发生亚稳态时则 D_PREV == X，这个上升沿将会错过。因此，加入跨时钟域处理设计是必须的。

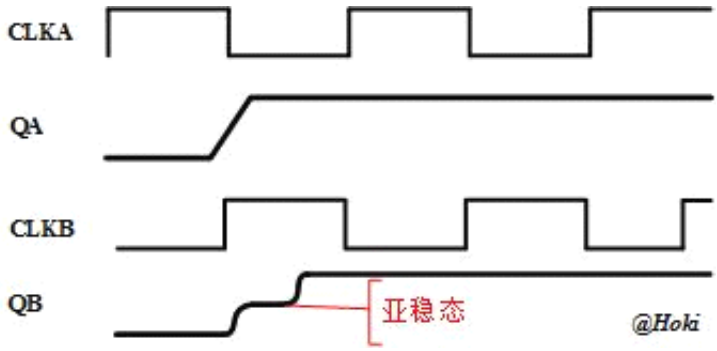


图3

对于**单比特信号**的跨时钟域处理，常用的方法是“打两拍”，即在捕获时钟域中加入两个寄存器进行时钟转换，如图4所示，加入 REG B1和 REG B2，虽然 REG B1处于亚稳态状态。

但是 REG B2的输出 QB2能稳定在正常的电平上，由于 REG B1和 REG B2之间没有多余的逻辑，REG B1能有充裕的时间稳定状态，此情况下 REG B2能完美地隐藏 REG B1的亚稳态。在捕获时钟的频率比较高的情况下，如果一个 REG B2还未能隐藏亚稳态，拍数也可以增加三个或者更多，当然一般情况下，两拍足矣。

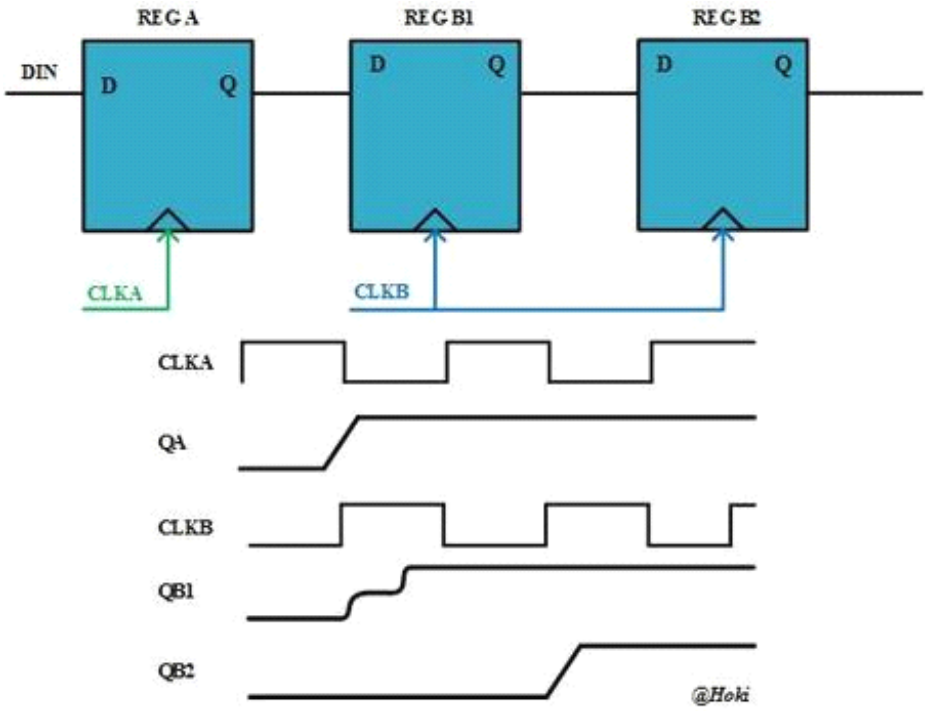


图4

对于**多比特总线数据**的跨时钟域处理，能否也使用“打两拍”的方法呢？答案见图5，虽然 REG B2的输出是稳定的，稳定在哪一个电平是不确定的，不过会在当前时钟或者下一个时钟输出正确电平，即偏差在一个时钟周期，也就是说不能保证所有比特位的状态一致，也是这个原因，导致传输多比特总线数据时各比特位不同步，常用的解决方法是加入 FIFO 隔离，如图6所示，FIFO 能有效地隔离两个时钟域，避免亚稳态的发生。

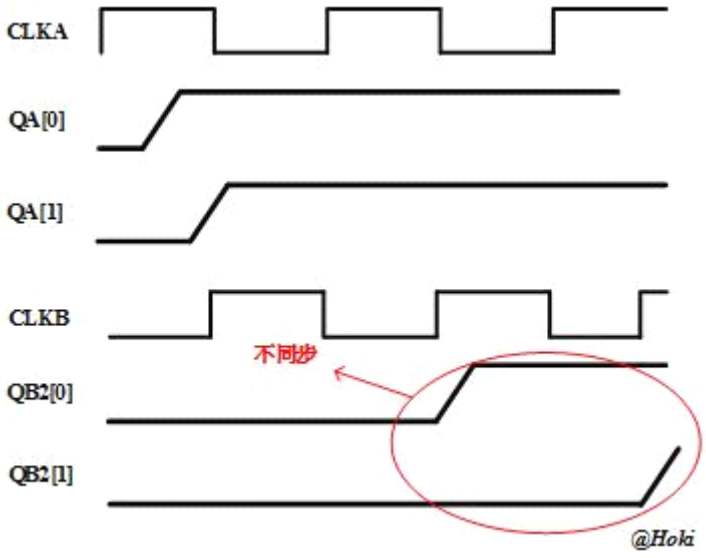


图5

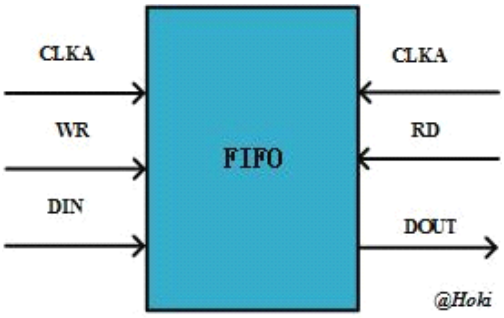


图6

2. FPGA 工具（Vivado）处理

Vivado 采用 XDC 对时序进行约束，默认情况下，会分析所有时钟的路径，当然也包括跨时钟域的路径。

设计经过综合实现后，在 Implementation 中点击 Report Clock Interaction（见图7），得到设计中所有时钟的交互情况，如图8所示，共有两个时钟：CLK_REG 和 CLK_USR，红色区域表示 Timed（unsafe），说明两个时钟间有时序路径。

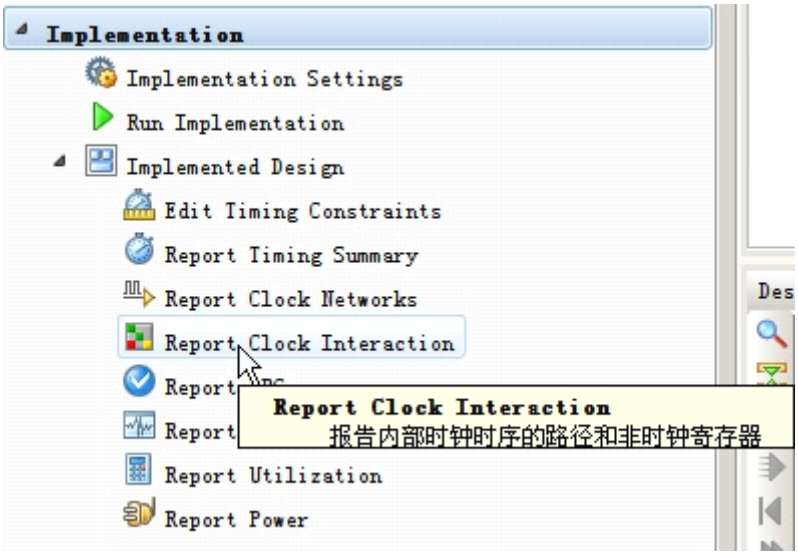
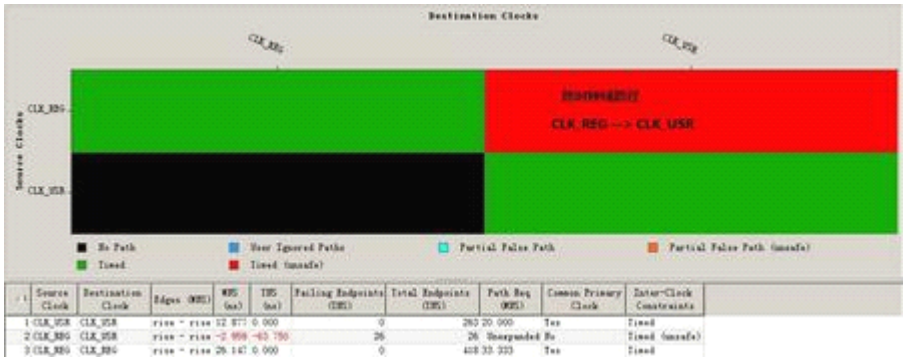


图7



可以发现在图8底部时序分析的结果，有红色报警说明有路径时序未收敛，打开时序分析报告（见图9），source clock 是 CLK_REG，destination clock 是 CLK_USR，说明是一条跨时钟域路径，其中 Requirement 只有0.001ns，显然对跨时钟域路径的分析不合理，因此通常在保证**设计者处理**完成后，添加时序约束，是 Vivado 忽略对跨时钟域路径进行时序分析。

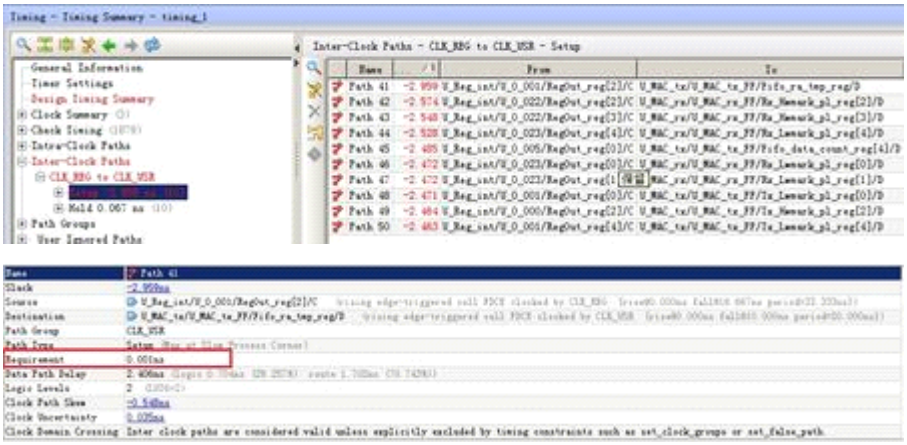


图9

可以通过设置时钟组（clock group）或者设置假路径（false path）处理跨时钟域路径。如图10所示，在 clock interaction 中红色区域右击，选择 Set Clock Groups 或者 Set False Path，

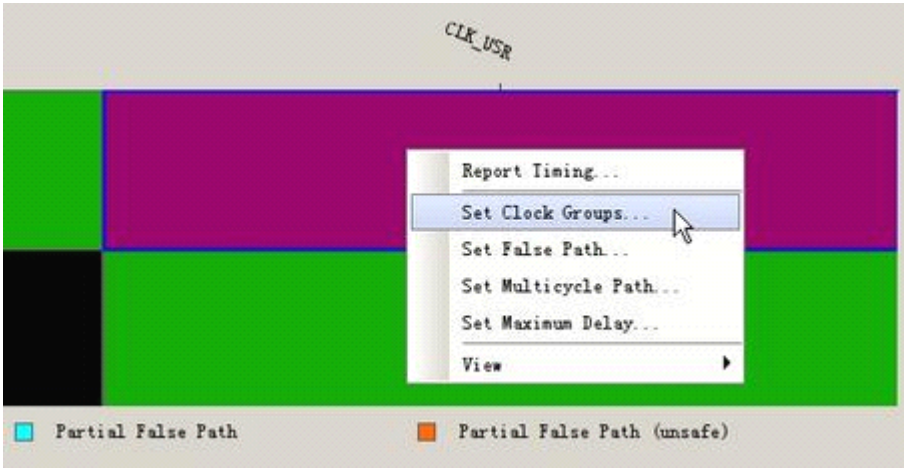


图10

Set Clock Group 和 Set False Path 的区别是，前者设置了双向（CLKA-TO-CLKB 和 CLKB-TO-CLKA）的路径，而后者只设置单向（CLKA-TO-CLKB 或者 CLKB-TO-CLKA）的路径，此例中因为只有单向区域有路径交互，因此使用 Set False Path 即可。

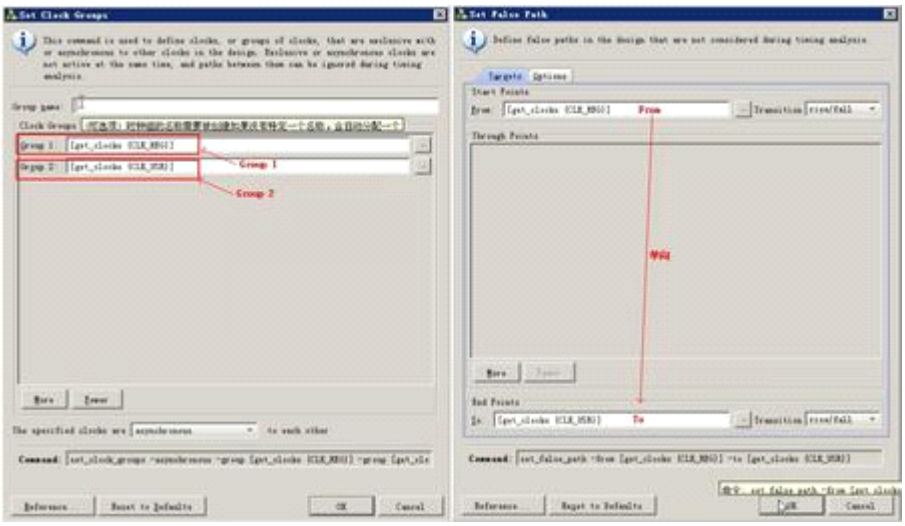


图11

添加完约束后，时序报告中 Inter-Clock Paths 没显示有路径，并且没有未收敛的时序路径，如图12所示，Clock Interaction 中原先红色区域变成了蓝色（User Ignored Paths），可以确认约束生效。

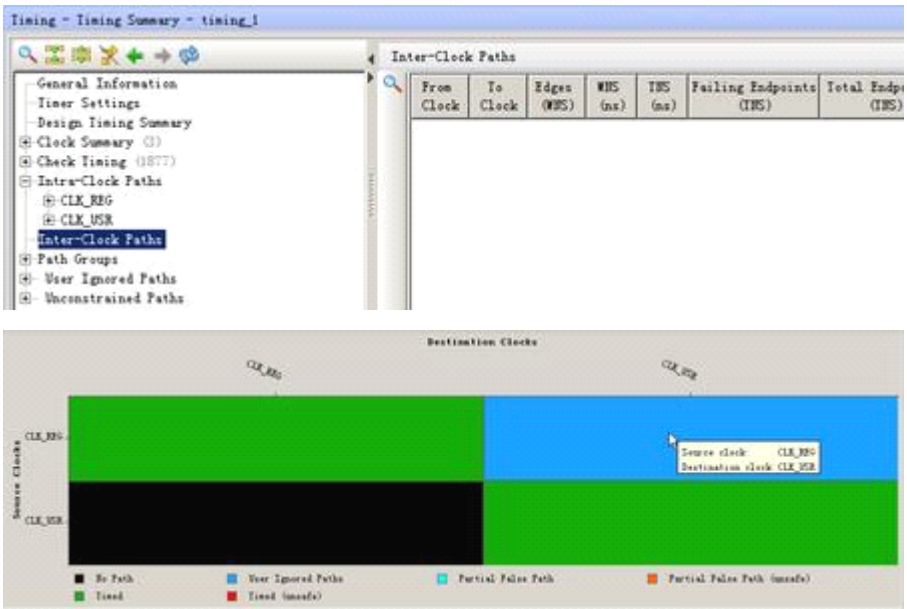


图12

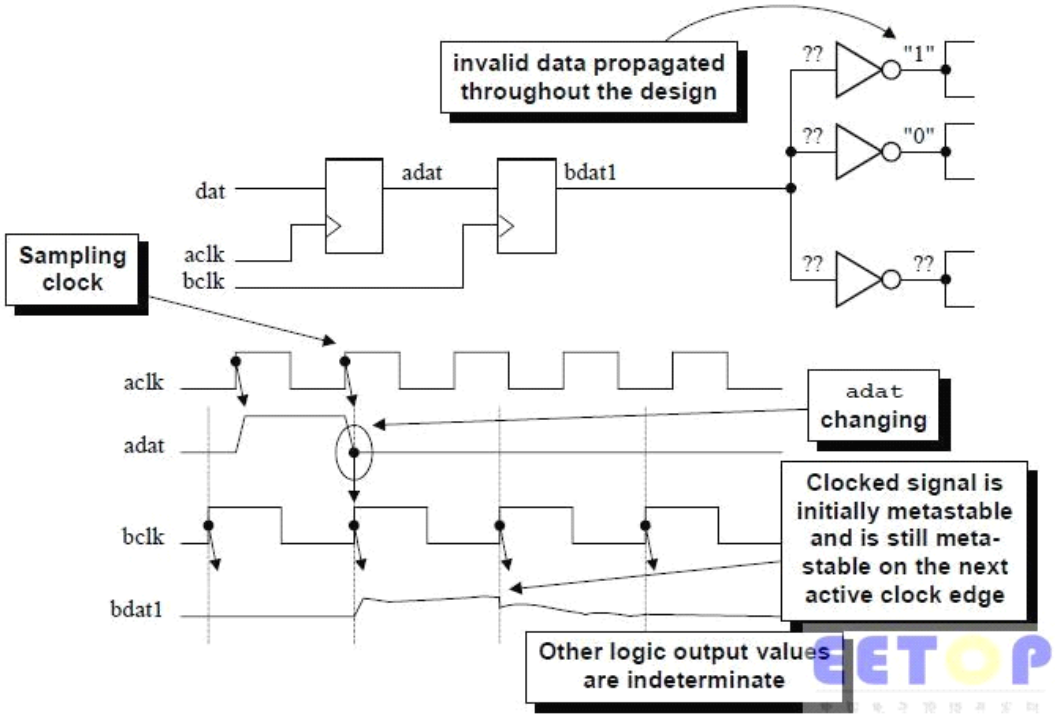
4.2 跨时钟域设计的一点总结（20160620）

1. 亚稳态的概念说明

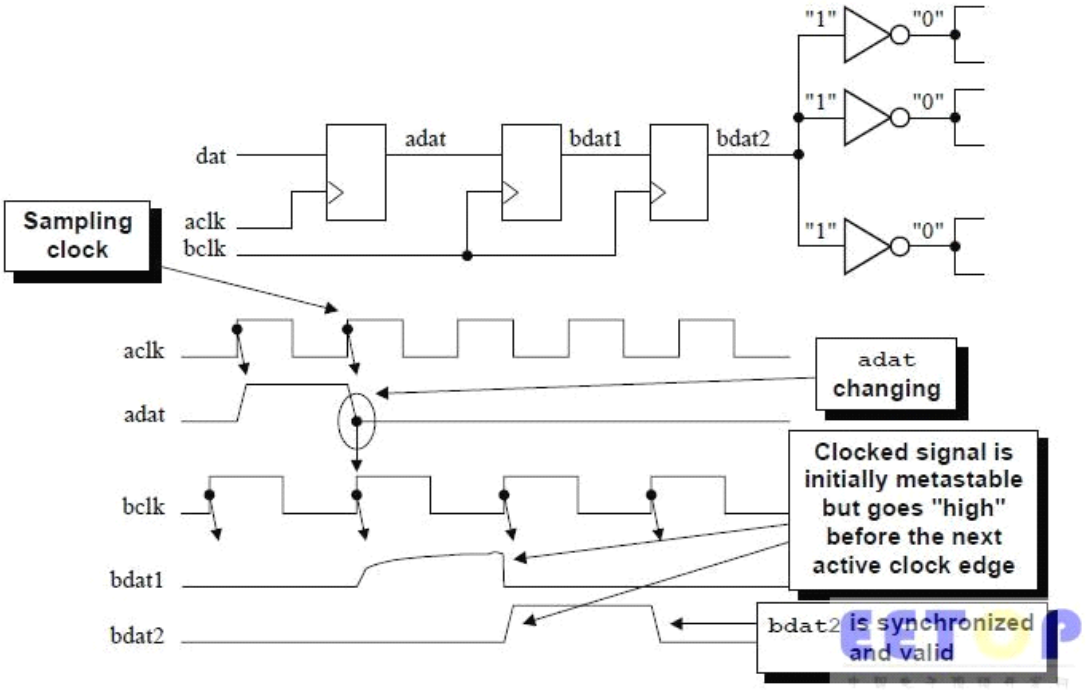
是指触发器无法在某个规定时间段内达到一个可确认的状态。当一个触发器进入亚稳态引时，既无法预测该单元的输出电平，也无法预测何时输出才能稳定在某个正确的电平上。在这个稳定期间，触发器输出一些中间级电平，或者可能处于振荡状态，并且这种无用的输出电平可以沿信号通道上的各个触发器级联式传播下去。

在同步系统中，如果触发器的 setup time / hold time 不满足，就可能产生亚稳态，此时触发器输出端 Q 在有效时钟沿之后比较长的一段时间处于不确定的状态，在这段时间里 Q 端毛刺、振荡、固定的某一电压值，而不是等于数据输入端 D 的值。这段时间称为决断时间（resolution time）。经过 resolution time 之后 Q 端将稳定到0或1上，但是究竟是0还是1，是随机的，与输入没有必然的关系。

在跨时钟域的设计中，由于时钟的不同步，及其容易产生亚稳态。



常用的同步方法是使用两级寄存器来进行同步：



2. 多时钟设计的命名规则

不同时钟域的时钟和模块以及模块输入输出信号命名，应当加上适当的前缀以区分。如控制部分 uclk，视频处理部分 vclk，NVM 接口部分 mclk。这样既便于区分，也便于在后端做时序约束和分析时的处理（比如某些命令的批量处理，set_false_path -from { u* }）。

3. 多时钟设计的模块划分

将整体的多时钟设计分割成多个单个独立时钟的功能模块和负责模块间同步的同步模块，这样非常利于后端的时序分析，程序结构也更加清晰。

4. 快时钟域到慢时钟域的同步

常见问题：快时钟域的信号持续时间太短，慢时钟域采不到

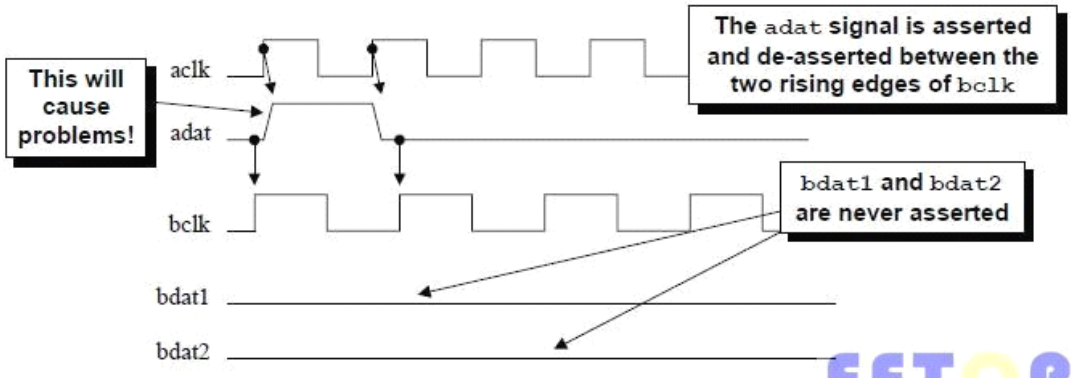


Figure 5 - Short control signal pulse missed during synchronization

解决方法 1： 将快时钟域传递的信号持续时间延长，使其大于慢时钟域的一个时钟周期

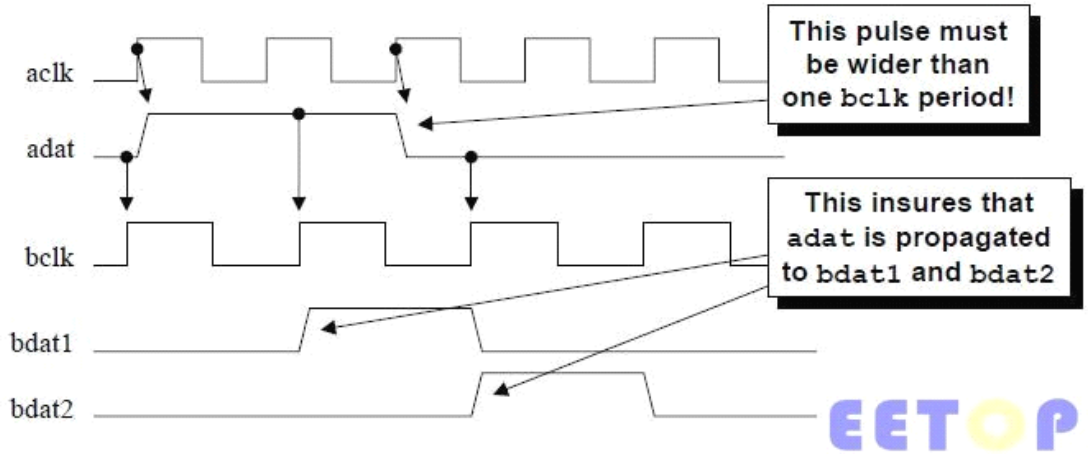


Figure 6 - Lengthened pulse to guarantee that the control signal will be sampled

解决方法 2：使用握手信号（反馈信号）

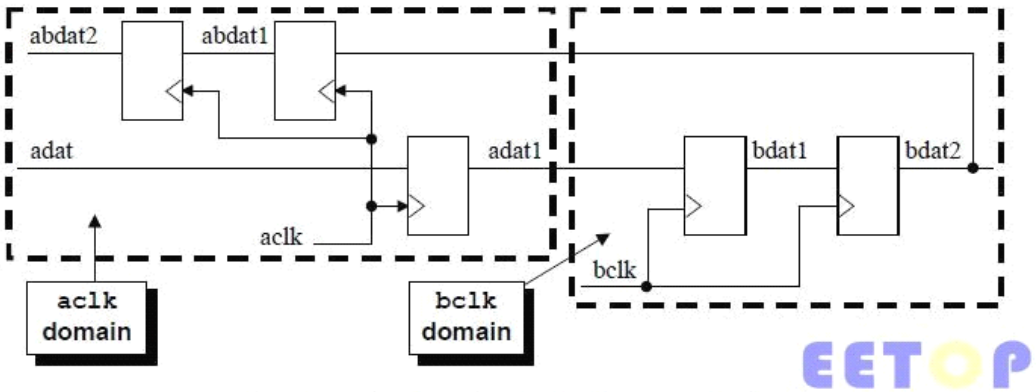


Figure 7 - Feedback synchronization of a control signal

5. 跨时钟域传递多个相关信号(如 enable 和 load 信号)

5.1 问题 1： 传递两个同时需要的信号(b_load 和 b_en)

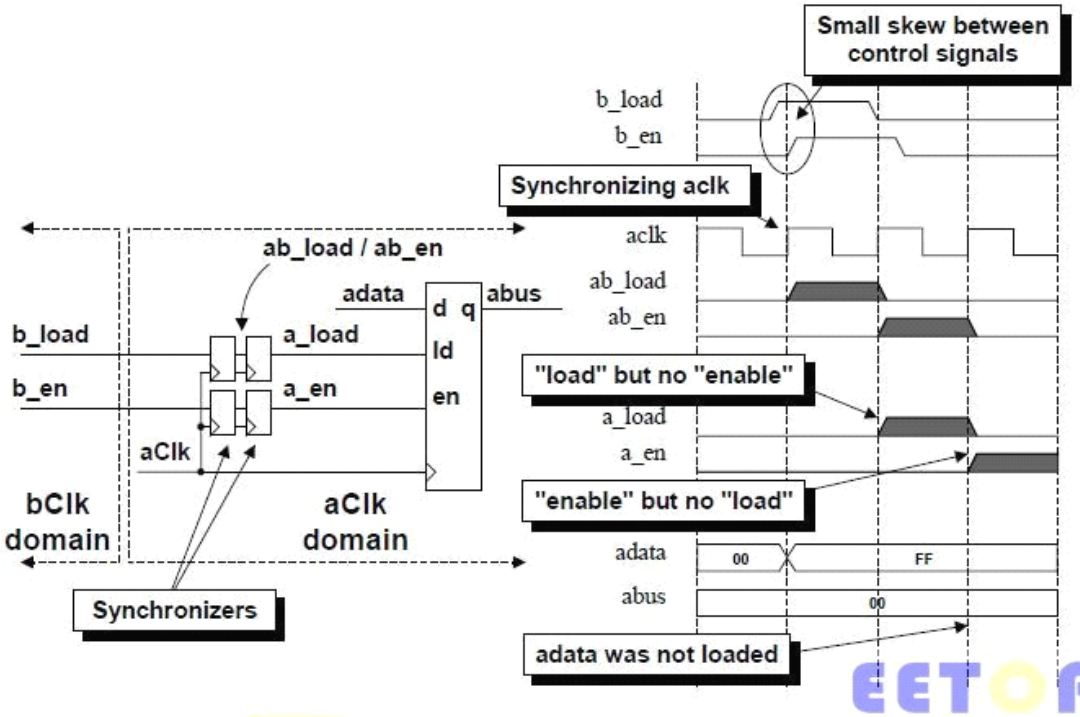


Figure 8 - Problem - Passing multiple control signals between clock domains

解决方法：只传递一个信号(b_lden)

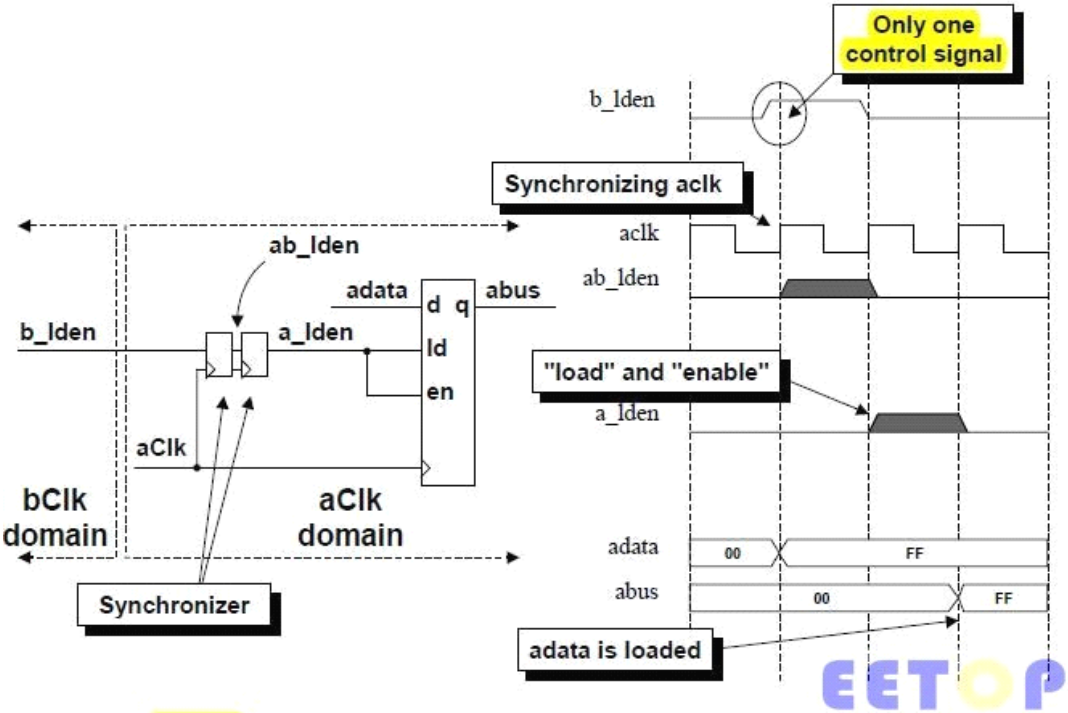


Figure 9 - Solution - Consolidating control signals before passing them between clock domains

5.2 问题 2：传递两个前后顺序控制的信号(ben1 和 ben2)

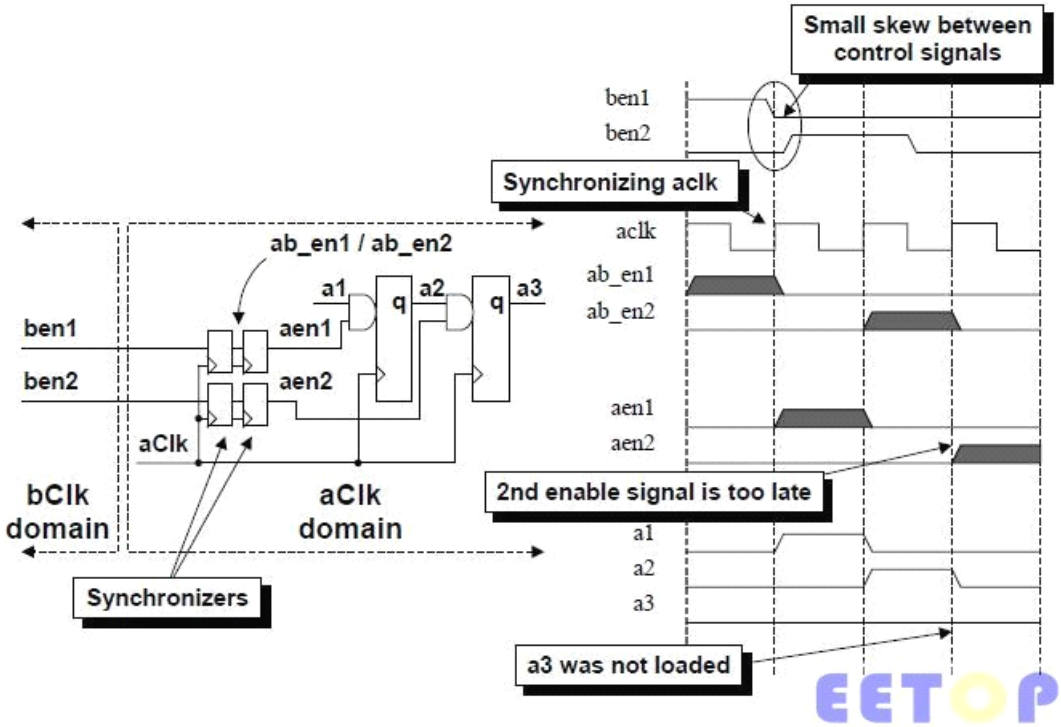


Figure 10 - Problem - Passing sequential control signals between clock domains

解决方法：只传递一个信号(ben1)

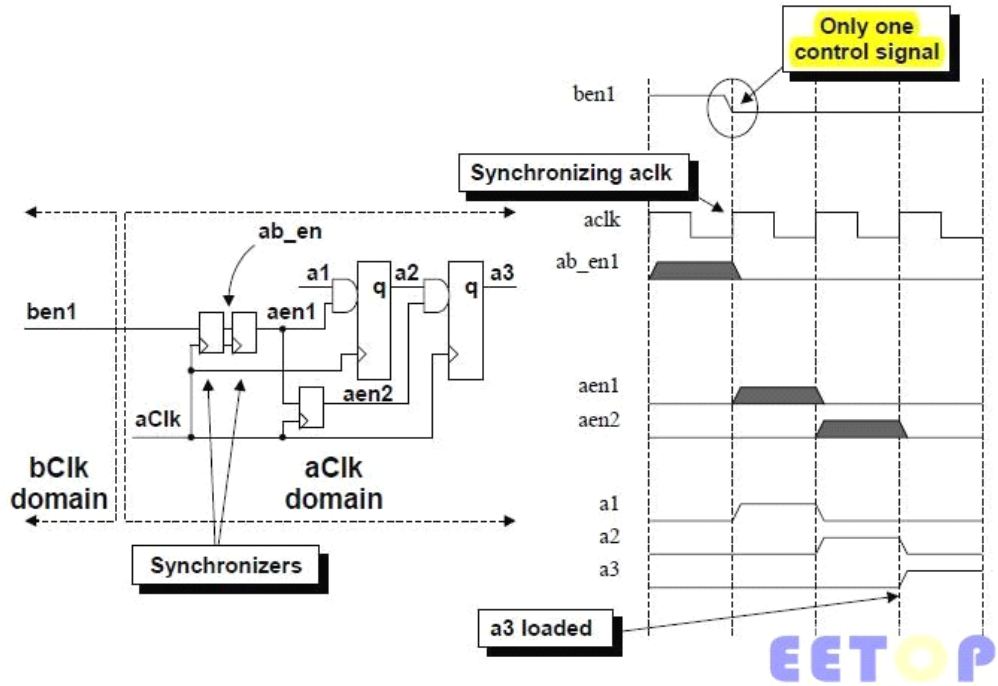


Figure 11 - Solution - Logic to generate the proper sequencing signals in the new clock domains

5.3 问题 3：传递两个编码控制信号(bdec[0]和 bdec[1])

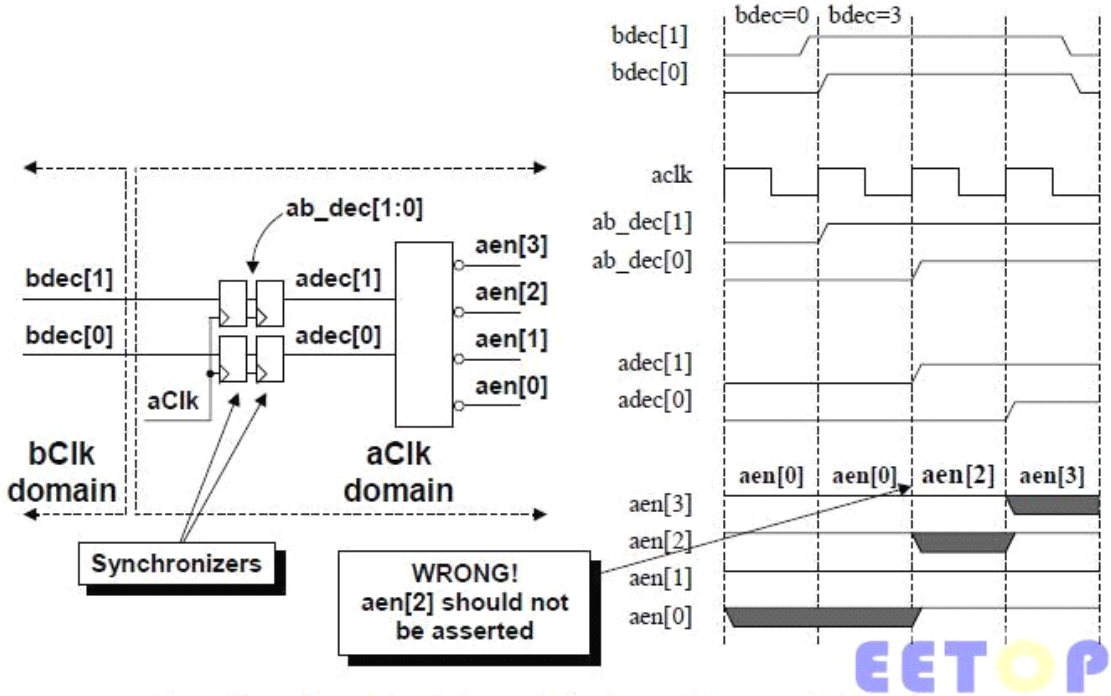


Figure 12 - Problem - Encoded control signals passed between clock domains

解决方法 1：产生一个 ready 信号(bden_n)来指示数据的有效性

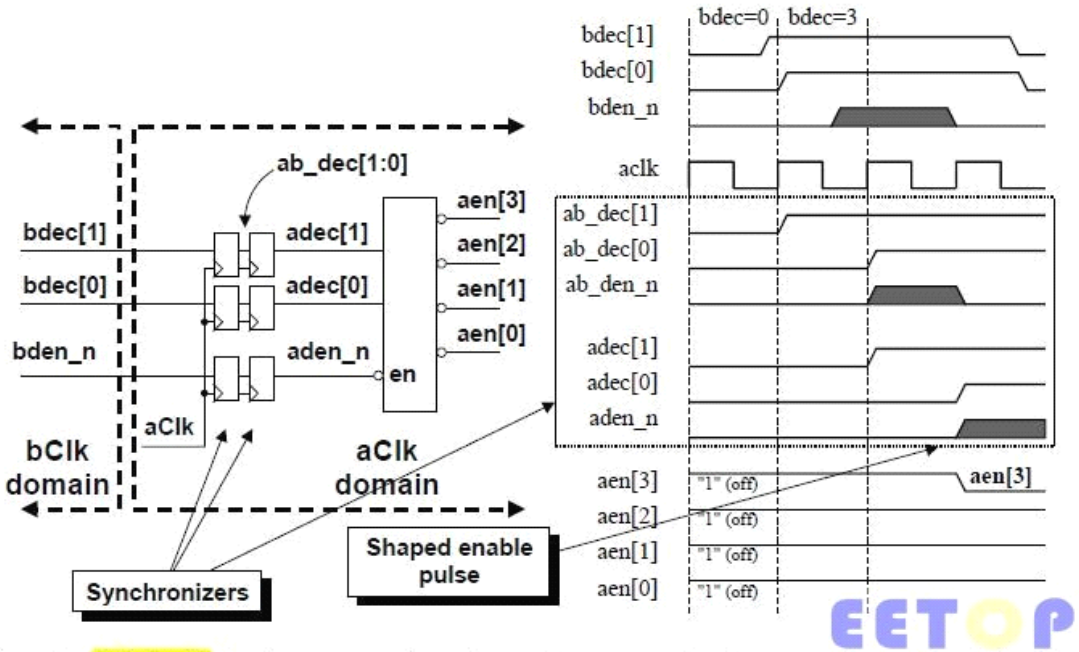


Figure 13 - Solution #1 - Logic to synchronize and wave-shape an enable pulse to pass between clock domains

解决方法 2：使用 one-hot key 信号结合状态机来处理接收到的信号

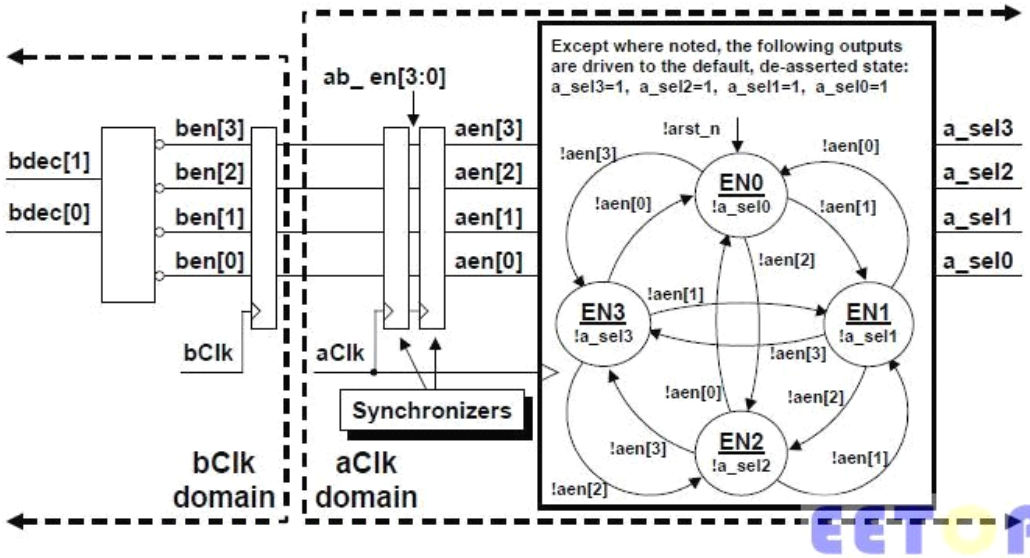


Figure 14 - Solution #2 - FSM logic to detect one-hot control signals passed from a different clock domain

附录（相关设计技巧）：

1. 慢时钟域到快时钟域的同步及上升（下降）沿检测电路

同步和上升沿检测电路：（注意输入 B 是被反向过的）

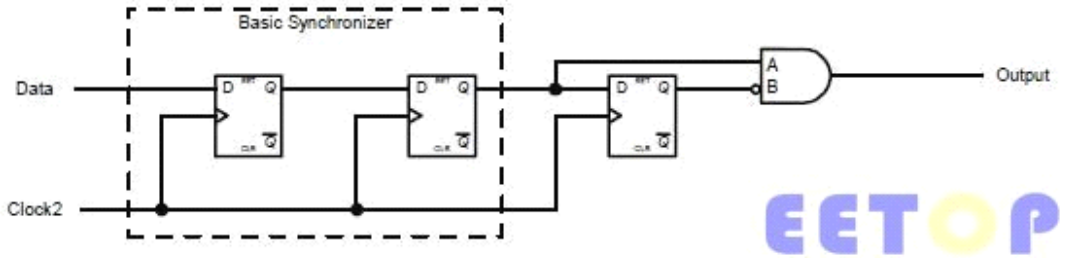


Figure 1-7: Edge-Detect Synchronizer

时序图：

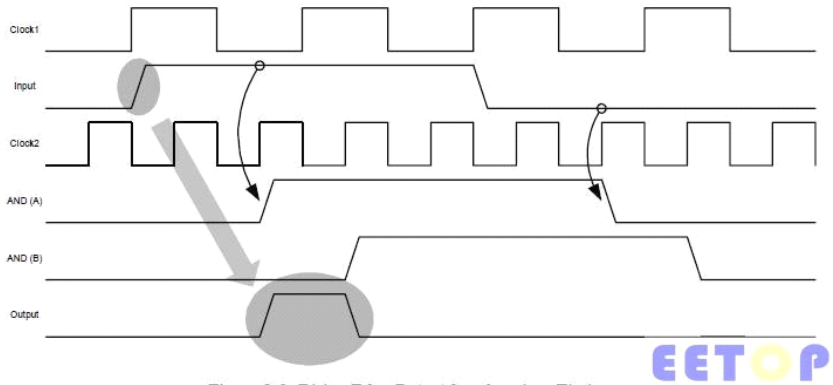


Figure 1-8: Rising Edge-Detect Synchronizer Timing

代码实现为：先将发送时钟域过来的信号用寄存器打两拍，然后将输出信号

A 和再打一拍的反向信号 B 相与（如果是下降沿检测，则将输出信号 A 反向和再打一拍的同向信号 B 相与）。

拓展：如果将先将发送时钟域过来的信号用寄存器打两拍，然后将输出信号 A 和再打一拍的信号 B 相一或，就得到的是上升沿和下降沿都检测的逻辑。

2.脉冲检测（将脉冲信号转换为电平信号，pulse-toggle-circuit）

基本电路为：

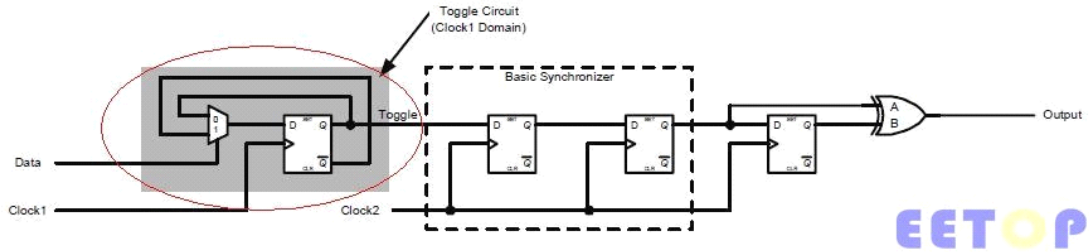


Figure 1-10: Pulse Synchronizer

时序图：

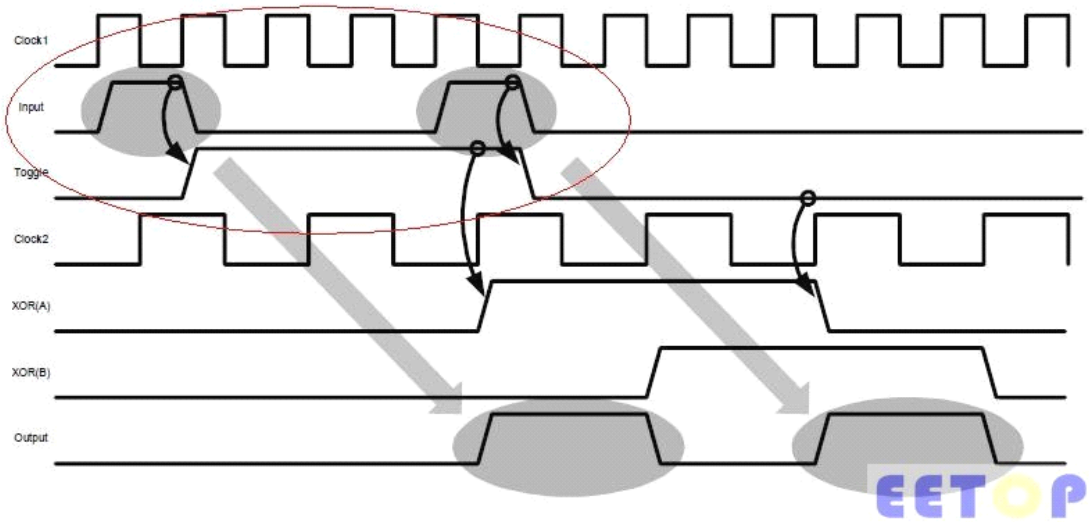


Figure 1-11: Pulse Synchronizer Timing

代码实现为：

```
always@(posedge clk1 or negedge nrst)
begin
if(!nrst)
Q <= 0;
else
```

Q <= D;

end

assign D = Data ? Q_bar : Q;

拓展：

3、完整握手流程

完整握手流程为：

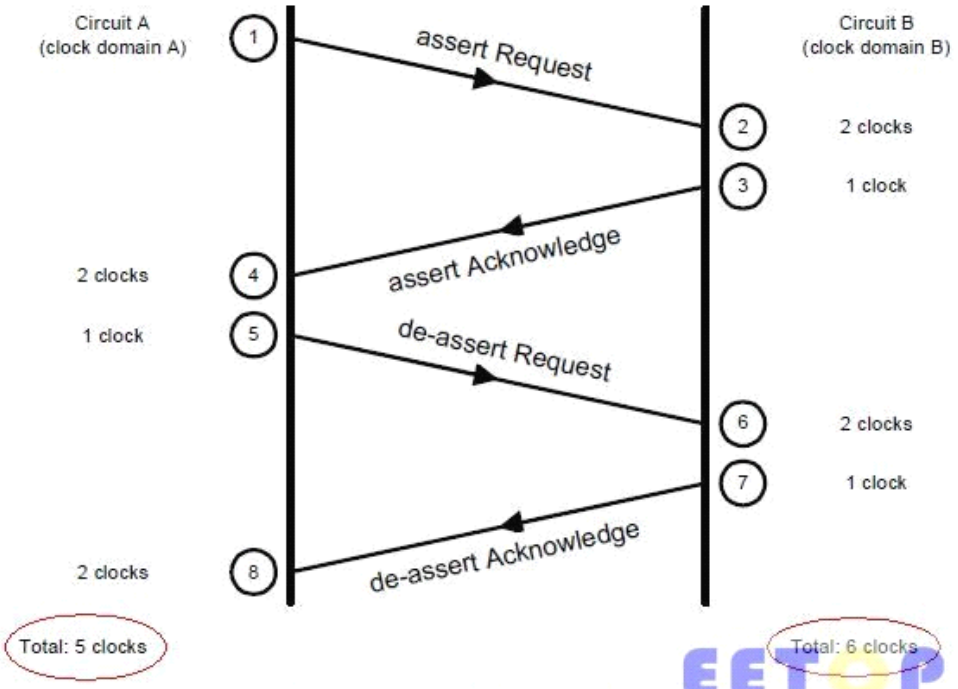


Figure 2-3: Full Handshake Flow

时序图：

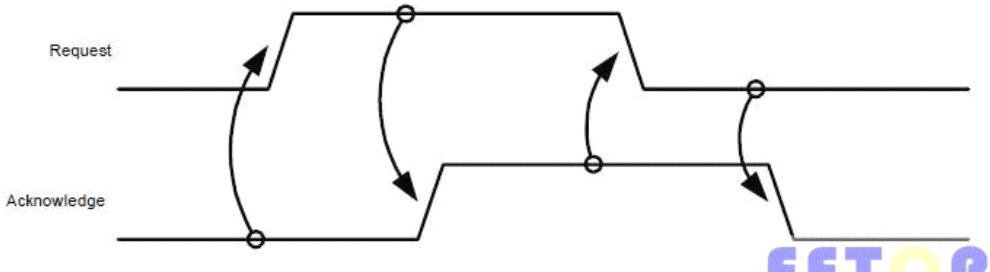


Figure 2-2: Full Handshake Signaling

代码实现为：.....

4. 部分握手流程 I（完整握手流程的简化版）

部分握手流程 I 为：

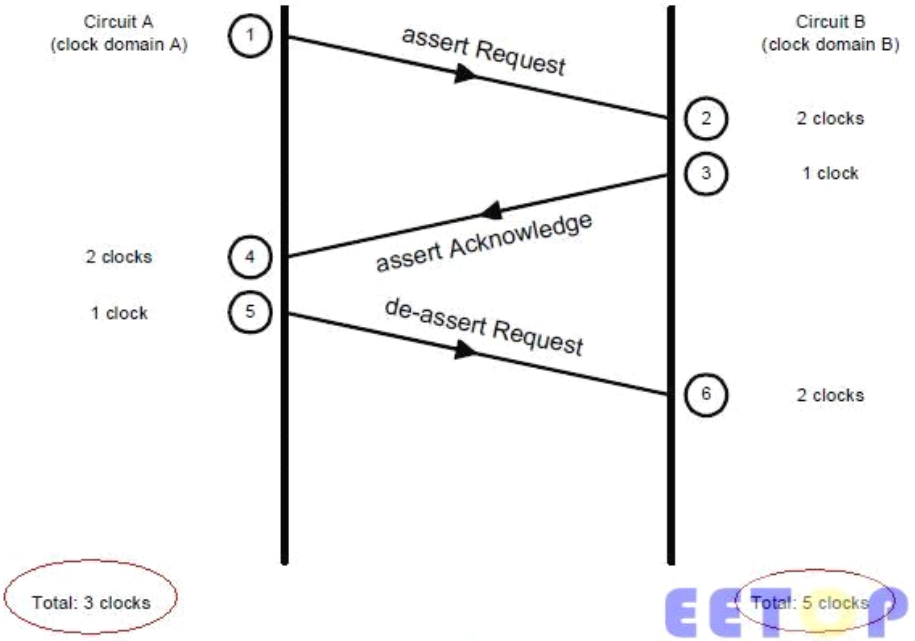


Figure 2-5: Partial Handshake I Flow

时序图：

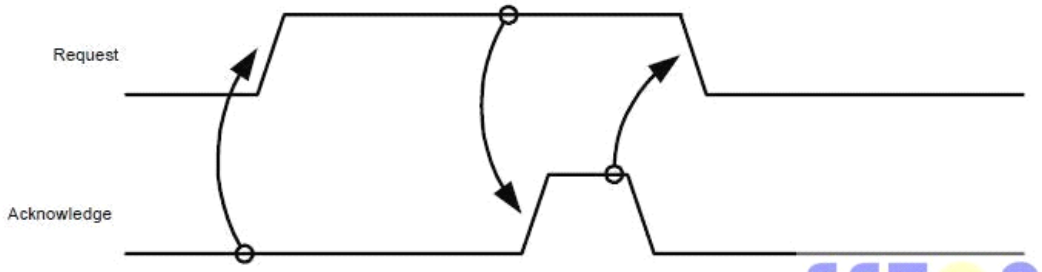


Figure 2-4: Partial Handshake I Signaling

说明：省去了完整握手流程里面的最后一步，也就是 ACK 信号自动会把自己 de-assert，而不是要等检测到 REQ 信号 de-assert 之后了。

In the first partial handshake scheme, Circuit A asserts its request signal and the Circuit B acknowledges it with a single clock-wide pulse. In this case, Circuit B does not care when Circuit A drops its request signal. However, to make this technique work, Circuit A must drop its request signal for at least one clock cycleotherwise Circuit B cannot distinguish between the previous and the new request.（A 发出的

REQ 信号必须至少无效持续一个时钟周期，否则 B 无法辨别两个响铃的 REQ 信号）

With this handshake, Circuit B uses a level synchronizerfor the request signal and Circuit A uses a pulse synchronizerfor the acknowledge signal. In this handshake protocol the acknowledge pulses only occur when Circuit B detects the request signal. This allows Circuit A to control the spacing of the pulses into the synchronizer by controlling the timing of its request signal.

5. 部分握手流程 II（完整握手流程的简化版）

部分握手流程 II 为：

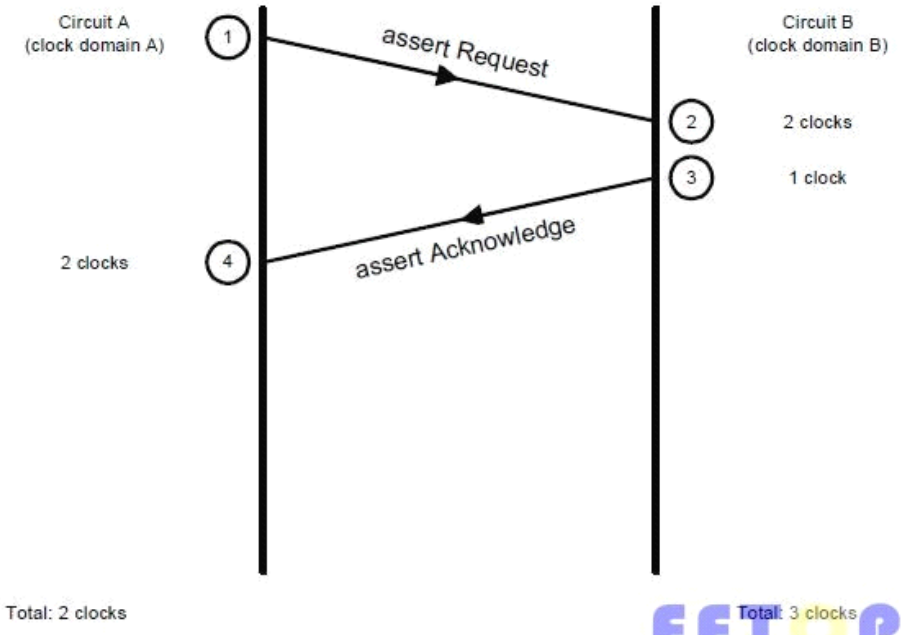


Figure 2-7: Partial Handshake II Flow

时序图：

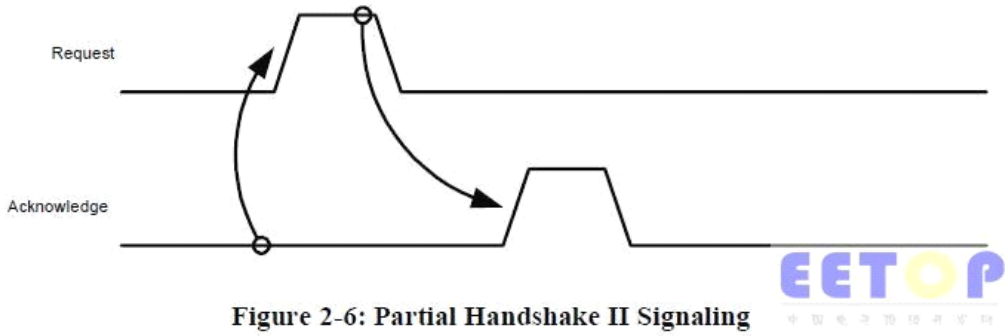


Figure 2-6: Partial Handshake II Signaling

说明：省去了完整握手流程里面的最后两步，两个信号在 assert 保持一段时间以后都是自动 de-assert，不在相互检测。

In this second partial handshake scheme, Circuit A asserts it request with a single clock-wide pulseand Circuit B acknowledges it with a single clock-wide pulse. In this case, both circuits need to save state to indicate that the request is pending.

This type of handshake uses pulse synchronizers but if one circuit has a clock that is twice as fast as the other, that circuit can use an edge-detect synchronizer instead. （如果有一个时钟域的时钟比另外一个时钟域的时钟快两倍以上，则可以使用边沿检测同步电路来代替握手电路）

6. Basic Data Path Design

基本电路为：

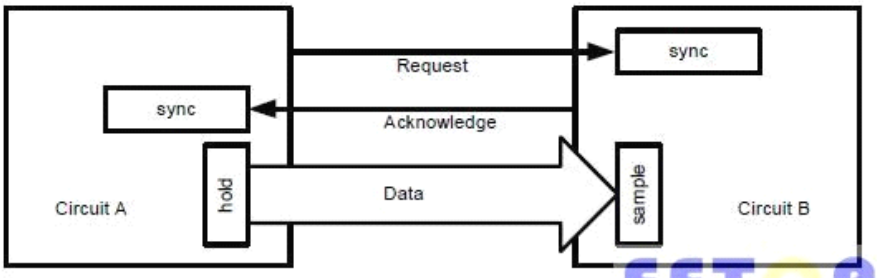


Figure 2-8: Data Path Synchronizer Diagram

时序图：

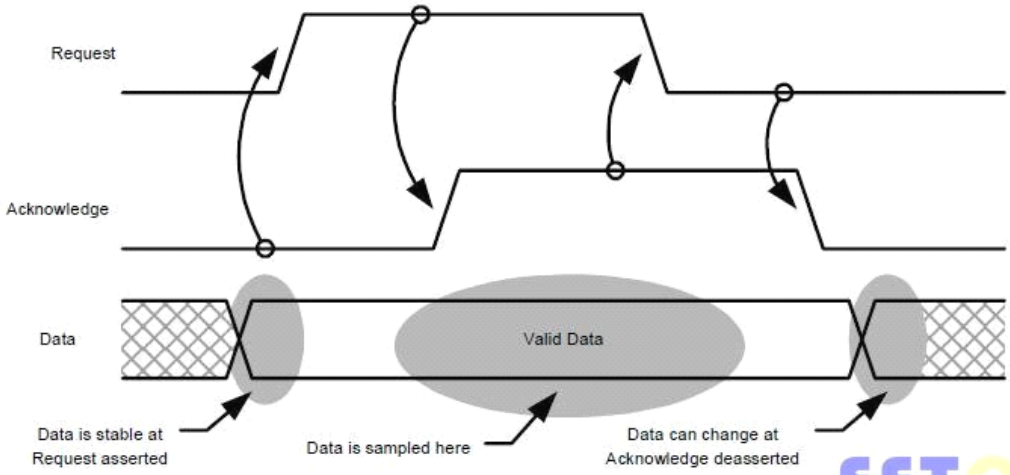


Figure 2-9: Data Path Timing Using Full Handshake 完整握手

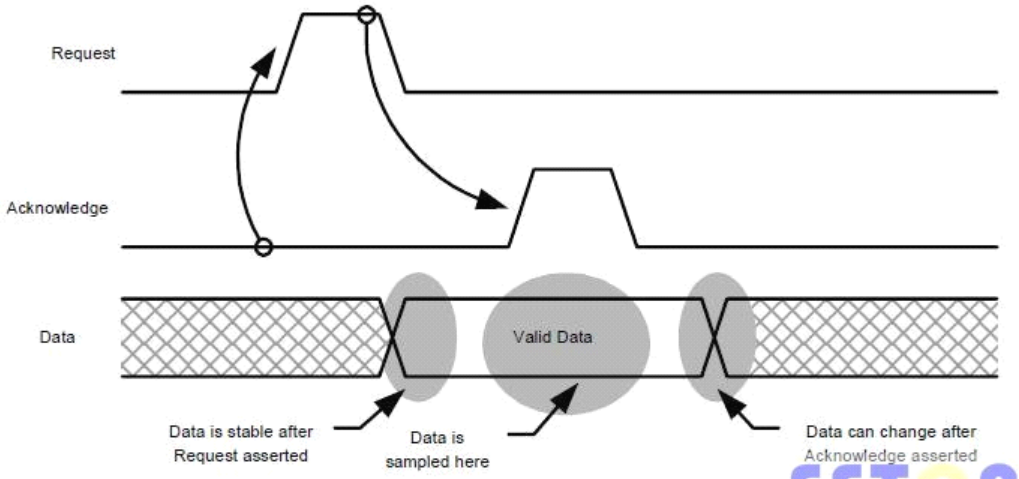


Figure 2-10: Data Path Timing Using Partial Handshake

说明：A design using full handshake signaling has a large window of time for the receiving circuit to sample the signal bus and is not very efficient. The same design can use a partial handshake instead of the full handshake to speed up the transfer.

4.3 源同步信号跨时钟域采集的两种方法 （20160620）

对于数据采集接收的一方而言，所谓源同步信号，即传输待接收的数据和时钟信号均由发送方产生。FPGA 应用中，常常需要产生一些源同步接口信号传输给外设芯片，这对 FPGA 内部产生时钟或数据的逻辑和时序都有较严格的要求。而内部的逻辑和时序。当然，无论何种情况，目的只有一个，保证信号稳定可靠的被传送或接收。

对于一个如图 1 所示的某视频芯片产生的源同步信号，当 FPGA 对其进行采集同步到另一个时钟域时，特权同学通常的做法有两种，特权同学称之为脉冲边沿检测采集法和异步 FIFO 采集法。下面简单的对这两种方法做一些讨论和说明。

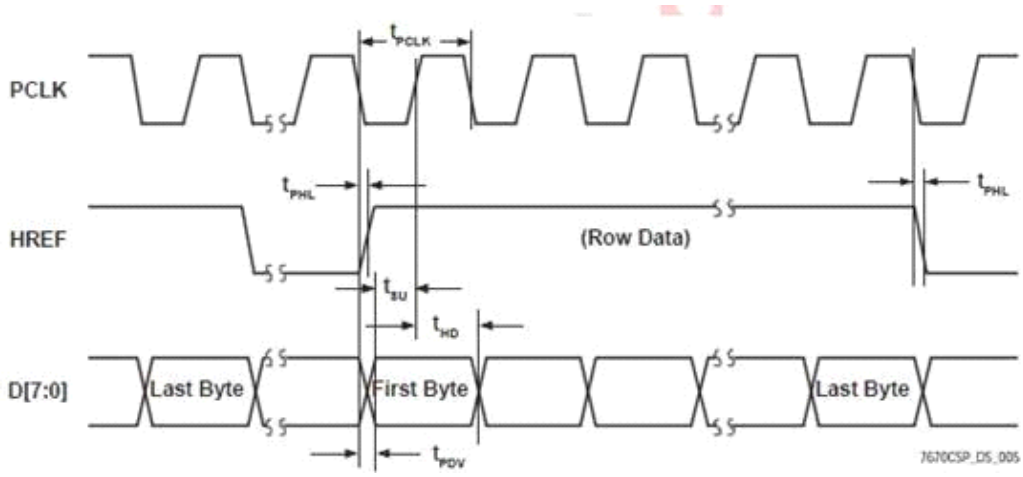


图 1

1.脉冲边沿检测采集法：

脉冲边沿检测采集法，顾名思义，一定是应用了经典的脉冲边沿检测法来帮助或者直接采集信号。对于脉冲边沿检测法，大家可以参考特权同学的《深入浅出玩转 FPGA》或者用 goooogle 摆渡一下。

而这里尤其需要提醒大家注意的是，著名的奈奎斯特采样定理告诉我们：要从采样信号中无失真的恢复原信号，采样频率应大于两倍信号最高频率。而特权同学通过实践得出的结论与此相仿：若想稳定有效的采集到脉冲（数字信号）变化的边沿，采样频率应大于被采样脉冲最大频率的 3 倍。注意是要大于 3 倍，甚至若是可能尽量采用 4 倍以上的采样频率才能够达到稳定的状态。

至于为什么，我想深谙此道（脉冲边沿检测法）的聪明人看完结论就已经明

白了，无需特权同学再废话解释一番。而具体的做法也很简单，把图 1 理想化就如图 2 所示。其中，待采集信号时钟 Tx Clock，待采集数据使能信号 Enable Signal，待采集数据总线 Data Bus。FPGA 内部信号采集时钟为 Rx Clock，该时钟为待采集时钟的 4 倍。

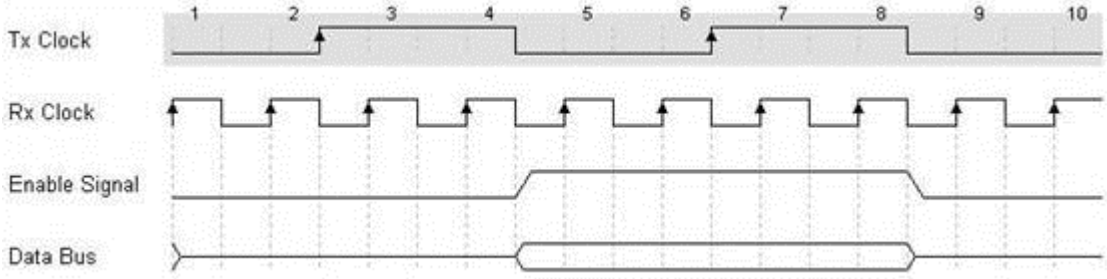


图 2

采用脉冲边沿检测法，使用 Rx Clock 去采集 Tx Clock，Rx Clock reg1 和 Rx Clock reg2 分别为第一级和第二级 Tx Clock 锁存信号。Tx Clock 上升沿对应的一个有效指示信号 Tx Clock pos 每个 Tx Clock 时钟周期产生一个 Rx Clock 脉宽的有效高电平使能信号。

从图 3 中可以看到，此时若用 Tx Clock pos 作为 FPGA 内部采样使能信号，虽然 Tx Clock pos 处于第 7 个 Tx Clock，但是真正采集 Data Bus 其实已经是第 8 个 Tx Clock 上升沿了。很明显，第 8 个 Tx Clock 上升沿对准的不是 Data Bus 的稳定信号中央，数据很可能采集到错误值。

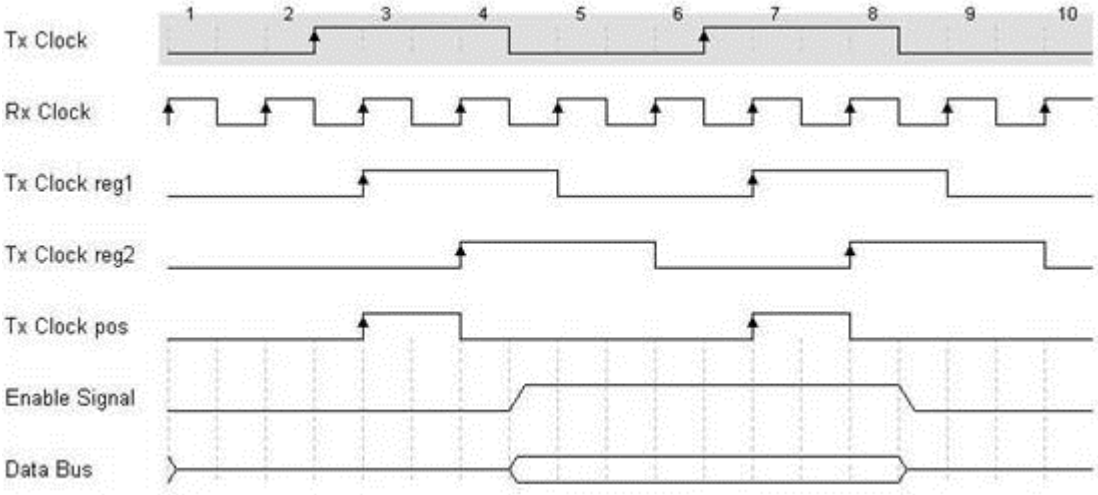


图 3

因此，通过上面的分析，还是可以采取一些变通的方式来保证第 8 个 Tx

Clock 上升沿采集到 Data Bus 的中央值。如图 4 所示，采用同样的方式对 Data Bus 做两级信号锁存，那么第 8 个 Tx Clock 上升沿就能够在 Data Bus reg2 的中央采集数据了。这样做只有一点小问题，相应的需要多付出 2 组寄存器来锁存 Data Bus。

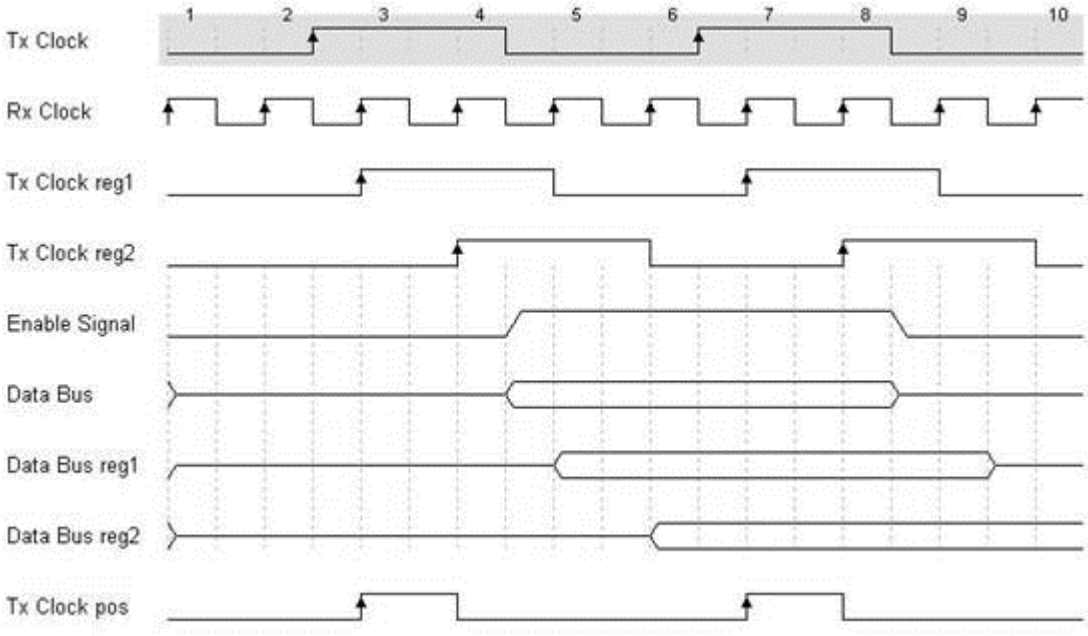


图 4

2.异步 FIFO 采集法

再说异步 FIFO 采集法，其实这种方法没什么新奇，只不过充分利用异步 FIFO 的同步特性来完成跨时钟域的数据交互。但是其中还是有几个非常关键的要点需要提醒设计者注意，无论如何 FIFO 的输入端数据和时钟信号（包括控制信号，如有效使能信号等）必须符合 FIFO 的数据锁存有效建立和保持时间，这个最重要的条件除了需要靠数据源端来保证外，还需要靠数据锁存端（FPGA 内部）设计者做好时序上的约束和分析，否则到源端再 nice 的波形恐怕都无法保证能够可靠的被 FIFO 锁存。

异步 FIFO 的基本通讯时序波形如图 5 所示。我们关心的是 FIFO 的写入。由图中不难发现，写入时钟 wrclk 的每个上升沿会判断写入请求信号 wrreq 是否有效，若是有效则 FIFO 会相应的锁存当前的写入数据 data。简单来看，从基本时序上分析，wrclk 的上升沿需要对准 wrreq 和 data 的中央，这是外部传输过来的源信号必须满足的基本关系。无论如何，即便是绞尽脑汁，也要想办法让这个

基本关系得到保证，否则，后面的 rdclk、rdreq 配合的再默契恐怕都不能得到稳定的 q 输出。

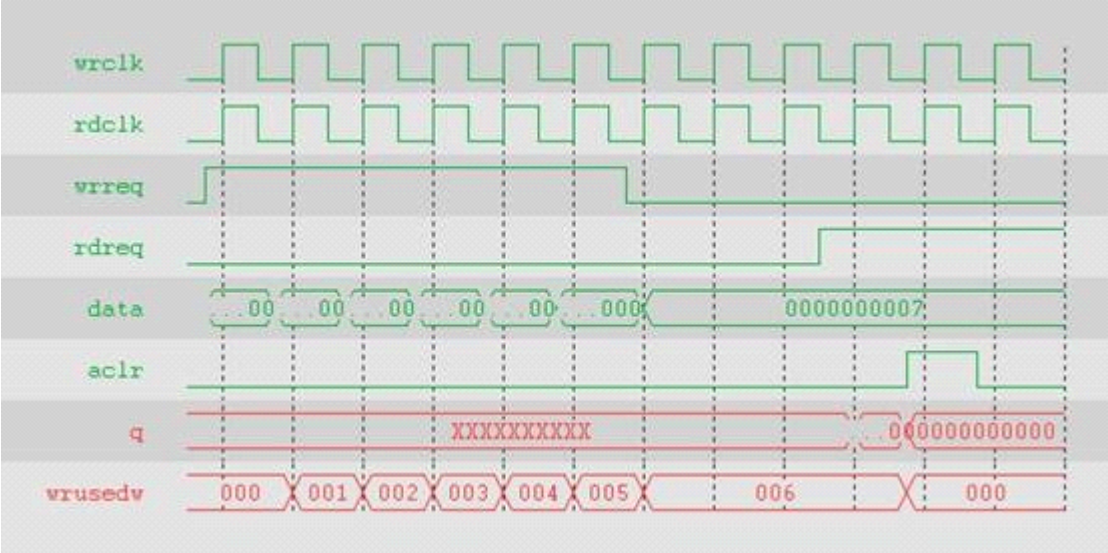


图 5

原型开发的前期，设计者必须首先验证写入信号的关系，哪怕是不惜动用示波器（⊙__⊙b 汗，连示波器都没有不要混了），源端给到 FPGA 输入端口的信号很多时候不是那么尽善尽美的，实践出真知，测试结果说了算。当然了，实在没有先进武器又想打胜仗的朋友恐怕只有不停的用代码测试采集到最稳定的数据了，这有点碰运气的成分在里面，不是非常推荐。

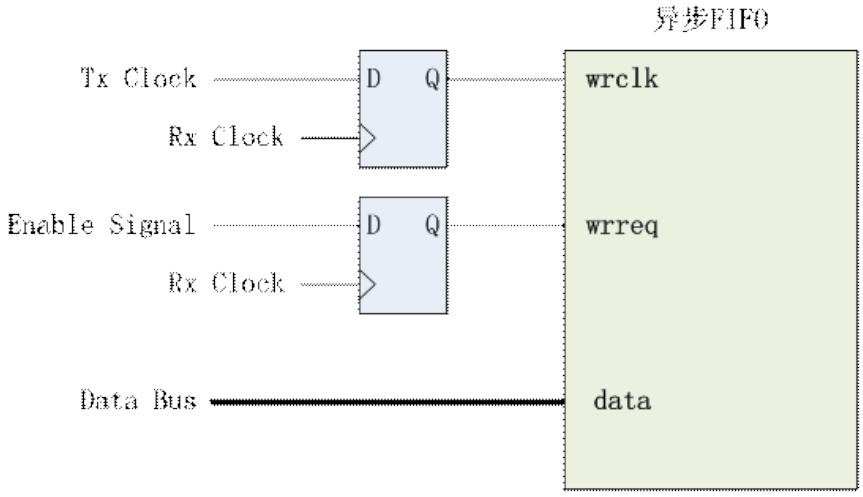


图 6

类似与开篇提到的应用，特权同学的实际信号采集如图 6 所示，把写入时钟 Tx Clock 和写入请求信号 Enable Signal 都先用同步时钟 Rx Clock 打了一拍，

然后再输入 FIFO 中，而数据总线 Data Bus 则直接送往 FIFO。这样从最终检测来看，能够保证时钟的上升沿对准数据和控制信号的中央，相对稳定和安全的把数据送往 FIFO 中。

工程实践中往往不是一招一式的生搬硬套理论，一定要抓住最关键的设计要点，并采取各种有效的手段保证设计的实现。