全面解析复位电路设计中的结构性缺陷及解决方案

随着数字化设计和 SoC 的日益复杂，复位架构也变得非常复杂。在实施如此复杂的架构时，设计人员往往会犯一些低级错误，这些错误可能会导致亚稳态、干扰或其他系统功能故障。本文讨论了一些复位设计的基本的结构性问题。在每个问题的最后，都提出了一些解决方案。

复位域交叉问题

1. 问题

在一个连续设计中，如果源寄存器的异步复位不同于目标寄存器的复位，并且在起点寄存器的复位断言过程中目标寄存器的数据输入发生异步变化，那么该路径将被视为异步路径，尽管源寄存器和目标寄存器都位于同一个时钟域，在源寄存器的复位断言过程中可能导致目标寄存器出现亚稳态。这被称为复位域交叉，其中启动和捕捉触发的复位是不同的。

在这种情况下，C 寄存器和 A 寄存器的起点异步复位断言是不同的。在 C 寄存器复位断言过程中而 A 触发器没有复位，如果 A 寄存器的输入端有一些有效数据交易，那么 C 寄存器的起点异步复位断言引起的异步变更可能导致目标 A 寄存器发生时序违规，从而可能产生亚稳态。

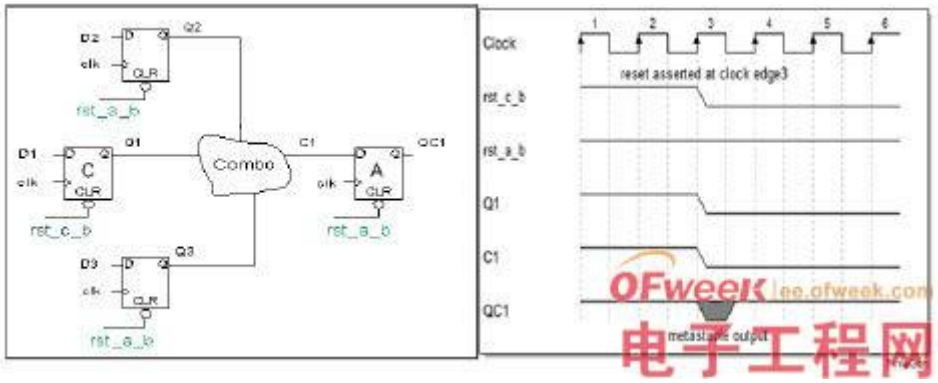

图 1:复位域交叉问题

在上面的时序图中，当有一些有效数据交易通过 C1 进行时，rst_c_b 获得断言，导致 C1 发生异步改变，w.r.t clk 从而使 QC1 进入亚稳态，这可能导致设计发生功能故障。

2. 解决方案

* 使用异步复位、不可复位触发器或 D1 触发器 POR.

* 如果复位源 rst_c_b 是同步的，那么则认为来自 C_CLR --> Q 的用于从 rst_c_b_reg -->C_CLR-->C_Q1-->C1-->A_D 进行设置保持检查的时序弧能够避免设计亚稳态。然而，通常在默认情况下 C_CLR-->Q 时序弧在库中不启用，需要在定时分析过程中明确启用。

* 在目的地（A）使用双触发器同步器，以避免设计中发生亚稳态传播。然而，设计人员应确保安装两个触发器引入的延迟不会影响预期功能。

由于组合环路导致复位源干扰

1. 问题

在 SoC 中，全局系统复位在设备中组合了软件或硬件生成的各种复位源。LVD 复位、看门狗复位、调试复位、软件复位、时钟丢失复位是导致全局系统复位断言的一些示例。 然而，如果由于任何复位源导致的全局复位断言是完全异步的，且复位发生源逻辑被全局复位清零，那么设计中会产生组合环路，这会在该复位源产生干扰。组合路径的传播延迟会根据不同的流程、电压或温度以及干扰范围而不同。如果设计中使用了组合信元用于复位断言和去断言，那么也会导致模拟中出现紊乱情况。这被视为设计人员的非常低级的错误。

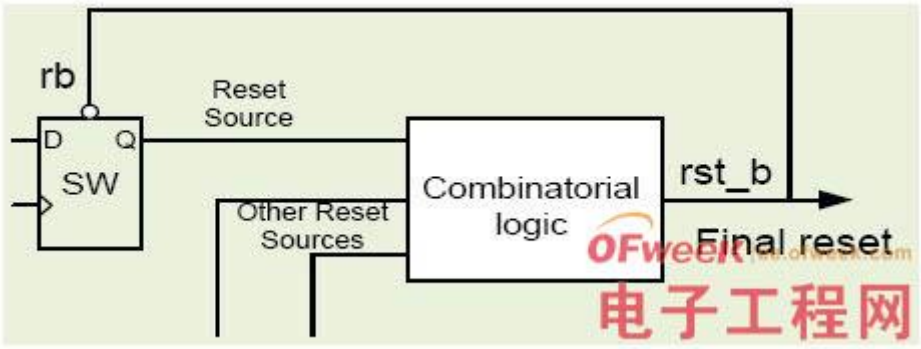

图 2:复位源干扰（基本问题）

在上图中，当复位源 SW_Q 断言时，会导致 rst_b 断言，这是全局复位。现在，如果全局复位本身被用于清除 “SW_Q” 复位断言，那么会在设计中在 SW_Q 输出和全局复位时产生干扰。此外，在模拟中，这会导致紊乱情况，因为复位源断言试图通过该组合逻辑去断言。

然而，如果复位源（SW_Q）在复位状态机（触发器的 SET/CLR 输入）为全局复位断言被异步使用，那么复位干扰可能能够复位整个系统（通过断言全局复位），因为全局系统复位去断言不仅仅与复位源去断言相关。当该复位源（有干扰）被同步使用或在触发器 D 输入使用的情况下可能依然有一个问题。干扰范围可能无法在至少一个周期内保持稳定，因此这不会被目标触发器捕获。此外，该复位源不能被用作任何电路的时钟（除了脉冲捕捉电路），因为它可能违反时钟宽度。

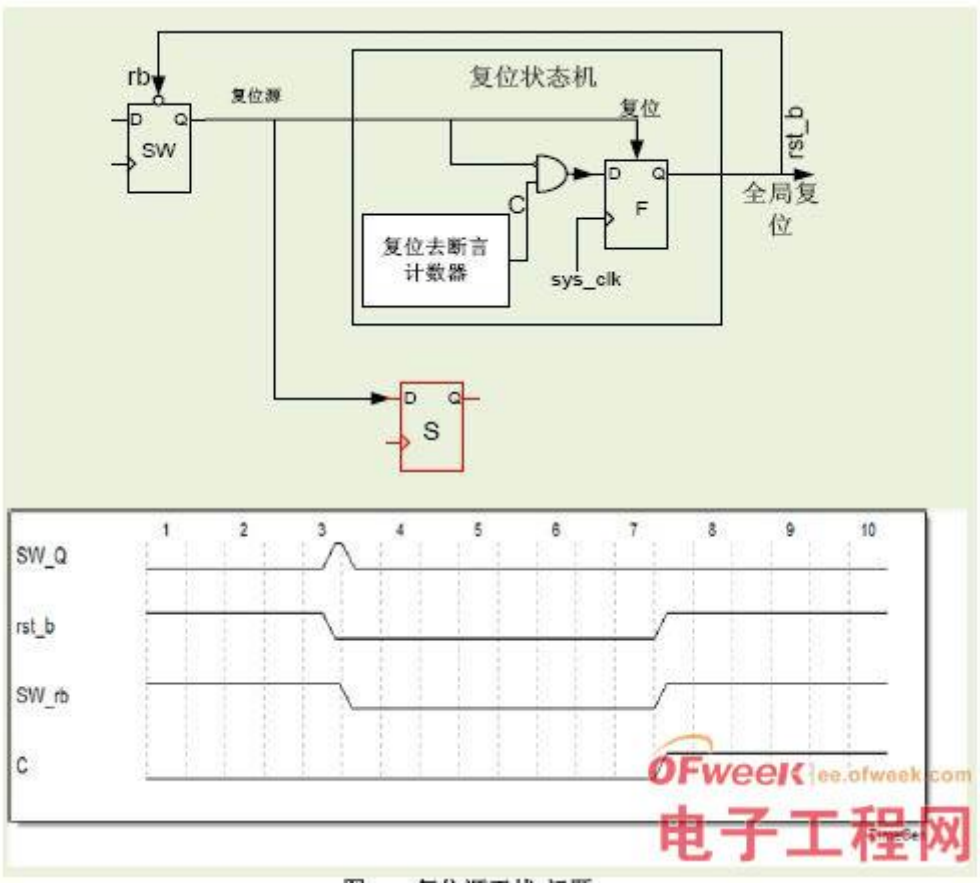

图 3:复位源干扰（问题 2）

在上图中，复位源 SW_Q 将出现干扰。虽然如果复位源 SW_Q 的干扰在某个触发器被捕捉作为复位事件状态（在 S）或用于其他目的，全局复位输出（rst_b）都没有干扰，但它将导致时序违反/亚稳态，或根本不可能被捕获。

2. 解决方案

* 设计人员永远都不应犯下上述（图 2）低级错误。

* 如果复位实现如图 3 所示，那么设计人员应保证复位源（在该示例中为 SW_Q）总是在触发器的 SET/CLR 输入使用，而不在 D 或 CLK 使用。

* 解决这个问题的最好的方法是在复位状态机中使用之前注册该复位源。虽然它将导致时钟依靠全局复位断言，但是无论如何，如果没有时钟，该内部复位（SW_Q）都不会断言。请参见图 4.

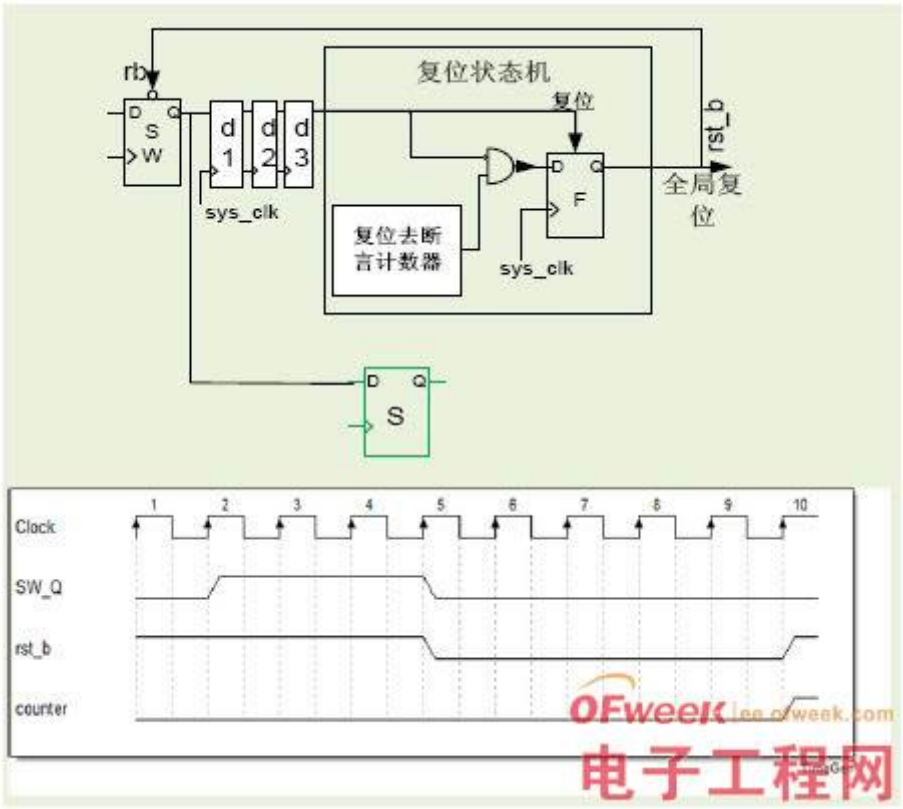

图 4:解决方案 1

此外，用户也可以扩展 SW_Q 断言，然后再在设计中使用它，复位断言与时钟无关。 请参见图 5.

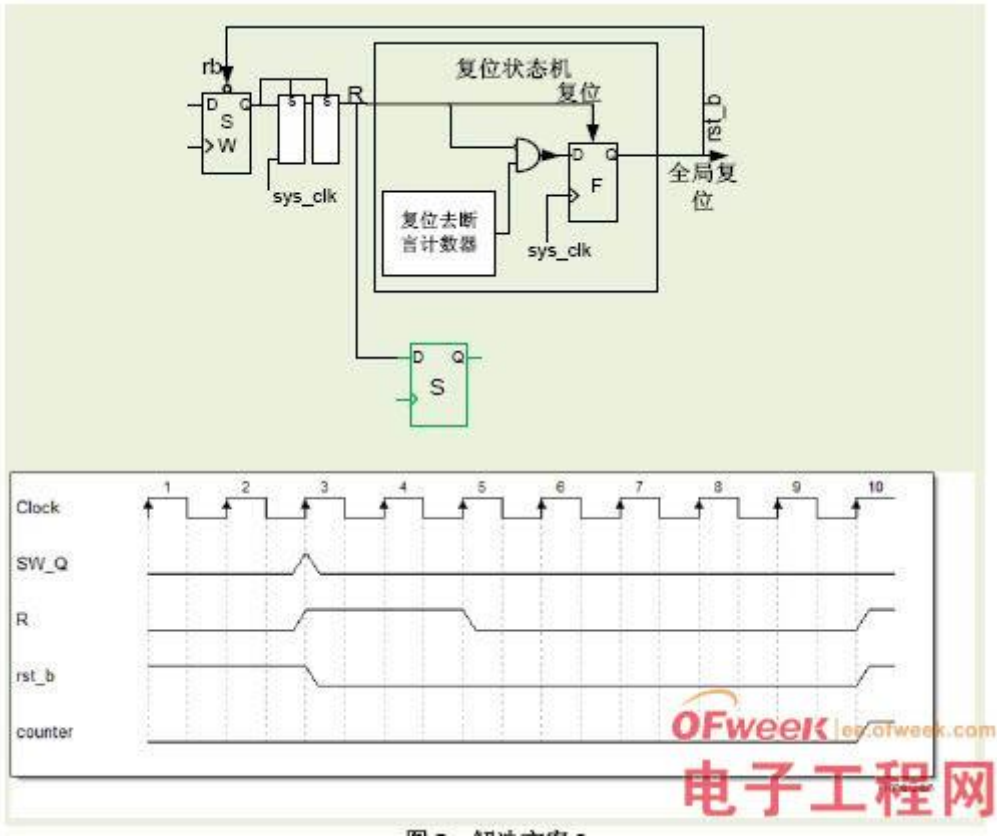

图 5:解决方案 2

复位路径的组合逻辑

1. 问题（I）

如果组合逻辑输入大约在同一时间发生变化，那么使用复位路径中的组合逻辑可能产生干扰，这可能在设计中触发虚假复位。下面是一个 RTL 代码，它会在设计中意外复位。

```
assign module_a_rstb = ! ( （slave_addr[7:0]==8'h02 & write_enable & （wdata[7:0]==00） )

always @（posedge clk or negedge module_rst_b）

if （! module_rst_b） data_q <= 1'b0;

else data_q <= data_d;
```

在上面的示例中，slave_addr,write_enable 和 wdata 改变它们的值 w.r.t system clock,使用静态时序分析，设计人员可以保证在目标触发器的设置时间窗口之前这些信号在一个时钟周期内的稳定性。然而，在该示例中，这些信号直接用作触发器的异步清零输入。

因此，即使在特定的时间 slave_addr[7:0]在逻辑上将其值从“00000110”改为“01100000”,但由于组合逻辑的传播延迟（净延迟和信元延迟）它可以用一个序列“00000110 --> 00000010 --> 00000000 --> 01000000 --> 01100000”生成过渡。

在这段时间里，salve_addr 为“00000010”,如果 wdata[7:0]始终为零且“write_enable” 已经被断言，那么它将在 module_rst_b 创建一个无用脉冲，从而导致虚假复位。

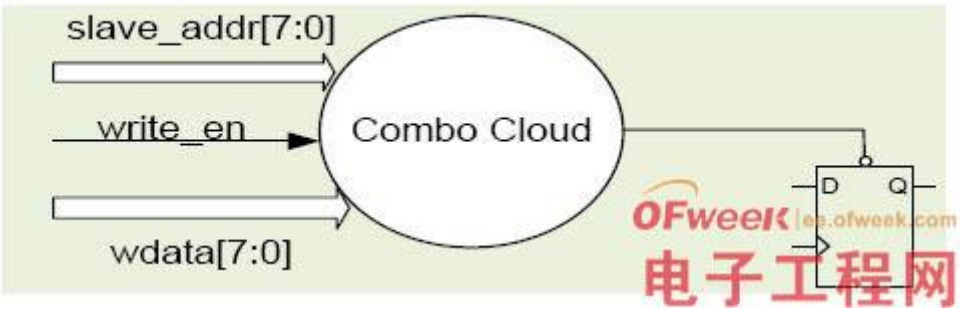

图 6:复位路径的组合逻辑

2. 解决方案

首先注册组合输出，然后再将其用作复位源（如图 7 所示）。

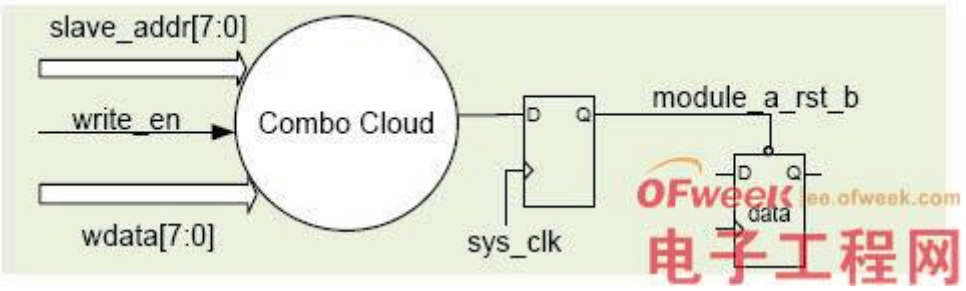

图 7:复位路径的组合逻辑解决方案

3. 问题（II）

在上面的示例中，复位路径的组合逻辑解决方案并不完善。如果组合逻辑输入大约在同一时间发生变化，那么它可能在设计中触发虚假复位。然而，如果组合逻辑的输入信号变化相互排斥，那么它可能不会引起任何设计问题。例如，测试模式和功能模式相互排斥。因此复位路径的测试复用是有效的设计实践。

然而，对于某些情况，变化相互排斥的静态信号或信号可能会导致设计出现虚假复位触发。下面的示例描述了此类设计可能出现问题。

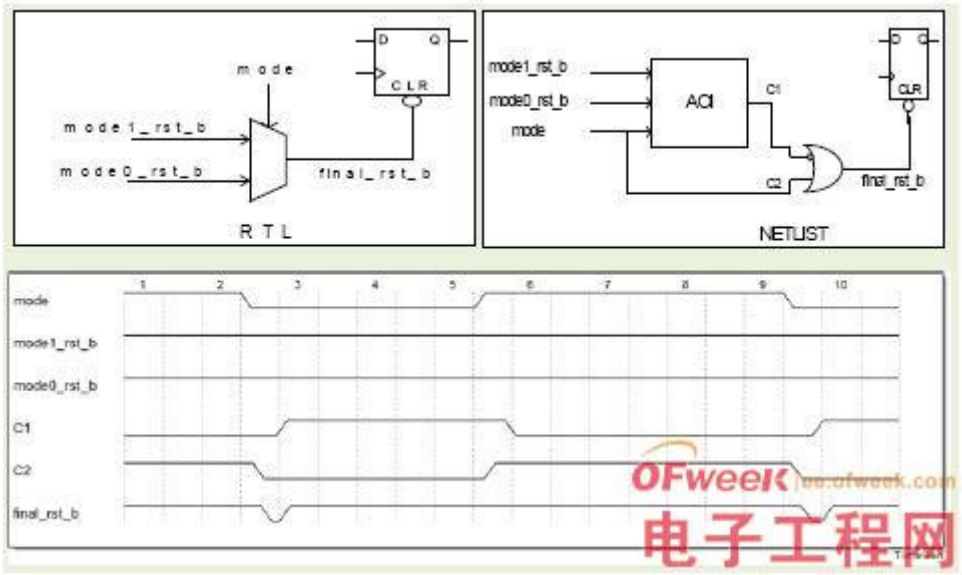

图 8:复位路径的组合逻辑（问题 2）

在上面的示例中，多路复用结构用于复位路径，同时进行 RTL 编码。其中“mode” 是一个控制信号，不频繁改变，而 mode0_rst_b 和 mode_1_rst_b 是两个复位事件，然而在合成 RTL 时，在门控级它被分解成不同的复杂的组合（And-Or-Invert[AOI]）信元。虽然在逻辑上它相当于一个多路复用器，但由于不同的信元和净延迟，每当信号“mode”从 1-->0 变化时，final_rst_b 都会产生干扰。

4. 解决方案

* 在合成过程中在复位路径保留多路复用结构，因为多路复用结构与其他组合逻辑相比易于产生干扰。MUX Pragma 可以在编码 RTL 时使用，这将有助于合成工具在复位路径中保留任何多路复用器。

设计中的同步复位问题

1. 问题（I）

在许多地方，设计人员在时钟方面喜欢同步复位设计。原因可能是为了节省一些芯片面积（带有异步复位输入的触发器比任何不可复位触发器都大）或让系统与时钟完全同步，也可能有一些其他原因。对于此类设计，当复位源被断言时需要向设计的触发器提供时钟，否则，这些触发器可能会在一段时间内都不进行初始化。但当该模块被插入一个系统时，系统设计人员可能选择在复位阶段禁用其时钟（如果在一开始不需要激活该模块），以节省整个系统的动态功耗。因此，该模块甚至在复位去断言后一段时间内都不进行初始化。如果该模块的任何输出直接在系统中使用，那么将捕获未初始化和未知的值（X），这可能会导致系统功能故障。

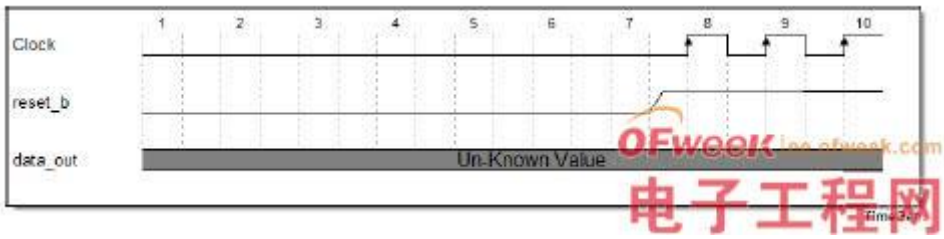

图 9:同步复位问题时序图

2. 解决方案

在复位阶段启用该模块的时钟且持续最短的时间，使该模块内的所有触发器都在复位过程中被初始化。 当系统复位被去断言时，模块输出不会有任何未初始化的值。

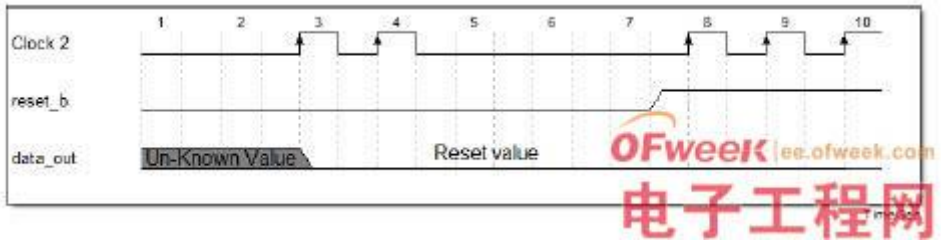

图 10:同步复位问题已解决

3. 问题（II）

在时钟域交叉路径使用两个触发同步器是常见做法。然而，有时设计人员对这些触发器使用同步复位。相同的 RTL 代码是

```
always @（posedge clk ）

if（! sync_rst_b） begin

sync1 <= 1'b0; sync2 <= 1'b0 ;

end

else begin

sync1 <= async_in; sync2 <= sync1

end
```

在硬件中进行了 RTL 合成后，上面的代码会在双触发器同步器的同步链中引入组合逻辑，这会带来风险，并缩短 sync2 触发器输入进入亚稳态的时间。

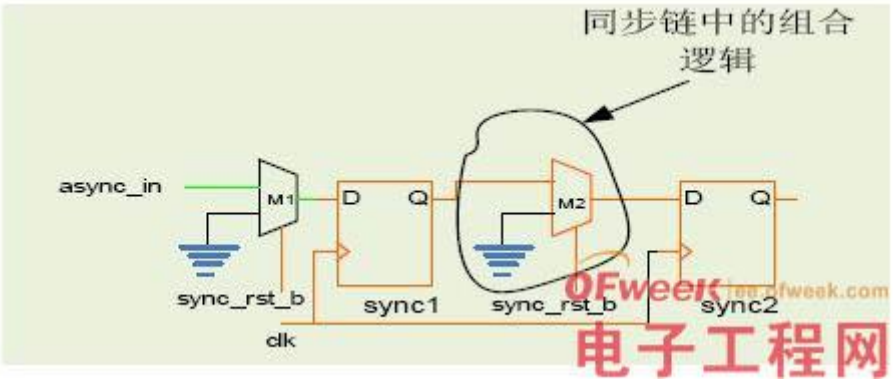

图 11:同步复位问题 2

2. 解决方案

可用以下方式编写 RTL 代码，以避免同步链的组合逻辑。

```
always @（posedge clk ）

if（! sync_rst_b） begin

sync1 <= 1'b0;

end

else begin

sync1 <= async_in; sync2 <= sync1

end
```

在上面的代码中，对 sync2 触发器不使用复位，因此在同步链中不会实现组合信元。然而，需要注意 sync2 需要一个额外的周期才能复位，这不应导致设计出现任何问题。

冗余复位同步器引起的问题

1. 问题

在使用多个异步时钟的设计中，设计人员需要确保在目标寄存器使用的时钟方面，异步复位的同步去断言，否则可能导致目标触发器发生时序违反，从而产生亚稳态。复位同步器被用来复位去断言，与目标时钟域同步。然而，只有在系统复位去断言过程中有目标时钟时才会发生复位去断言时序违反。如果在复位去断言时没有时钟，那么便不会有任何时序违反。因此，在设计多时钟域模块时，设计人员可以让编译时间选项绕过该模块中的那些复位同步器，并让系统集成商根据对该模块的时钟可用性决定是否需要使用复位同步器。

此外，如果系统时钟和异步时钟比非常高，冗余同步器甚至会造成设计功能性问题。下面描述了这个问题。

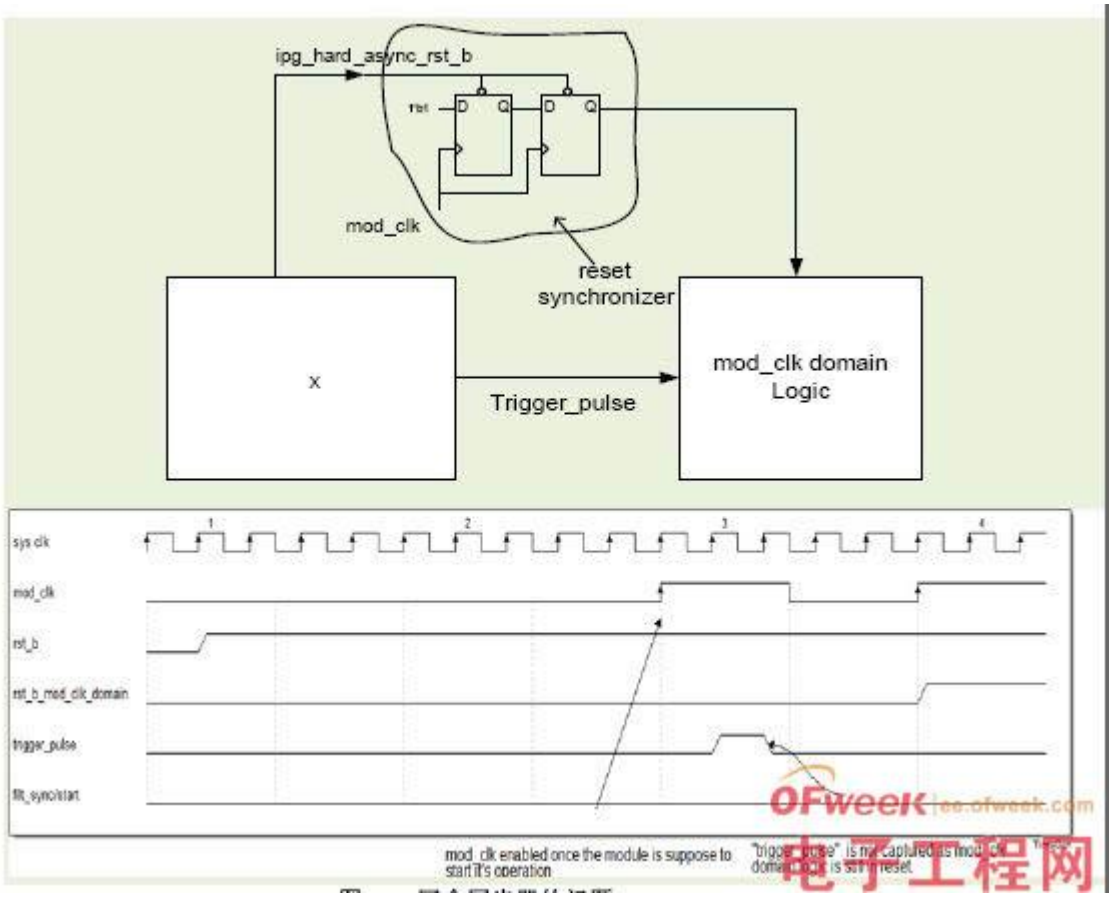

图 12:冗余同步器的问题

在上面的设计中，去断言与 sys clk 同步的系统复位被馈送到(mod_clk 域)的复位同步器，然后在 mod_clk 域逻辑中使用该复位。让我们假定 sys clk : mod_clk 的时钟频率比大于 6:1.默认不启用 mod_clk,以节省动态功率。当用户想要启用 mod_clk 域逻辑的功能时，便启用该时钟。在启用了该时钟后，有两个 mod_clk 周期的延迟，其中，由于复位同步器导致整个 mod_clk 域逻辑都处于复位状态。在该阶段，如果一些数据交易从 sys clk 域开始，将在 mod_clk 域丢失。

2. 解决方案

虽然这不是大问题，但有时会在客户一端造成混淆，因为该延迟对客户不可见。 因此消除混淆的更好的方式是：

* 如果在全局复位去断言过程中没有时钟，则在设计中绕过/删除冗余复位同步器。 这当然会节省一定的门控数。

* 如果动态功耗不是问题，用户可以在 mod_clk 域逻辑开始运作之前很长时间在启动代码选择启用 mod_clk. 因此，复位去断言将有足够的时间传播。

* 这也可以在软件中处理，在任何有效操作之前启用了 mod_clk 后，设置两三个 mod_clk 周期的延迟。

由于罕见的时钟路径导致复位去断言时序问题

1. 问题

设计的复位架构根据系统而不同。在一些安全关键设备中，整个复位状态机在安全时钟上工作，安全时钟默认启用。该时钟也被用作设备的默认系统时钟。

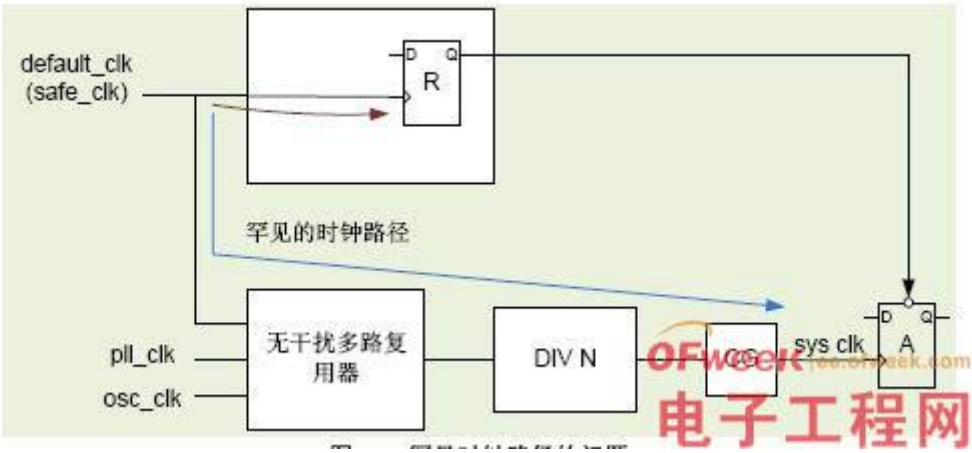

图 13:罕见时钟路径的问题

在上图中，复位状态机（R 触发器）在 default_clk 上工作。此外，在复位去断言过程中，default_clk 是 sys clk 的源。因此，在逻辑上，这两个时钟（clk1 和 clk2）在复位去断言过程中同步。但是，由于 clk1 和 clk2 之间存在巨大的罕见路径，因此很难平衡这两个时钟并视其为同步。 因此，满足 A 触发器的复位去断言变得具有挑战性。

2. 解决方案

异步对待 clk1 和 clk2，并在 A 触发器中使用复位之前放置复位同步器。现在需要从 S2-->A 满足复位去断言时序（见图 14）。这不应是个问题。

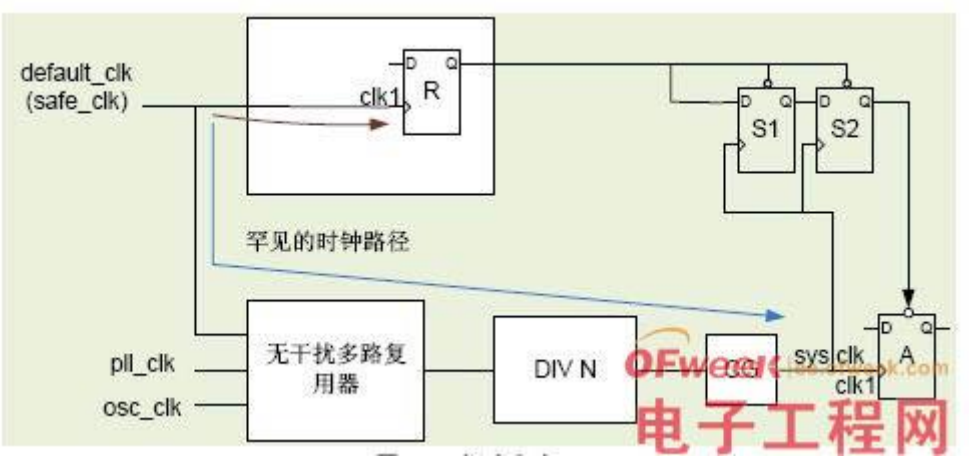

图 14:解决方案

结束语

这部分主要专注于复位设计中的故障以及克服这些问题的可能的解决方案。然而，上述解决方案并非唯一的解决方案，也不普遍适用于所有设计。这些是一些通用的解决方案和建议的指导方针，在特殊情况下可能需要进行修改。在这些情况下，此类问题不仅导致功能故障，还会增加一些额外的调试时间和工作，从而增加执行周期时间。因此，设计人员需要在设计的早期阶段考虑此类问题。