Clock Domain Crossing

跨时钟域处理 3 大方法揭秘

作者：开源骚客工作室

跨时钟域处理是 FPGA 设计中经常遇到的问题，而如何处理好跨时钟域间的数据，可以说是每个 FPGA 初学者的必修课。**如果是还在校的本科生，跨时钟域处理也是面试中经常常被问到的一个问题。**

在本篇文章中，主要介绍 3 种跨时钟域处理的方法，这 3 种方法可以说是 FPGA 界最常用也最实用的方法，这三种方法包含了**单 bit 和多 bit 数据的跨时钟域处理**，学会这 3 招之后，对于 FPGA 相关的跨时钟域数据处理便可以手到擒来。

本文介绍的 3 种方法跨时钟域处理方法如下：

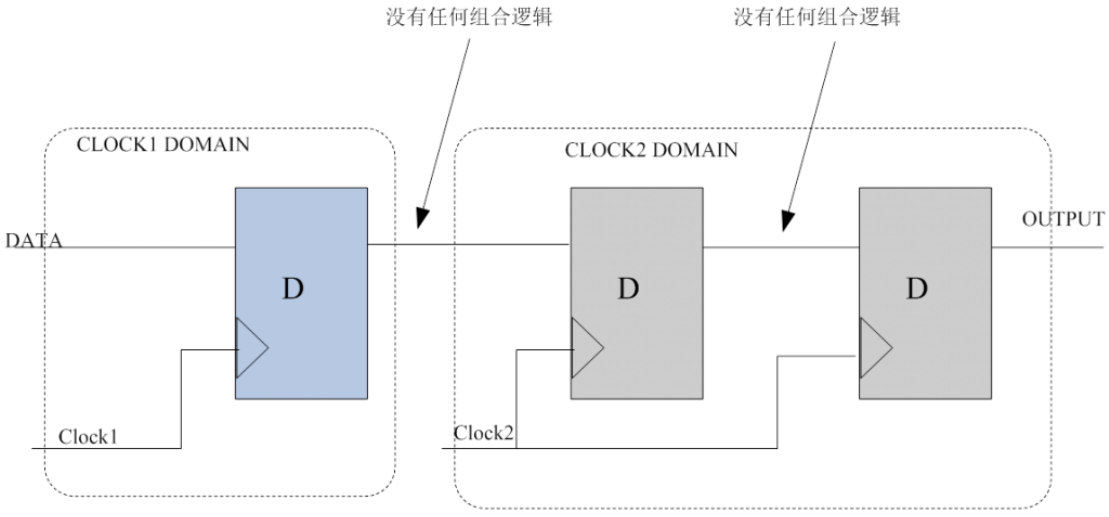
1. **打两拍；**
2. **异步双口 RAM；**
3. **格雷码转换。**

Clock Domain Crossing

第一种方法：打两拍

大家很清楚，处理跨时钟域的数据有单 bit 和多 bit 之分，而打两拍的方式常见于处理单 bit 数据的跨时钟域问题。

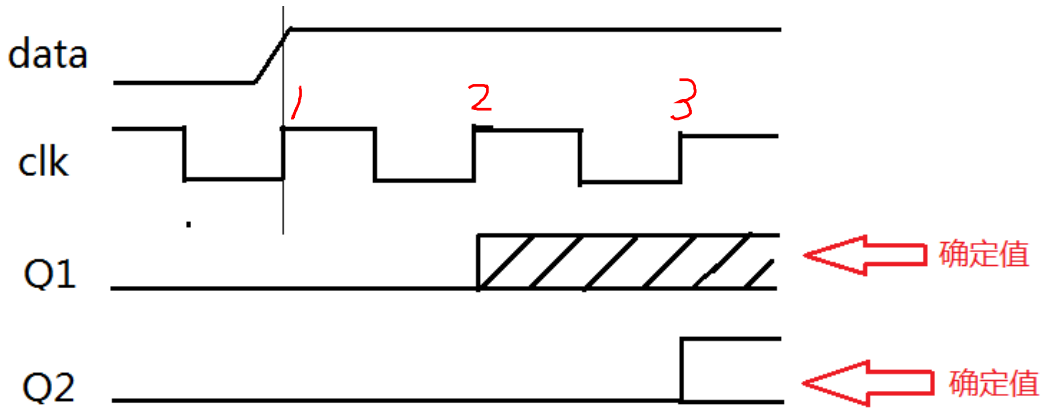
打两拍的方式，其实说白了，**就是定义两级寄存器，对输入的数据进行延拍。**如下图所示。



应该很多人都会问，**为什么是打两拍呢，打一拍、打三拍行不行呢？**

先简单说下两级寄存器的原理：**两级寄存是一级寄存的平方，两级并不能完全消除亚稳态危害，但是提高了可靠性减少其发生概率。总的来讲，就是一级概率很大，三级改善不大。**

这样说可能还是有很多人不够完全理解，那么请看下面的时序示意图：



data 是时钟域 1 的数据，需要传到时钟域 2（clk）进行处理，寄存器 1 和寄存器 2 使用的时钟都为 clk。假设在 clk 的上升沿正好采到 data 的跳变沿（从 0 变 1 的上升沿，实际上的数据跳变不可能是瞬时的，所以有短暂的跳变时间），那这时作为寄存器 1 的输入到底应该是 0 还是 1 呢？这是一个不确定的问题。所以 Q1 的值也不能确定，但至少可以保证，在 clk 的下一个上升沿，Q1 基本可以满足第二级寄存器的保持时间和建立时间要求,出现亚稳态的概率得到了很大的改善。

如果再加上第三级寄存器，由于第二级寄存器对于亚稳态的处理已经起到了很大的改善作用，第三级寄存器在很大程度上可以说只是对于第二级寄存器的延拍，所以意义是不大的。

可能对于这部分的解释不是很到位，不过还是希望大家能够多思考一下，欢迎大家批评指正。

第二种方法：异步双口 RAM

处理多 bit 数据的跨时钟域，一般采用异步双口 RAM。假设我们现在有一个信号采集平台，ADC 芯片提供源同步时钟 60MHz，ADC 芯片输出的数据在 60MHz 的时钟上升沿变化，而 FPGA 内部需要使用 100MHz 的时钟来处理 ADC 采集到的数据（多 bit）。

在这种类似的场景中，我们便可以使用异步双口 RAM 来做跨时钟域处理。先利用 ADC 芯片提供的 60MHz 时钟将 ADC 输出的数据写入异步双口 RAM，然后使用 100MHz 的时钟从 RAM 中读出。

对于使用异步双口 RAM 来处理多 bit 数据的跨时钟域，相信大家还是可以理解的。**当然，在能使用异步双口 RAM 来处理跨时钟域的场景中，也可以使用异步 FIFO 来达到同样的目的。**

第三种方法：格雷码转换

对于第三种方法，小编在大学里边从没接触过，也是在工作中才接触到。

我们依然继续使用介绍第二种方法中用到的 ADC 例子，将 ADC 采样的数据写入 RAM 时，需要产生 RAM 的写地址，但我们读出 RAM 中的数据时，肯定不是一上电就直接读取，而是要等 RAM 中有 ADC 的数据之后才去读 RAM。这就需要 100MHz 的时钟对 RAM 的写地址进行判断，当写地址大于某个值之后再去读取 RAM。

在这个场景中，其实很多人都是使用直接用 100MHz 的时钟于 RAM 的写地址进行打两拍的方式，但 RAM 的写地址属于多 bit，如果单纯只是打两拍，那不一定能确保写地址数据的每一个 bit 在 100MHz 的时钟域变化都是同步的，肯定有一个先后顺序。**如果在低速的环境中不一定会出错，在高速的环境下就不一定能保证了。所以更为妥当的一种处理方法就是使用格雷码转换。**

对于格雷码，相邻的两个数间只有一个 bit 是不一样的（格雷码，在本文中不作详细介绍），如果先将 RAM 的写地址转为格雷码，然后再将写地址的格雷码进行打两拍，之后再在 RAM 的读时钟域将格雷码恢复成 10 进制。这种处理就相当于对单 bit 数据的跨时钟域处理了。

对于格雷码与十进制互换的代码，仅提供给大家作参考：

```
// GRAY_W
function [WADDRWIDTH:0] GRAY_W(input op, input [WADDRWIDTH:0] addr_in);
    integer i;
    begin
        if(op==ENCODE)
            GRAY_W = (addr_in >> 1) ^ addr_in;
        else if(op==DECODE) begin
            GRAY_W[WADDRWIDTH] = addr_in[WADDRWIDTH];
            for(i=WADDRWIDTH-1;i>=0;i=i-1)
               GRAY_W[i] = GRAY_W[i+1] ^ addr_in[i];
        end
    end
endfunction

// GRAY_R
function [RADDRWIDTH:0] GRAY_R(input op, input [RADDRWIDTH:0] addr_in);
    integer i;
    begin
        if(op==ENCODE)
            GRAY_R = (addr_in >> 1) ^ addr_in;
        else if(op==DECODE) begin
            GRAY_R[RADDRWIDTH] = addr_in[RADDRWIDTH];
            for(i=RADDRWIDTH-1;i>=0;i=i-1)
               GRAY_R[i] = GRAY_R[i+1] ^ addr_in[i];
        end
    end
endfunction
```

代码使用的是函数的形式，方便调用，**op 表示编码或者译码，WADDRWIDTH 和 RADDRWIDTH 表示位宽。**

特别提醒#1

邓堪文博客将于【开源骚客】公众号优势互补为大家分享 FPGA 相关知识。

邓堪文博客：http://dengkanwen.com

【开源骚客（微信号：OpenSoc）】公众号: