FPGA/CPLD 数字电路设计经验分享

摘要：在数字电路的设计中，时序设计是一个系统性能的主要标志，在高层次设计方法中，对时序控制的抽象度也相应提高，因此在设计中较难把握，但在理解 RTL 电路时序模型的基础上，采用合理的设计方法在设计复杂数字系统是行之有效的，通过许多设计实例证明采用这种方式可以使电路的后仿真通过率大大提高，并且系统的工作频率可以达到一个较高水平。

关键词：FPGA 数字电路 时序 时延路径 建立时间 保持时间

1 数字电路设计中的几个基本概念：

1.1 建立时间和保持时间：

建立时间（setup time）是指在触发器的时钟信号上升沿到来以前，数据稳定不变的时间，如果建立时间不够，数据将不能在这个时钟上升沿被打入触发器；保持时间（hold time）是指在触发器的时钟信号上升沿到来以后，数据稳定不变的时间， 如果保持时间不够，数据同样不能被打入触发器。 如图 1 。 数据稳定传输必须满足建立和保持时间的要求，当然在一些情况下，建立时间和保持时间的值可以为零。 PLD/FPGA 开发软件可以自动计算两个相关输入的建立和保持时间（如图 2）

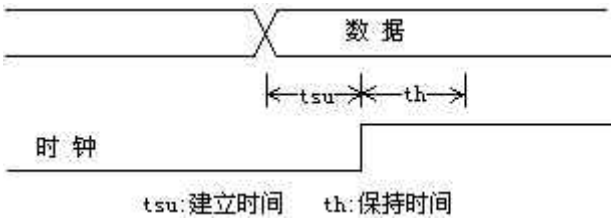


图 1 建立时间和保持时间关系图

注：

- 在考虑建立保持时间时，应该考虑时钟树向后偏斜的情况，在考虑建立时间时应该考虑时钟树向前偏斜的情况。在进行后仿真时，最大延迟用来检查建立时间，最小延时用来检查保持时间。
- 建立时间的约束和时钟周期有关，当系统在高频时钟下无法工作时，降低时钟频率就可以使系统完成工作。保持时间是一个和时钟周期无关的参数，如果设计不合理，使得布局布线工具无法布出高质量的时钟树，那么无论如何调整时钟频率也无法达到要求，只有对所设计系统作较大改动才有可能正常工作，导致设计效率大大降低。因此合理的设计系统的时序是提高设计质量的关键。在可编程器件中，时钟树的偏斜几乎可以不考虑，因此保持时间通常都是满足的。

1.2 FPGA中的竞争和冒险现象

信号在 FPGA 器件内部通过连线和逻辑单元时，都有一定的延时。延时的大小与连线的长短和逻辑单元的数目有关，同时还受器件的制造工艺、工作电压、温度等条件的影响。信号的高低电平转换也需要一定的过渡时间。由于存在这两方面因素，多路信号的电平值

发生变化时，在信号变化的瞬间，组合逻辑的输出有先后顺序，并不是同时变化,往往会出现一些不正确的尖峰信号，这些尖峰信号称为"毛刺"。如果一个组合逻辑电路中有"毛刺"出现，就说明该电路存在"冒险"。（与分立元件不同，由于 PLD 内部不存在寄生电容电感，这些毛刺将被完整的保留并向下一级传递，因此毛刺现象在 PLD、FPGA 设计中尤为突出）图 2 是一个逻辑冒险的例子，从图 3 的仿真波形可以看出，"A、B、C、D"四个输入信号经过布线延时以后，高低电平变换不是同时发生的，这导致输出信号"OUT"出现了毛刺。（我们无法保证所有连线的长度一致，所以即使四个输入信号在输入端同时变化，但经过 PLD 内部的走线，到达或门的时间也是不一样的，毛刺必然产生）。可以概括的讲，只要输入信号同时变化，（经过内部走线）组合逻辑必将产生毛刺。 将它们的输出直接连接到时钟输入端、清零或置位端口的设计方法是错误的，这可能会导致严重的后果。 所以我们必须检查设计中所有时钟、清零和置位等对毛刺敏感的输入端口，确保输入不会含有任何毛刺

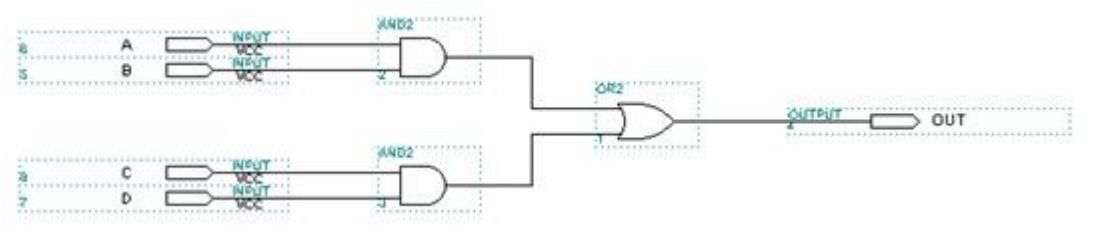


图 2 存在逻辑冒险的电路示例

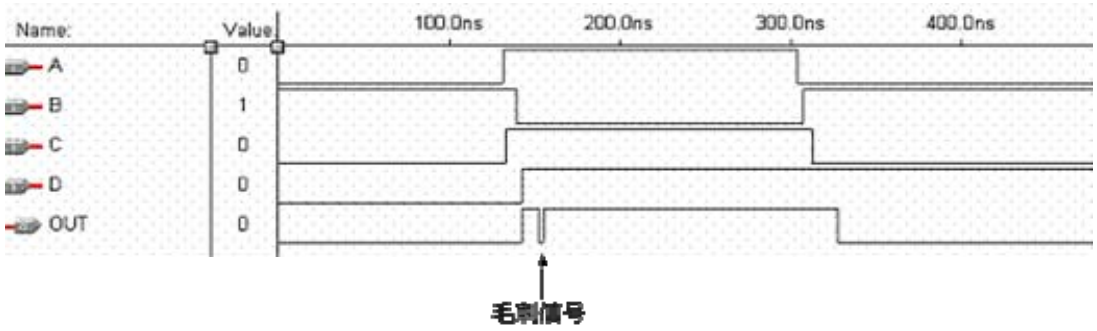


图 3 图 2 所示电路的仿真波形

冒险往往会影响到逻辑电路的稳定性。时钟端口、清零和置位端口对毛刺信号十分敏感，任何一点毛刺都可能会使系统出错，因此判断逻辑电路中是否存在冒险以及如何避免冒险是设计人员必须要考虑的问题。

如何处理毛刺

我们可以通过改变设计，破坏毛刺产生的条件，来减少毛刺的发生。例如，在数字电路设计中，常常采用格雷码计数器取代普通的二进制计数器，这是因为格雷码计数器的输出每次只有一位跳变，消除了竞争冒险的发生条件，避免了毛刺的产生。

毛刺并不是对所有的输入都有危害，例如 D 触发器的 D 输入端，只要毛刺不出现在时钟的上升沿并且满足数据的建立和保持时间，就不会对系统造成危害，我们可以说 D 触发器的 D 输入端对毛刺不敏感。 根据这个特性，我们应当在系统中尽可能采用同步电路，这

是因为同步电路信号的变化都发生在时钟沿，只要毛刺不出现在时钟的沿口并且不满足数据的建立和保持时间，就不会对系统造成危害。 （由于毛刺很短，多为几纳秒，基本上都不可能满足数据的建立和保持时间）

去除毛刺的一种常见的方法是利用 D 触发器的 D 输入端对毛刺信号不敏感的特点，在输出信号的保持时间内，用触发器读取组合逻辑的输出信号，这种方法类似于将异步电路转化为同步电路。 图 4 给出了这种方法的示范电路，图 5 是仿真波形。

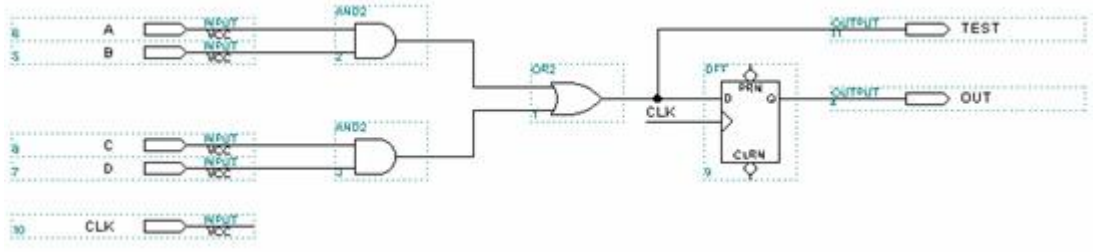


图 4 消除毛刺信号方法之二

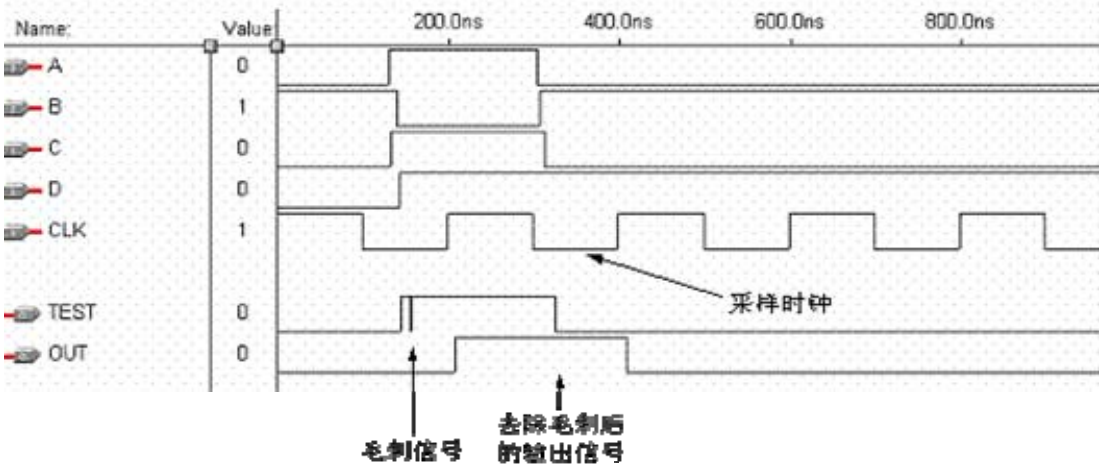


图 5 图 4 所示电路的仿真波形

如前所述，优秀的设计方案，如采用格雷码计数器，同步电路等，可以大大减少毛刺，但它并不能完全消除毛刺。 毛刺并不是对所有输入都有危害，例如 D 触发器的 D 输入端，只要毛刺不出现在时钟的上升沿并且满足数据的建立和保持时间，就不会对系统造成危害。因此我们可以说 D 触发器的 D 输入端对毛刺不敏感。但对于 D 触发器的时钟端，置位端，清零端，则都是对毛刺敏感的输入端，任何一点毛刺就会使系统出错，但只要认真处理，我们可以把危害降到最低直至消除。下面我们就对几种具体的信号进行探讨。

1.3 清除和置位信号

在 FPGA 的设计中，全局的清零和置位信号必须经过全局的清零和置位管脚输入，因为他们也属于全局的资源，其扇出能力大，而且在 FPGA 内部是直接连接到所有的触发器的置位和清零端的，这样的做法会使芯片的工作可靠、性能稳定，而使用普通的 IO 脚则不能保证该性能。

在 FPGA 的设计中，除了从外部管脚引入的全局清除和置位信号外在 FPGA 内部逻辑的处理中也经常需要产生一些内部的清除或置位信号。清除和置位信号要求象对待时钟那样小心地考虑它们，因为这些信号对毛刺也是非常敏感的。

在同步电路设计中，有时候可以用同步置位的办法来替代异步清 0。在用硬件描述语言的设计中可以用如下的方式来描述：

异步清 0 的描述方法：

```
process(rst,clk)
begin
  if rst='1' then
     count<=(others=>'0');
  elsif clk'event and clk='1' then
     count<=count+ 1;
  end if;
end process;
```

同步清 0 的描述方法：

```
process
begin
  wait until clk'event and clk='1';
     if rst='1' then
        count<=(others=>'0');
     else
        count<=count+ 1;
     end if;
end process;
```

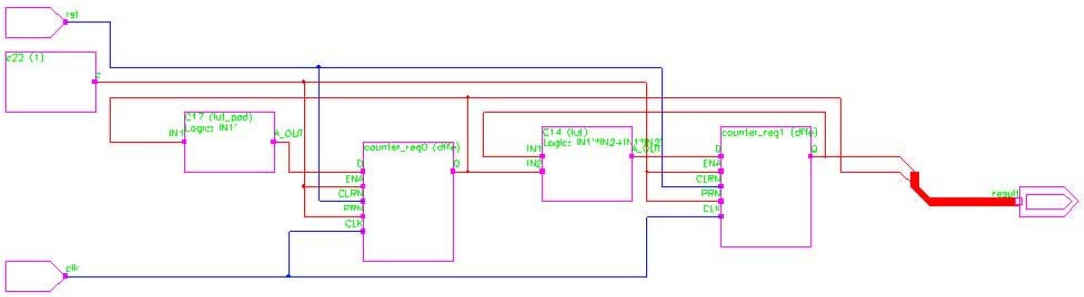


图 6 异步清 0、置位逻辑图

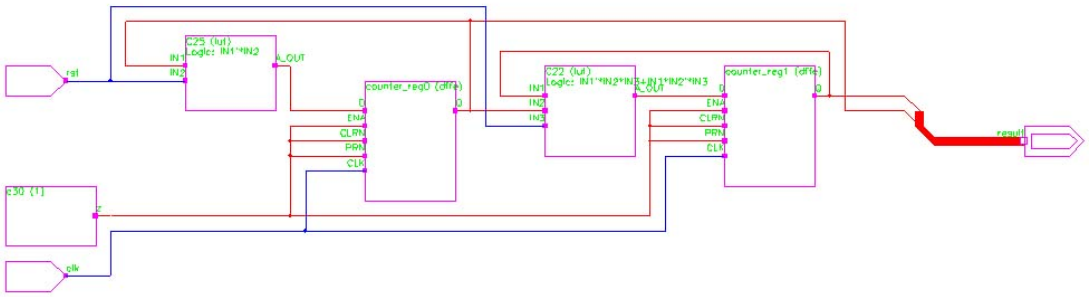


图 7 同步清 0、置位关系图

1.4 触发器和所存器：

我们知道，触发器是在时钟的沿进行数据的锁存的，而所存器是用电平使能来锁存数据的。所以触发器的 Q 输出端在每一个时钟沿都会被更新，而所存器只能在使能电平有效器件才会被更新。在 FPGA 设计中建议如果不是必须那么应该尽量使用触发器而不是所存器。

那么在使用硬件描述语言进行电路设计的时候如何区分触发器和所存器的描述方法哪？其实有不少人在使用的过程中可能并没有特意区分过，所以也忽略了二者在描述方法上的区别。下面是用 VHDL 语言描述的触发器和所存器以及综合器产生的电路逻辑图。

触发器的语言描述：

```
process
begin
  wait until clk'event and clk='1';
     q<=d;
  end process;
```

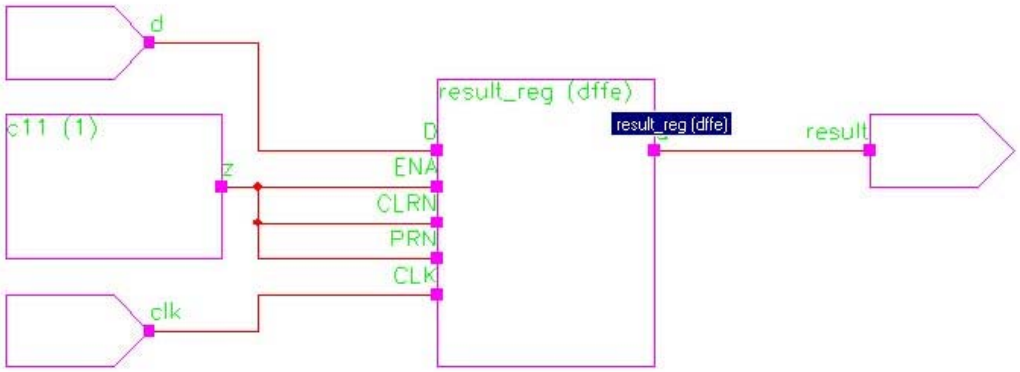


图 触发器

所存器的语言描述：

```
process(en,d)
begin
  if en='1' then
       q<=d;
  end if;
end process;
```

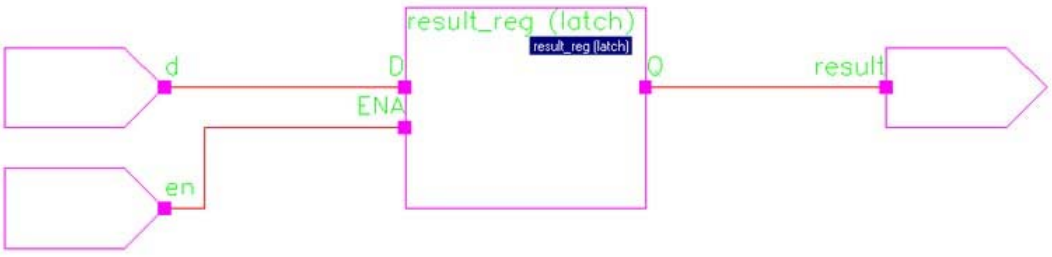


图 所存器

由上述对 Latch 的描述可见，其很容易于选择器的描述相混淆，用 VHDL 语言对选择器的描述方法如下：

```
process(en,a,b)
begin
  if en='1' then
      q<=a;
  else
      q<=b;
  end if;
end process;
```

2 FPGA/CPLD 中的一些设计方法

2.1 FPGA 设计中的同步设计

异步设计不是总能满足(它们所馈送的触发器的)建立和保持时间的要求。因此，异步输入常常会把错误的数据锁存到触发器，或者使触发器进入亚稳定的状态,在该状态下，触发器的输出不能识别为 1 或 0。如果没有正确地处理，亚稳性会导致严重的系统可靠性问题。

另外，在 FPGA 的内部资源里最重要的一部分就是其时钟资源（全局时钟网络），它一般是经过 FPGA 的特定全局时钟管脚进入 FPGA 内部，后经过全局时钟 BUF 适配到全局时钟网络的，这样的时钟网络可以保证相同的时钟沿到达芯片内部每一个触发器的延迟时间差异是可以忽略不计的。

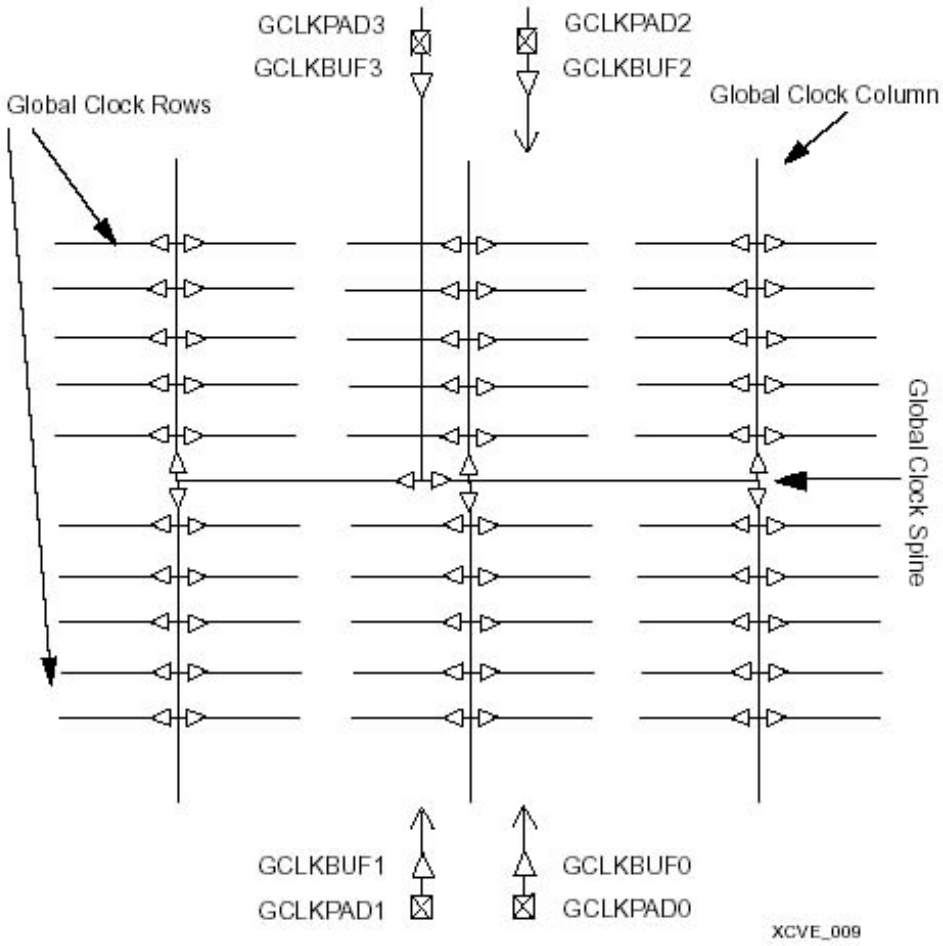


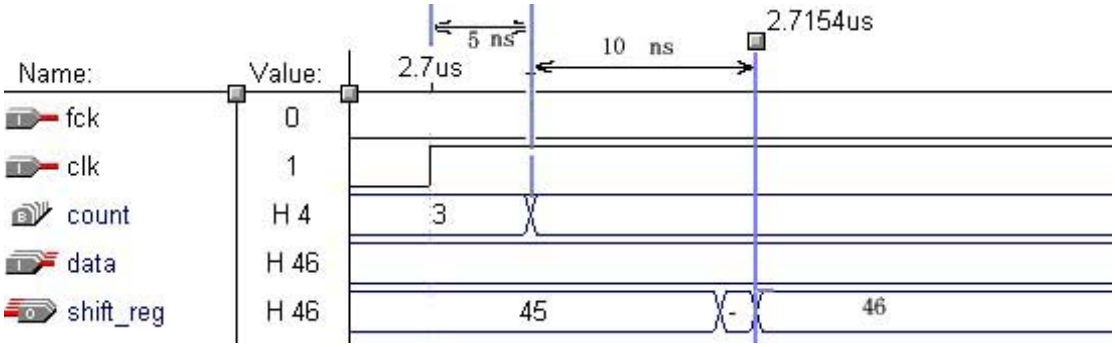
Figure 9: Global Clock Distribution Network

在 FPGA 中上述的全局时钟网络被称为时钟树，无论是专业的第三方工具还是器件厂商提供的布局布线器在延时参数提取、分析的时候都是依据全局时钟网络作为计算的基准的。如果一个设计没有使用时钟树提供的时钟，那么这些设计工具有的会拒绝做延时分析有的延时数据将是不可靠的。

在我们日常的设计中很多情形下会用到需要分频的情形，好多人的做法是先用高频时钟计数，然后使用计数器的某一位输出作为工作时钟进行其他的逻辑设计。其实这样的方法是不规范的。比如下面的描述方法：

```
process
begin
  wait until clk'event and clk='1';
    if fck='1' then
       count<=(others=>'0');
    else
       count<=count+ 1;
    end if;
end process;

process
begin
    wait until count(2)'event and count(2)='1' ;
       shift_reg<=data;
end process;
```

在上述的第一个 process 电路描述中，首先计数器的输出结果（count(2)）相对于全局时钟 clk 已经产生了一定的延时（延时的大小取决于计数器的位数和所选择使用的器件工艺）；而在第二个 process 中使用计数器的 bit2 作为时钟，那么 shift_reg 相对于全局 clk 的延时将变得不好控制。布局布线器最终给出的时间分析也是不可靠的。这样产生的结果波形仿真如下图所示：



正确的做法可以将第二个 process 这样来写。

```
process
begin
    wait until clk'event and clk='1' ;
        if count(2 downto 0)="000" then
            shift_reg<=data;
        end if;
end process;
```

或者分成两步来写：

```
process(count)
begin
    if count(2 downto 0)="000" then
        en<='1';
    else
        en<='0';
    end if;
end process;

process
begin
    wait until clk'event and clk='1' ;
        if en='1' then
            shift_reg<=data;
        end if;
end process;
```

这样做是相当于产生了一个 8 分频的使能信号，在使能信号有效的时候将 data 数据采样到 shift_reg 寄存器中。但此种情形下 shift_reg 的延时是相对于全局时钟 clk 的。下面的图形更能看得清楚。

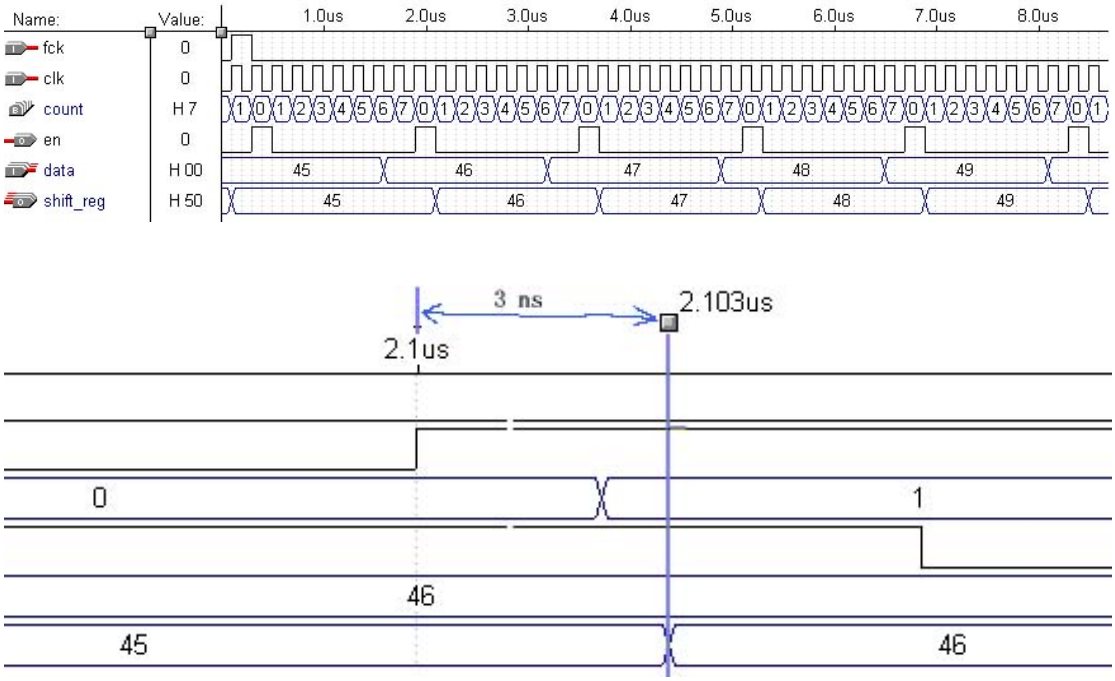


图 上图中波形的局部放大

2.2 FPGA 设计中的延时电路的产生：

在日常的电路设计中，有时候我们需要对信号进行延时处理来适应对外接口的时序关系，最经常也是最典型的情况是做处理机的接口；因为与处理的接口时序关系是异步的，而一个规范的 FPGA 设计应该是尽可能采用同步设计。那么遇到这种情况该如何处理呢？

首先在 FPGA 中要产生延时，信号必须经过一定的物理资源。在硬件描述语言中有关键词 Wait for xx ns，需要说明的是该语法是仅仅用于仿真而不能用于综合的，可综合的延时方法有：

- 使信号经过逻辑门得到延时（如非门）；
- 使用器件提供的延时单元（如 Altera 公司的 LCELL，Xilinx 公司的）；

注意：当使用多级非门的时候综合器往往会将其优化掉，因为综合器会认为一个信号非两次还是它自己。

需要说明的是在 FPGA/CPLD 内部结构是一种标准的宏单元，下图是 Xilinx 公司的 Spartans II 系列器件的一个标准宏单元。虽然不同的厂家的芯片宏单元的结构不同，但概括而言都是由一些组合逻辑外加一或二个触发器而构成。在实际应用中，当一个模块内的组合逻辑被使用了那么与其对应的触发器也就不能用了；同样如果触发器单元被用了那么组合逻辑单元也就废了。这就是有时候（特别是使用 CPLD）虽然设计使用的资源并不多但布局布线器却报告资源不够使用的原因。

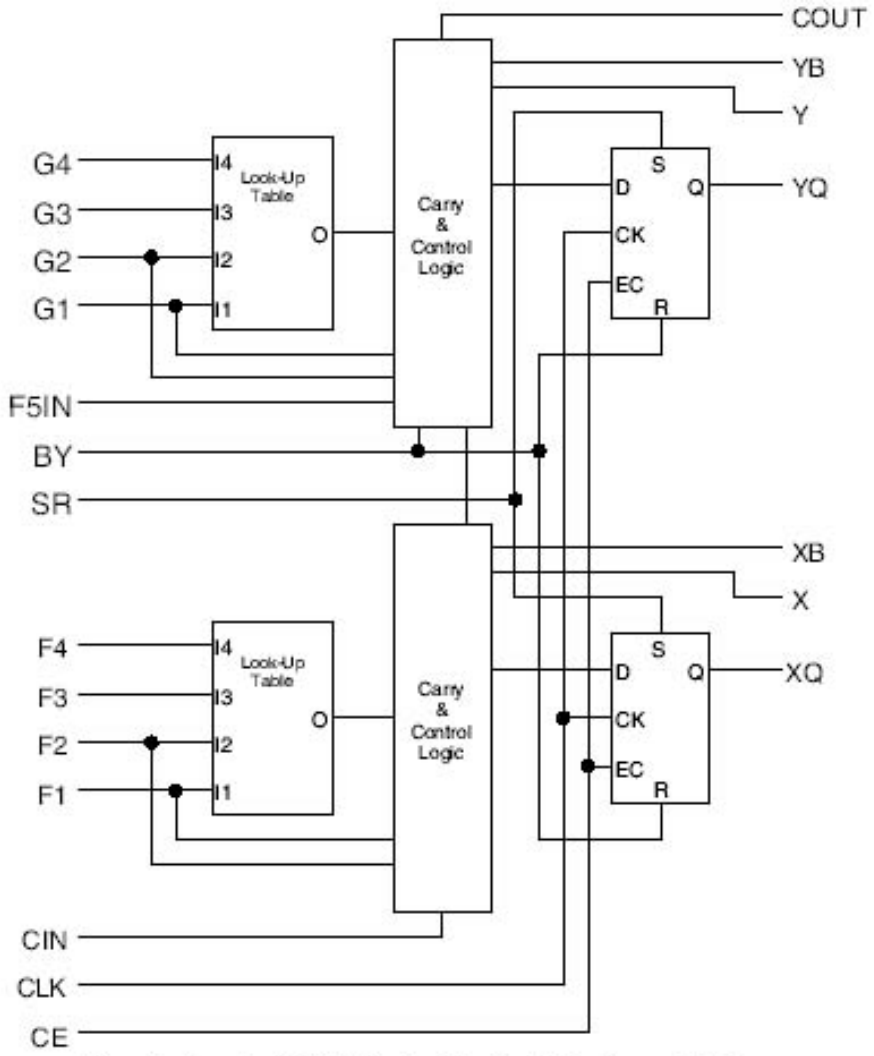
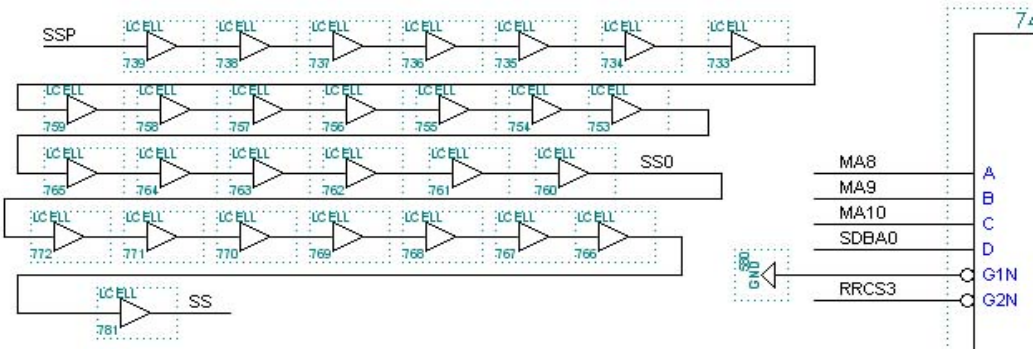


Figure 4: Spartan-II CLB Slice (two identical slices in each CLB)

现面的一个例子是前一段时间我在公司遇到的一个设计。设计使用 Altera 公司的 EPM7256 型号的 CPLD。该设计实际使用的寄存器资源只有 109 个，占整个器件资源的 42%。可是该设计使用了如下图所示的延时方法来做处理器接口的时序：



在该电路的设计中使用了大量的 LCELL 来产生 100 多纳秒的延时，这样做的后果是虽然整个电路的触发器资源只使用了 42%，可是用 MaxplusII 进行布局布线已经不能够通过了。而且我怀疑经过这么多逻辑的延时后所产生的信号还能保持原来的性能不。

当需要对某一信号作一段延时时，初学者往往在此信号后串接一些非门或其它门电路，此方法在分离电路中是可行的。但在ＦＰＧＡ中，开发软件在综合设计时会将这些门当作冗余逻辑去掉，达不到延时的效果。用 ALTERA 公司的 MaxplusII 开发ＦＰＧＡ时，可以通过插入一些ＬＣＥＬＬ原语来产生一定的延时，但这样形成的延时在ＦＰＧＡ芯片中并不稳定，会随温度等外部环境的改变而改变，因此并不提倡这样做。在此，可以用高频时钟来驱动一移位寄存器，待延时信号作数据输入，按所需延时正确设置移位寄存器的级数，移位寄存器的输出即为延时后的信号。此方法产生的延时信号与原信号比有误差，误差大小由高频时钟的周期来决定。对于数据信号的延时，在输出端用数据时钟对延时后信号重新采样，就可以消除误差。

对于这样大的延时我建议的实现方法是采用时钟锁存来产生延时的方法，我们知道当一个信号用时钟锁存一次，将会占用一个触发器资源，信号会向后推移一个时钟周期；该同事的设计里 CPLD 芯片正好连接有 32MHz 的时钟，那么每用时钟锁存一次 ssp 信号就会推移 31ns，这样只需多使用 3 个触发器资源就可以达到目的了。电路图和仿真波形如下图所示：当然这样做对原来信号高低电平的宽度会稍有改变，但只要是在与其接口的芯片的容许范围之内就不会影响到功能的实现。

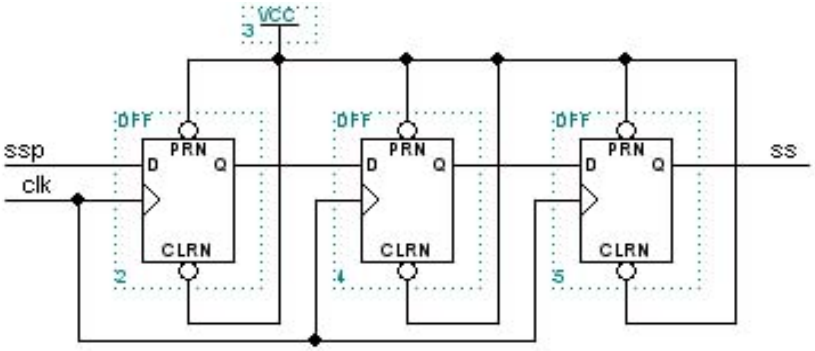
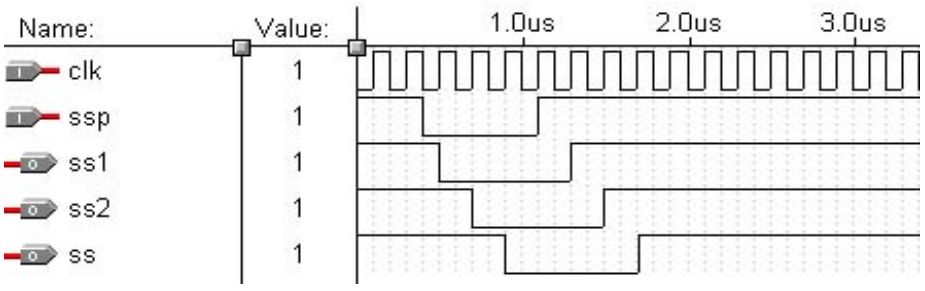


图 用于延时的电路图

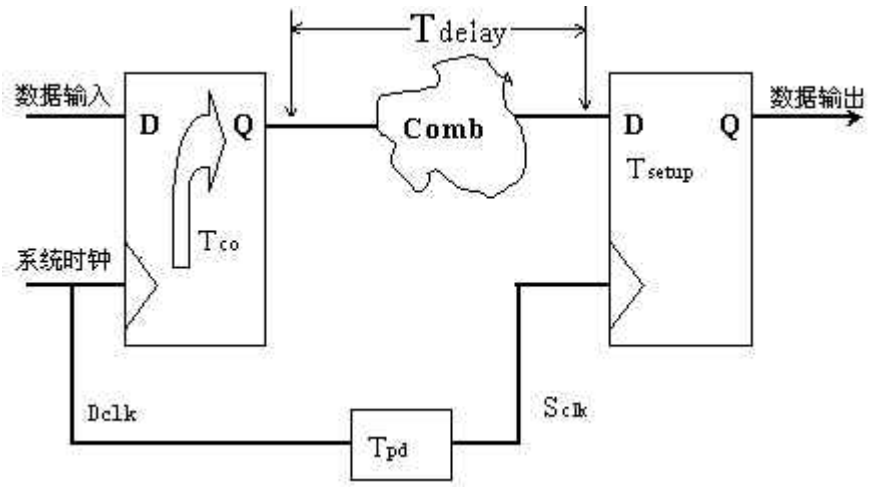


上图仿真波形

2.3 如何提高系统的运行速度

同步电路的速度是指同步时钟的速度。同步时钟愈快，电路处理数据的时间间隔越短，电路在单位时间处理的数据量就愈大.

我们先来看一看同步电路中数据传递的一个基本模型，如下图：



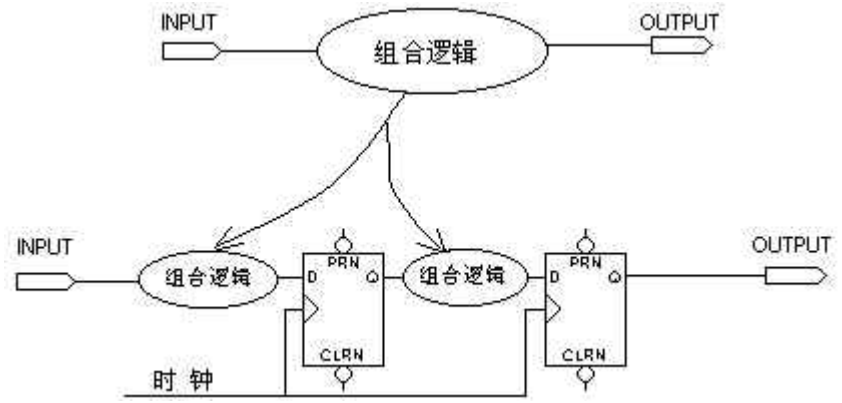
（Tco 是触发器时钟到数据输出的延时；Tdelay 是组合逻辑的延时；Tsetup 是触发器的建立时间）

假设数据已经被时钟的上升沿打入 D 触发器，那么数据到达第一个触发器的 Q 端需要 Tco，再经过组合逻辑的延时 Tdelay 到达的第二个触发器的 D 端，要想时钟能在第二个触发器再次被稳定的锁入触发器，则时钟的延迟不能晚于 Tco+Tdelay+Tsetup，（我们可以回顾一下前面讲过的建立和保持时间的概念，就可以理解为什么公式最后要加上一个 Tdelay） 由以上分析可知：最小时钟周期：T=Tco+Tdelay+Tsetup 最快时钟频率 F= 1/T PLD 开发软件也正是通过这个公式来计算系统运行速度 Fmax

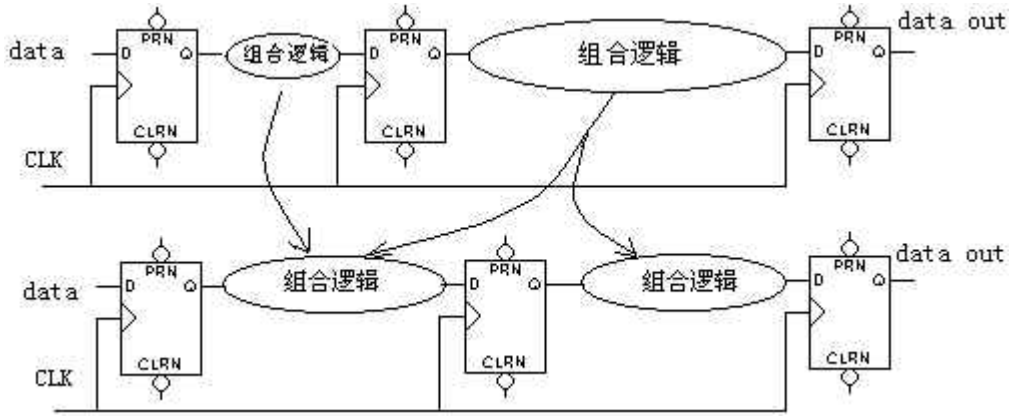
注：在这个逻辑图中有个参数：Tpd ,即时钟的延时参数，我们在刚才做时间分析的时候，没有提这个参数，（如果使用 PLD 的全局时钟型号，Tpd 可以为 0，如果是普通时钟，则不为 0）。所以如果考虑到时钟的延时，精确的公式应该是 T=Tco+Tdelay+Tsetup-Tpd。当然以上全部分析的都是器件内部的运行速度，如果考虑芯片 I/O 管脚延时对系统速度的影响，那么还需要加一些修正。

由于 Tco、Tsetup 是由具体的器件和工艺决定的，我们设计电路时只可以改变 Tdelay。所以缩短触发器间组合逻辑的延时是提高同步电路速度的关键。由于一般同步电路都不止一级锁存（如图 3），而要使电路稳定工作，时钟周期必须满足最大延时要求，缩短最长延时路径，才可提高电路的工作频率。

如图 2 所示：我们可以将较大的组合逻辑分解为较小的几块，中间插入触发器，这样可以提高电路的工作频率。这也是所谓“流水线”（pipelining）技术的基本原理。



对于图 3 的上半部分，它时钟频率受制于第二个较大的组合逻辑的延时，通过适当的方法平均分配组合逻辑，可以避免在两个触发器之间出现过大的延时，消除速度瓶颈。



FPGA/CPLD 开发软件中也有一些参数设置，通过修改这些设置，可以提高编译/布局布线后系统速度，但是根据经验这种速度的提高是很有限的，假如按照要求我们需要设计一个可以工作到 50MHz 的系统，实际布局布线器报告出来的 Fmax 只有 40MHz，此时如果我们使用布局布线器的设置选项最多可以提高到 45MHz，这还是运气比较好的情况。而且你必须了解这些选项的含义、使用背景等。

其实在一个设计里影响速度的瓶颈经常只会有几条，我们将延时最大的路径称作关键路径。当设计的运行速度不符合系统设计要求的时候我们可以首先找到不能满足要求的关键路径，按照上述的方法将关键路径上的组合逻辑拆分成多个中间用触发器隔开，这样很容易就可以从根本上提升系统的运行速度了。

有的设计在设计开始就知道那部分电路会产生比较大的组合逻辑，导致速度瓶颈的产生，那么就应该在开始就想好解决办法。比如现在设计需要产生一个 32 位的加法器，并且要求能够工作在 50MHz。根据经验直接用 32 位加法器肯定是达不到 50MHz 的要求的，这时我们可以将其分成 3 个 12 位计数器来操作，后面的计数器只要将前面计数器结果的高位(进位位）相加就可以了。

下面是原来在宽带接入服务器设计中的流量统计单元中的 32 位加法器的描述：

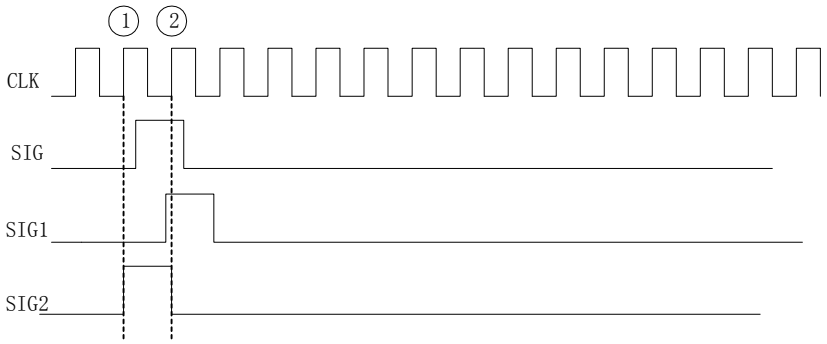
```
----     flow count element
------------------------------------------------------------
-----temporary computing 12 bits adder
process(Count_0_en,count_buffer,Len,Carry_0_0,Carry_0_1)
begin
    case Count_0_en is
        ---1st Step addition       (10 downto 0) + (10 downto 0)
        when "001"    => add_12_a_0 <= ('0' & count_buffer(0)(10 downto 0));
                         add_12_b_0 <= ('0' & Len(10 downto 0));
        ---2nd Step addition       (21 downto 11) + Carry_0_0
        when "010"    => add_12_a_0 <= ('0' & count_buffer(0)(21 downto 11));
                         add_12_b_0 <= ("00000000000" & Carry_0_0);
        ---3rd Step addition       (31 downto 22) + Carry_0_1
        when "100"    => add_12_a_0 <= ("00" & count_buffer(0)(31 downto 22));
                         add_12_b_0 <= ("00000000000" & Carry_0_1);
        when others   => add_12_a_0 <= (others=>'X') ;
                         add_12_b_0 <= (others=>'X') ;
    end case;
end process;

------12 bits adder
add_12_result_0 <=  add_12_a_0 + add_12_b_0;
------Bytes Count
process(RST,CLK_25MHz,IO,OE_bar,data_sel,Count_0_en)
begin
  if(RST = '1')then     -----system Reset
    count_buffer(0) <= (others => '0');
    Carry_0_0          <= '0';
    Carry_0_1          <= '0';
    Carry_0_2          <= '0';
  elsif(CLK_25MHz'event and CLK_25MHz = '0')then
    if(OE_bar = '0' and data_sel = '0')then
      count_buffer(0) <= IO;
      Carry_0_2        <= '0';
    else
      case Count_0_en is
                ---1st Step addition       (10 downto 0) + (10 downto 0)
        when "001"    => count_buffer(0)(10 downto 0) <= add_12_result_0(10 downto 0);
                         Carry_0_0      <= add_12_result_0(11);--first step carry
                ---2nd Step addition       (21 downto 11) + Carry_0_0
        when "010"    => count_buffer(0)(21 downto 11) <= add_12_result_0(10 downto 0);
                         Carry_0_1      <= add_12_result_0(11);--Second step carry
                ---3rd Step addition       (31 downto 22) + Carry_0_1
        when "100"    => count_buffer(0)(31 downto 22) <= add_12_result_0(9 downto 0);
                         Carry_0_2      <= add_12_result_0(10);--Third step carry
        when others =>   Carry_0_2      <= '0';
      end case;
    end if;
  end if;
end process;
```

2.4 信号输出

当你需要将 FPGA/CPLD 内部的信号通过管脚输出给外部相关器件的时候，如果不影响功能最好是将这些信号通过用时钟锁存后输出。因为通常情况下一个板子是工作于一种或

两种时钟模式下，与 FPGA/CPLD 相连接的芯片的工作时钟大多数情形下与 FPGA 的时钟同源，如果输出的信号经过时钟锁存可以起到如下的作用：

- 容易满足芯片间信号连接的时序要求；
- 容易满足信号的建立保持时间；



如上图所示，比如 FPGA/CPLD 在 CLK 的时钟沿 1 锁存一个信号得到 SIG 所示的波形，SIG 信号需要给另外的一个与其接口的芯片，那么该芯片将**一定会**在 CLK 的时钟沿 2 正确采样到 SIG 信号。但是如果该信号在 FPGA/CPLD 中输出的时候不是用时钟沿锁存的，那将有可能出现 SIG1/SIG2 所示的时序关系，则与其接口的芯片在时钟沿 2 处采样该信号的时候有可能出现建立保持时间不满足要求而出现采样不可靠、沿打沿等情况。另外通过组合逻辑输出还有可能出现毛刺的情况。所有这些不规范的设计都会引起系统工作时的不可靠、不稳定的情形。

2.5 寄存异步输入信号

我们在日常的设计工作中，FPGA/CPLD 总是要与别的芯片相连接的，FPGA/CPLD 会给别的芯片输出信号，同时也要处理别的芯片送来的信号，这些信号往往对 FPGA/CPLD 内部的时钟系统而言是异步的，为了可靠的采样到这些输入信号，建议将这些输入信号使用相应的时钟锁存后在处理，这样做：

- 将原来的异步信号转化成同步来处理；
- 去除输入信号中的毛刺（特别是对于数据总线）；

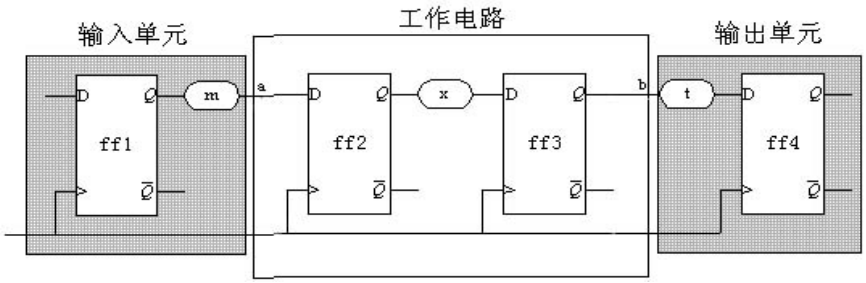


图 FPGA/CPLD 中信号的输入、输出锁存

2.6 FPGA/CPLD 中的时钟设计

无沦是用离散逻辑、可编程逻辑，还是用全定制硅器件实现的任何数字设计，为了成功地操作，可靠的时钟是非常关键的。设计不良的时钟在极限的温度、电压或制造工艺的偏差情况下将导致错误的行为，并且调试困难、花销很大。在设计 FPGA/CPLD 时通常采用几种时钟类型。时钟可分为如下四种类型：全局时钟、门控时钟、多级逻辑时钟和波动式时钟。多时钟系统能够包括上述四种时钟类型的任意组合。

无论采用何种方式，电路中真实的时钟树也无法达到假定的理想时钟，因此我们必须依据理想时钟，建立一个实际工作时钟模型来分析电路，这样才可以使得电路的实际工作效果和预期的一样。在实际的时钟模型中，我们要考虑时钟树传播中的偏斜、跳变和绝对垂直的偏差以及其它一些不确定因素。

对于寄存器而言，当时钟工作沿到来时它的数据端应该已经稳定，这样才能保证时钟工作沿采样到数据的正确性，这段数据的预备时间我们称之为建立时间（setup time）。数据同

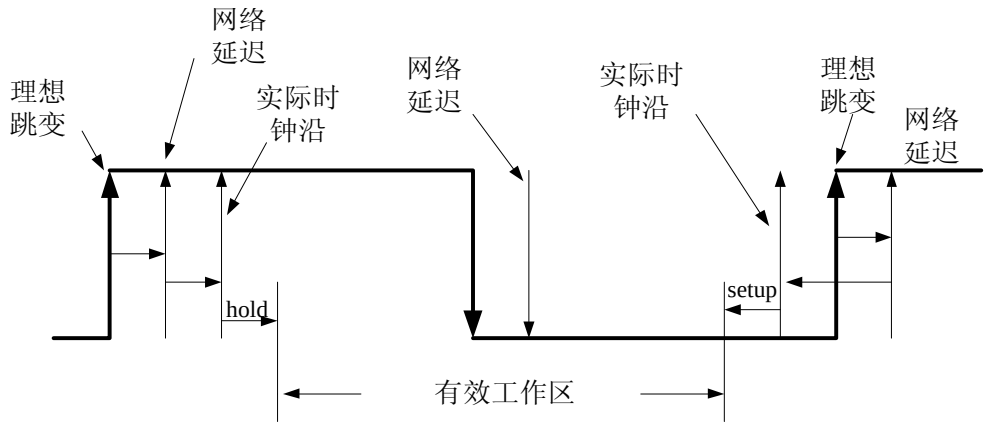


图 5　工作时钟模型

样应该在时钟工作沿过去后保持一段时间，这段时间称为保持时间（hold time）。因此具体的时钟如图 5 所示。其中网络延迟是指时钟的传播延时以及因为跳变不垂直等效的偏差，在此基础上考虑一些不确定因素实际的工作时钟沿如图中所示。保持时间（hold）和建立时间（setup）都是相对于实际时钟跳变而言的。因此在确定电路时序时，必须要考虑到这些因素，使得建立时间和保持时间符合要求。

为了使电路正常工作，建立时间和保持时间应该分别满足：

$$t_{hold} + t_{skew} < t_{clock\_Q\_\min} + t_{logic\_\min}$$

$$t_{clock} > t_{clock\_Q\_\max} + t_{logic\_\max} + t_{setup} + t_{skew}$$

其中 $t_{clock\_Q\_max}$ 是时钟沿变化到数据输出端变化的最慢变化情况，t_{logic_max} 是寄存器间组合逻辑的最大可能延迟，$t_{clock\_Q\_min}$ 和 t_{logic_min} 表示最快情况。在考虑建立保持时间时，应该考虑时钟树向后偏斜的情况，在考虑建立时间时应该考虑时钟树向前偏斜的情况。在进行后仿真时，最大延迟用来检查建立时间，最小延时用来检查保持时间。

2.6.1 全局时钟

对于一个设计项目来说，全局时钟(或同步时钟)是最简单和最可预测的时钟。在 PLD/FPGA 设计中最好的时钟方案是：由专用的全局时钟输入引脚驱动的单个主时钟去钟控设计项目中的每一个触发器。只要可能就应尽量在设计项目中采用全局时钟。PLD/FPGA 都具有专门的全局时钟引脚，它直接连到器件中的每一个寄存器。这种全局时钟提供器件中最短的时钟到输出的延时。

图 1 示出全局时钟的实例。图 1 定时波形示出触发器的数据输入 D[1..3]应遵守建立时间和保持时间的约束条件。建立和保持时间的数值在 PLD 数据手册中给出，也可用软件的定时分析器计算出来。如果在应用中不能满足建立和保持时间的要求，则必须用时钟同步输入信号(参看下一章“异步输入”)。

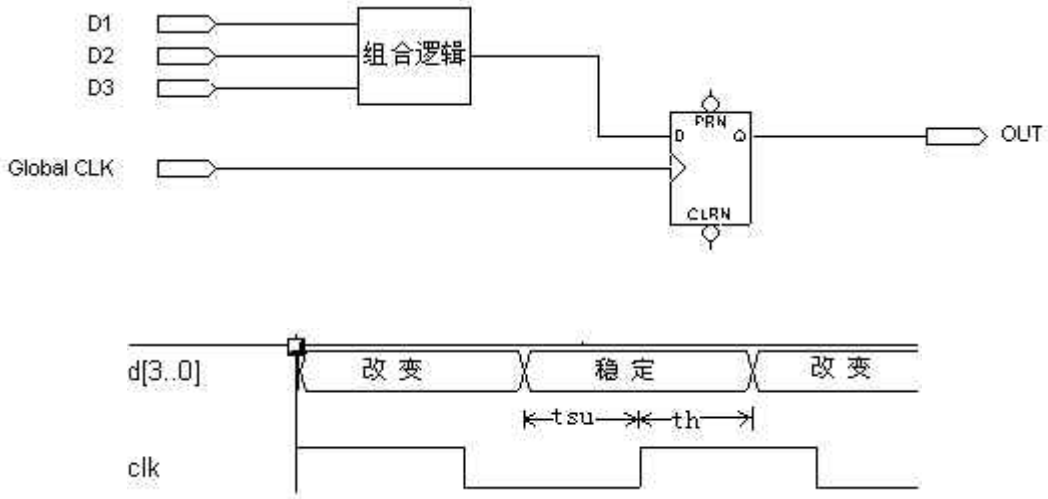


图 1 全局时钟

（最好的方法是用全局时钟引脚去钟控 PLD 内的每一个寄存器，于是数据只要遵守相对时钟的建立时间 tsu 和保持时间 th）

2.6.2 门控时钟

在许多应用中，整个设计项目都采用外部的全局时钟是不可能或不实际的。PLD 具有乘积项逻辑阵列时钟（即时钟是由逻辑产生的），允许任意函数单独地钟控各个触发器。然而，当你用阵列时钟时，应仔细地分析时钟函数，以避免毛刺。

通常用阵列时钟构成门控时钟。门控时钟常常同微处理器接口有关，用地址线去控制写脉冲。然而，每当用组合函数钟控触发器时，通常都存在着门控时钟。如果符合下述条件，门控时钟可以象全局时钟一样可靠地工作：

- 驱动时钟的逻辑必须只包含一个“与”门或一个“或”门。如果采用任何附加逻在某些工作状态下，会出现竞争产生的毛刺。

- 逻辑门的一个输入作为实际的时钟，而该逻辑门的所有其它输入必须当成地址或控制线，它们遵守相对于时钟的建立和保持时间的约束。

图 2 和图 3 是可靠的门控时钟的实例。在 图 2 中，用一个“与”门产生门控时钟，在 图 3 中，用一个“或”门产生门控时钟。在这两个实例中，引脚 nWR 和 nWE 考虑为时钟引脚，引脚 ADD[o. . 3]是地址引脚，两个触发器的数据是信号 D[1..n]经随机逻辑产生的。

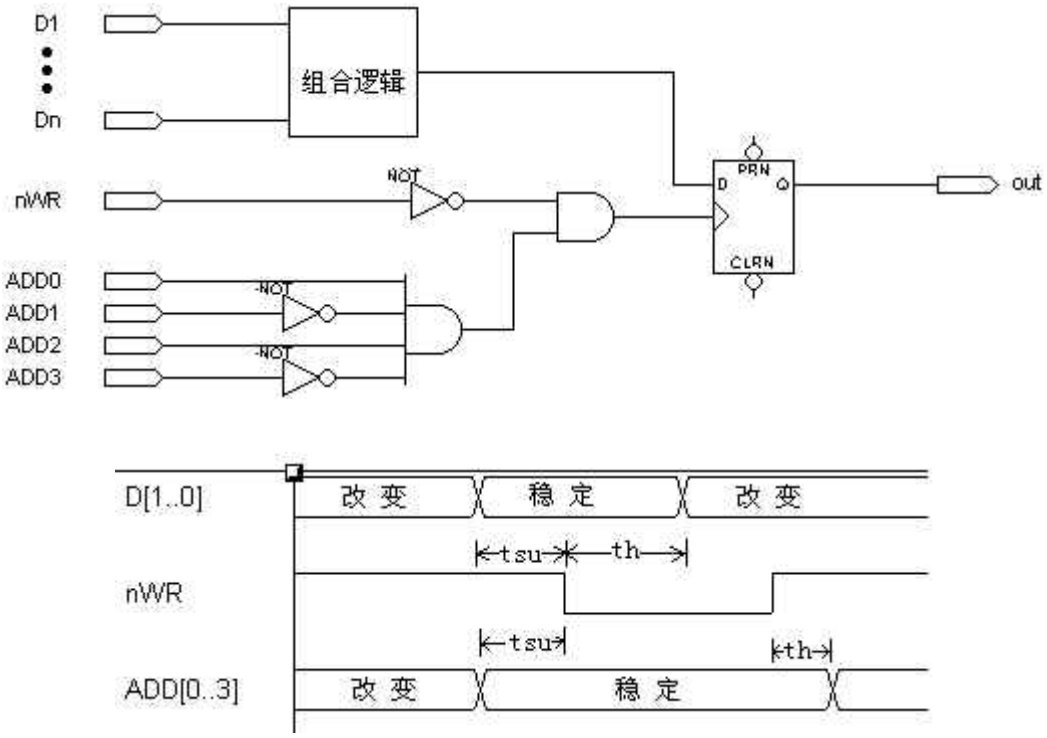
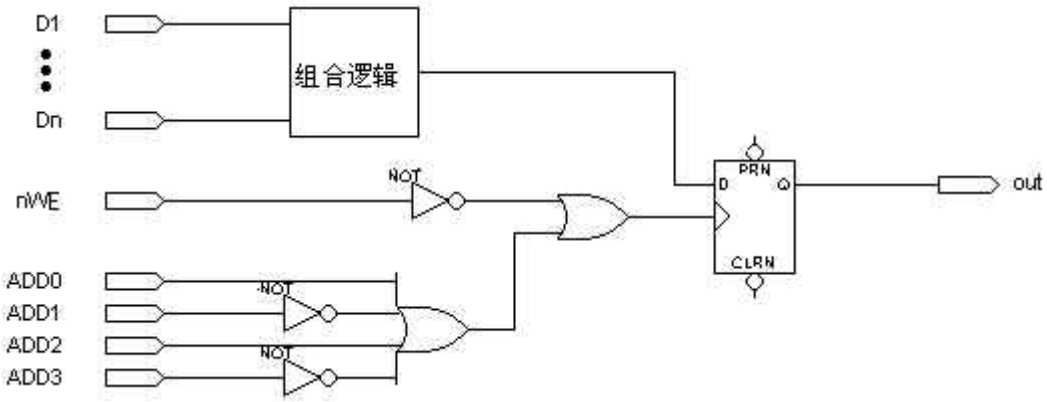


图 2 “与”门门控时钟



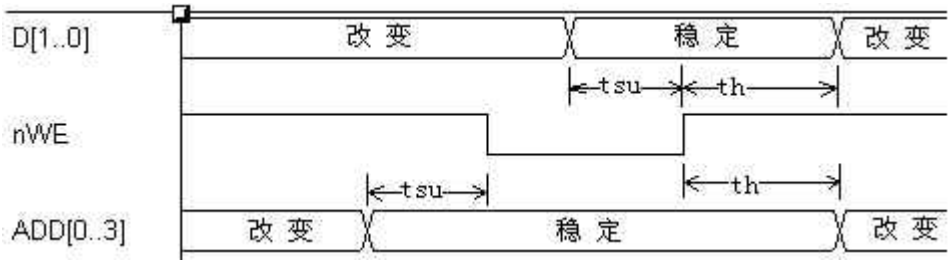


图 3 “或”门门控时钟

图 2 和图 3 的波形图显示出有关的建立时间和保持时间的要求。这两个设计项目的地址线必须在时钟保持有效的整个期间内保持稳定(nWR 和 nWE 是低电平有效)。如果地址线在规定的时间内未保持稳定，则在时钟上会出现毛刺，造成触发器发生错误的状态变化。另一方面，数据引脚 D[1. . n]只要求在 nWR 和 nWE 的有效边沿处满足标准的建立和保持时间的规定。

我们往往可以将门控时钟转换成全局时钟以改善设计项目的可靠性。图 4 示出如何用全局时钟重新设计 图 2 的电路。地址线在控制 D 触发器的使能输入，许多 PLD 设计软件，如 MAX+PLUSII 软件都提供这种带使能端的 D 触发器。当 ENA 为高电平时，D 输入端的值被钟控到触发器中：当 ENA 为低电平时，维持现在的状态。

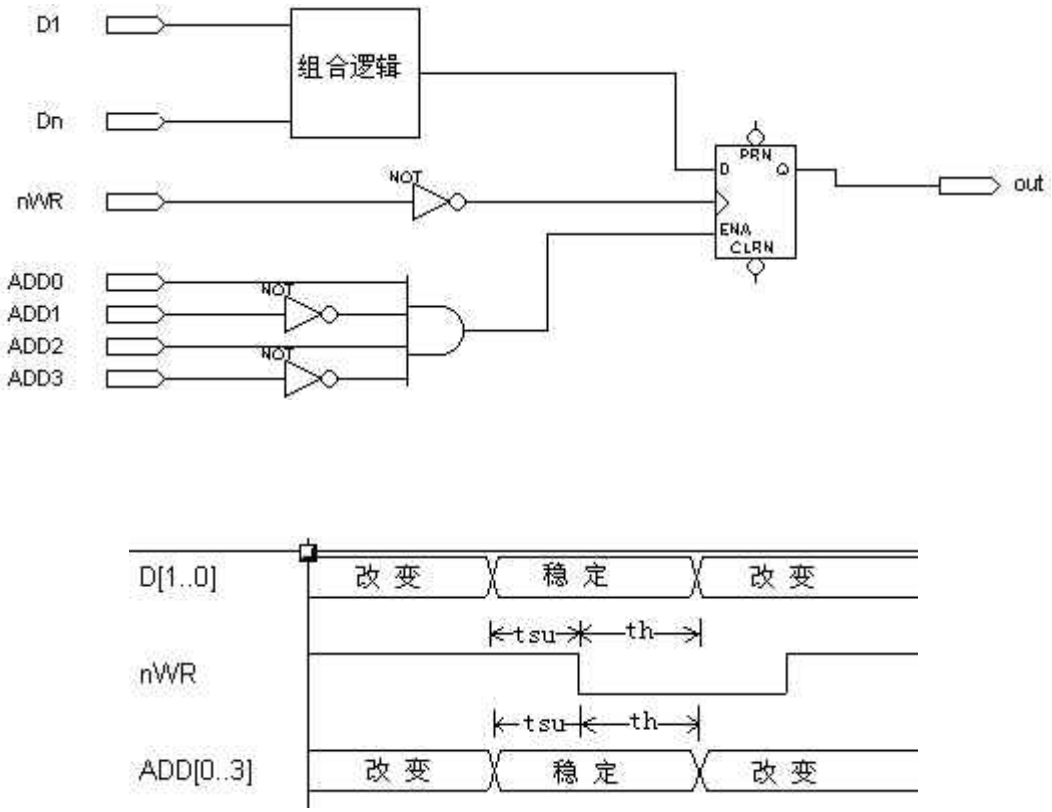


图 4 “与”门门控时钟转化成全局时钟

图 4 中重新设计的电路的定时波形表明地址线不需要在 nWR 有效的整个期间内保持稳定；而只要求它们和数据引脚一样符合同样的建立和保持时间，这样对地址线的要求就少很多。

图 5 给出一个不可靠的门控时钟的例子。3 位同步加法计数器的 RCO 输出用来钟控触发器。然而，计数器给出的多个输入起到时钟的作用，这违反了可靠门控时钟所需的条件之一。在产生 RCO 信号的触发器中，没有一个能考虑为实际的时钟线，这是因为所有触发器在几乎相同的时刻发生翻转。而我们并不能保证在 PLD/FPGA 内部 QA,QB,QC 到 D 触发器的布线长短一致，因此，如 图 5 的时间波形所示，在器从 3 计到 4 时，RCO 线上会出现毛刺（假设 QC 到 D 触发器的路径较短，即 QC 的输出先翻转）。

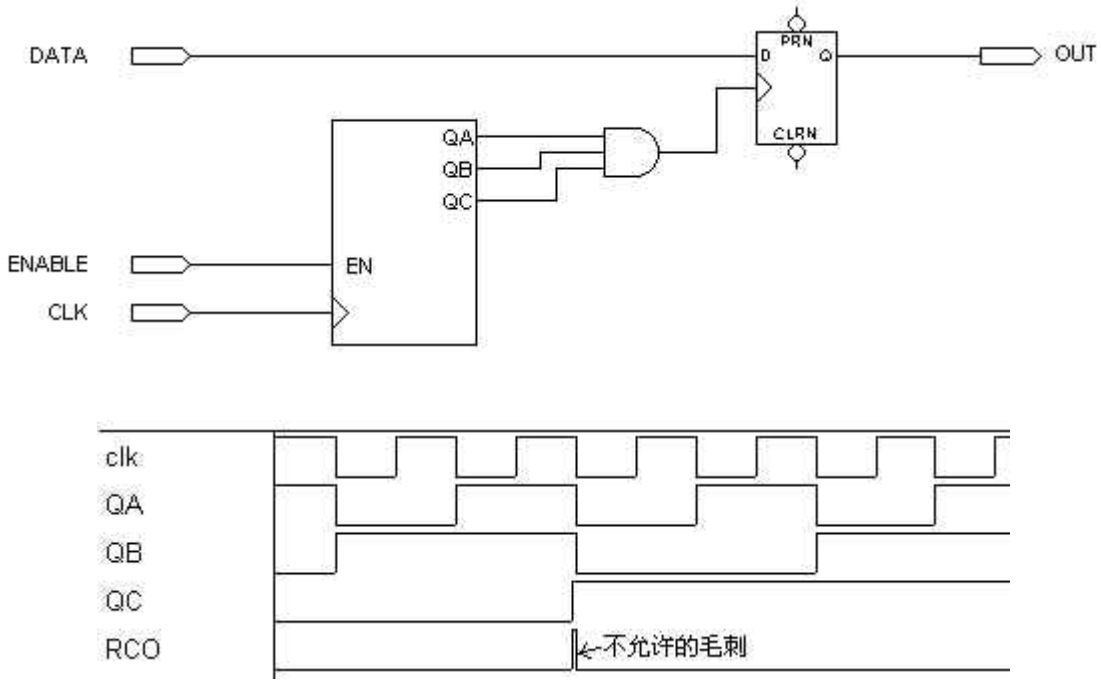


图 5 不可靠的门控时钟
（定时波形示出在计数器从 3 到 4 改变时，RCO 信号如何出现毛刺的）

图 6 给出一种可靠的全局钟控的电路，它是图 5 不可靠计数器电路的改进，RCO 控制 D 触发器的使能输入。这个改进不需要增加 PLD 的逻辑单元。

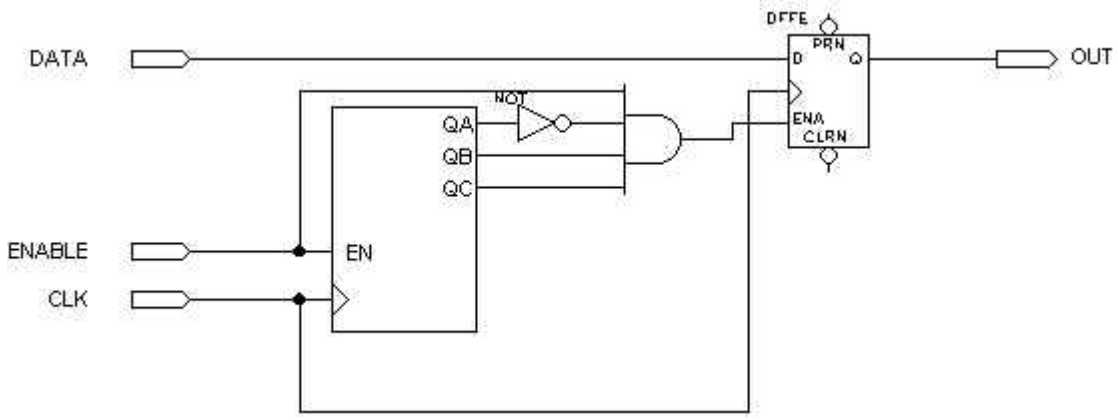


图 6 不可靠的门控时钟转换为全局时钟

2.6.3 多级逻辑时钟

当产生门控时钟的组合逻辑超过一级(即超过单个的“与”门或“或”门)时，证设计项目的可靠性变得很困难。即使样机或仿真结果没有显示出静态险象，但实际上仍然可能存在着危险。通常，我们不应该用多级组合逻辑去钟控 PLD 设计中的触发器。

图 7 给出一个含有险象的多级时钟的例子。时钟是由 SEL 引脚控制的多路选择器输出的。多路选择器的输入是时钟(CLK)和该时钟的 2 分频(DIV2)。由图 7 的定时波形图看出，在两个时钟均为逻辑 1 的情况下，当 SEL 线的状态改变时，存在静态险象。险象的程度取决于工作的条件。 多级逻辑的险象是可以去除的。例如，你可以插入“冗余逻辑”到设计项目中。然而，PLD/FPGA 编译器在逻辑综合时会去掉这些冗余逻辑，使得验证险象是否真正被去除变得困难了。为此，必须应寻求其它方法来实现电路的功能。

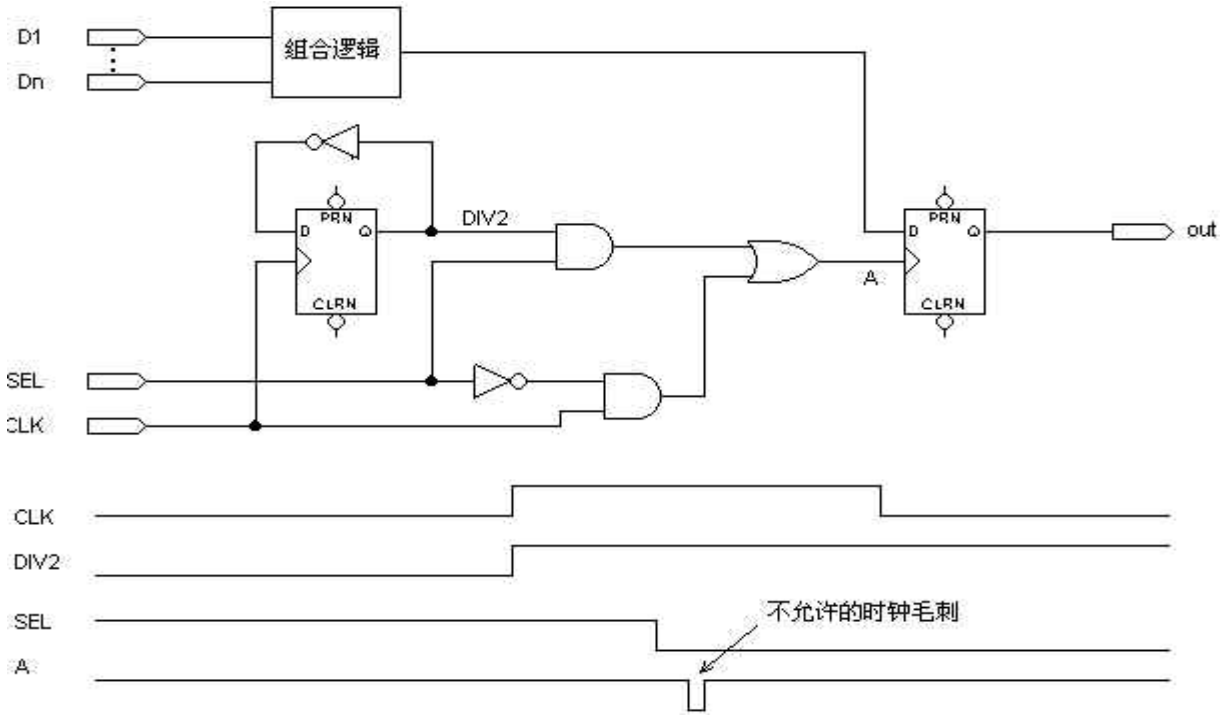


图 7 有静态险象的多级时钟

图 8 给出 图 7 电路的一种单级时钟的替代方案。图中 SEL 引脚和 DIV2 信号用于使能 D 触发器的使能输入端，而不是用于该触发器的时钟引脚。采用这个电路并不需要附加 PLD 的逻辑单元，工作却可靠多了。 不同的系统需要采用不同的方法去除多级时钟，并没有固定的模式。

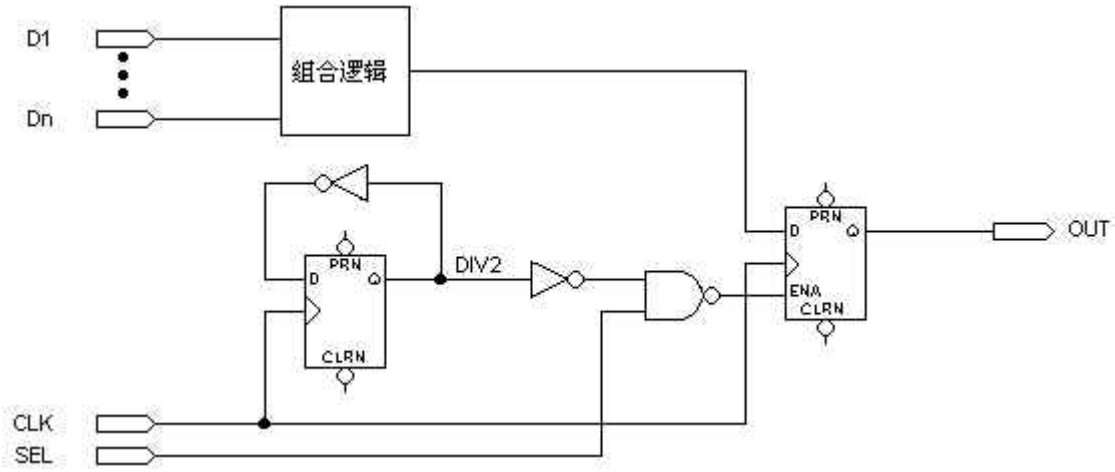


图 7 无静态险象的多级时钟

（这个电路逻辑上等效于图 7，但却可靠的多）

2.6.4 行波时钟

另一种流行的时钟电路是采用行波时钟，即一个触发器的输出用作另一个触发器的时钟输入。如果仔细地设计，行波时钟可以象全局时钟一样地可靠工作。然而，行波时钟使得与电路有关的定时计算变得很复杂。行波时钟在行波链上各触发器的时钟之间产生较大的时间偏移，并且会超出最坏情况下的建立时间、保持时间和电路中时钟到输出的延时，使系统的实际速度下降。

用计数翻转型触发器构成异步计数器时常采用行波时钟，一个触发器的输出钟控下一个触发器的输入，参看图 9 同步计数器通常是代替异步计数器的更好方案，这是因为两者需要同样多的宏单元而同步计数器有较快的时钟到输出的时间。图 10 给出具有全局时钟的同步计数器，它和 图 9 功能相同，用了同样多的逻辑单元实现，却有较快的时钟到输出的时间。几乎所有 PLD 开发软件都提供多种多样的同步计数器。

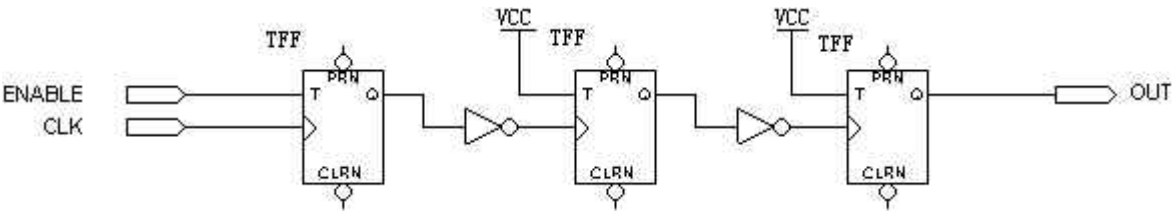


图 9 行波时钟

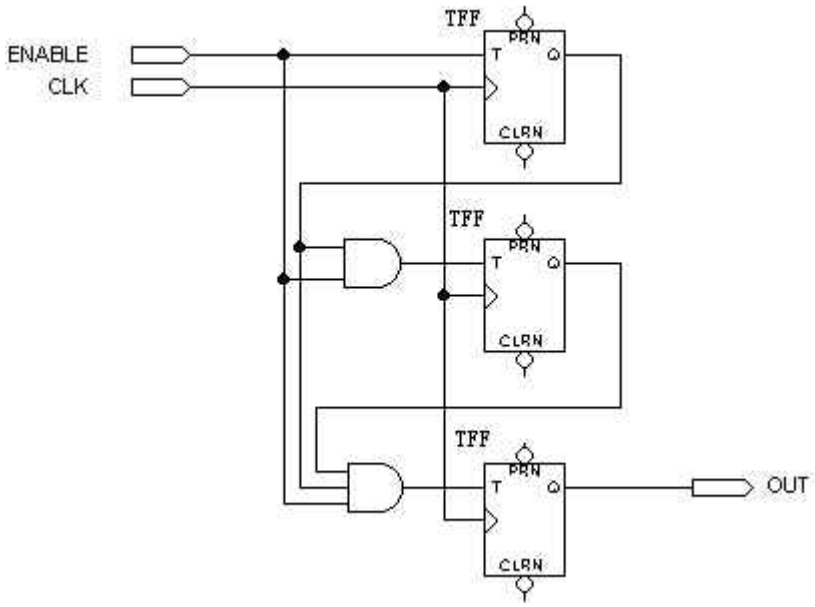


图 10 行波时钟转换成全局时钟

(这个 3 位计数器是图 9 异步计数器的替代电路，它用了同样的 3 个宏单元，但有更短的时钟到输出的延时)

2.6.5 多时钟系统

许多系统要求在同一个 PLD 内采用多时钟。最常见的例子是两个异步微处理器器之间的接口，或微处理器和异步通信通道的接口。由于两个时钟信号之间要求一定的建立和保持时间，所以，上述应用引进了附加的定时约束条件。它们也会要求将某些异步信号同步化。

图 11 给出一个多时钟系统的实例。CLK_A 用以钟控 REG_A，CLK_B 用于钟控 REG_B，由于 REG_A 驱动着进入 REG_B 的组合逻辑，故 CLK_A 的上升沿相对于 CLK_B 的上升沿有建立时间和保持时间的要求。由于 REG_B 不驱动馈到 REG_A 的逻辑,CLK_B 的上升沿相对于 CLK_A 没有建立时间的要求。此外，由于时钟的下降沿不影响触发器的状态，所以 CLK_A 和 CLK_B 的下降沿之间没有时间上的要求。如图 4，2．II 所示，电路中有两个独立的时钟，可是，在它们之间的建立时间和保持时间的要求是不能保证的。在这种情况下，必须将电路同步化。图 12 给出 REG_A 的值(如何在使用前)同 CLK_B 同步化。新的触发器 REG_C 由 GLK_B 触控，保证 REG_G 的输出符合 REG_B 的建立时间。然而，这个方法使输出延时了一个时钟周期。

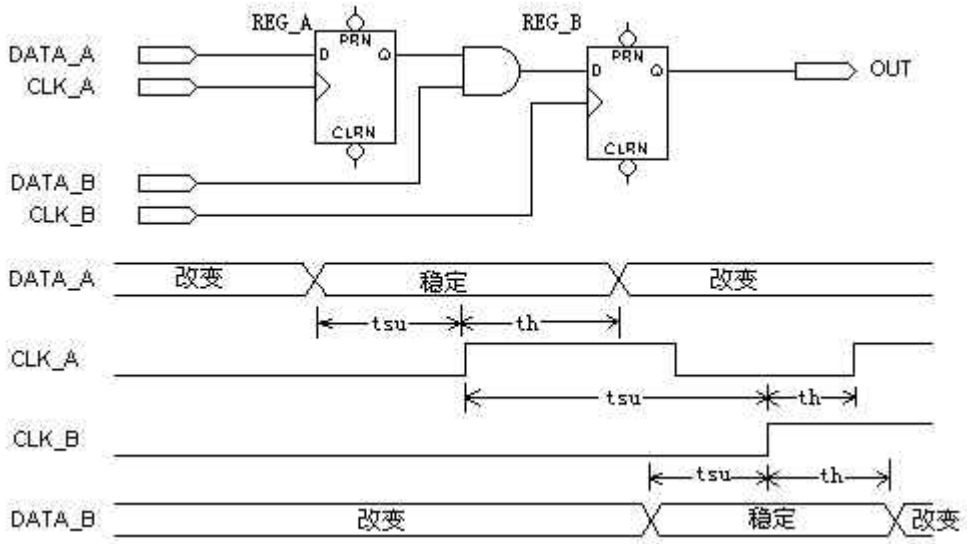


图 11 多时钟系统

(定时波形示出 CLK_A 的上升沿相对于 CLK_B 的上升沿有建立时间和保持时间的约束条件)

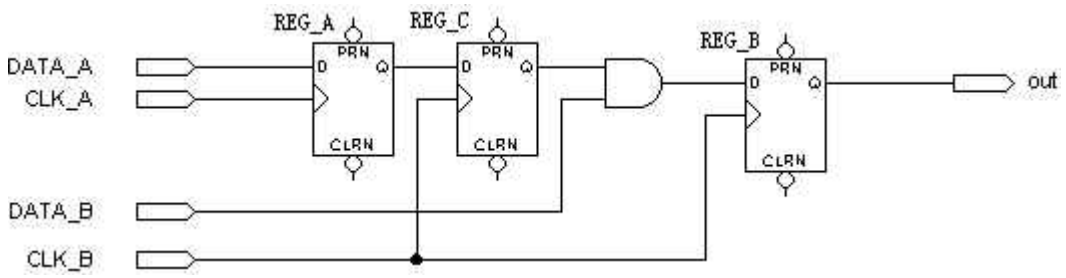


图 12 具有同步寄存器输出的多时钟系统

(如果CLK_A和CLK_B是相互独立的,则REG—A的输出必须在它馈送到1REG_B之前,用REG_C同步化)

在许多应用中只将异步信号同步化还是不够的，当系统中有两个或两个以上非同源时钟的时候，数据的建立和保持时间很难得到保证，我们将面临复杂的时间问题。最好的方法是将所有非同源时钟同步化。使用 PLD 内部的锁项环（PLL 或 DLL）是一个效果很好的方法，但不是所有 PLD 都带有 PLL、DLL，而且带有 PLL 功能的芯片大多价格昂贵，所以除非有特殊要求，一般场合可以不使用带 PLL 的 PLD。 这时我们需要使用带使能端的 D 触发器，并引入一个高频时钟。

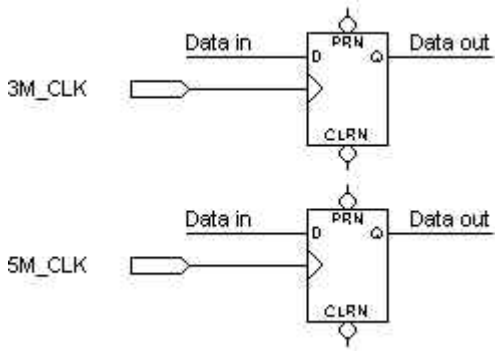


图 13 不同源时钟

如图 13 所示，系统有两个不同源时钟，一个为 3MHz，一个为 5MHz，不同的触发器使用不同的时钟。为了系统稳定，我们引入一个 20MHz 时钟，将 3M 和 5M 时钟同步化，如图 15 所示。 20M 的高频时钟将作为系统时钟，输入到所有触发器的的时钟端。3M_EN 和 5M_EN 将控制所有触发器的使能端。即原来接 3M 时钟的触发器，接 20M 时钟，同时 3M_EN 将控制该触发器使能 ，原接 5M 时钟的触发器，也接 20M 时钟，同时 5M_EN 将控制该触发器使能。 这样我们就可以将任何非同源时钟同步化。

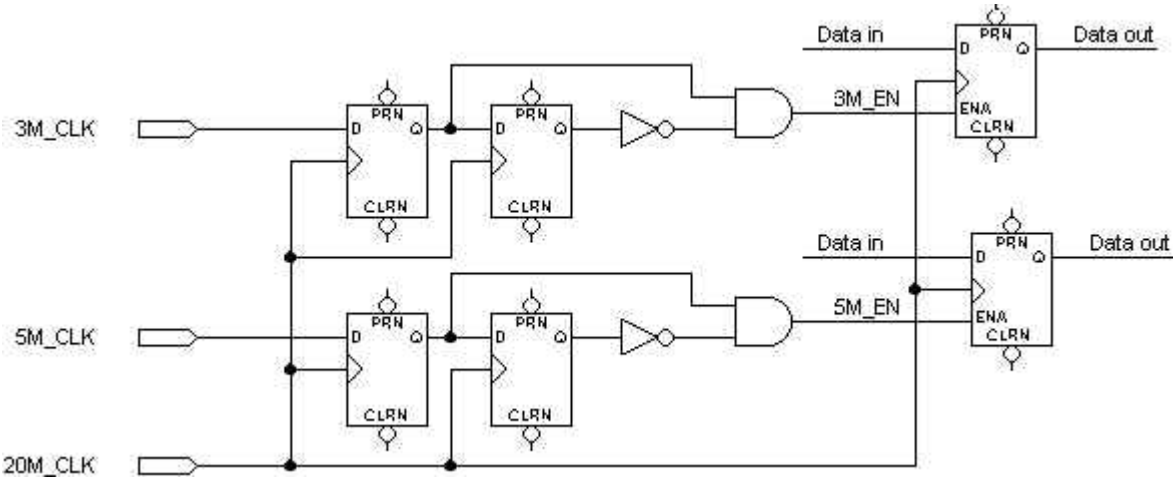


图 13 同步化任意非同源时钟

（一个 DFF 和后面非门，与门构成时钟上升沿检测电路）

另外，异步信号输入总是无法满足数据的建立保持时间，容易使系统进入亚稳态，所以也建议设计者把所有异步输入都先经过双触发器进行同步化.

小结：稳定可靠的时钟是系统稳定可靠的重要条件，我们不能够将任何可能含有毛刺的输出作为时钟信号，并且尽可能只使用一个全局时钟，对多时钟系统要注意同步异步信号和非同源时钟。

2.6.6 多时钟系统设计的一些方法：

如果时钟间存在着固定的频率倍数，这种情况下它们的相位一般具有固定关系，可以采用下述方法处理；

- 使用高频时钟作为工作时钟，使用低频时钟作为使能信号，当功耗不作为首要因素时建议使用这种方式；
- 在仔细分析时序的基础上描述两个时钟转换处的电路；

如果电路中存在两个不同频率的时钟，并且频率无关，可以采用如下策略：

- 利用高频时钟采样两个时钟，在电路中使用高频时钟作为电路的工作时钟，经采样后的低频时钟作为使能；
- 在时钟同步单元中采用两次同步法
- 使用握手信号
- 使用双时钟 FIFO 进行数据缓冲

时钟同步化，如果系统中存在两个时钟 clk_a 和 clk_b,设计者可以使用频率高于 max(clk_a,clk_b)两倍的时钟来作为采样时钟，两个低频时钟经过处理后可以作为触发器的使能信号，采用这种方案的好处是整个电路采用单时钟工作，但需要一个额外的高频时钟，当电路有功耗要求时，设计者应该仔细考虑；

使用 20M 采样 3M 和 5M,syn_5M 作为原来 5M 信号驱动寄存器的使能信号；

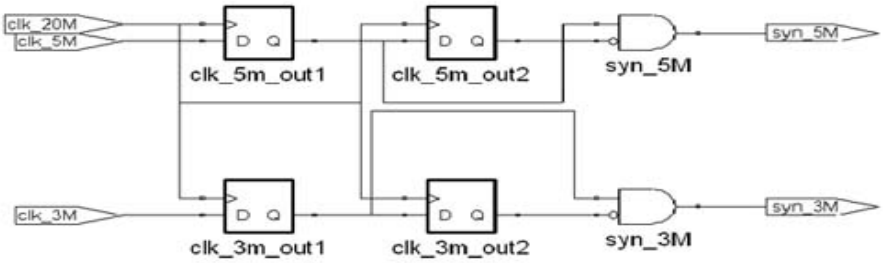


图 使用高频时钟采样 2 个低频时钟原理图

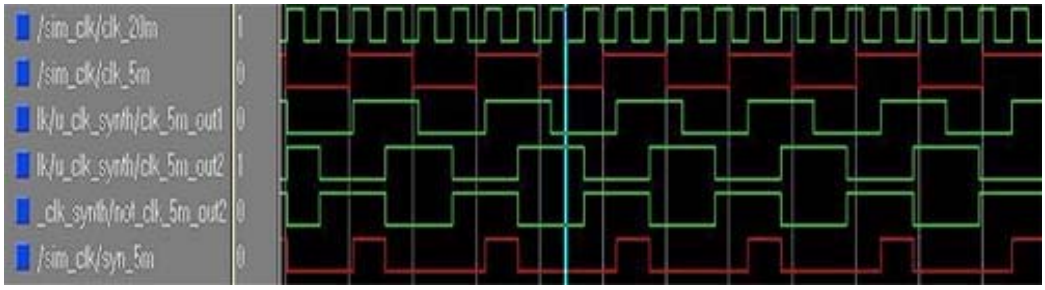
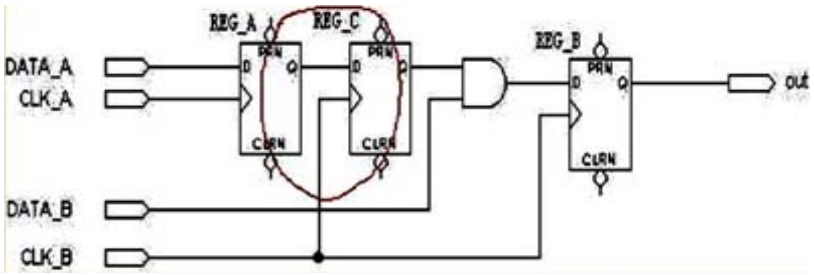


图 使用高频时钟采样 2 个低频时钟波形图

在构件由两个不同系统时钟控制工作的模块之间的同步模块时，应该遵守下面原则：两个采用不同时钟工作的寄存器之间不应该再出现逻辑电路，而应该仅仅是一种连接关系，具体如下图所示，这种方法有利于控制建立保持时间的满足。



握手信号机制是异步系统之间通信的基本方式，我们在处理不同时钟之间的接口时，也可以采用这种方式，但需要注意的是设计者应该仔细分析握手和应答信号有效持续的时间，确保采样数据的正确性。

目前各种器件中提供的双时钟 FIFO 宏单元很好的提供了对异步双时钟的访问，单元的内部有协调两个时钟的电路，确保读写的正确性。可以利用这个器件完成数据的同步。

1. 采用全局时钟，不要将时钟参与运算。系统提供一定数量的全局时钟线，在布局布线时，尽量满足这些信号的要求以减小时钟偏移和倾斜。如果时序安排不合理使用了较多 gated clock，那么这些时钟的偏斜就会较大，不能保障建立时间和保持时间，导致电路工作频率降低或无法工作。
2. 以寄存器为边界划分工作模块。在设计较大规模的电路时，分模块设计是必不可少的，在各模块通过之后再进行系统的联调。但由于在单模块调试和联调时布线资源的占用紧张程度不同，使得每个模块的输出无法保持与单独布线时相同，在联调时造成困难。如果每一个模块的输出端口都采用寄存器输出，那么即使在整体布局布线后，各模块的输出依然可以保证原来的时序，这使得联调的工作效率大大提高。加入这些寄存器也使得电路的可测性有所提高。
3. 组合逻辑尽量采用并行结构，降低寄存器间组合路径的延迟是提高系统工作频率最主要的手段，因此在完成相同功能的前提下应该尽量使用并行逻辑，如图 6 所示。

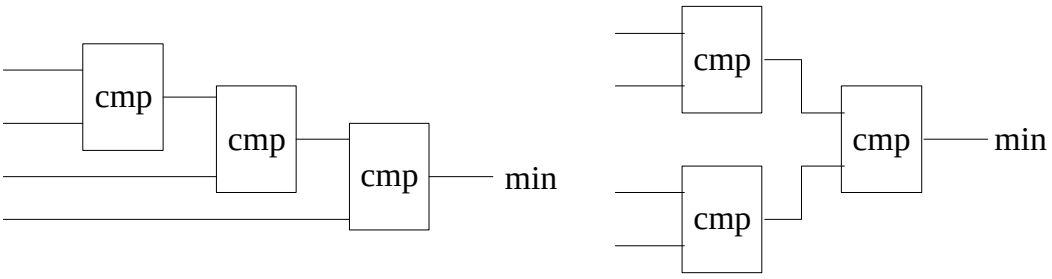


图 6 四输入比较器的串行和并行实现

如果没有优先级要求应该尽量采用 case 语句来描述，这样综合出来的电路并行度要大一些，如果采用 if-then-else 结构，综合出来的电路都是串行的，增大了时延路径。

4. 在描述中应该消除锁存器，如果某个数据需要保存应该合理安排使用寄存器，因为锁存器在整个工作电平有效期间都对输入敏感，输入中的任何毛刺经过锁存器后都不会消除，这样使得在其后的组合电路发生竞争冒险的可能性大为提高，影响电路性能。一些不适当的描述也会使得产生不必要的锁存器，增加了电路的面积。
5. 在设计中应该尽量采用同步设计，信号被时钟采样后再参与逻辑运算，这样可以隔断组合路径，也可以消除毛刺。在设计中，组合信号的输出不允许反馈作为该组合逻辑的输入，这样可以避免组合环。