深圳市华为技术有限公司 研究管理部文档中心	文档编号	产品版本	密级
		V1.0	秘密
	产品名称：同步电路设计技术及规则		共27页

同步电路设计技术

及规则

（仅供内部使用）

FPGA GROUP

文 档 作 者：	周志坚	日期：	1999/11/18
研 究 部：		日期：	
文档管理员：		日期：	

深圳市华为技术有限公司

修订记录[P1]

日期	修订版本	描述	作者
1999/11/18	1.00	初稿完成	周志坚

目　录

同步电路设计技术及规则

1 设计可靠性

为了增加可编程逻辑器件电路工作的稳定性，一定要加强可编程逻辑器件设计的规范要求，要尽量采用同步电路设计。对于设计中的异步电路，要给出不能转换为同步设计的原因，并对该部分异步电路的工作可靠性(如时钟等信号上是否有毛刺，建立-保持时间是否满足要求等）作出分析判断，提供分析报告。

2 时序分析基础

电路设计的难点在时序设计，而时序设计的实质就是满足每一个触发器的建立/保持时间的要求。

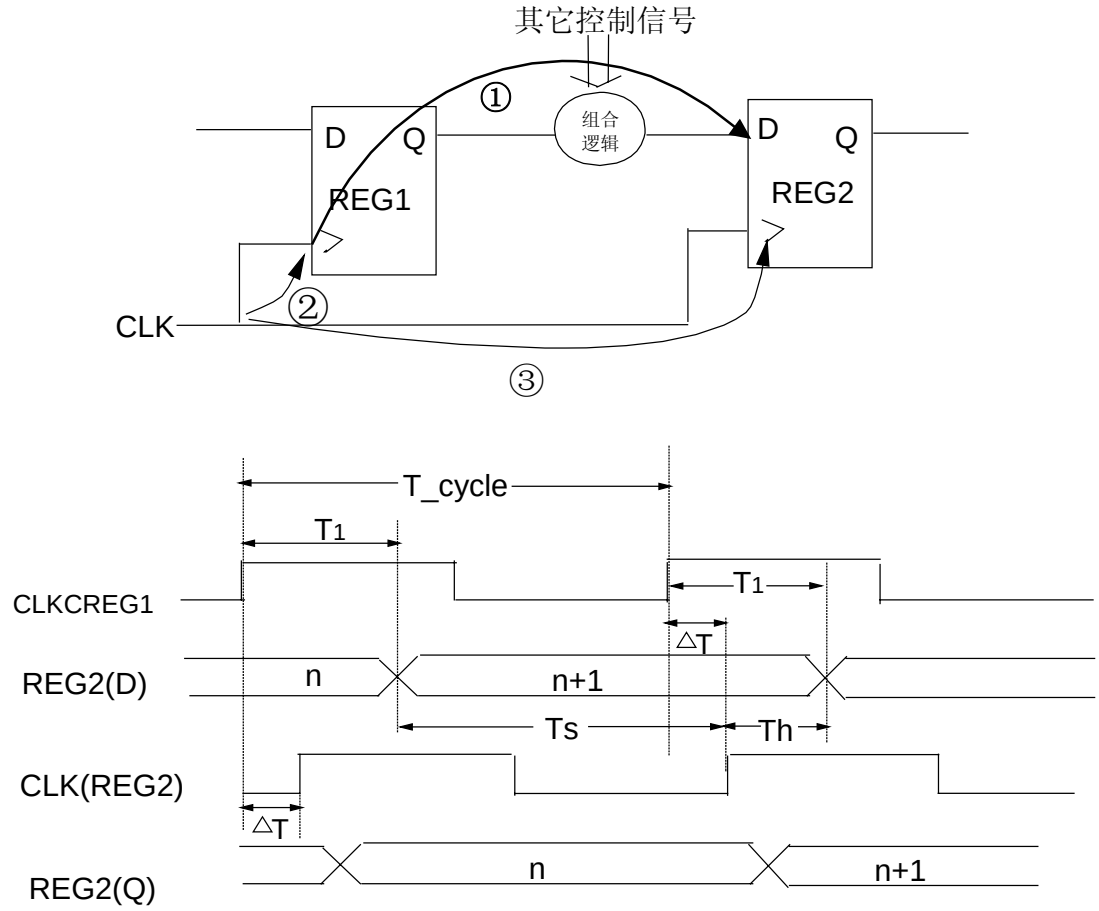


图1.1

如上图所示，以REG2为例，假定

触发器的建立时间要求为：T_setup，保持时间要求为：T_hold，

路径①延时为：T1，路径②延时为：T2，路径③延时为：T3，

时钟周期为：T_cycle，

Ts ＝（T_cycle ＋ △T）－T1，Th ＝T1－△T，

令 △T ＝T3－T2，则

条件1. 如果T_setup ＜ Ts ，即 T_setup ＜ （T_cycle ＋ △T）－T1，这说明信号比时钟有效沿超过T_setup 时间到达REG2的D端，满足建立时间要求。反之则不满足；

条件2. 如果T_hold ＜ Th ，即T_hold ＜ T1－△T ，这说明在时钟有效沿到达之后，信号能维持足够长的时间，满足保持时间要求。反之则不满足。

从条件1和2我们可以看出，当△T ＞ 0 时，T_hold受影响；当△T ＜ 0 时，T_setup 受影响。

如果我们采用的是严格的同步设计电路，即一个设计只有一个CLK，并且来自时钟PAD或时钟BUFF（全局时钟），则△T对电路的影响很小，几乎为0；如果采用的是异步电路，设计中时钟满天飞，无法保证每一个时钟都来自强大的驱动BUFF（非全局时钟），如下图所示，则△T影响较大，有时甚至超过人们想象。这就是为什么我们建议采用同步电路进行设计的重要原因之一。

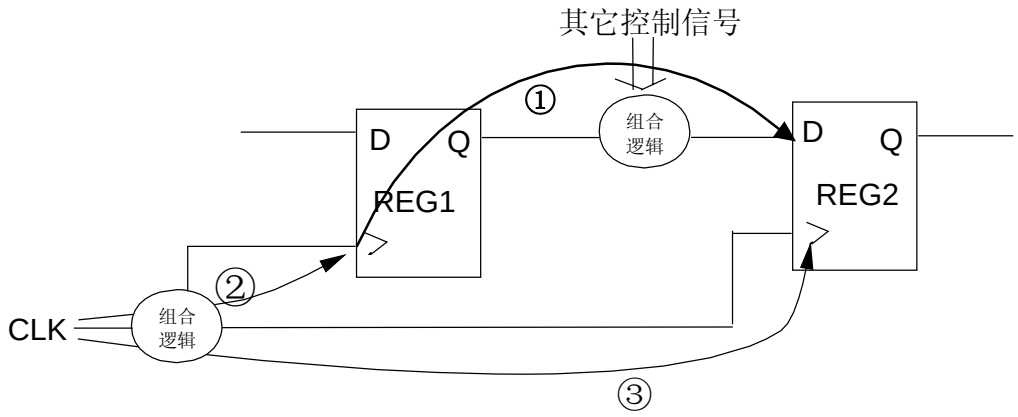


图1.2

3 同步电路设计

3.1 同步电路的优越性

1.同步电路比较容易使用寄存器的异步复位/置位端，以使整个电路有一个确定的初始状态；

2.在可编程逻辑器件中，使用同步电路可以避免器件受温度，电压，工艺的影响，易于消除电路的毛刺，使设计更可靠，单板更稳定；

3.同步电路可以很容易地组织流水线，提高芯片的运行速度，设计容易实现；

下图是一个设计中所要准备采用的电路，该设计采用Xilinx的FPGA器件4062xla来实现，工作频率是32.768MHz（即图中CLK频率）。设计原打算在每隔60ns输出一个数据，即DATA。然而，我们在设计之前，考虑到256x7的同步RAM延时可能比较大，如果在加上其后的同步RAM延时的话，估计在60ns之内很难完成。该部分电路是整个设计中的一个关键路径，因此，我们在进行具体设计之前，先对这种电路结构进行了验证，事实证明我们的担心是对的。正确的做法是，采用流水线方法，在256x7的RAM之后再加一个触发器，每个RAM都按60ns的速度读取数据，整个流程滞后60ns输出DATA。其它相关信号（在其它模块中）也随之滞后60ns输出。

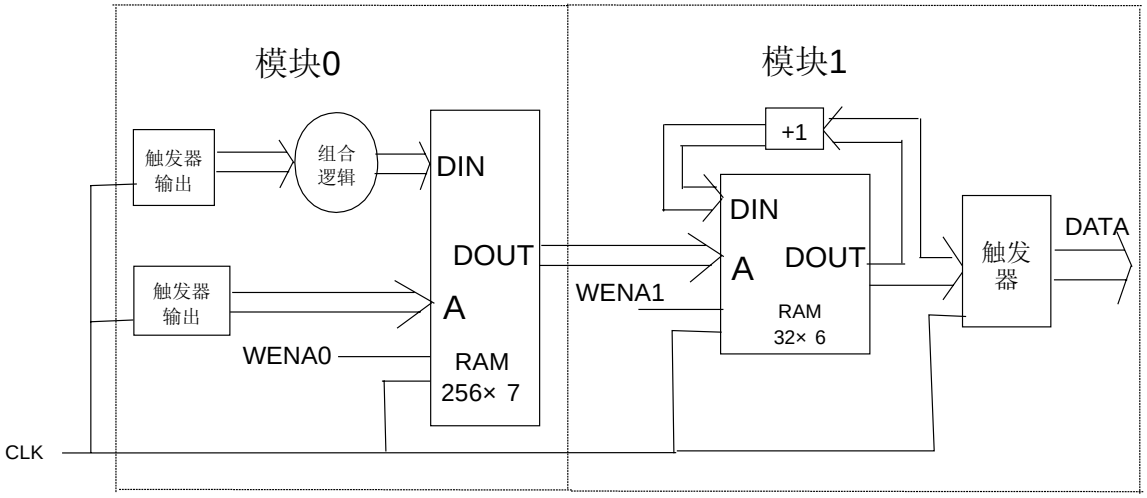


图1. 3

4. 同步电路可以很好地利用先进的设计工具，如静态时序分析工具等，为设计者提供最大便利条件，便于电路错误分析，加快设计进度。

1.1 同步电路的设计规则

1.尽可能在整个设计中只使用一个主时钟，同时只使用同一个时钟沿，主时钟走全局时钟网络。

2.在FPGA设计中，推荐所有输入、输出信号均应通过寄存器寄存，寄存器接口当作异步接口考虑。

3.当全部电路不能用同步电路思想设计时，即需要多个时钟来实现，则可以将全部电路分成若干局部同步电路（尽量以同一个时钟为一个模块），局部同步电路之间接口当作异步接口考虑。

4.当必须采用多个时钟设计时，每个时钟信号的时钟偏差（△T）要严格控制。

5.电路的实际最高工作频率不应大于理论最高工作频率，留有设计余量，保证芯片可靠工作。

6.电路中所有寄存器、状态机在单板上电复位时应处在一个已知的状态。

1.2 异步设计中常见问题及其解决方法

异步电路设计主要体现在时钟的使用上，如使用组合逻辑时钟、级连时钟和多时钟网络；另外还有采用异步置位、复位、自清零、自复位等。这些异步电路的大量存在，一是增加设计难度，二是在出现错误时，电路分析比较困难，有时会严重影响设计进度。

很多异步设计都可以转化为同步设计，对于可以转化的逻辑必须转化，不能转化的逻辑，应将异步的部分减到最小，而其前后级仍然应该采用同步设计。下面给出一些异步逻辑转化为同步逻辑的方法：

1. 组合逻辑产生的时钟

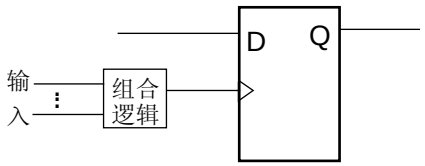


图1. 4　组合逻辑产生的时钟

组合逻辑的时钟如果产生毛刺，易使触发器误翻转。

2.行波计数器/行波时钟

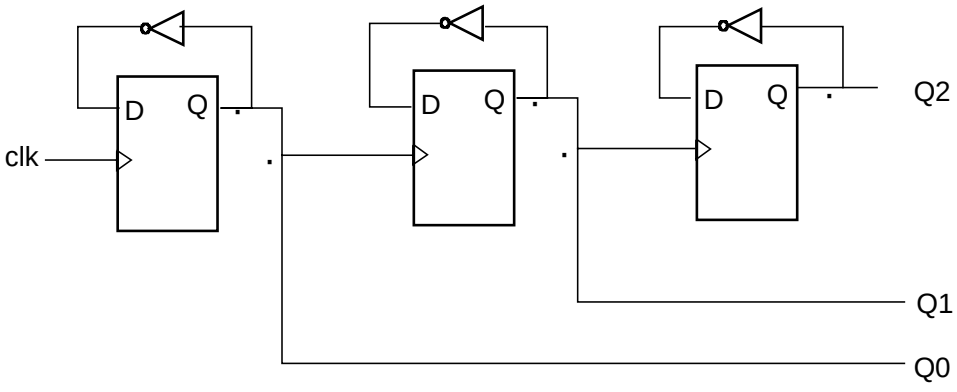


图1.5　行波计数器

行波记数器虽然原理简单，设计方便，但级连时钟（行波时钟）最容易造成时钟偏差（△T），级数多了，很可能会影响其控制的触发器的建立/保持时间，使设计难度加大。转换的方法是采用同步记数器，同步计数器用原理图描述可能较难，但用VHDL很简单就可以描述一个4位计数器：

```
Counter4:
Process(nreset,clk)
Begin
  If    nreset = '0' then
        Cnt <= ( others => "0" );
   Elsif clk = '1' and clk'event then
        Cnt <= cnt + 1;
  End if;
End process counter4;
```

通常逻辑综合工具都会对上述描述按不同器件的特点进行不同的优化，我们并不需要关心它是逐位进位计数器还是超前进位计数器。

4.不规则的计数器

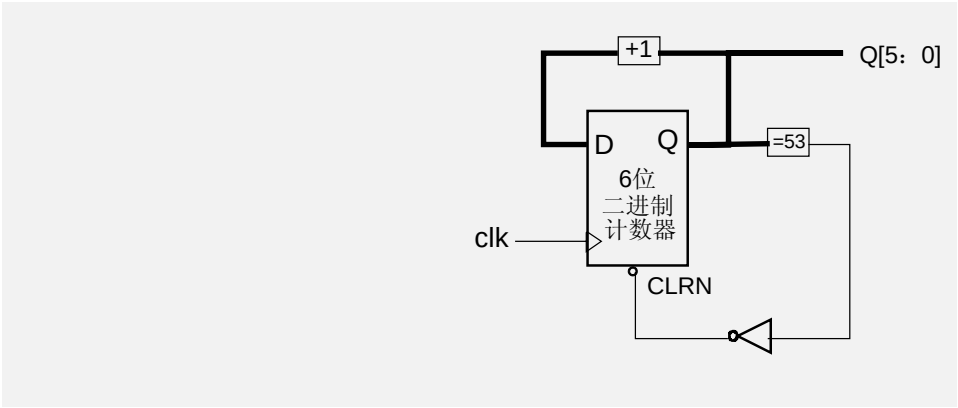


图1.6　　不规则的计数器

这是一个53计数器，采用计到53后产生异步复位的办法实现清0，产生毛刺是必然的。然而最严重的是，当计数器所有bit或相关bit均在翻转时，电路有可能出错，例如：计数器从“110011”—>“110100”，由于电路延时的原因，中间会出现“110101”状态，导致计数器误清0。

采用同步清0的办法，不仅可以有效地消除毛刺，而且能避免计数器误清0。电路如下图所示。

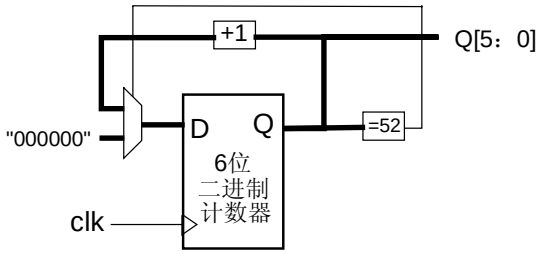


图1.7　　规则的计数器

5.分频器

这是3和4的特例，我们推荐使用同步计数器最高位的方法，如果需要保证占空比，可以使用图1.8所示电路进行最后一次二分频。下图是19.44MHz分频到8kMHz(分频数为2430)的电路：

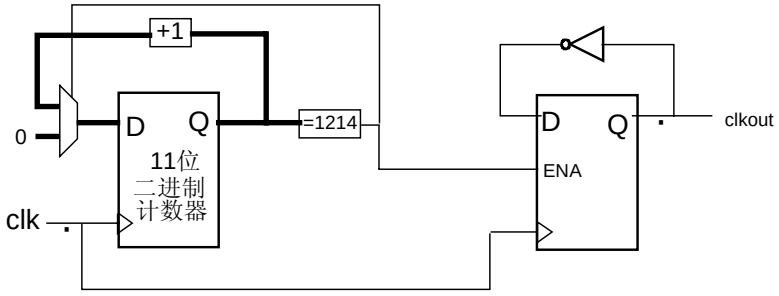
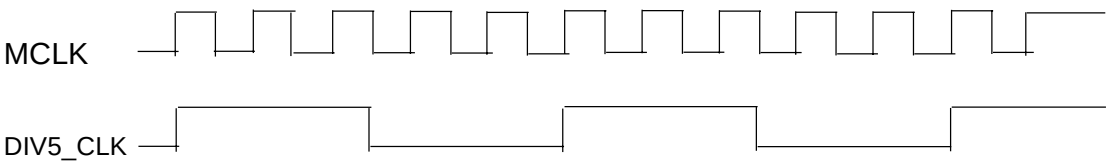


图1.8　　分频数为2430的电路

若是奇数分频，则处理比较特殊，以5分频器为例，其要求产生的时序关系如下图所示，



很显然，该电路要用上MCLK的上沿和下研，对上图时序进行分解，得下图

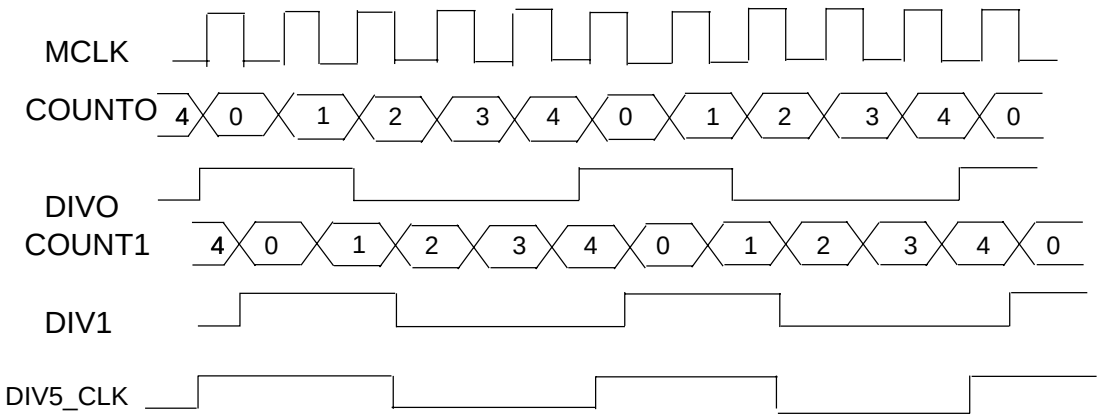


图1.9 5分频信号时序分解

图中，COUNT0采用上沿计数，COUNT1采用下沿计数，DIV0和DIV1是分别是上沿触发器和下沿触发器的输出，DIV5_CLK是DIV0和DIV1的或门输出。读者可根据该时序图，画出相应的原理图，或者用HDL语言进行描述。

在使用该电路时，需要注意：

（1）DIV0和DIV1到DIV5_CLK的约束要严，越快越好。不然，无法保证1:1的占空比。

（2）MCLK频率要求较高，尽量不要出现窄脉冲，尤其是在高频电路里。

（3）COUNT1可有可无，视时钟频率高低而定。频率越高，COUNT1越需要。

6.多时钟的同步化

我们在设计中，经常预见这种情况：一个控制信号来自其它芯片（或者芯片其它模块），该信号相对本电路来讲是异步的，即来自不同的时钟源。其模型可用图1.10表示。

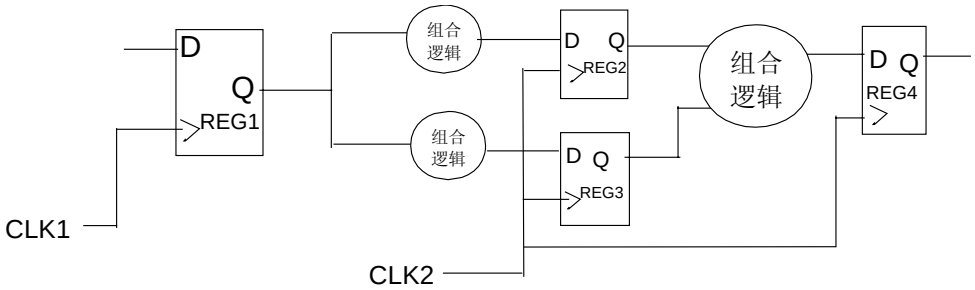


图1.10

在图1.10中，CLK1与CLK2来自不同的时钟源，该电路即可能出现在同一芯片里，又可能出现在不同芯片里。但效果是一样的，即存在危险性：由于时钟源不同，对REG2和REG3来讲，在同一时刻，一个“认为”REG1的输出是“1”，另一个认为是“0”。这必定造成电路判断出现混乱，导致出错。

这种错误的实质是内部其它电路对输入控制信号（也可认为是状态信号）认识不一致，导致不同的电路进入不同的状态。正确的做法是在REG1之后再加一个触发器，用CLK2的时钟沿去采样，然后用该触发器的输出参与其后同步电路“活动”。如图1.11所示。

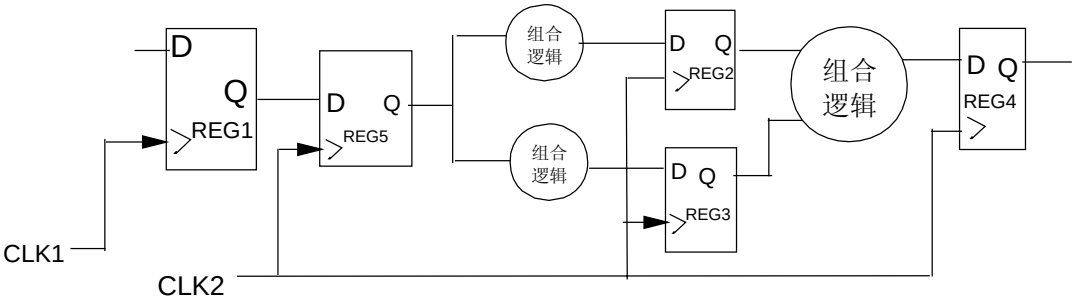


图1.11

如果输入信号是两根以上信号线，如下图所示，则该处理方法不准确。应引入专门的同步调整电路或其它特殊处理电路。我们在设计时，会对总线数据进行同步调整，却往往忽略了对一组控制信号进行同步调整。

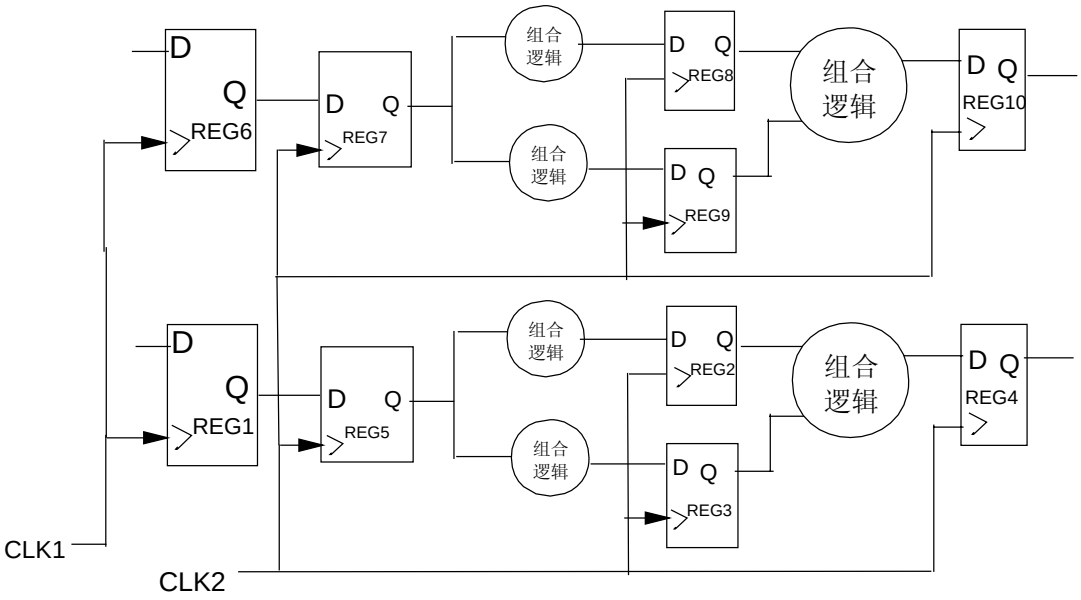


图1.12 问题电路

7.RS触发器

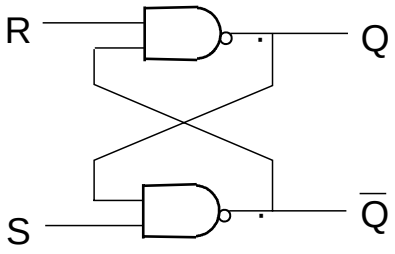


图1.13

RS触发器是一种危险的触发器，R=S=1会导致不稳定态，初始状态也不确定。在设计时尽量避免采用这种电路，或用如图1.14电路改进

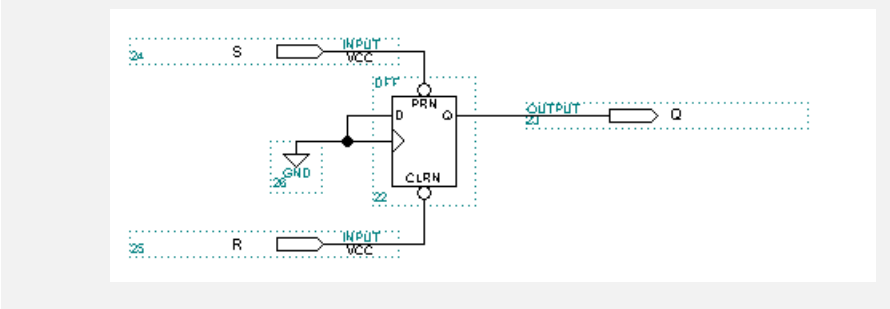


图1.14

可编程逻辑器件一般选用结构中的D触发器来完成设计。

我们认为，设计时最好从系统的角度来考虑，实现电路的功能，建议使用用VHDL直接描述所需要的设计。这样作，既安全，又具有极大的灵活性：

```
Process(nreset,clk)
Begin
 If    nreset = '0' then
       Rs <= '0';
 Elsif clk = '1' and clk'event then
            Case r&s is
                 When "00" =>
                     Rs <= rs;
                 When "01" =>
                     Rs <= '1';
                 When "10" =>
                     Rs <=    '0';
```

```
When "11" =>
   --这里可以自定义R=S=1的行为
When others =>
    Rs <= '0';
End case;
End if;
End precess;
```

8.上升沿检测

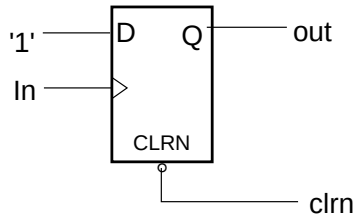


图1.15　上升沿检测

不建议采用这种电路检测信号的上升沿，因为IN为数据信号，上边容易有毛刺，使触发器误动作。

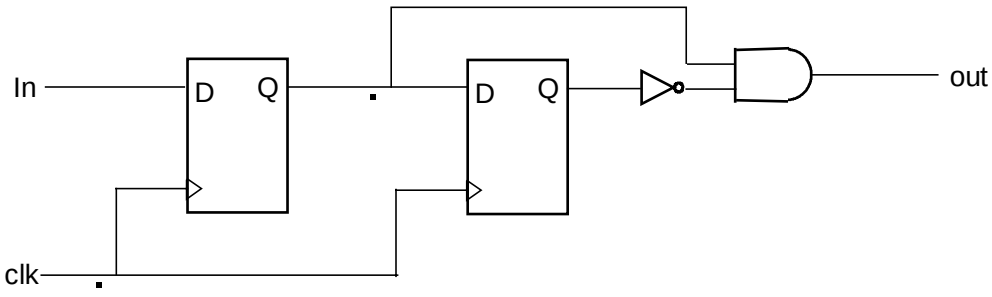


图1.16

采用时钟检测信号，出现0->1的变化即为上升沿。当被测信号与时钟相关时，可以不用第一个触发器。

9.下降沿检测

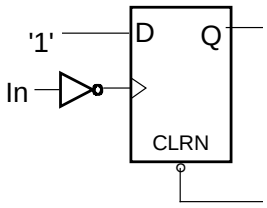


图1.17

同上升沿检测电路，这种电路对毛刺、干扰极为敏感。

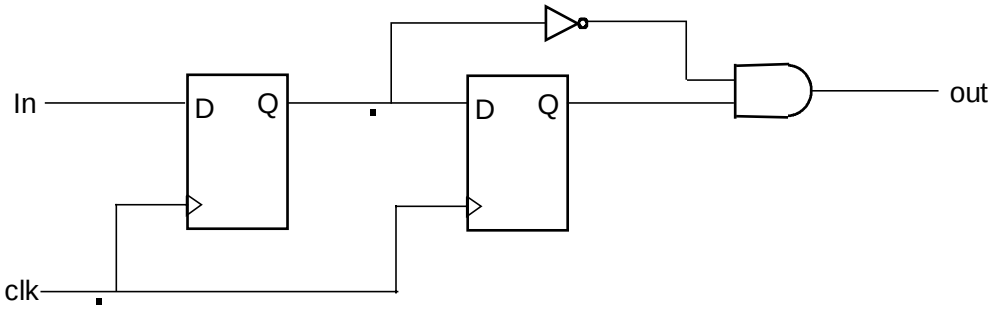


图1.18

采用时钟检测信号，出现1->0的变化即为下降沿。 当被测信号与时钟相关时，可以不用第一个触发器。

10.上升/下降沿检测

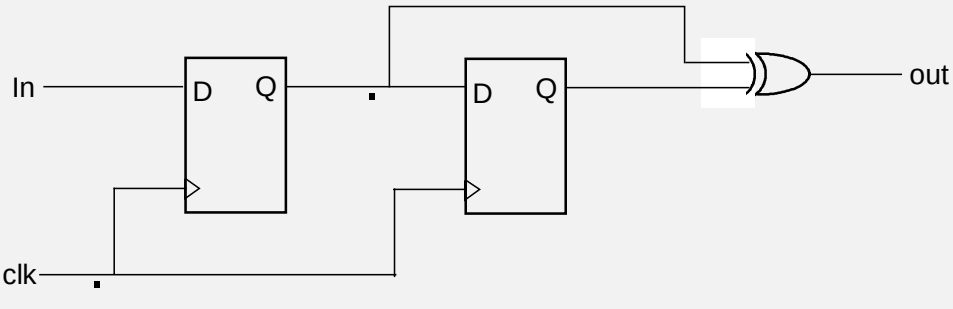


图1.19

采用时钟检测信号，出现变化即为上升/下降沿。 当被测信号与时钟相关时，可以不用第一个触发器。

11.对计数器的译码

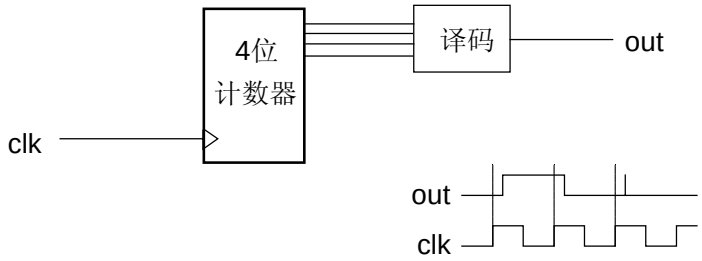


图1.20

对计数器译码，可能由于竞争冒险产生毛刺。如果后级采用了同步电路，我们完全可以对此不予理会。如果对毛刺要求较高，推荐采用Gray编码(PLD)或One-hot编码(FPGA)的计数器，一般不要采用二进制码.具体描述中，我们可以用状态机来描述，而利用逻辑综合工具来编码，有经验的选手可以自己强制定义状态机的编码。

12.门控时钟

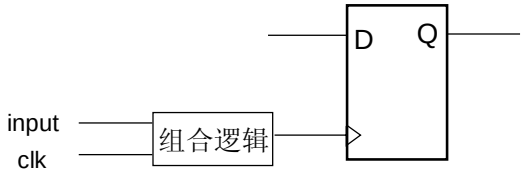


图1.21　门控时钟

门控时钟是非常危险的，极易产生毛刺，使逻辑误动作。 在可编程逻辑器件中，一般使用触发器的时钟使能端，而这样，并不增加资源，只要保证建立时间，可使毛刺不起作用。

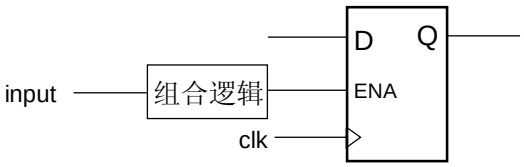


图1.22

13.锁存器

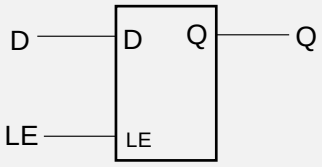


图1.23

锁存器是较危险的电路，没有确定的初始状态，输出随输入变化，这意味着毛刺可以通过锁存器。若该电路与其它D触发器电路相连，则会影响这些触发器的建立保持时间。除非有专用电路特别需要（其实总线锁存之类的功能用373之类的小规模IC更好），在设计内部，不要使用锁存器。

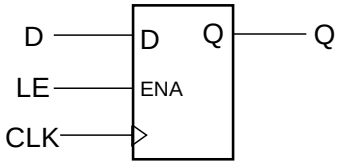


图1.24

引入时钟，可以实现锁存器的功能。

14 多级时钟或多时钟网络

由于时钟建立-保持时间的限制，FPGA设计中应尽量避免采用多时钟网络，或尽量减少时钟的个数。

图1.25是有问题的电路，这是一个含有险象的多级时钟的例子，多路选择器的输入是clk和clk的2分频，时钟是有SEL引脚控制的多路选择器输出的，在这两个时钟均为逻辑1时，当SEL线的状态改变时，存在静态冒险竞争现象，冒险竞争险象的程度取决于工作的条件。 图1.26是 改 进后的 电 路。

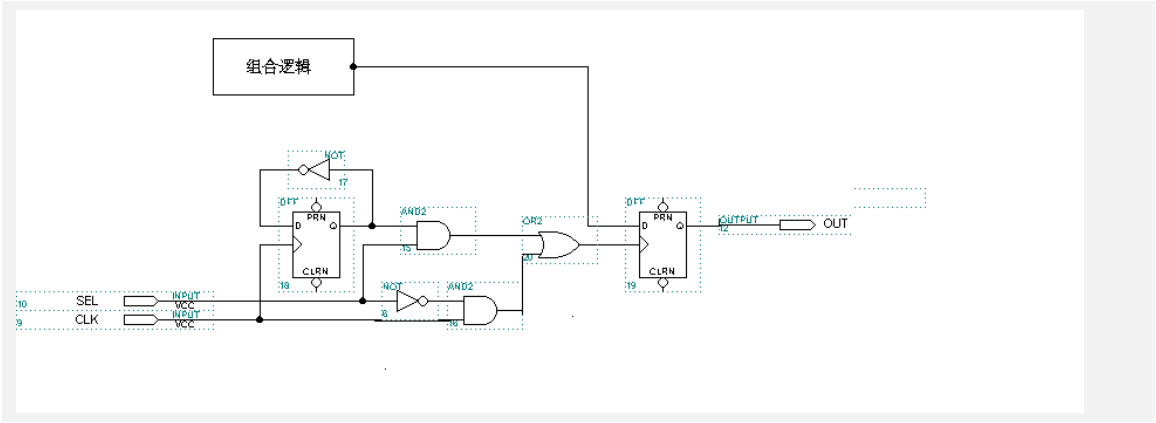


图1.25

改进：

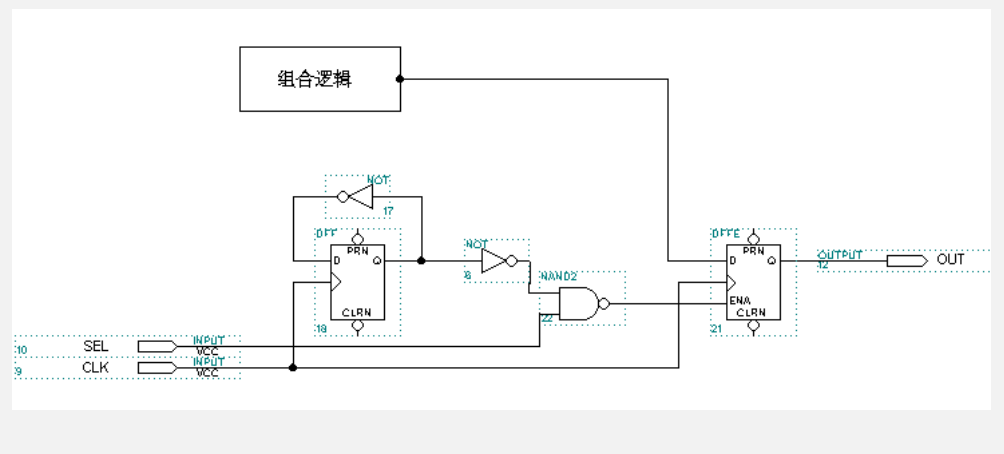


图1.26

总之，任何数字设计，为了成功地操作，时钟是关键。设计不良的时钟在极限的温度、电压或制造工艺的偏差下将导致错误的行为，而设计良好的时钟应用，则是整个数字系统长期稳定工作的基础。

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，在真正的同步系统情况下，由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。几乎所有的可编程逻辑器件中都有专用的全局信号引脚，它的特殊布线方式可以直接连到器件中的每一个寄存器。只要保证输入数据的建立和保持时间满足要求（相应可编程器件的数据手册中可查到），整个同步系统就可以在全局时钟的驱动下可靠工作，所以在可能的情况下，一定要使用全局时钟。

1.3 不建议使用电路

1 不建议使用组合逻辑时钟或门控时钟

组合逻辑很容易产生毛刺，用组合逻辑的输出作为时钟很容易使系统产生误动作。

2 不建议使用行波时钟

3 尽量避免采用多个时钟，多使用触发器的使能端来解决。

在可编程逻辑器件设计时，由于时钟建立应尽量避免采用多时钟网络，或者采用适当的措施减少时钟的个数，使用频率低的时钟尽量简化消除。

4 触发器的置/复位端尽量避免出现毛刺，及自我复位电路等，最好只用一个全局复位信号。

5 电路中尽量避免“死循环”电路，如RS触发器等。

6 禁止时钟在不同可编程器件中级连，尽量降低时钟到各个器件时钟偏差值。

同一个时钟在不同可编程器件中使用时不允许级连，而且应该保证该时钟到达不同可编程器件输入引脚的时钟偏差足够小。因为经过多个可编程器件的延时后，难以估计该时钟在链上各个触发器时钟

之间产生的大量时间偏移，如果这种时间偏移引起建立时间、保持时间难以满足的话，那么设计的逻辑就极有可能要失败了。

2 SET和RESET信号处理

在设计时应尽量保证有一全局复位信号，或保证触发器、计数器在使用前已经正确清零和状态机处于确知的状态。

寄存器的清除和置位信号，对竞争条件和冒险也非常敏感。在设计时，应尽量直接从器件的专用引脚驱动。另外，要考虑到有些器件上电时，触发器处于一种不确定的状态，系统设计时应加入全局复位/Reset。这样主复位引脚就可以给设计中的每一个触发器馈送清除或置位信号，保证系统处于一个确定的初始状态。需要注意的一点是：不要对寄存器的置位和清除端同时施加不同信号产生的控制，因为如果出现两个信号同时有效的意外情况，会使寄存器进入不定状态。

对于状态机设计，由于有可能存在一些状态对于系统而言是“非法的”（或称“无关的”），所以除了在状态机设计时充分考虑各种可能出现的状态以及一旦进入“非法”状态后可以强迫状态机在下一个时钟周期内进入“合法”状态（一般时初始状态）外，一定要保证系统初始化时状态机就处于“合法”的初始状态，这里最好的办法仍然是使用全局主复位信号强迫状态机进入已知的“合法”状态。下面给出一段包含状态机的全局时钟和复位的AHDL描述：

```
SUBDESIGN example
    (
        clock           : INPUT ;
        /reset          : INPUT ;
    )
    VARIABLE
     -- State Machine declaration
        poll                    : MACHINE WITH STATES
                                (Idle,s1,s2);
     -- Varible & Counter declaration
        var1                    : DFF;
        var2                    : DFFE;
        counter[5..0]           : DFF;

    BEGIN
     -- global reset & clock, initial set to 1
```

```
        var1.clk = GLOBAL(clock);
        var1.prn = GLOBAL(/reset);

-- global reset & clock, initial set to 0
        var2.clk = GLOBAL(clock);
        var2.clrn = GLOBAL(/reset);

-- global reset & clock for counter,    initial value set to 0
        counter[].clk = GLOBAL (clock);
        counter[].clrn = GLOBAL (/reset);
        IF counter[] == 53 THEN
                counter[] = 0;
        ELSE
                counter[] = counter[] + 1;
        END IF;

-- state machine global clock & reset, initial state set to idle
        poll.clk = GLOBAL (Clk25m);
        poll.reset = not GLOBAL (/Reset);
        CASE poll IS
                WHEN idle =>
                        。
                        poll = s1;
                        。
                WHEN s1 =>
                        。
                        poll = s2;
                        。
                WHEN s2 =>
                        。
                        poll = s1;
                        。
                WHEN OTHERS =>
                        poll = idle;
```

```
                END CASE;

END
```

3 时延电路处理

时延电路是指在可编程器件的设计中，为了能够满足电路之间时序配合的要求，利用可编程器件的内部资源而进行时序调整，如：利用线延时或者若干串联 Buffer 电路，使时钟或数据滞后一段时间。典型的，该时间即不能太长，又不能太短，以便满足“特定”需要。

然而，这种时延电路严重依赖器件工艺，也依赖每次的布线结果，给设计带来许多麻烦，有时会引起严重后果。因此，必须正确处理时延电路：

1 在设计中应尽量避免时延电路，绝大多数时延电路是由设计者在设计之初考虑不完善造成的。例如，在设计中没必要地存在多个时钟电路，模块划分不合理，关键电路关键时序考虑不周等。

我们的设计经常是在功能模块划分完成之后，就急急忙忙去做具体电路，许多关键时序并没有考虑清楚，这不可避免的造成设计多次反复。正确的做法是：

设计目标分析 ⟶ 功能模块划分 ⟶ 确定关键电路时序和模块间接口时序 ⟶ 具体电路设计

设计电路，尤其是数字电路，最关键的一环是：设计各模块间的接口时序，确定关键电路的时序。这个工作必须在具体电路设计之前确定下来。

“时序是事先设计出来的，而不是事后测出来的，更不是凑出来的”。

2 若实在无法，则尽量采用高频电路，对所需信号加触发器进行延时。该延时只跟时钟频率和触发器个数有关，而与工艺基本无关。

4 全局信号的处理方法

全局信号处理的原则是：时钟信号、异步清零、置位信号上不允许存在毛刺；不允许异步清零、置位信号同时有效。

在下述几种情况下，时钟信号、异步清零、置位信号上可能会有毛刺：

（1） 时钟信号、异步清零、置位信号为组合逻辑输出

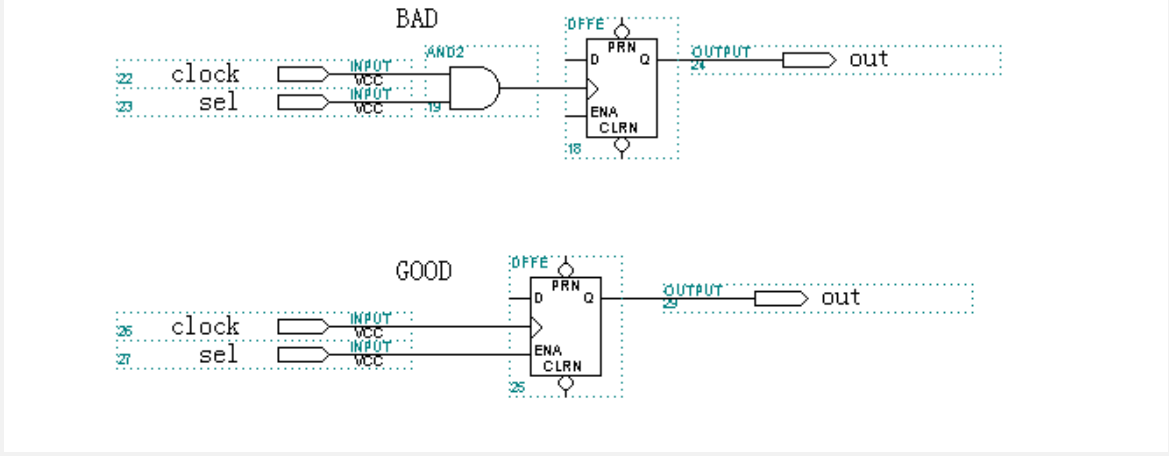
由于组合逻辑是电平敏感的，比较容易产生毛刺，而组和逻辑的细小毛刺一旦经过时序电路则其对电路的影响则会放大。因此在设计中对时钟信号、异步清零、置位信号这些对时序电路来讲非常重要的信号应尽量采用同步电路，而对于非用组合逻辑不行的地方则必须用卡诺图严格的分析时序电路，确定彻底消除竞争与冒险后才可引入到时序电路中使用。下面举例说明：

- 对于必须用组合逻辑的输出作为时钟、异步清零、置位信号的电路，对其组合电路的输出必须采用卡诺图进行严格的分析，保证彻底消除了竞争与冒险后才可引入到时序电路中使用。对于时钟信号、异步清零、置位信号是多个信号中选择的情况可按下述方法同步化

对组合电路产生的时钟信号的处理：

情况1：同一个时钟源，通过组合逻辑控制它的通断，如下图：

图 1.27　　通过组合逻辑控制它的通断



上面的图中，触发器的时钟是由CLOCK与SEL相与后的输出，目的是通过控制触发器的时钟的通断达到控制触发器的输出的目的，这种情况下由于CLOCK与SEL不是严格同步的，则有可能会在触发器的时钟脚产生毛刺，而在FPGA中，触发器对时钟端的毛刺是敏感的，上述毛刺可能导致触发器的误动作。

在下面的图中，触发器的时钟是CLOCK，而将SEL信号与触发器的使能端连接，这样同样达到了通过SEL信号控制触发器的输出的目的，但由于CLOCK不会产生毛刺，可以保证触发器的可靠触发。

情况2：通过组合逻辑对触发器的时钟在多个时钟中选择一个，如下图所示：

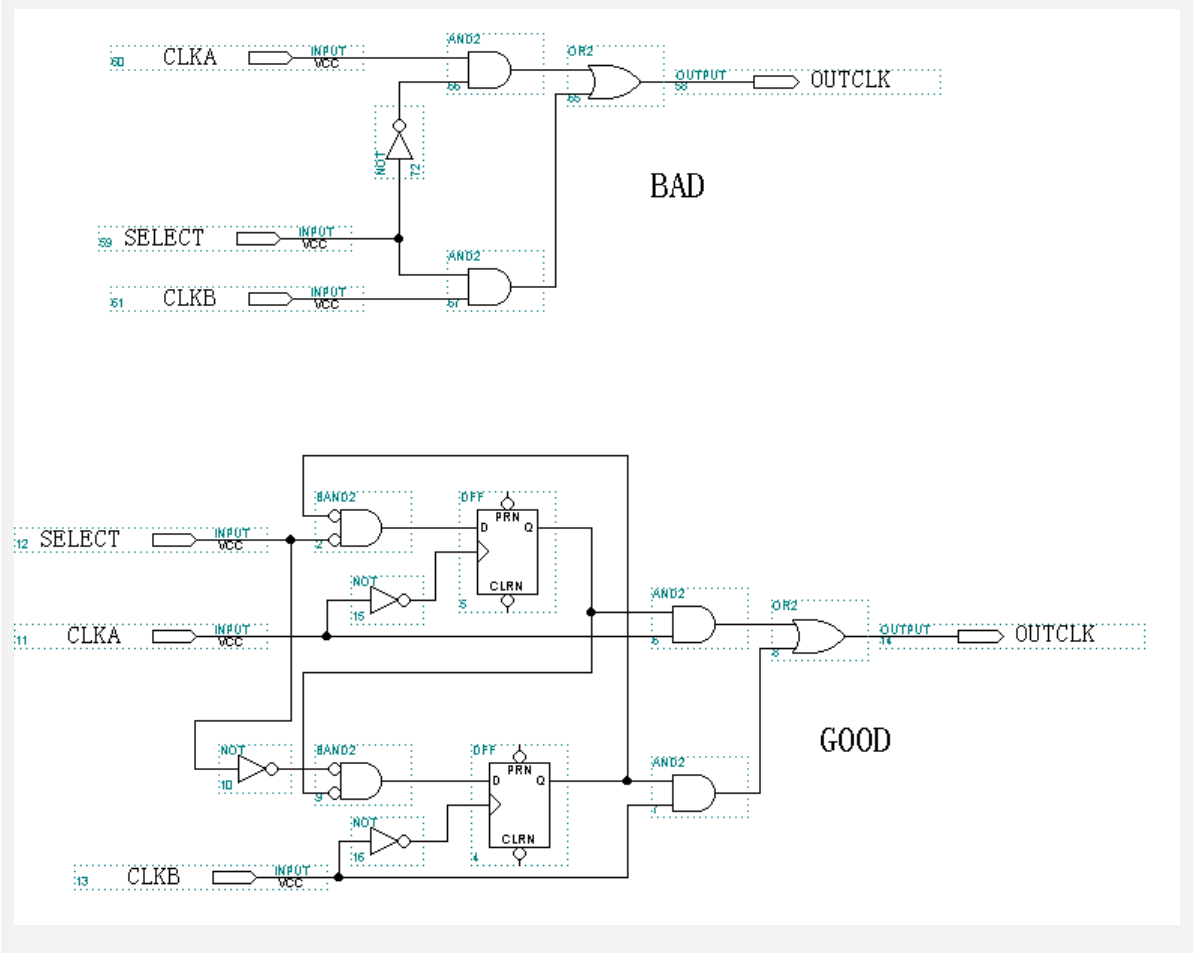


图1.28　组合逻辑在多个时钟中选择一个

上图中，标为BAD的一个由于直接用组合逻辑实现在CLKA、CLKB两个时钟中二选一的功能，而组合电路由于不同路径的延迟不同，所以在电路的时钟输出很容易产生毛刺；而标为“GOOD”的一个由于对选择信号SELECT分别用两个触发器进行了同步化，所以在时钟的输出端不会产生毛刺。

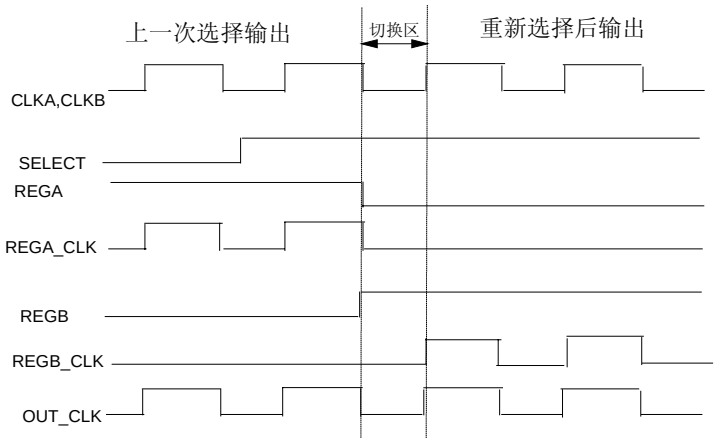


图 1.29

图1.29是图1.28中GOOD电路的原理时序图。其中REGA表示接CLKA的触发器，其对应的与门输出为REGA_CLK；同样定义REGB和REGB_CLKB。很显然，当REGA或REGB前后有限“漂移”时，OUT_CLK是不会出现毛刺得。

对GOOD电路，有两个问题：

1. SELECT如果在CLK下降沿左右发生变化，该电路能否正常？

2. 该电路的触发器能采用上升沿触发吗？

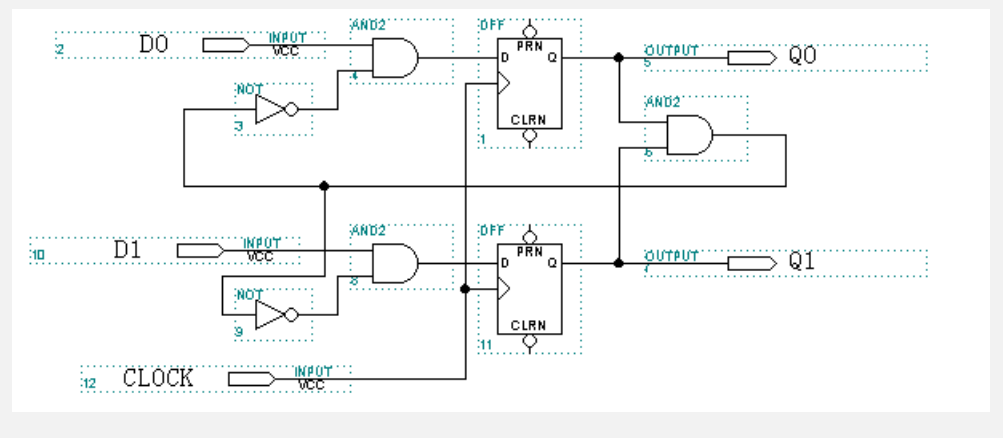
（2）使用自我清除、自我置位和自我钟控的寄存器；

• 建议尽量使用触发器的使能端来达到钟控的目的。

对于自我清除、自我置位，最好通过组合逻辑将其转变成对输入端数据的置位与清除，从而提高电路的可靠性，具体说明如下二图：

图1.30是一个自我清除的电路图，当输出端均为“1”时，电路复位，这样消除了产生毛刺的隐患，但使输出端体现出复位延迟了一个CLOCK。

图1.31是一个自我置位的电路图，当输出端均为“0”时，电路置位，这样做输出端体现出置位延迟了一个CLOCK，但消除了产生毛刺的隐患。



图

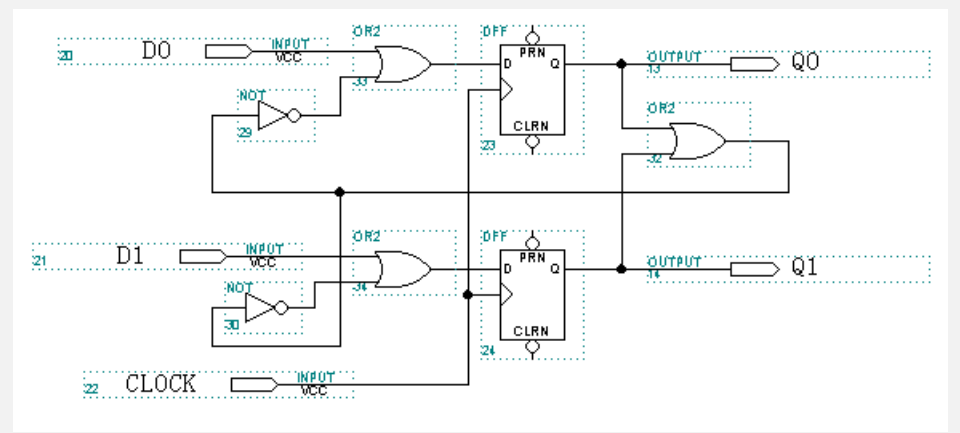


图 1.31

（3）时钟信号、异步清零、置位信号由外部输入，输入时有毛刺；

对于由外部输入的有毛刺的时钟、异步清零、置位信号通常的处理方法是使用内部触发器锁存，使之同步化。

1 时序设计的可靠性保障措施

（1）时钟偏差要加以控制

在同步电路里，时钟信号要连接到所有的寄存器，触发器以及锁存器等器件上。这些巨大的负载就象一个大电容加在时钟线上，再加上时钟线本身的分布电容和电阻，这样时钟线就象分布的RC线。由于RC线的延时是线长的函数，这样就使得连到同一根时钟线上的时钟由于距离时钟源的远近不一而产生不同的延时。因而造成了同一时钟到达各个器件的时间不一致，使得各个以时钟为基准器件的动作也不一致，而造成时序上的混乱。这就是同步电路时钟偏差。

要让同步电路可靠地运行，就要对时钟偏差进行控制，以使时钟偏差减小到可用的范围。影响时钟偏差的主要有以下几个因素：

1、用于连接时钟树的连线

2、时钟树的拓扑结构

3、时钟的驱动

4、时钟线的负载

5、时钟的上升及下降时间

在通常的FPGA设计中对时钟偏差的控制主要有以下几种方法：

1、控制时钟信号尽量走可编程器件的全局时钟网络。在可编程器件中一般都有专门的时钟驱动器及全局时钟网络，不同种类、型号的可编程器件，它们中的全局时钟网络数量不同，因此要根据不同的设计需要选择含有合适数量全局时钟网络的可编程器件。一般来说，走全局时钟网络的时钟信号到各使用端的延时小，时钟偏差很小，基本可以忽略不计。

2、若设计中时钟信号数量很多，无法让所有的信号都走全局时钟网络，那么可以通过在设计中加约束的方法，控制不能走全局时钟网络的时钟信号的时钟偏差。如在UCF文件中加上：NET *** MAXSKEW=3；

一般而言，时钟偏差的控制值应按保持时间要求来计算： △T ＜ T1 － T_hold 。

注意：Xilinx器件应用中，MAXSKEW约束必须直接加在触发器的时钟端上，MAXSKEW约束不能通过任何门电路，尤其是对芯片PAD进行约束时，应对IBUF之后的信号加约束，许多人在初学时经常忘记这一点。

3、异步接口时序裕度要足够大

局部同步电路之间接口都可以看成是异步接口，比较典型的是设计中的高低频电路接口、I/O接口，那么接口电路中后一级触发器的建立-保持时间要满足要求，时序裕度要足够大。

4、在系统时钟大于30MHz时，设计难度有所加大，建议采用流水线等设计方法。

采用流水线处理方式可以达到提高时序电路的速度，但使用的器件资源也成倍增加。

5、要保证电路设计的理论最高工作频率大于电路的实际工作频率；

最高频率是指设计软件经综合，布局，布线后，软件计算出的可工作的最高频率。比如经过综合及布局布线后，最长的延时为20ns,则其理论最高工作频率为50MHz。需要指出的是：此频率更设计有很大关系，良好的设计可使器件工作在更高的频率范围。

所谓电路的实际工作频率是指所设计的逻辑电路实际要达到的工作频率，既实际电路中所使用的时钟频率。

要使时序电路可靠地工作就必须使理论最高工作频率大于电路的实际的工作频率，但理论要高于实际的值是多少，这需要根据实际情况而定。主要要考虑的是时钟的最大可能抖动是多少。

2 ALTERA参考设计准则

Ensure Clock, Preset, and Clear configurations are free of glitches.
Never use Clocks consisting of more than one level of combinatorial logic.
Carefully calculate setup times and hold times for multi-Clock systems.
Synchronize signals between flipflops in multi-Clock systems when the setup and hold time requirements cannot be met.
Ensure that Preset and Clear signals do not contain race conditions.
Ensure that no other internal race conditions exist.
Register all glitch-sensitive outputs.
Synchronize all asynchronous inputs.
Never rely on delay chains for pin-to-pin or internal delays.
Do not rely on Power-On Reset. Use a master Reset pin to clear all flipflops.
Remove any stuck states from state machines or synchronous logic.