门规

别担心，这里不是黑帮。

我喜欢把寄存器比喻为门。所谓的电路就是由一堆这样的门包围着的。每个门都有一个相应的时钟来驱动，在相同间隔后，门将会打开。一些信号从一个门出来，经过一系列的变化（组合逻辑）变成一个新的信号，走到另一个门口，当门打开的时候，进入。这就是 FPGA 里面发生的一切。或许速度会快一些，或许情况会复杂一些，但是大体就是这么回事而已。而我们要保证的就是所有的信号都可以高高兴兴来，安安全全回去。不会迟到，也不会早退，更不会被门夹到，就万事大吉了。但是事情容易说不容易做，所以需要注意一些细节方面的问题。如果因为你把逻辑给做错了，那我们就不提了。而有些事情，即使你感觉一切都做得足够完美了，却还是没有结果。在这里提一些会有好处。

门规之组合电路篇

虽然我们要讨论的是门，但其实我们更关注的应该是门与门之间发生的一切。因为门的作用只是把阶段和另一个阶段分割开来，而更多的事情，其实是在这个阶段里面做的。门与门之间，或者门外面的，就都是这些组合电路了。组合电路最终影响了你的时钟能够飙到多快。

1. 进来的时候，请随手关门

这是一个再简单不过的规则。在进来的时候，请关一下门。相信如果你呆在一个有一部分没有被包围在墙和门里面的房子里，一定很没安全感。尤其是在门外面的东西吧，你会整夜整夜的睡不着觉。所以进来的时候，最好可以随手关门，把自己包围在一个安全的环境里面。所以信号也是这样，信号是很敏感很没安全感的东西。所以，进来的时候最好被时钟打一下。这在 FPGA 设计中是非常重要的，尤其是复位信号。外部信号的触发事件是随机的，连它自己都不晓得什么时候那个 reset 键就被按了一下，加上进来以后的线路会有一点延迟，恰好在时钟触发的时候，前面一个信号被触发了，后一个信号恰好被门夹到。就好像指挥抬手的时候，钢琴开始弹奏了，小提琴手还在摆 pose。 这样导致的结果就是大家不是同时开始工作。

门分很多种，材质不一样（时钟频率），大小不一样（时钟相位），还有很多属性不同(全局时钟？)。所以为了进入不同的区域，你必须关一下相应的门。这其实很容易理解，你进厨房，关厕所的门，就有点奇怪了。还是以复位信号为例，*对所有你需要用到复位信号的时钟域，最好可以被相应的时钟打一下。*

提出这样一点，是因为它经常被忽略，而且发生问题的状况可能变成随机的。如果你的设计仿真没有问题，而在板子上发生一些随机的事件，那么很有可能是你的异步信号造成的。

2. 在门与门之间请不要来回走动

门（寄存器）与门之间是什么？是组合逻辑电路。信号的传递就是不断的打开一个门，然后进入另外一个门，这样一级一级的不断传递，你才可以一直从起点走到终点。在有些时候也许你并没有注意，你在到达另外一个门之前，你转了一个圈又回来了。虽然这只是一个简单

的圈，但是这个动作会被无限循环，因为这个动作没有被时钟限制。所以导致的后果可想而知了。举一个最简单的例子：

D = A + B + C;

Reg C = D + B;

A 和 B 都是输入，而 C 是输出，你会发现其实这个 C 偷偷的回去转了一圈，而结果是使自己又多做了几次循环。而最终结果是多少要看这段逻辑自身的延迟和你的时钟频率，对不起，谁都不可能预测结果。而且每次编译的结果都会不同，因为编译导致的电路延迟是随机的。但是如果我们把 D 也用门关一下（寄存器），那么结果就会舒服了。

Reg D = A + B + C;

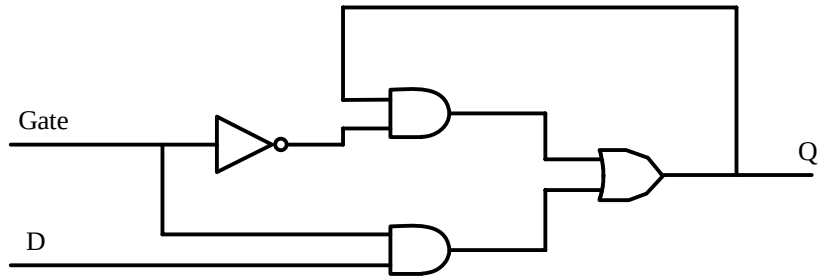
Reg C = D + B;

不妨在 QuartusII 里面尝试这两种电路。 工具会报告一个 combinational loop 的警告给你。也可以尝试做一些仿真体会一下。

3. 不要为了省钱，用假门 （Latch Vs Register）

让我们把这个门再好好的研究一下，首先是定义。我这里说的门更多是 flip-flop register. 作为现成的资源，它遍布于器件的四面八方。另外有一种叫 Latch 的东西，我把它叫做假门。在普通数字电路中，人们会喜欢用到它。因为它的结构相对简单，所以资源上比较节约。所以我们需要说说这个假门的问题。

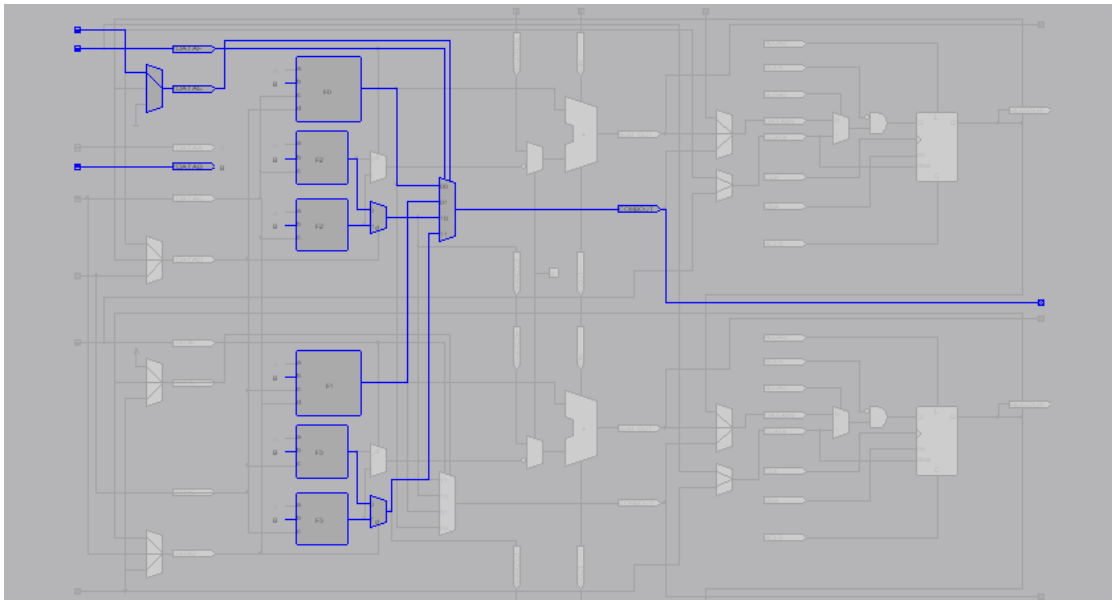
那么首先需要了解一下什么是假门，其实就是一个偷工减料的门。我们来看一个简单的图：



这确实是一个再简单不过的逻辑了，比 flip-flop 的那个寄存器要简单许多。当 gate 为高的时候，Q 变成 D，否则，就保持 Q 的值。很显然这是一个用电平来保存数据的一个逻辑，但是为什么我们不喜欢它呢？

1. 不稳定，和 flip-flop 做一个比较，register 是在开门的时候数据进去的，然后这个门马上就关掉了，所以数据只要在开门的前后保持稳定，那么从输出看都是可以保持一致的。但是 Latch 的结构中，当 Gate 为高的时候，Q 的数据会随着 D 的变化而变化。所以他无法保持一个稳定的状态。
2. 它给工具制造了麻烦。因为电路是由你设计的，工具并无法知道你的思路，所以他不会知道你想要的到底是什么时候的输入。因为你现在门是一直那么开着的。在这过程中，狐狸也来过了，狼也来过了，老鼠也来过了，工具不知道究竟以什么为标准来判断这个电路是不是能符合时序要求。这样的话，或许你会很幸运的在一颗片子上面完成任务。但是一旦换一颗，情况就变了。
3. 最后，作为一个精打细算的人，我们需要知道他是不是真的就省了那些传说中的资源了。大家不妨自己做一个 Latch 出来（QuartusII 的 megacore 中是可以找到 LPM_LATCH 的）。

然后把它编译一下出来看看。你会发现事实上你根本找不到这样的一个电路（ Resource property editor ）你只能看到这样一个有点庞大的图:



真是不划算，你发现其实整个查找表都给用上了。而后面一个简简单单的 register 可怜巴巴的放在那里用不到。

作为 FPGA 设计中比较独特的一个原则就是，这个世界不是你创造的，所以你必须去适应它，而不是顽固自己的意图。

换句话说：坏人也是人，不是说你不能做，但都已经告诉你坏人不好了，你干嘛还要去做呢？

留一个问题给大家去尝试，或者玩一下。什么状况下，或者说怎么写代码的时候，可以把 Latch 写出来。

4. 除了门，其他的一切过程都只是暂时的

在 FPGA 中，除了门与门之间的延迟是固定的（时钟频率）外，一切的组合电路的延迟都是不确定的。你不能依靠一次编译的延迟结果来做你的设计，虽然这样看上去的效率会非常高。就好像接力跑一样，可以一棒一棒的传下去。但是电路是需要可以不断重复的一个过充。相信每次接力跑交棒的时间点都是不同的吧，你再做一次编译以后，你会发现整个世界都变了。所以在设计中，尽量避免使用这种手段。

延迟链，经常会使用这种手段，比如几级非门来增加延迟。但是它也是具有不稳定性的，所以在不到万不得已的状况下，不要用它来增加认为的延迟。延迟链在 ASIC 中另外一种用途是增加扇出。而这在 FPGA 设计中是画蛇添足了。因为布线资源中已经加入了 buffer 了。

记住这样一个规律就好了，凡是没有被门关过的信号都是不稳定的，都只是暂时的。