門規

別擔心，這裏不是黑幫。

我喜歡把寄存器比喻爲門。所謂的電路就是由一堆這樣的門包圍著的。每個門都有一個相應的時鐘來驅動，在相同間隔後，門將會打開。一些信號從一個門出來，經過一系列的變化（組合邏輯）變成一個新的信號，走到另一個門口，當門打開的時候，進入。這就是 FPGA 裏面發生的一切。或許速度會快一些，或許情況會複雜一些，但是大體就是這麼回事而已。而我們要保證的就是所有的信號都可以高高興興來，安安全全回去。不會遲到，也不會早退，更不會被門夾到，就萬事大吉了。但是事情容易說不容易做，所以需要注意一些細節方面的問題。如果因爲你把邏輯給做錯了，那我們就不提了。而有些事情，即使你感覺一切都做得足夠完美了，卻還是沒有結果。在這裏提一些會有好處。

門規之組合電路篇

雖然我們要討論的是門，但其實我們更關注的應該是門及閘之間發生的一切。因爲門的作用只是把階段和另一個階段分割開來，而更多的事情，其實是在這個階段裏面做的。門及閘之間，或者門外面的，就都是這些組合電路了。組合電路最終影響了你的時鐘能夠飆到多快。

1. 進來的時候，請隨手關門

這是一個再簡單不過的規則。在進來的時候，請關一下門。相信如果你呆在一個有一部分沒有被包圍在牆和門裏面的房子裏，一定很沒安全感。尤其是在門外面的東西吧，你會整夜整夜的睡不著覺。所以進來的時候，最好可以隨手關門，把自己包圍在一個安全的環境裏面。所以信號也是這樣，信號是很敏感很沒安全感的東西。所以，進來的時候最好被時鐘打一下。這在 FPGA 設計中是非常重要的，尤其是重定信號。外部信號的觸發事件是隨機的，連它自己都不曉得什麼時候那個 reset 鍵就被按了一下，加上進來以後的線路會有一點延遲，恰好在時鐘觸發的時候，前面一個信號被觸發了，後一個信號恰好被門夾到。就好像指揮抬手的時候，鋼琴開始彈奏了，小提琴手還在擺 pose。 這樣導致的結果就是大家不是同時開始工作。

門分很多種，材質不一樣（時鐘頻率），大小不一樣（時鐘相位），還有很多屬性不同(全局時鐘？)。所以爲了進入不同的區域，你必須關一下相應的門。這其實很容易理解，你進廚房，關廁所的門，就有點奇怪了。還是以重定信號爲例，*對所有你需要用到重定信號的時鐘域，最好可以被相應的時鐘打一下。*

提出這樣一點，是因爲它經常被忽略，而且發生問題的狀況可能變成隨機的。如果你的設計仿真沒有問題，而在板子上發生一些隨機的事件，那麼很有可能是你的非同步信號造成的。

2. 在門及閘之間請不要來回走動

門（寄存器）及閘之間是什麼？是組合邏輯電路。信號的傳遞就是不斷的打開一個門，然後進入另外一個門，這樣一級一級的不斷傳遞，你才可以一直從起點走到終點。在有些時候也許你並沒有注意，你在到達另外一個門之前，你轉了一個圈又回來了。雖然這只是一個簡單

的圈，但是這個動作會被無限迴圈，因爲這個動作沒有被時鐘限制。所以導致的後果可想而知了。舉一個最簡單的例子：

D = A + B + C;

Reg C = D + B;

A 和 B 都是輸入，而 C 是輸出，你會發現其實這個 C 偷偷的回去轉了一圈，而結果是使自己又多做了幾次迴圈。而最終結果是多少要看這段邏輯自身的延遲和你的時鐘頻率，對不起，誰都不可能預測結果。而且每次編譯的結果都會不同，因爲編譯導致的電路延遲是隨機的。但是如果我們把 D 也用門關一下（寄存器），那麼結果就會舒服了。

Reg D = A + B + C;

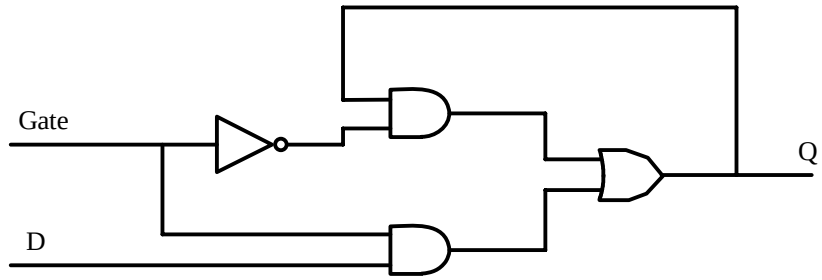
Reg C = D + B;

不妨在 QuartusII 裏面嘗試這兩種電路。 工具會報告一個 combinational loop 的警告給你。也可以嘗試做一些仿真體會一下。

3. 不要爲了省錢，用假門 （Latch Vs Register）

讓我們把這個門再好好的研究一下，首先是定義。我這裏說的門更多是 flip-flop register. 作爲現成的資源，它遍佈於器件的四面八方。另外有一種叫 Latch 的東西，我把它叫做假門。在普通數位電路中，人們會喜歡用到它。因爲它的結構相對簡單，所以資源上比較節約。所以我們需要說說這個假門的問題。

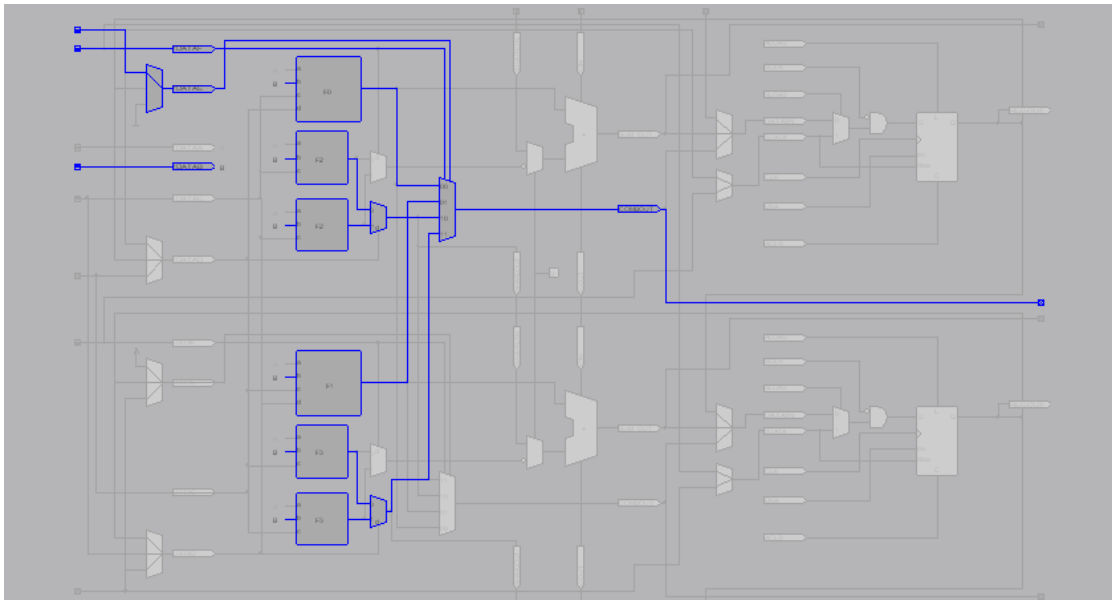
那麼首先需要瞭解一下什麼是假門，其實就是一個偷工減料的門。我們來看一個簡單的圖：



這確實是一個再簡單不過的邏輯了，比 flip-flop 的那個寄存器要簡單許多。當 gate 爲高的時候，Q 變成 D，否則，就保持 Q 的值。很顯然這是一個用電平來保存資料的一個邏輯，但是爲什麼我們不喜歡它呢？

1. 不穩定，和 flip-flop 做一個比較，register 是在開門的時候資料進去的，然後這個門馬上就關掉了，所以資料只要在開門的前後保持穩定，那麼從輸出看都是可以保持一致的。但是 Latch 的結構中，當 Gate 爲高的時候，Q 的資料會隨著 D 的變化而變化。所以他無法保持一個穩定的狀態。
2. 它給工具製造了麻煩。因爲電路是由你設計的，工具並無法知道你的思路，所以他不會知道你想要的到底是什麼時候的輸入。因爲你現在門是一直那麼開著的。在這過程中，狐狸也來過了，狼也來過了，老鼠也來過了，工具不知道究竟以什麼爲標準來判斷這個電路是不是能符合時序要求。這樣的話，或許你會很幸運的在一顆片子上面完成任務。但是一旦換一顆，情況就變了。
3. 最後，作爲一個精打細算的人，我們需要知道他是不是真的就省了那些傳說中的資源了。大家不妨自己做一個 Latch 出來（QuartusII 的 megacore 中是可以找到 LPM_LATCH

的）。然後把它編譯一下出來看看。你會發現事實上你根本找不到這樣的一個電路（ Resource property editor ）你只能看到這樣一個有點龐大的圖:



真是不划算，你發現其實整個查找表都給用上了。而後面一個簡簡單單的 register 可憐巴巴的放在那裏用不到。

作爲 FPGA 設計中比較獨特的一個原則就是，這個世界不是你創造的，所以你必須去適應它，而不是頑固自己的意圖。

換句話說：壞人也是人，不是說你不能做，但都已經告訴你壞人不好了，你幹嘛還要去做呢？

留一個問題給大家去嘗試，或者玩一下。什麼狀況下，或者說怎麼寫代碼的時候，可以把 Latch 寫出來。

4. 除了門，其他的一切過程都只是暫時的

在 FPGA 中，除了門及閘之間的延遲是固定的（時鐘頻率）外，一切的組合電路的延遲都是不確定的。你不能依靠一次編譯的延遲結果來做你的設計，雖然這樣看上去的效率會非常高。就好像接力跑一樣，可以一棒一棒的傳下去。但是電路是需要可以不斷重複的一個過充。相信每次接力跑交棒的時間點都是不同的吧，你再做一次編譯以後，你會發現整個世界都變了。所以在設計中，儘量避免使用這種手段。

延遲鏈，經常會使用這種手段，比如幾級反閘來增加延遲。但是它也是具有不穩定性的，所以在不到萬不得已的狀況下，不要用它來增加認爲的延遲。延遲鏈在 ASIC 中另外一種用途是增加扇出。而這在 FPGA 設計中是畫蛇添足了。因爲佈線資源中已經加入了 buffer 了。

記住這樣一個規律就好了，凡是沒有被門關過的信號都是不穩定的，都只是暫時的。