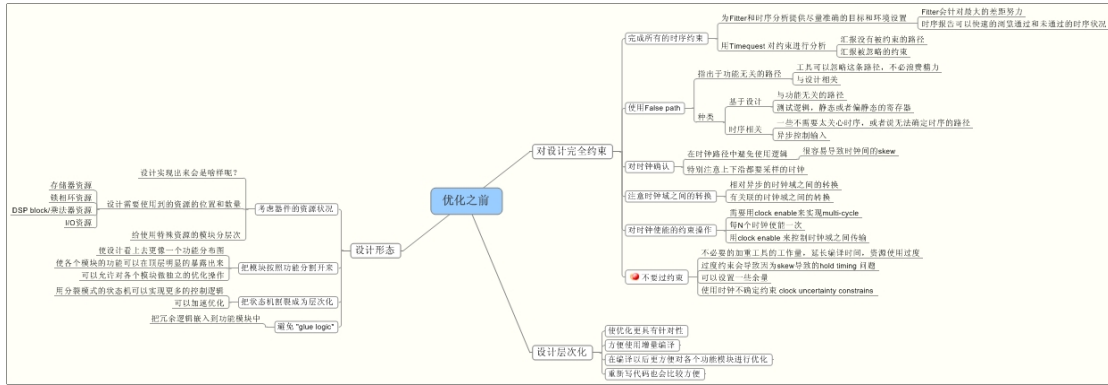


8. 優化那些事兒

優化是一個很麻煩的問題，因為這個話題非常的雜，細碎。好像我們介紹一個人，你會一下子不知道從什麼方面說比較好，因為他會同時擁有不同的身份，存在於社會當中。所以，我儘量的完成了這麼一張圖表，讓大家對優化這個懸浮于平衡中間的一個遊戲方法有一定的瞭解。平衡，其實是優化中的一個最關鍵的詞。當然針對不同的應用，我們會有完全不同的需求，平衡點也會有算偏向。但是畢竟不能矯枉過正，太過苛求，否則只能是過猶不及了。這麼說，似乎很沒意思，讓我們來說三個考量，時序，資源，功耗。這就是優化中的三個平衡極限。在有的設計中，演算法對時序有要求，所以會要求設計能跑在一定的時鐘頻率上，這就需要對時序進行優化。有的公司在設計的後期開始考慮成本的問題，會希望選擇儘量小的器件，那麼這個時候，資源消耗變成了重點。而在手提式器件的設計中，功耗是至關重要的。而這三點，是沒可能同時做到的。爲了達到某種目的，你必須要付出其他的代價。所以，在做優化之前，做好一個優化目標是很有必要的。當然最基本的是時序，和資源。在這裏我們比較重點的討論這兩方面的話題。大家一定看到了前面這種讓你暈得亂七八糟的圖，我的任務，就是把他們解釋一下。



● 优化之前

在提优化之前，我們當然需要有一個提供优化的基本形態，就是你的設計。如果你的設計還沒怎麼完成，大可不必就著急的開始优化。因為每次編譯都會把你的优化努力隨機掉。而最好的优化方法，其實就是可以不优化。那就是把代碼寫的很优化，退而求其次，就是把代碼寫的容易优化。這裏又要提老掉牙的事情了，代碼要寫得有層次化，好處就不囉嗦了。那麼在寫代碼的時候需要考慮什麼問題呢？

首先是你使用的目標器件的資源狀況。通過一些小實驗，你可以知道，你寫出來的代碼，大概會實現成什麼樣。這對於你寫代碼有一種感官上的映射非常有幫助。然後就是一些特殊零件的數目和位置，比如記憶體（memory），計算器（DSP block），特殊的管腳資源(LVDS)。

其實是把模組按照功能分割開來。從頂層電路看起來，真個設計就是一些功能模組的組合，看上去和規劃的功能圖一模一樣。這樣做的好處，自然是不言而喻的。也比較符合常規的美學思想。

然後是狀態機的問題，儘量不要寫出太大的狀態機，寧願用一些小型的狀態機來相互關聯。（除非你希望不被老闆替換掉而寫出很炫的，不過那樣你自己以後也會很麻煩，因為你的記性並不像你想像中那麼可靠）工具會耗盡心機去實現你一個很宏偉的狀態機，結果可能還不能讓人足夠滿意。

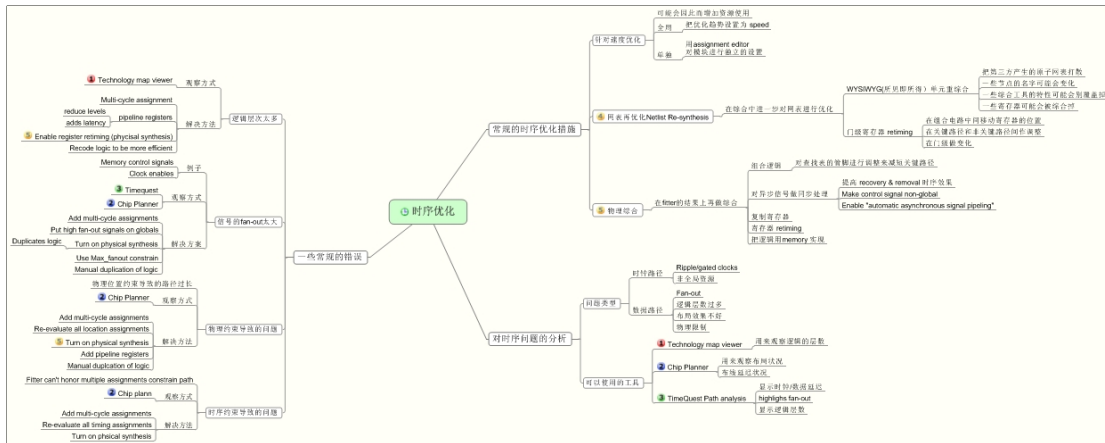
Glue logic. 就是兩個模組之間的黏合邏輯。這或許是很難避免的。但他們往往會成為優化的一個盲點。這種三不管地區，最容易導致髒亂差等社會問題。所以，儘量把它們放進某個模組的勢力範圍。

在知道了寫代碼的大概規則以後，我們來看看約束。約束本身並不是优化，而是給优化制定的一個目標。你需要工具達到什麼樣的目標，你當然盡可能的要告訴工具，否則讓他這麼放任自流下去，後果會很嚴重。所以最簡單的，你必須要提供所有的時序資訊。好比，所有的時鐘頻率，所有的管腳時序要求。這樣 fitter 才會有一個著眼點，針對距離目標最遠的路徑去努力。時序報告也才有可能報告哪些路徑是沒有通過要求的。我還是推薦大家使用 timequest 來做時序約束，好處是，它可能對你的時序約束和你的設計對照做分析，在做時序分析之前，先對你的約束做分析，然後告訴你，你有多少該做的事情而沒有做的（為被約束的路徑）還有多少你要求做的，而沒有被做的（被忽略的時序要求）。

這裏提出一個話題，false path，有人叫假路，有人叫錯路，都是英語惹得禍，我們還是叫它 false path。具體來說，就是不該存在的路，或者說，即使存在，也沒人去走的路。那麼對這種路，我們是可以讓工具看都不用看的。關於 false path，其實倒不用太過著急。大可以在發生問題以後，再去看是不是。否則讓你每條 false path 挑出來，其實也是個蠻無聊的事情。

對時鐘的約束，要重點關注兩個現象。首先是儘量少的在時鐘路徑上引入邏輯。這樣認為的造成了時鐘和時鐘之間的 skew。我們都知道這不是什麼好事情。另外就是一種上下沿都需要用來採集資料的時鐘。對於時鐘的約束有很多的地方需要注意，否則你的電路都不知道會飛到哪里去。不過這東西需要一些體驗，靠嘴巴說是說不出感覺的。

現在來說約束中最重要的一個關鍵，不要過約束。過約束的壞處一大堆，增加編譯時間（你可能不太在乎），資源使用過度（可能也還可以忍受），導致其他的時序問題（那可是個大麻煩了）如果你對自己的約束有些不太放心，又或者說可能器件和器件之間會有很細微的差別，你可以給約束做一些餘量，但是過約束是萬萬要不得的。



● 时序优化:

寫了半天，終於開始寫优化了，首先我們看看這個时序优化的問題。還是先看幾個關鍵字吧。

优化目標：一個是全局的設置，告訴工具你這個設計的优化偏向。當時序优化偏重要的時候，可以設置為 Speed。當然這使以犧牲 area, power 為代價的。也可以對某個獨立模組做局部設置，就是說在這個模組中，优化目標是時序。

網表再綜合(netlist re-synthesis): QuartusII 支持一些第三方的綜合工具（說實話，Quartus 自己的綜合工具，也還是可以的）。他們把代碼變成可以映射到 FPGA 上的網表檔，就是由一堆查找表和寄存器組成的東西。再綜合會把這些網表還原成爲反及閘結構的電路，然後再根據 quartus 綜合工具的演算法再做綜合。這當然未必是一件好事，可能會因此而破壞了原來工具中一些好的演算法做出來的邏輯而導致結果更差。

物理綜合：這是時序工具中效果最明顯的工具。它對 fitter 以後的結果，在關鍵路徑上，對電路進行調整。既然佈局佈線都做好了，還有什麼可以做的呢？有很多。首先是組合電路。組合電路，無非就是一堆查找表的連接。而佈局佈線本身是一個比較隨機的過程。在完成以後，我們會發現一些路徑是比較差的，那麼我們可以對這些路徑做一些調整，他們選擇更短的一條路徑，同時功能保持不變。對非同步控制信號做一級同步流水線。這樣可以規避一些 recovery 和 removal 的時序問題。還有一個功能是複製寄存器，比如說一個寄存器的 fanout 比較多，或者說，其中的一條線會連到比較遠的地方。那麼我們可以在那個比較遠的地方複製一個和這個寄存器完全一樣的寄存器，這樣可以大大提高時序效果。當然這需要犧牲一個寄存器了。還有一個好玩的事情，是寄存器的 re-timing，如果我們把寄存器看成流水線中的一層的話，它其實就是位於一條路徑中的某個點而已。他的前後兩個組合電路的延遲可能是不同的。而可能恰巧那個長的變成了一個關鍵路徑（就是大大的壞的一條），那麼我們是不是可以移動一下這個寄存器的位置，讓大家的時間可以平衡一下，而結果是時序上去了。但是在功能上會有一點小小的變化，所以這個選項還是慎用。

可能造成的幾種時序問題：

時鐘路徑導致問題，這個時鐘可能是個 gated clock,或者非全局時鐘。這個調整的餘地相對比較小。

資料路徑導致問題，這個比較複雜，首先可能是 Fan-out 太大導致的延遲過大。也有可能是邏輯門的層數過多導致。或者佈局佈線的隨機隨得不好，兩個點的距離很大。還有就是一些物理限制，比如 DSP block 之間的相對位置。

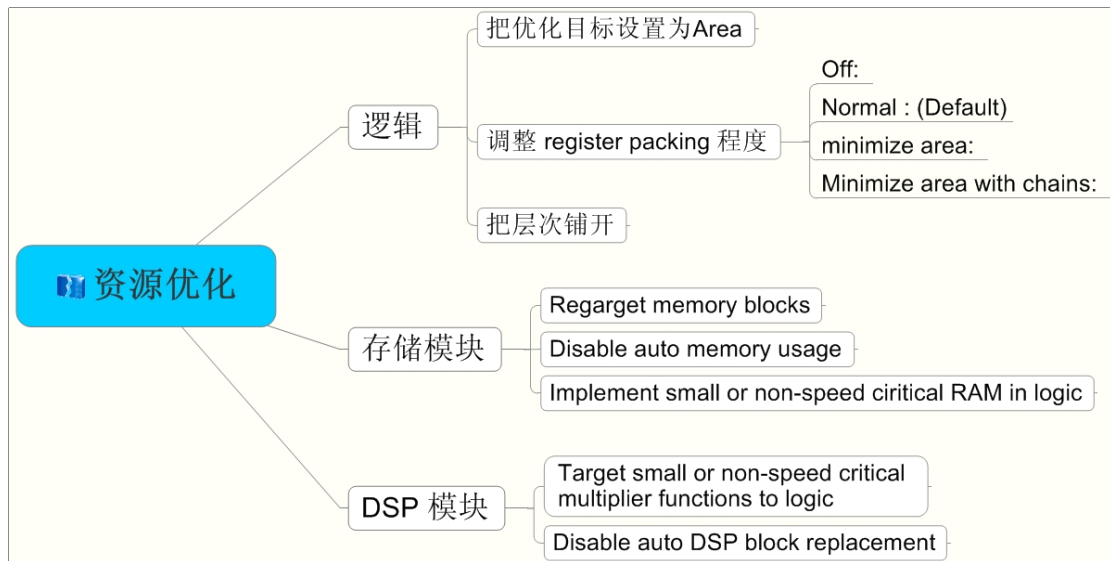
用來觀察時序問題的工具：

Technology map viewer: 這東西，實在是沒辦法用中文說了，我私自反了，恐怕你會恨死我，因為在 Quartus 裏面死活找不到了。它是可以用來觀察你的綜合結果以後的電路實現的。所以用它，你可以看到你的電路經過了多少層次的路徑。

Chip planner: 這是用來看你的電路具體在晶片上的實現的，通過它你可以觀察到相對之間的位置的距離，和連線上的延遲。

TimeQuest: 這是時序分析的最佳工具，無論是觀察時鐘或者路徑延遲，fanout, 或者邏輯層次，都可以使用它。雖然不一定比其他的工具更直觀，但是它是最全面的。

針對一些經典的可能性，表格裏面列出了一些，大家可以自己看一下。

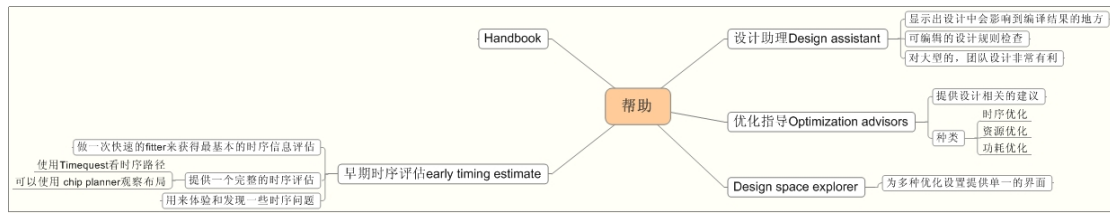


● 資源優化

和時序優化相對的，我們可以給 Quartus 一個優化指導方向，當你把它設置為 area 的時候，工具會儘量考慮資源的使用。可以做全局設置，也可以做局部設置。

Register packing: 這是一個針對 LE 資源優化的工具，在新的 Quartus 中一共有七級設置。比如把兩個互不相干的電路擠到同一個 LE 中。根據等級設置的不同，會有不同的資源消耗結果，當然付出的代價就是時序的降低。

當然對特定模組的使用，是需要一個整體規劃的。什麼樣的狀況下去使用記憶體資源，什麼樣的狀況下使用 DSP-block，都需要有一定的評估。更多的時候，其實在寫代碼的時候就已經做好了計畫。



● 幫手：

相信大家已經有點暈了，幸運的時候，我們不是一個人在戰鬥。我們還是有很多幫手的。讓我們來看看這些傢伙：

Design assistant：這是一個會提供給你一些提示的助手工具。他會顯示出設計中會導致編譯結果的地方。你可以自己定制一些設計規則，他會幫助你去監督你自己，尤其在一個團隊設計中，對大家設計的同一年性有比較好的效果。

Optimization advisor: 優化指導，會提出一些相關的優化選項的指導。當然這種指導是非常通用的，不可能非常針對你自己的應用，所以在是否使用這些建議的時候，還是要自己權衡一下。所以簡單的瞭解一些優化選項，還是有必要的。否則，莫名其妙的就被工具耍了。

Early timing estimate: 這個工具會做一些最基本的佈局佈線，這樣它可以很快的得到一些時序方面的資訊，這在你做時序約束，和logic lock設置的時候有很多的指導意義，同時也節省了一些時間。

Seeds: 誰都靠不上，我們靠老天。佈局佈線基本上是一個隨機的事情，沒有人能夠瞭解結果能使什麼樣子的。而seeds就好像一個賭博用的骰子一樣，告訴編譯從什麼狀況開始佈局。沒有人知道seeds 中的具體某個數字代表什麼樣的狀況，所以就是不斷的碰運氣吧。給大家一個迷信，似乎17這個數字特別好，結果會比較理想一些。

Design Space Explorer: 這個是一個實在沒辦法的辦法，但凡還有一線生路，最好不要用這個東西。他會自動的對你的設計根據一些設置，不斷的嘗試佈局佈線。比如使用不同的優化方法，不同的演算法，甚至不同的種子來做編譯，最後給你一個最符合約束要求的編譯結果。但是這樣的工作量有多大，大家可想而知了。如果你有比較多的時間，你可以試試把它扔在那裏跑個幾天。

Handbook: 啥也不用說了。